STEPS TOWARD A REVISED COMPILER-MONITOR SYSTEM II (CMS-2)

Vincent Cecil Secades

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

STEPS TOWARD A REVISED COMPILER-MONITOR SYSTEM II (CMS-2)

by

Vincent Cecil Secades

and

David Clark Rummler

Thesis Advisor: G. A. Kildall

June 1973

Steps Toward a Revised Compiler-Monitor System II (CMS-2)

by

Vincent Cecil Secades
Lieutenant Commander, United States Navy
B.S., Naval Postgraduate School, 1972

and

David Clark Rummler
Lieutenant, United States Navy
B.A., University of Texas, 1964

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1973

ABSTRACT

This paper describes a proposal for a revised Compiler Monitor System II (CMS-2). Primary emphasis is placed on design improvements to the CMS-2 compiler and language. Changes to the Monitor and Librarian which support the above improvements are discussed or implied where appropriate. A new concept is proposed, called multi-level programming, which allows the system designer to define the levels of language constructs which are appropriate for the various types of program modules in a large self-contained software system. The approach taken is to design a language and compiler-monitor system (CMS-2RS) which will facilitate the multi-level programming concept and the top-down programming method of software engineering in a production library environment.

TABLE OF CONTENTS

I. INTRCDUCTION 8

A. BACKGROUND 8

1. History of Compiler Monitor System II 8

2. History of the CMS-2 Language 9

B. GOALS AND OBJECTIVES 9

1. Thesis Objectives 9

2. Project Goals 10

II. CMS-2 SYSTEM AND LANGUAGE DESIGN 11

A. CMS-2 SYSTEM DESIGN 11

1. MS-2 Monitor System 11

2. CMS-2 Compiler System 11

3. CMS-2 Librarian 14

4. Object Code Loaders 15

5. Tape Utility 15

6. CMS-2 Flowcharter 15

7. Assembler 15

B. CMS-2 LANGUAGE DESIGN 15

1. Major Features 15

2. Program Structure 16

3. Header Structure 20

4. Data Structure 20

5. Control Structure 25

III. REVISED COMPILER AND LANGUAGE DESIGN 32

A. PROGRAM STRUCTURE 32

1. Structured Programming 32

2. Program Structure Revisions 33

3. Multi-level Programming 34

4. Declarative Delimiters 36

B. HEADER STRUCTURE 36

1. EQUALS and NITEMS Statement 36

2. CSWITCH Statement 39

3. DEP and EXECUTIVE Statements ------------- 40

4. SPILL Statement ------------------------- 40

5. Summary --------------------------------- 40

C. DATA STRUCTURES ------------------------- 41

1. Data Types ------------------------------ 41

2. Data Elements --------------------------- 41

D. CCNTROL STRUCTURE ----------------------- 44

1. BEGIN and END Declaratives -------------- 44

2. VARY Statement -------------------------- 44

3. CASE Statement -------------------------- 44

4. WHILE Statement ------------------------- 45

5. IF Statement ---------------------------- 45

6. Label Declaratives ---------------------- 45

7. SEARCH Statement ------------------------ 45

8. UNPCK Statement ------------------------- 46

9. Procedure CALL Statement ---------------- 46

10. GOTO, RETURNTO, and RESUME Statements ----- 46

11. RETURN Statement ------------------------ 46

12. INPUT and OUTPUT Statements ------------- 46

13. EXEC Statement -------------------------- 47

14. STOP Statement -------------------------- 47

15. COBEGIN and COEND Declaratives ---------- 47

16. RESERVE Statement ----------------------- 47

17. SET Statement --------------------------- 48

E. EXPRESSIONS ----------------------------- 48

1. Real ------------------------------------ 48

2. Bits ------------------------------------ 49

3. Character ------------------------------- 49

4. Boolean --------------------------------- 50

IV. LANGUAGE ANALYSIS ----------------------- 52

A. SLR(K) CONTEXT-FREE GRAMMARS ------------ 52

1. Grammar --------------------------------- 52

2. Context-Free Grammars ------------------- 52

3. Direct Productica or Reduction ----------- 53

4. Production or Reduction ----------------- 53

5. Sentential Form ------------------------- 53

6. Sentence -- 53

7. Language -- 53

8. Phrase -- 53

9. Canonical Derivation ---------------------------- 53

10. Parse -- 54

11. Left-right, Bottom-up Parse ------------------- 54

12. Rightmost Derivation --------------------------- 54

13. LR(K) Grammar ---------------------------------- 54

B. SLR(K) PARSING ALGORITHIM ---------------------- 54

1. Context-Free Parsing ---------------------------- 54

2. LR(0) and SLR(1) Parsing ----------------------- 56

C. SLR(1) SYNTAX ANALYSIS of CMS-2RS -------------- 57

V. DESIGN OF THE TWO-PASS CMS-2RS COMPILER ---------- 58

A. FIRST PASS ------------------------------------- 58

1. Lexical Analyzer -------------------------------- 58

2. Symbol Table Design ----------------------------- 59

3. Syntax Analyzers -------------------------------- 62

4. Backus-Naur Form (BNF) -------------------------- 63

5. Intermediate Language (IL) ---------------------- 63

6. First Pass Output ------------------------------- 64

B. SECOND PASS ------------------------------------ 64

VI. CONCLUSIONS ------------------------------------- 65

A. RESULTS -- 65

B. FUTURE WORK ------------------------------------ 66

APPENDIX A BNF DESCRIPTION OF CMS-2RS ------------- 67

APPENDIX B CMS-2RS LANGUAGE DESCRIPTION ----------- 77

APPENDIX C SAMPLE CMS-2RS PROGRAMS ---------------- 109

APPENDIX D DESCRIPTOR'S PROTOCOL ------------------ 113

APPENDIX E OPERATION CODES FOR IL ----------------- 119

APPENDIX F CMS-2RS COMPILER LISTING --------------- 123

LIST OF REFERENCES -------------------------------- 220

INITIAL DISTRIBUTION LIST ------------------------- 221

FORM DD 1473 -------------------------------------- 222

LIST OF TABLES

I. SAMPLE PROGRAM SYSTEMS ------------------------------ 17

II. SAMPLE SYSTEM PROCEDURES --------------------------- 18

III. CMS-2 PROGRAM STRUCTURE DELIMITERS --------------- 21

IV. CONTROL STATEMENT OPERATORS ----------------------- 27

V. FUNCTIONAL OPERATORS ------------------------------- 28

VI. CMS-2 MIXED EXPRESSION EVALUATION ---------------- 30

VII. INPUT-OUTPUT STATEMENT OPERATORS ----------------- 31

VIII. PROGRAM DEBUG STATEMENT OPERATORS --------------- 28

IX. CMS-2RS PROGRAM STRUCTURE DELIMITERS ------------- 38

X. CMS-2RS MIXED EXPRESSION EVALUATION -------------- 51

LIST OF DRAWINGS

1. CMS-2 JCB FLOW -- 12

2. CMS-2 LANGUAGE TRANSLATICN PROCESSES --------------- 13

3. TABLE ITEM AND ARRAY ASSIGNMENTS ------------------ 23

4. TABLE FIELD ASSIGNMENTS ----------------------------- 24

5. TABLE STRUCTURES ------------------------------------ 26

6. SCOPE OF DATA DESIGNS ------------------------------- 35

7. MULTI-LEVEL PROGRAMMING ----------------------------- 37

I. INTRODUCTION

The origins and design capabilities of the CMS-2 System must be explored before proposing any steps toward a revised language and compiler. This section describes the history of CMS-2 and the goals and objectives of this paper. The CMS-2 System and language are briefly described in Section II. Section III describes the revised language and compiler which will be called CMS-2RS. In Section IV, a detailed discussion of SLR(1) grammars, parsing algorithms, and CMS-2RS language analysis is provided. Section V describes the first pass of the CMS-2RS compiler and offers recommendations for intermediate language and second pass designs. Finally, the thesis conclusions are presented in Section VI.

A. BACKGROUND

Compiler Monitor System II (CMS-2) is an integrated group of computer program modules which comprise a Monitor System (MS-2) and a Compiler System (CMS-2). The Monitor System supervises the compiling, library, and loading processes. The Compiler System translates CMS-2 language source code into object code for any one of five target computer systems. The following sections provide a historical background of the CMS-2 System and language.

1. History of Compiler Monitor System II (CMS-2)

The primary reason for developing the CMS-2 System resulted in large part from the decision to develop the Navy AN/UYK-7 third generation computer. The two systems developed were designed to be hosted in the UNIVAC CP-642B and AN/UYK-7 computers. The CP-642B version generates object code for those two computers, as well as the Litton L-304, UNIVAC-1830A, and UNIVAC-1218/1219 computers.

The CMS-2 System was built by Computer Sciences Corporation under contract to Fleet Combat Direction Systems Support Activity, Pacific (FCDSSAPAC), in San Diego, Cali-

fornia. FCDSSAPAC provides System production, maintenance, and support services for all Navy and contractor activities involved in Command and Control Systems software development.

2. History of CMS-2 Language

The CMS-2 language extends the capabilities of Compiler System-1 language (CS-1) and includes some of the features of FORTRAN, JOVIAL, and PL/I. CMS-2 language capabilities were implemented specifically for Command and Control Communications problems which include internal and external message processing, table update and search, coordinate conversion, transcendental and hyperbolic functions and data display processing requirements. The language was initially implemented in 1969 and is now the Navy standard for all Command and Control Systems applications.

B. GOALS AND OBJECTIVES

This section describes the thesis objectives and the project goals to implement and validate those objectives.

1. Thesis Objectives

The CMS-2 System and Language were specifically developed to provide a high level language processor system for building and maintaining deliverable computer programs from a production library. In its current form, the CMS-2 System and language are in large part capable of accomplishing that requirement, but not without some inefficiencies and limitations in the System, as well as redundancies, ambiguities, and limitations in the language. It is with these deficiencies in mind that the following objectives are defined.

a. Determine what ambiguities, redundancies, and limitations exist in the CMS-2 program, data, and control structures, and define the necessary structure changes to correct them.

b. Develop an unambiguous Backus-Naur Form (BNF) description of the proposed language, as an SLR(1) Context Free grammar, that is appropriate to automated compiler construction.

c. Determine if an automatic SLR(1) grammar analysis and parsing table generation system can be applied to the proposed language and system to simplify compiler design and maintenance.

d. Determine what fundamental changes need to be made to the language and system to support the structured programming concept in a modular library environment.

e. Determine if the CMS-2 language can be segmented to allow design and implementation of a modular compiler, thus allowing ease of future language extension and maintenance.

2. Project Goals

a. Produce an operational first pass lexical scanner and syntax analyzer that will implement and validate the above thesis objectives.

b. Design an intermediate language (IL) that can be used by the second pass of the compiler to generate optimized object code for either a single or multiple address target computer.

II. CMS-2 SYSTEM AND LANGUAGE DESIGN

This section describes the CMS-2 System including the Monitor, Compiler, Librarian, Loader, Assembler, Flowcharter and Tape Utility. Additionally, program, header, data and control structures of the CMS-2 language are described.

A. CMS-2 SYSTEM DESIGN

The CMS-2 System consists of the MS-2 Monitor and subordinate systems whose operations are controlled by the monitor program. This section describes the CMS-2 System as presented by references 1, 2, and 3. The CMS-2 Systems include the following.

1. MS-2 Monitor System

The MS-2 Monitor System is a batch-processing operating system designed to control execution of its subsidiary components and user jobs being run on the computer. The monitor provides the environmental interfaces necessary for all programs running under its control. These interfaces include a job control card processor, an interrupt processor, an input/output system, an operator communication module, a debug module with dump, patch, and snap capabilities, and a job accounting package. In addition, the monitor maintains a library of the system component programs and data base definitions that can be called by the user upon request to be added to his compile or execution package. The CMS-2 job flow is shown in Figure 1.

2. CMS-2 Compiler System

The CP-642B version Compiler is a two-phase langauge processor that analyzes a dual syntax source program (CS-1 and CMS-2 languages) and generates object code for any one of the computers mentioned above. The phases of compiler translation are described below and illustrated in Figure 2.

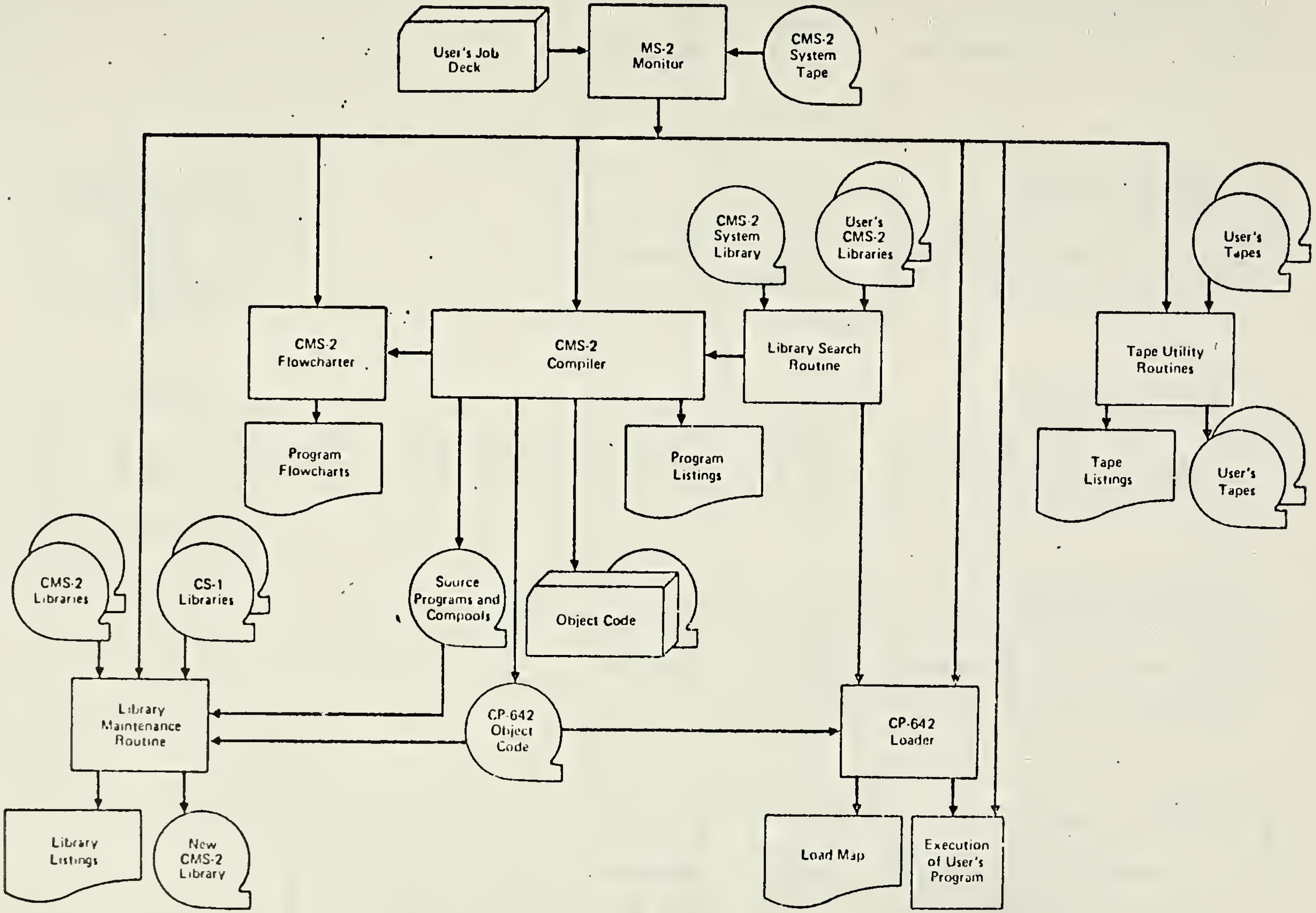


FIGURE 1 - CMS-2 JOB FLOW
(Figure 1 was extracted from Ref. 1)

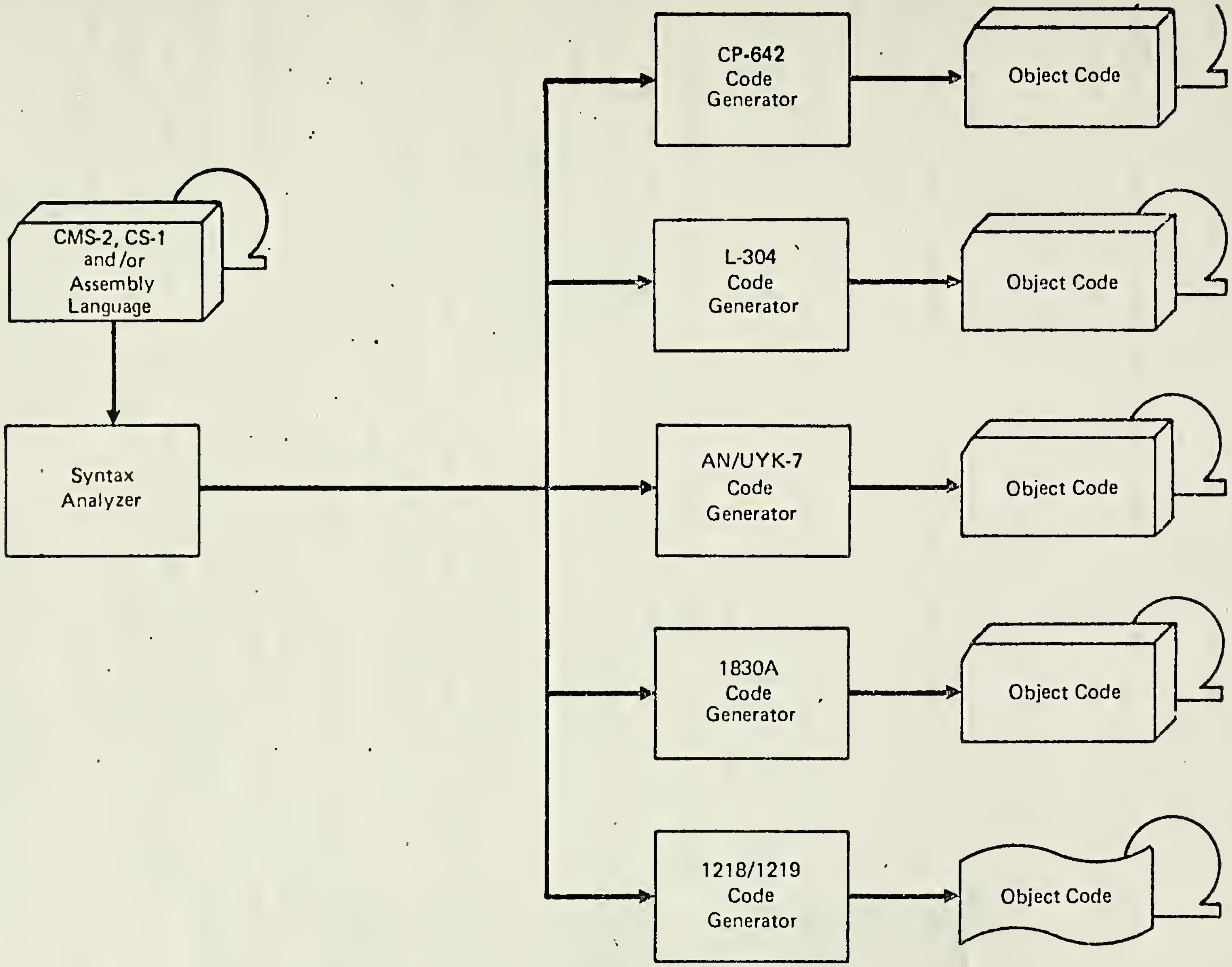


FIGURE 2 - CMS-2 LANGUAGE TRANSLATION PROCESSES
(Figure 2 was extracted from Ref. 1)

a. First Phase (Syntax Analyzer)

The first phase of the compiler processes the user's source program which consists of CMS-2 and CS-1 language statements and properly bracketed machine code instructions. The source statements are checked for validity, and an intermediate form of the program (IL) and symbol table are generated.

b. Second Phase (Code Generator)

The second phase processes the IL and symbol table to produce the final output listings and object code for the target computer. Each code generator produces the appropriate object code for each target computer in either absolute or relocatable format. The code may reference input, output, debug, and built-in functional routines. These routines are then linked to the object code of the calling program either by the compiler in an absolute mode or by the linking-loader in a relocatable mode. The built-in functional routines are either added in-line to the object code where referenced or are added as procedures to be linked together at run-time.

3. CMS-2 Librarian

The librarian is a file management system that provides storage, retrieval, and correction functions for a programmer's source programs and object code.

a. Library Maintenance

The library maintenance or executive control routine (LIBEXEC) is used to create, modify, or reproduce libraries for CMS-2 programmers. They contain source programs, object modules, and predefined data designs. A library translator routine, under control of LIBEXEC, is used to convert existing CS-1 programs or libraries into a CMS-2 library format.

b. Library Search

The library search routine is responsible for retrieving data from a previously created CMS-2 library.

4. Object Code Loaders

The CMS-2 System includes two loader programs for linking and loading object code produced by the CMS-2 compiler. The absolute loader loads all instructions and data at the addresses assigned during the compilation. Similarly, the relocatable loader processes relocatable object code directly from the compiler or from a CMS-2 library by assigning all program memory addresses and linking program segments to produce an executable object program. The loader, in the case of the AN/UYK-7, combines independently compiled program segments under a common base register or registers.

5. Tape Utility

The CMS-2 system provides a set of utility routines to assist the programmer with the manipulation of data recorded on magnetic tape. The routines provide the capability to construct, duplicate, compare, list, and reformat data files on tape.

6. CMS-2 Flowcharter

The flowcharter is designed to process unique flowcharter statements in a user's CMS-2 source program and output a flowchart of the program logic to the high-speed printer.

7. Assembler

The assembler, in the case of the AN/UYK-7 version, accepts mnemonic oriented instructions and provides a macro instruction capability.

B. CMS-2 LANGUAGE DESIGN

This section presents an overview of the major features of CMS-2, and its program, header, data, and control structures, as presented by references 1, 2, and 3.

1. Major Features

The major features of CMS-2 are: modular procedure-oriented program structures; high-level control structures; reentrant procedures; dynamically allocated data designs; separate definition of data and control structures; state-

ment oriented card input processing; Fixed-point, Floating-point, Boolean, Hollerith, and Status data types; and Character and Bitstring manipulation.

2. Program Structure

A CMS-2 Program is composed of ordered sets of statements organized into header, data, and control structures. These structures can be grouped to form System Elements called System Data Designs, System Procedures, or System Reentrant Procedures. Any System Element or Elements may be separately compiled as a System, which may be a complete or partial execution package, as shown in Table I.

The System Procedures and Reentrant Procedures are composed of ordered sets of Local and Auto Data Designs and Function and Procedure subroutines as shown in Table II. The local procedure within a System Procedure which shares its name is known as the Prime Procedure and is the entry point for System Procedure calls. A System Reentrant Procedure may only contain Auto Data Designs and its local procedures and functions are composed of reentrant statements. Each procedure that calls a Reentrant Procedure is dynamically assigned a private copy of the Auto Data Design during execution.

The headers are groups of statements that precede System Elements and specify control of compiler and loader processing of source and object code.

a. Program Statement Design

Program statements are composed of various sets of symbols and delimiters which are further composed of elements from the revised U. S. ASCII Standard Character Set. The symbols are catagorized as operators, control identifiers, data identifiers, and constants.

Operators are described by special characters and reserved words, and are used for unary, binary, and functional operations. They may operate on defined data fields and are used for control of program execution. Control identifiers are described by user declared words which define program locations where program execution flow may

SYSTEM_A	SYSTEM_B	SYSTEM_C
HEADER DECLARATION	SYSTEM DATA DESIGN	SYSTEM PROCEDURE
SYSTEM DATA DESIGN	EXTERNAL REFERENCE	SYSTEM DATA DESIGN
SYSTEM PROCEDURE	SYSTEM PROCEDURE	SYSTEM REENTRANT PROCEDURE
SYSTEM PROCEDURE	SYSTEM PROCEDURE	SYSTEM DATA DESIGN
SYSTEM REENTRANT PROCEDURE	SYSTEM DATA DESIGN	EXTERNAL REFERENCE
SYSTEM PROCEDURE	SYSTEM PROCEDURE	SYSTEM PROCEDURE

TABLE I - SAMPLE PROGRAM SYSTEMS

SYSTEM PROCEDURE A	SYSTEM REENTRANT PROCEDURE B
LOCAL DATA DESIGN	AUTO DATA DESIGN
PROCEDURE A	PROCEDURE B
PROCEDURE B	AUTO DATA DESIGN
LOCAL DATA DESIGN	FUNCTION A
PROCEDURE C	PROCEDURE C
AUTO DATA DESIGN	AUTO DATA DESIGN
PROCEDURE D	FUNCTION D
LOCAL DATA DESIGN	PROCEDURE E
FUNCTION E	PROCEDURE F
PROCEDURE F	

TABLE II - SAMPLE SYSTEM PROCEDURES

be transferred. These locations may be specified by statement labels or procedure and function names.

Data identifiers define simple variables, statement switches, tables, and arrays. Constants are ordered sets of numeric characters with known, fixed values. They may be reals (decimal or octal), Hollerith strings, Status values, or Boolean. Another type of constant, called called a Tag, is represented by sets of alphanumeric characters and can be used throughout a compile-time System in place of literal constants.

b. Scope of Identifiers

The scope of an identifier is the range of program structure within which an identifier can be referenced or has meaning.

(1) Forward and Backward References. Within a Data Design, a data structure may be referenced either before (forward) or after (backward) it has been defined. Outside a Data Design, a data structure may only be referenced after it has been defined.

A reference to statement labels within System Procedures; calls to local procedures and functions within their System Procedure; and calls to Prime Procedures not having formal parameters or abnormal exits may be either forward or backward.

(2) Local and Global Definitions. Local definitions are those identifiers which can be referenced only from within the System Element in which they are defined. Global definitions apply to those identifiers that can be referenced both from inside and outside the System Element in which they are defined.

Local Data Designs are local only to the System Procedure in which they are declared. All System Data Designs and Prime Procedures are global to their compile-time System.

(3) External References and External Definitions. An external definition (EXTDEF) is a prefix that allows an identifier to be referenced outside of the System Element

where it is defined. All System Data Designs and Prime Procedures are automatically defined as external. An external reference (EXTREF) is a prefix that allows an identifier to be referenced in a System Element that is local tc another System Element. These external references must obey the local/global ranges of the identifiers they reference. The following cases apply.

(a) The identifer is a data structure in a Local or System Data Design that follows the reference.

(b) The identifier is a local procedure name, statement label, or data structure in a Lccal Data Design, and is defined in another System Procedure.

(c) The identifier is defined in another compile-time System that will be linked with the current System at load time. This case is identified by the prefix TRANSREF in the AN/UYK-7 System.

(4) Declarative Delimiters. Table III shows the delimiters which declare the beginning and ending of the varicus program structures.

3. Header Structure

The header structure contains compiler control declaratives which specify action as to base register allccation modes, library retrieval and update options, program debug features, object code addressing, cutput format and listing options, system index-register interpretation, and the computer mode of operation within which the designated program is expected to run. These statements may be located in major headers, if the contrcl applies to the entire compile-time System, and in minor headers, if the control applies only to a System Element.

4. Data Structure

The data structure consists of precise definitions of temporary and permanent data storage areas, input areas, cutput areas, and special data units such as program switches. These structures can be grouped together to form System, Local, and Auto Data Designs. A data declaration defines the type, precision, identifier, and opticnally, the

BEGIN DELIMITER	END DELIMITER	DELIMITED ELEMENT
SYSTEM	END-SYSTEM	Compile-time System
SYS-DD	END-SYS-DD	System Data Design
SYS-PROC	END-SYS-PROC	System Procedure
LOC-DD	END-LOC-DD	Local Data Design
PROCEDURE	END-PROC	Local Procedure
FUNCTION	END-FUNCTION	Local Function
SYS-PROC-REN	END-SYS-PROC	Reentrant System Procedure
AUTO-DD	END-AUTO-DD	Auto Data Design
HEAD	END-HEAD	Header Declaration

TABLE III - CMS-2 PROGRAM STRUCTURE DELIMITERS

scaling and preset value of a data element in a Data Design. The five major data structures are switches, variables, Tables, Arrays and Files. An Overlay declaration is used to define an identifier which is packed with the values of a list of variables or Fields in the order in which they are listed from left-to-right.

a. Switches and Variables

Switches provide for transfer of program control to statements or procedures depending upon the value of a programmer supplied index. A variable is a singular piece of data of length one bit, multiple bits, or computer words. If the length is not specified then a default parameter is assumed for the specified target computer. Variables may also be preset to a compatible value.

b. Tables

Tables hold crdered sets of identically structured information. The common unit of data structure in a Table is the Item.

(1) Item. An Item consists of k ccmputer words, where k is selected by the programmer or compiler. A Table may contain n Items, where n is a declared parameter. Tables may be declared horizontally such that all words 0 of all Items are stored together sequentially follcwed by all words 1 of all Items up to word n. The words of a vertically defined Table are stored such that all words of every Item are stored together sequentially. Item assignments are shown in Figure 3.

(2) Fields. Items may be subdivided into Fields, which are the smallest subdivision of a Table. A Field may be a partial wcrd, full word, or multiwcrd subdivision. Fields have the same data types as variables and may be preset and altered dynamically as variables. In addition, Fields may overlap each other, but must be type compatible. Field assignments are shown in Figure 4.

(3) Subtables, Like-tables, and Item-areas. The Table declaration structure also allows the programmer to define a subset of adjacent Items within a Table as a

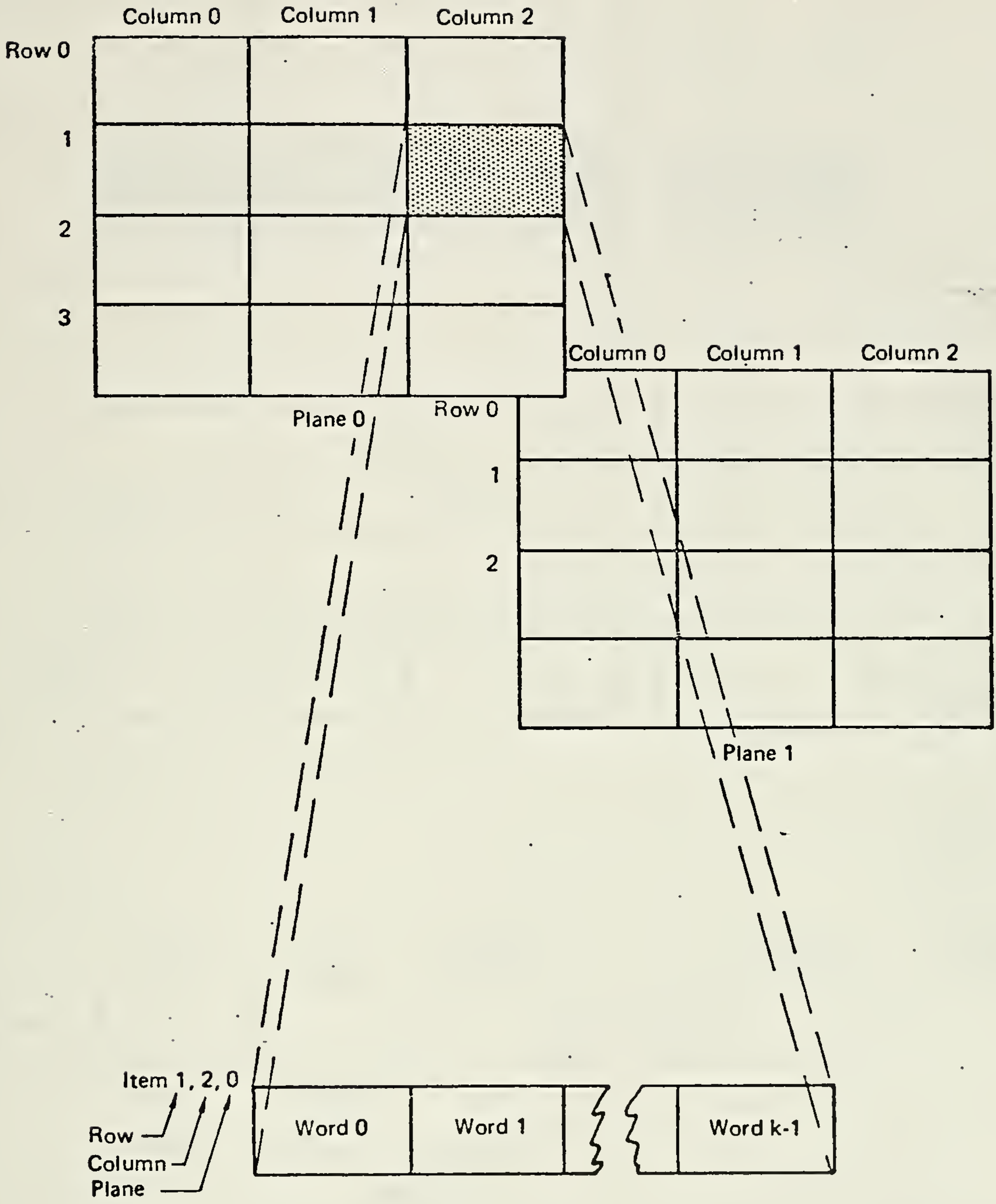


FIGURE 3 - TABLE ITEM AND ARRAY ASSIGNMENTS

(Figure 3 was extracted from Ref. 1)

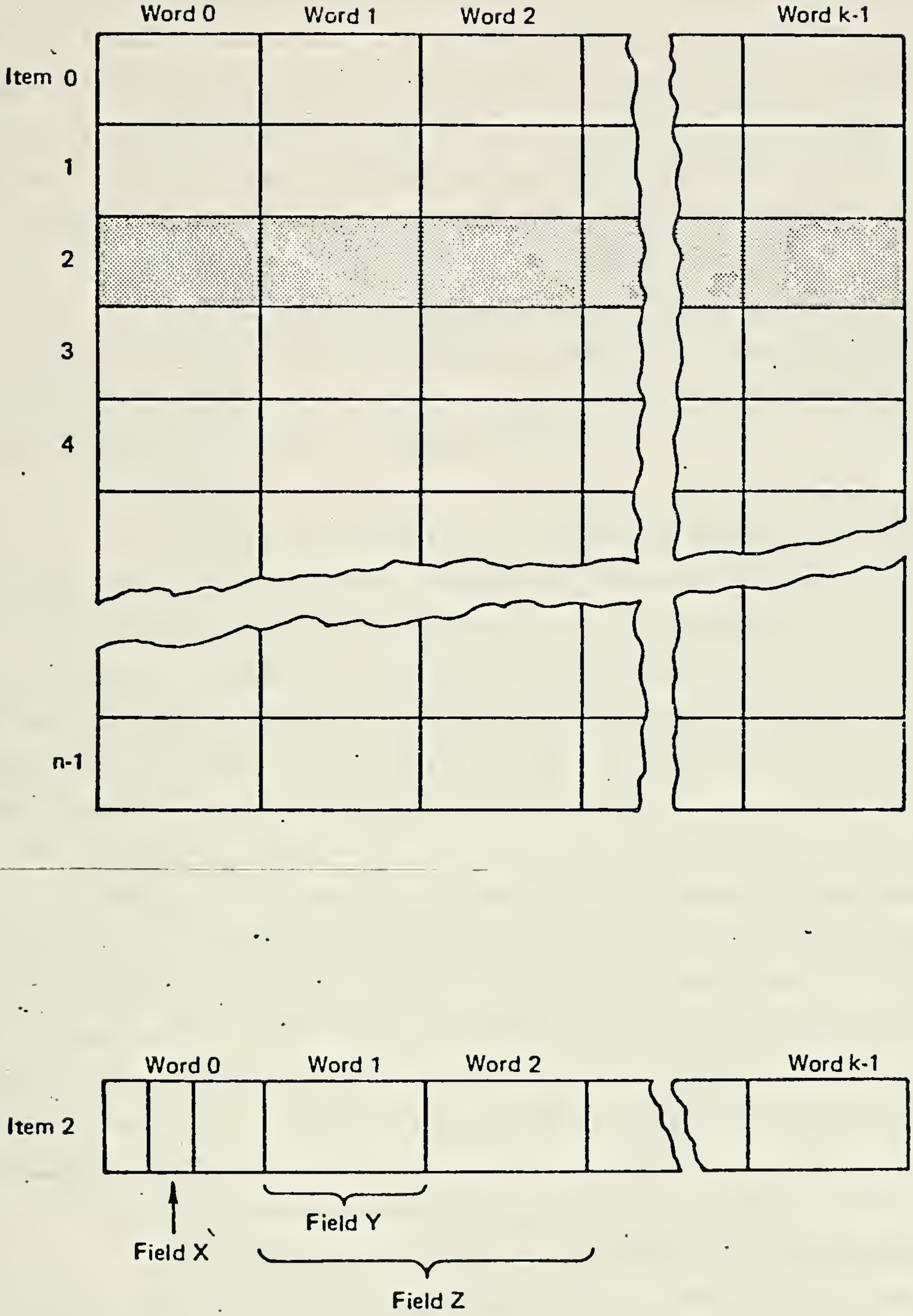


FIGURE 4 - TABLE FIELD ASSIGNMENTS

(Figure 4 was extracted from Ref. 1)

subtable. The programmer may also allocate a working storage area outside the table, called an Item-area, which automatically takes on the same Field format as the Table Item. Additionally, the programmer may declare Tables having the identical Field format as the parent Table, called like-Tables, but having a different number of Items. The Table structures are shown in Figure 5.

c. Arrays

An array is a multi-dimensional extension of the Table concept for storing ordered sets of identically structured information previously defined as Items. The array structures are shown in Figure 3.

d. Files

A File declaration defines a data structure environment in which one or more physical files are to be processed. The declaration assigns a File name for dynamic statement referencing, identifies the symbolic name assigned to the actual hardware device, and declares that all data to be processed on the named hardware device is physically organized as described in the declarative statement.

5. Control Structure

The CMS-2 control structure or dynamic statement specifies processing operations and results in executable code. Each statement consists of an operator followed by a list of operands and additional operators. An operand may be a single name, a constant, a data unit reference, or an expression. The data units may be variables, subscripted variables, Tables, or Files.

a. Statement and Functional Operators

The major statement operators are described in Table IV. The functional operators in CMS-2 are used to facilitate references to and operations on data structures. These operators are described in Table V.

b. Expressions

Real expressions may include standard addition, subtraction, multiplication, and division operators, as well as exponentiation, mixed-mode values,

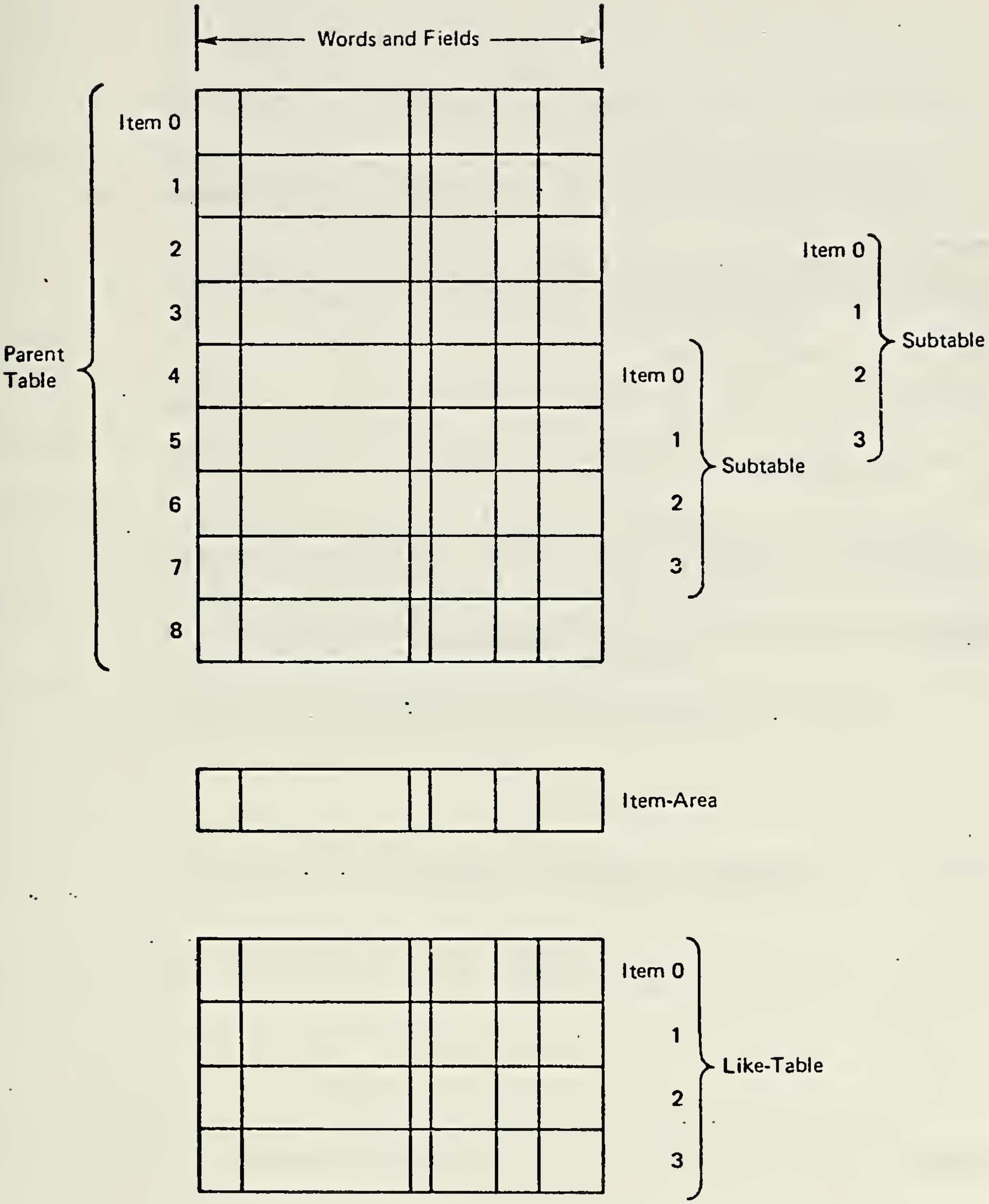


FIGURE 5 - TABLE STRUCTURES

(Figure 5 was extracted from Ref. 1)

OPERATOR	MEANING
SET	Performs claculations or assigns a value to one or more data units. The assignemnt may be Real, Hollerith, Status, Boolean or Multi-word.
SWAP	Exchanges the contents of two data units.
GOTO	Alters program flow directly or via a statement switch.
IF	Expresses a Boolean test situation for conditional execution of one more statements.
VARY	Establishes a program loop to repeat execution of one or more statements by varying one or more indexes by a specified increment until a test value is satisfied.
FIND	Searches a table for data that satisfies specified conditions, and assigns to subscript variables the index values pointing to its location.
PACK	Transfers bits strings into a data area.
SHIFT	Shifts a string of bits right or left a specified amount.
RESUME	Specifies a transfer of execution control to the increment and test steps within a VARY block.
RETURN	A transfer of execution control from a procedure call that specifies normal return to a label if a special condition is met, or a return of the value of an expression to the point of call.
EXEC	A call to an executive program which passes one or two parameters indicating what action to be taken and on what data unit or address.
STOP	Temporarily suspends program execution until manually restarted on the computer.

TABLE IV - CONTROL STATEMENT OPERATORS

OPERATOR	MEANING
BIT	To reference a string of bits in a data unit.
CHAR	To reference a string of characters in a hollerith data unit.
CORAD	To reference a data unit's relative core address.
ABS	To obtain absolute value of an arithmetic primary.
POS, FIL	To move a magnetic tape record, file a specified number of positions forward or backwards.
LENGTH	To obtain a record's length for the last input-output operation.
CNT	To obtain count of bits set to one in a data unit.
CAT	To concatenate character strings.

TABLE V - FUNCTIONAL OPERATORS

OPERATOR	MEANING
DISPLAY	Causes the contents of machine registers and specified data units to be formatted and printed on the system output device.
SNAP	Contents of a data unit are printed and stored. Subsequent executions cause a printout only when the data contents are modified.
RANGE	High and low values are specified for a data unit, and each time the data unit is modified in the program, a message is printed if the value falls outside the range.
TRACE	A printout is generated for the execution of each CMS-2 statement between TRACE and END-TRACE.
PTRACE	Each CMS-2 procedure call encountered in the program being executed is identified by calling and called procedure names.

TABLE VIII - PROGRAM DEBUG STATEMENT OPERATORS

and in-line redefinition of the scaling of fixed-point numbers.

A relational expression performs a comparison between two similar operands as specified by a relational operator. There are four types of comparisons:

(1) Real, involving the comparison of signed Real values (fixed, floating, or mixed),

(2) Hollerith, involving a left-to-right, character-by-character comparison,

(3) Boolean, involving the comparison of single bits, and

(4) Status, involving the comparison of status values.

A CMS-2 expression may include algebraic, relational, and Boolean operators. The heirarchy of expressional evaluation is shown in Table VI.

c. Input-output Statements

The CMS-2 Input-output statements permit the program to communicate with the various hardware devices while running in a monitor environment. The operators are described in Table VII.

d. Program Debug Statements

The CMS-2 Debug statements are placed in the source language to cause run-time debug routines to be available for program execution analysis. The debug statement operators are described in Table VIII.

PRIORITY	OPERATOR	DEFINITION
1	**,-	EXPONENTIATION, UNARY MINUS
2	*,/	MULTIPLICATION, DIVISION
3	+,-	ADDITION, SUBTRACTION
4	EQ, NOT GT, LTEQ GTEQ, LT	EQUAL, NOT EQUAL GREATER THAN, LESS THAN OR EQUAL GREATER THAN OR EQUAL, LESS THAN
5	COMP	LOGICAL COMPLEMENT
6	AND	LOGICAL MULTIPLICATION
7	OR	LOGICAL SUM
8	XOR	LOGICAL DIFFERENCE

TABLE VI - CMS-2 MIXED EXPRESSION EVALUATION

OPERATOR	MEANING
FILE	Defines the environment and pertinent information concerning an input or output operation, and reserves a buffer area for record transmission.
OPEN	Prepares an external device for input, output, or scratch (both) operations.
CLOSE	Deactivates a specified file and its external device, sends last unfinished buffer to output.
INPUT	Directs an input operation from an external device to a FILE buffer area.
OUTPUT	Directs an output operation from a FILE buffer area to an external device.
FORMAT	Defines the desired conversion between data blocks and internal data definitions.
ENCODE	Directs transformation of data elements into a common arrea, with conversion in accordance with a specified FORMAT.
DECODE	Directs unpacking of a common area and transmittal to data units as specified by a FORMAT declaration.
ENDFILE	Places and end-of-file mark on appropriate recording mediums.
CHECKID	Directs checking an id header or label on a FILE if the device is at load point.
SEARCH	Directs a search operation for specific data within a FILE.

TABLE VII - INPUT-OUTPUT STATEMENT OPERATORS

III. REVISED COMPILER AND LANGUAGE DESIGN

This section describes the CMS-2RS Compiler System and Language design capabilities which differ from the CMS-2 System. A more complete description of the CMS-2RS language is provided in Appendix B.

A. PROGRAM STRUCTURE

In a large system of programming modules, the problems of isolation of logic design error and verification of program correctness tend to increase exponentially as the number of combinations of program procedure call parameters and transfers of execution control increase. To counter this problem, a well-defined method of software engineering and suitable language-compiler system are needed.

1. Structured Programming

The term "Structured Programming" is defined in references 4 and 5 as a top-down method of program building with the top program modules being at the highest level of the program logic design and referencing the next level modules as procedure calls in its control structures. This process continues until the modules at the lowest level are referencing basic machine and operating system constructs.

In reference 6, Bohm and others have shown that the control logic of any programming problem can be represented by the three basic control structures: simple statement sequences, IF THEN ELSE statements, and DO WHILE statements. The CASE structure of ALGOL W may also be required to prevent multiple nesting of IF THEN ELSE statements.

To ensure program reliability under varying loading conditions, its logical correctness must be easily verifiable. The first requirement must therefore be to restrict each program segment to exactly one entry and exit point. Another requirement is that library substitution facilities be provided at compile and load-time so that the segemnts can

be stored and retrieved by symbolic names. Finally, scope of identifiers and GOTO statements must be controlled in their application to prevent unlimited access to data and control structures.

2. Program Structure Revisions

The program structure of CMS-2RS retains the modular structure of CMS-2, but includes several revisions to its conventions for procedure exit, and identifier referencing to facilitate implementation of the top-down programming concept.

a. Procedure Exit

The convention of allowing unrestricted abnormal exits or returns to a calling procedure has been modified. A return which precedes the point of call may result in an infinite loop condition, given the abnormal exit condition still exists. Abnormal exits have therefore been restricted to return points which follow the point of call and lie within the calling procedure.

b. External Referencing and External Definition

The concept of externally defining an identifier in one element to be externally referenced by another element destroys the integrity of the System Procedure and its Local Data Designs. A program logic error related to the above concept may be very difficult and costly to locate. Therefore, external references are only allowed between different compile-time Systems and then only by specifying the identifiers of System Data Designs, System Procedures or Reentrant Procedures. The prefix EXTREF now applies to System Data Designs in symbol table format and to System Procedures in source code format, and at load-time to System Procedures in object code format. The Elements thus specified are then added as complete sections to the compile or load-time System as appropriate.

If it is necessary to reference another System Procedure's Local Data Design structures at compile-time, then the data structure must be removed and put into an appropriate System Data Design so that it is global to

both procedures. The external definition, EXTDEF, and transient reference, TRANSREF, prefixes have been eliminated to simplify language and compiler design.

c. Forward and Backward References

The forward reference of identifiers within a data design is not necessary and has therefore been deleted. The Tag declaration, which is used to name a constant throughout compile-time, is sufficient to allow presetting of all data structures within a data design.

d. Local and Global Definitions

The local and global definitions of identifiers allow program structure to develop in a logical fashion. The System Procedures at the lowest physical program level have access to their Local Data Designs and all other previously declared System Data Designs. The System Procedures at the highest level in the program, however, have access only to their Local Data Designs and all System Data Designs declared above. This concept of scope of identifiers provides for implementation of the structured programming concept by allowing the top procedure to be written at the highest logic level and the bottom at the lowest level as shown in Figure 6. What is required though, is a means of prescribing what levels of control structures (how close to the machine) should be allowed within each System Procedure in an executable System.

3. Multi-level Programming

A Multi-level programming system is one in which all the language constructs are expressed on a high lexical level. They are, however, capable of implementing systems as well as applications programs. If these constructs are to be controlled in their use in all types of programming, a matrix of levels of programming verses language constructs is required in the syntax analyzer to allow only the appropriate code to be generated for each programming level. The Syntax analyzer can thereby control which grammar reductions are allowed at each programming level in a System. A further requirement exists to define in the Monitor System a matrix specifying which levels are accessible by each user, module

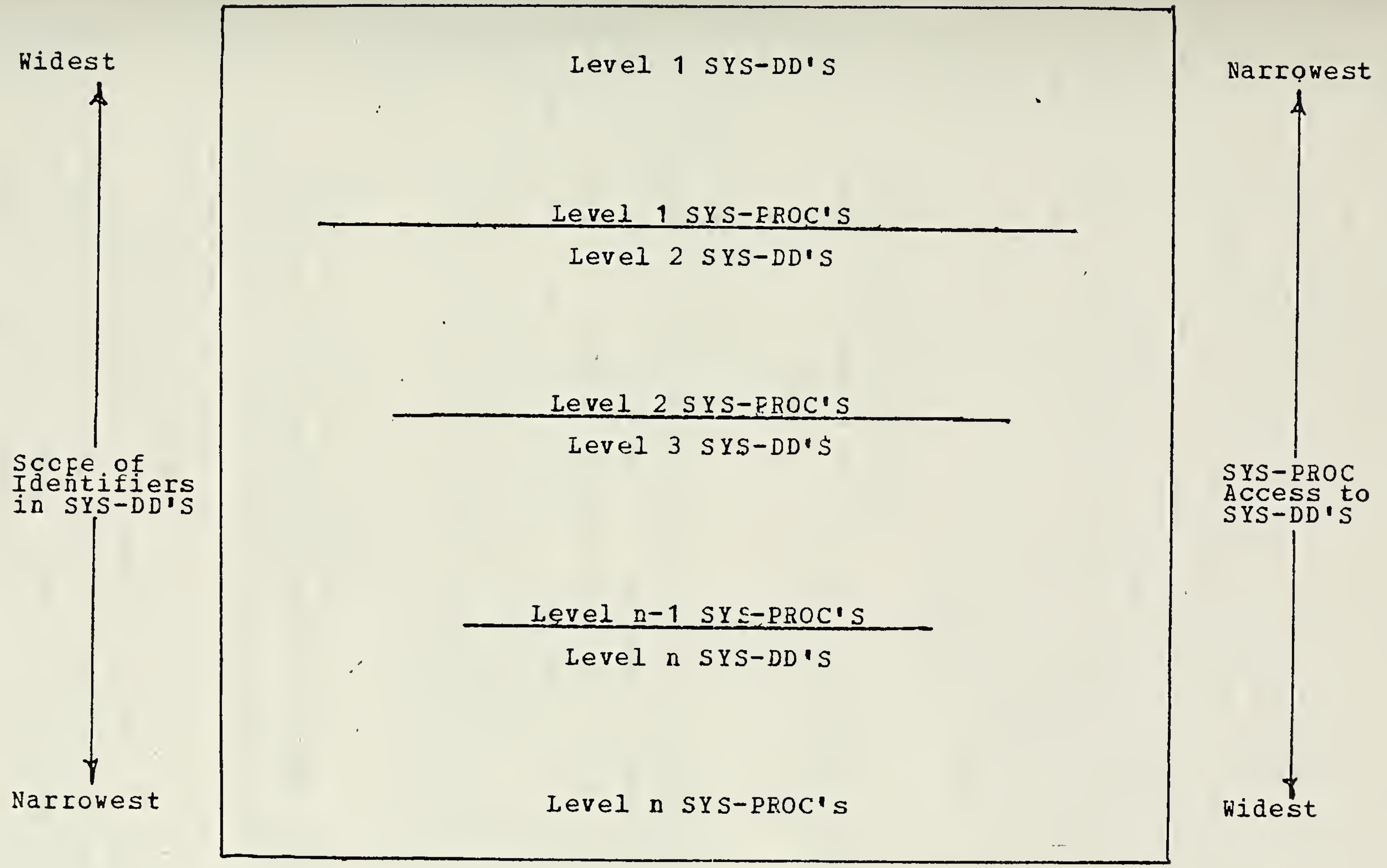


FIGURE 6 - SCOPE OF DATA DESIGNS

and corresponding project. The resulting system allows a programmer to specify that other levels of program modules, for which he is not authorized to generate code, are to be added at the appropriate level to his compile-time or load-time System. The concept of System executive or supervisor state and task state are thereby expanded to levels of states. For example, the use of System Registers as variables in expressions and assignment statements is limited to those levels which require register access such as I/O routines. Thus the Monitor System has complete control over which user generates what level of code for which System and provides a means of security over use of program control constructs. This concept is illustrated in Figure 7.

4. Declarative Delimiters

Table IX shows the revised delimiters which declare the beginning and ending of program structures.

B. HEADER STRUCTURE

The minor header declaration has been eliminated and major headers now only apply to the entire compile or load-time System. The header declarations were not implemented in the current CMS-2RS compiler, but could easily be included in the same manner as the data declarations. The following CMS-2 statements have not been implemented in the header structure in order to simplify the CMS-2RS language and compiler.

1. EQUALS and NITEMS Statement

The Equals statement is used to assign numeric values to variables and to specify their allocation relative to other variables.

a. Numeric Assignment

The assignment of numeric values to symbols by simple parenthesis-free expressions requires the generation of compile-time code to preset symbols to the desired values. This practice is not necessary since symbols can be initialized in the data design by presets or in the procedure itself by dynamic code. Allowing this method of pre-initialization

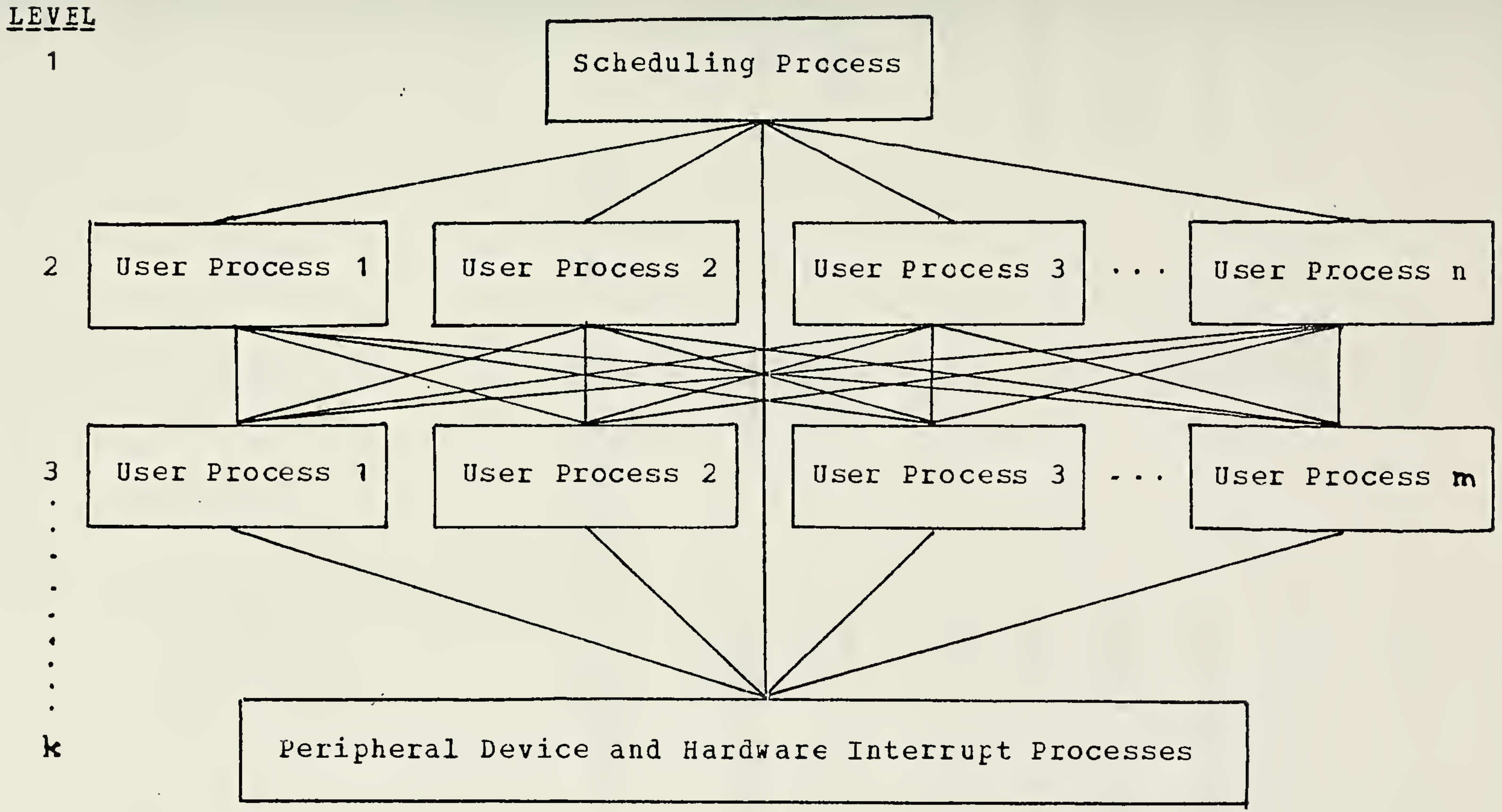


FIGURE-7 --MULTI-LEVEL PROGRAMMING

BEGIN_DELIMITER	END_DELIMITER	DELIMITED_ELEMENT
SYSTEM	END	Compile-time System
SYS_DD	END_DD	System Data Design
SYS_PROC	END	System Procedure
SYS_PROC_R	END	System Reentrant Procedure
PROCEDURE		Local Procedure
FUNCTION		Local Function
LOC_DD	END_DD	Local Data Design
AUTO_DD	END_DD	Auto Data Design
HEAD	END_HEAD	Header Declaration

TABLE IX - CMS-2RS PROGRAM STRUCTURE DELIMITERS

introduces a possible source of error that may be difficult to detect and violates the structured programming concept of program segment integrity. Further, initializing symbols at compile-time requires that header declaration information always be a part of the System Procedure or Data Design in order to maintain library configuration control.

b. Relative Allocation

The relative allocation feature allows a data unit to be assigned the same relative location as another data unit. This relative allocation capability can also be accomplished at run-time by the CORAD functional operator, thus preserving library segment independence.

The NITEMS declaration specifies that an identifier is to be assigned a value for compile-time purposes, but may also be assigned another value at load-time. It is used only for establishing the number of items in a vertical Table whose size is to be determined at load-time.

The EQUALS and NITEMS statements were not implemented since most of their capability could be more easily provided by the Tag declaration.

2. CSWITCH Statement

The CSWITCH (compile switch) statement provides selective compilation of blocks of statements within a System Element. The CSWITCH parameter identifies a group of statements within a CSWITCH block. In the header, the list of parameters following the CSWITCH identifier specifies to the compiler those CSWITCH blocks which are to be compiled in the System Element or Elements that follow. The parameter DELETE specifies that all CSWITCH code blocks in the following Element or Elements, which are not being compiled, are to be deleted from the source output and listings of the compiler.

The CSWITCH feature was not implemented since the ability to selectively delete code from compilation complicates the problem of program segment configuration control and in some cases duplicates the library CORRECT feature.

3. DEP And EXECUTIVE Statements

The DEP statement informs the compiler of a list of System Elements which are dependent or subordinate to the System Element following the header declaration in a compile or load-time System.

The EXECUTIVE statement function informs the compiler that the following System Element or Elements are allowed to generate control memory references to index registers and accumulators using the symbolic identifier specified in the System Index (register,name) declaration.

These statements will not be necessary if a multi-level compiling concept is implemented. Instead what is required is a declaration which informs the compiler of the program level and project type of the following Element or Elements.

4. SPILL Statement

The SPILL statement causes every identifier within a System Procedure to be declared as an External Definition. This statement has not been implemented since its capability can be achieved by relativley simple program changes.

5. Summary

In summary, the header declarations should be used to inform the compiler of the type of Element allocation (static or relocatable) and the type of output required for the compile and load-time Systems. They should not be used to modify the Element's internal data allocation and control structure representation. Header statements are also used to specify library retrieval-correction, debug, and data design pooling changes to the compile or load-time Systems' default parameters.

C. DATA STRUCTURE

The data structure of a program must allow a compact definition of a data unit's attributes and provide for presetting it to a constant or Tag value. The following changes have been made to simplify CMS-2RS.

1. Data Types

The data types available in CMS-2 do not explicitly include a type that allows operations on the mathematical set of binary digits. The data type Boolean is used primarily for the logical operations of conjunction, disjunction, and negation. Therefore, it has been limited to a fixed length set which is defined as either True or False and no longer can be used for bitstring manipulation.

To provide a Bitstring capability, a new data type has been defined, called Bits, which allows the logical operations of "and", "or", complementation, as well as SHIFT and substring functional operations. These operations provide the capability for programming at the bit level in addition to programming at other data type levels.

The data type Integer, as well as Fixed-point and Floating-point, are now treated as Reals and may only be used in Real expressions.

2. Data Elements

The data element constructs in CMS-2RS provide a means of organizing data into the most commonly used structures with the exception of lists.

a. Switches

The Index switch declaration for statements is composed of an index variable followed by a list of n statement labels to which control may transfered depending upon the value of the index (0 to n-1). An alternative form of the declaration contains two index variables and a column of labels for each. The item switch declaration is composed of a variable followed by a list of constant statement-label pairs.

The Procedure switch declaration is composed of procedure names to which calls may be made, and a list of formal input and output parameters for each procedure.

The statement switch and procedure switch capabilities have been incorporated intc the CASE statement.

b. DATA and MODE Declarations

The DATA declaration has been incorporated into the data unit preset capability in the data designs. To facilitate this, a TAG declaration has been added which allows the programmer to assign a constant value to an identifier which can be used throughout a compile-time System.

The MODE declaration, which speciifes that subsequent data declarations have the same data type, has been superceded by the requirement that type specifications appear in every data declaration.

c. Variables

The precision specification for variables is nc longer cptional. It must now be specified and each code generator will translate it into the appropriate length for its target machine.

The V(x,y) specification requires the compiler to preset-scale a variable starting at the y bit position by defining that position with a magnitude of x and each subsequent position as one half the preceding positions' value. This capability has been provided by the Bits data type expressions.

d. Tables

The option tc store one dimensional Tables in either vertical or horizontal alignment has not been implemented in favor of aligning all Items vertically (all words stored sequentially). An optimum search capability is now provided by the compiler that allows Table searching to be independent of Item word storage alignment.

The optional counter (major index), which keeps track of the actual number of items within the table, has not been implemented. This feature was programmer maintained and could just as easily be accomplished by a normally declared Integer variable.

Like-tables have not been implemented since the same capability can be achieved by declaring either another Table with a different number of items or a Sub-table with the desired dimensions.

The Sub-table concept has been extended to include arrays or multi-dimensional Tables, as well as one dimensional Tables.

The INDIRECT Table implementation has been simplified by not requiring the CORAD functional operator in making procedure calls by address. The fact that the formal input parameter is an INDIRECT Table name or that the corresponding formal and actual input parameters are of the same name is sufficient to achieve a procedure call in which no values are passed. Further, the INDIRECT option may not be used as an actual parameter.

The Field declaration has been changed to allow only type "b" word packing, that is the compiler assembles the Fields within Table Items sequentially in the order in are declared.

e. Overlay

The Overlay declaration has not been implemented since dynamic packing of a variable with a list of variables can be accomplished by the PACKIN instruction. The static packing of variables is of questionable value in a language with a message processing capability and requires an unnecessarily complicated compiler data declaration capability. If a list processing capability is required, it should be added as a complete feature of the language and compiler including list declarations and logical set operators such as union, intersection, add-to, and delete-from.

D. CONTROL STRUCTURE

The CMS-2RS control structure provides the capability to specify parallel as well as sequential processes. Further, a structure has been provided which groups statements into blocks for sequential processing. This section describes the control statements which have been added to the CMS-2RS language to implement the above capabilities. Additional changes have been made to provide the capability to implement the structured and multi-level programming concepts.

1. BEGIN and END Declaratives

The BEGIN and END block structure declaratives, which are described by Bauer in reference 7, have been implemented to allow grouping of statements for sequential execution. There is no imposed limit on the number of levels of block structure that can be nested within a System Procedure since data designs are declared separately from control structures.

2. VARY Statement

The VARY Statement has been changed to allow an optional increment value (other than 1) for each index variable on the same loop level. The loop execution will cease only when all indexes have reached their test value.

3. CASE Statement

The CASE statement, which is described by Bauer in reference 7, has been added to the language to provide the capability for selective statement execution. A case index value is compared at run-time to a list of case labels to determine which statement will be executed. Each statement in a case block may have more than one label and labels may be either Integer or Status constants. An ELSE case capability has also been provided in the event the index value falls outside the range of the case label list. An alternate version of the CASE statement has been implemented which does not require statement labels. An implied labeling sequence of 0 to n-1 is used to select one of n statements in a case block.

4. While Statement

The While statement has been added to provide for loop controlled execution of a statement or block of statements based upon the continued True condition of a Boolean expression.

5. IF Statement

The IF statement has been modified to allow nesting of IF THEN ELSE statements. Only simple (not CASE, VARY, WHILE, or IF) statements are allowed within the nesting structure. All types of statements, however, may be used for the last statement after the last ELSE.

The "IF DATA FOUND THEN" and "IF DATA NOTFOUND THEN" statements have not been implemented since the indicies of a Table or File in a SEARCH statement will be set to minus one if the search is unsuccessful, otherwise they will be set to the location of the desired data unit.

The "IF (subscripted data structure) VALID/INVALID THEN" statements have not been implemented since the Boolean functional operator SUBSCRIPT, with parameter subscripted data structure, provides a True of False indication of subscript range validity, and can be implemented by using it within an "IF THEN ELSE" statement.

The "IF (data unit) ODDP/EVENP THEN" statements have not been implemented since the COUNT(data unit) functional operator provides the same capability when checked for even or odd value in an IF statement.

6. Label Declaratives

Labels are allowed on every statement type except statements within VARY, CASE, RESERVE, and WHILE statement blocks. Multi-level programming, however, may be used to restrict their use to specified program levels.

7. SEARCH Statement

The SEARCH and FIND statements have been combined to simplify searching a Table or File structure for a particular data unit. If found, the index variables in the data sturcture's dimension list will be set at run-

time to point to the data unit's location, otherwise they will be set to minus one.

8. UNPCK Statement

The UNPCK statement has been implemented to allow a data unit to be unpacked into a list of data units in sequence from left-to-right. If the combined length of the receiving data units is longer than the source, then they are filled with zeroes.

9. Procedure CALL Statement

The procedure CALL statement has been changed to allow calls by value, value-result, result, address, and address-result. All input and output actual parameters are optional and need not agree in number, but must agree in type. An expression may be used as an input actual parameter but only a data unit may be used as an output actual parameter.

10. GOTO, RETURNTO, and RESUME, Statements

The GOTO, RETURNTO, and RESUME statements have been implemented to allow transfers only to labels within a System Procedure. The GOTO and RETURNTC statements, with special hardware condition checks, have not been implemented since the same capability is available by using the INTERRUPT Boolean functional operator within an IF THEN GCTC statement. The GOTC statement with an Item or Index-switch name check has not been implemented since the same capability is available with the CASE statement.

11. RETURN Statement

The RETURN statement has been changed to allow transfer with a data unit instead of transfer with the result of an evaluated expression.

12. INPUT and OUTPUT Statements

The INPUT statement has been implemented to allow moving of Character and Binary records from READ, OCM (Operator communication medium), and user declared Files to the following user specified receiving data units: Tables, Fields, Items, or variables. The OUTPUT statement has been implemented in a similar manner for PRINT, PUNCH, OCM, and

user declared Files. Both statements allow data to be moved back and forth from internal to external structures.

13. EXEC Statement

The EXEC Statement has not been implemented since a normal call to a procedure, with the proper input parameters and at the appropriate programming level, accomplishes the same effect.

14. STOP Statement

The STOP statement, with a hardware special condition check, has not been implemented since the same capability is available by using the INTERRUPT functional operator in an IF statement.

15. COBEGIN and COEND Declaratives

The COBEGIN and COEND declaratives which are described by Hansen in reference 8, have been implemented to allow simultaneous execution by more than one processor of several statements including procedure calls. Each process may refer to shared resources such as variables, subscripted variables, File records, and non-reentrant procedures only if their values are not changed. Reading from and adding to the end of data structures, however, are legal accesses to a shared resource.

GOTO statements are not allowed within a COBEGIN-COEND block to prevent errors arising from the transfer of control from one process to another. In addition, COBEGIN and COEND declaratives may be nested to any level within a System Procedure or Reentrant Procedure. Thus, they may be used to describe any combination of parallel and sequential processes.

16. RESERVE and WAIT Statements

The RESERVE statement has been added to provide a means for the programmer to specify critical sections in each process within a COBEGIN-COEND block. The parameters for the RESERVE statement are those resources which may be change-accessed by two or more processes. Each resource thus specified, is identified as belonging to a particular process. If another process desires access to the same

resource then that process will be enqueued on a wait list for later access when the resource is freed. The WAIT statement has been added to allow synchronization of processes, that is, to specify that the controlling procedure is to suspend execution until one or more procedures have completed their execution. These statements are intended to allow the programmer to explicitly control the execution of parallel and synchronized processes.

17. SET Statement

The SET statement allows assignment of evaluated expressions and Table values to a compatible data unit or list of units including other Tables. The data units allowed are variables, subscripted variables, and Tables. Table assignments must be size and Item-length compatible or the desired result may not be obtained. Subscripted variables must specify Field name, Table name, and dimensions.

The "SET (receptacle(s)) TO (expression) THEN" statement has not been implemented since the same capability is available by using an IF THEN SET statement.

The "SET (receptacle(s)) TO (expression) SAVING (Real data unit) OVERFLOW (label)" statement has not been implemented since the same capability is available by using the REM operator within an IF THEN GOTO statement.

E. EXPRESSIONS

The expression types implemented in CMS-2FS include Real, Bits, Status, Character, and Boolean. Real expressions include Integer, Fixed-point, and Floating-point data types.

1. Real

In Real expressions, a binary operator has been added which gives the remainder after division of two real primaries (REM). In addition, two functional operators have been added: The operator NUMBER, with parameter Bits primary, converts a Bits value into an Integer value; and the operator CHARCODE, with parameter Character primary, converts a character symbol code into an internal integer representation.

The type conversion conventions for mixed operand expressions are,

a. Decimal and octal operands will be converted to decimal results.

b. Integer and Fixed-point operands will be converted to Fixed-point results.

c. Integer and Floating-point operands will be converted to Floating-point results.

d. Fixed-point and Floating-point operands will be converted to Floating-point results.

Where the receptacle is of smaller precision than the evaluated expression, the next least significant digit will be rounded and the remaining digits truncated.

2. Bits

The SHIFT statement has been replaced by the following unary operators in Bits expressions: shift left-logical (SHLL), shift right-logical (SHRL), circular end-around shift left (CIRSHLL), and circular end-around shift right (CIRSHRL).

The binary operators, logical and (ANDL), logical or (ORL), and logical complementation (NOTL), have been implemented to allow logical operations on Bits operands. If the operands are of unequal length, the shorter one is right justfied and filled with zeroes on the left before a binary operation is completed.

The functional operator BIT has been renamed BITSTRING, but performs the same function of converting a Real primary to a Bits value and extracting a specified number of binary digits from it starting at any position in the string.

3. Character

One functional operator has been added to Character expressions. The CODECHAR operator, with parameter Real primary, converts an integer value into a single character symbol code. The CHAR operator has been renamed SUBCHAR, but performs the same function of extracting a specified

number of character symbols from a Character primary starting at any character position in the string.

4. Boolean

The unary operator COMP or complementation has been renamed NOT. The binary operators allowed in Boolean expressions are conjunction (AND) and disjunction (OR).

The comparison operators in relational expressions have been changed to: equal (=), not equal (¬=), less than (<), greater than (>), less than or equal (<=), and greater than or equal (>=). The following restrictions are placed on operand comparisons: Real operands are converted to the same type before they are compared, Bits operands of all lengths are compared from right-to-left and only for equal or not equal, Character operands are compared from left-to-right for all lengths with longer strings having a value of greater than if all other characters are equal, and Status operands are compared by their Integer value. In a Status variable declaration, the first Status constant associated with the variable is assigned a value of zero increasing by increments of one for each additional Status constant.

If an expression involves operations of the same priority they are performed from left-to-right. In parenthesized expressions, the innermost parenthesized expressions are evaluated first. A hierarchy of evaluation is shown in Table X.

PRIORITY	OPERATOR	DEFINITION
1	ABS,-	ABSOLUTE VALUE, UNARY MINUS
	CAT	STRING CONCATENATION
	NOTL	BITSTRING COMPLEMENTATION
2	**	EXPONENTIATION
	SHLL	SHIFT LEFT LOGICAL
	SHRL	SHIFT RIGHT LOGICAL
	CIRSHLL	CIRCULAR SHLL
	CIRSHRL	CIRCULAR SHRL
3	*,/	MULTIPLICATION, DIVISION
	REM	REMAINDER AFTER DIVISION
	ANDL	LOGICAL AND
	ORL	LOGICAL OR
4	+,-	ADDITION, SUBTRACTION
5	=,¬=	EQUAL, NOT EQUAL
	<,>=	LESS THAN, GREATER THAN OR EQUAL
	>,<=	GREATER THAN, LESS THAN OR EQUAL
6	NOT	NEGATION
7	AND	CONJUNCTION
	OR	DISJUNCTION

TABLE X - CMS-2RS MIXED EXPRESSION EVALUATION

IV. LANGUAGE ANALYSIS

The following sections describe Simple left-to-right parsing, with k symbols of look-ahead, Context-Free grammars (SLR(k)), an SLR(1) parsing algorithm, and SLR(1) syntax analysis of CMS-2RS.

A. SLR(k) CONTEXT-FREE GRAMMARS

The class of LR(k) grammars is the largest class of grammars which can be naturally parsed left-to-right and bottom-up using a deterministic pushdown automaton. Among the various sub-classes of LR(k) grammars are precedence and bounded-right-context grammars. Since LR(k) grammars have these properties, CMS-2RS was defined as an SLR(1) context-free grammar. This section describes context-free grammars and SLR(k) grammars as presented by references 10 and 9.

1. Grammar

A Grammar is defined by a quadruple of sets of terminal symbols (VT), non-terminal symbols (VN), a start symbol (S), and productions (P). The letters VT denote a finite set of terminal or non-reduceable symbols from the vocabulary (V) of the grammar. The letters VN denote a finite set of non-terminal or variable symbols from which various strings of terminals and non-terminals can be derived. The letter S denotes the start non-terminal symbol from which all strings of terminals in the grammar are derived. The letter P denotes the finite set of productions or relations between the left part symbols in VN and the right part strings of symbols in V*, where V=VN U VT and V* denotes the set of all strings composed of symbols in V including the empty string.

2. Context-free Grammars

A Context-free grammar is one in which every pro-

duction is of the form v=>W, where v is a single variable and w is any string other than the empty string.

3. Direct Production or Reduction

For a grammar G, a string v Directly Produces a string w or w Directly Reduces to v, written v=>w, means: if U=>u is a rule of G, and x and y are strings which may be empty, then v=xUy, and w=xuy.

4. Production or Reduction

A string v produces a string w or w reduces to v, written v=>+w, means: if n>0 and n is the number of productions from v to w, then v=u(0)=>u(1)=>u(2)=>...=>u(n)=w, where if v=w or v=>+w then v=>*w, and u(i) denotes the ith production.

5. Sentential Form

A string x is called a sentential form if x is derivable from the start symbol S, written S=>*x.

6. Sentence

A sentence is a sentential form consisting only of terminal symbols.

7. Language

The language of the grammar, written L(G(S)), is the set of sentences: L(G)=(x|S=>*x and x is in VT+).

8. Phrase

If w=xuy is a sentential form then u is called a phrase of the sentential form w for the non-terminal U if U=>+u and S=>*xUy. In addition, u is called a simple phrase if S=>*xUy and U=>u. That is, not only must a reduction be possible using a production in the grammar, but the reduced sentential form must also be derivable from the start symbol. In addition, u is called a simple phrase if S=>*xUy and U=>u, that is, not only must a reduction be possible using a production in the grammar, but also the reduced sentential form must be derivable from the start symbol.

9. Canonical Derivation

A direct derivation xUy=>xuy is canonical, written xUy=|>xuy, if y contains only terminals. A derivation w=>+v

is canonical, written w=|>v, if every direct derivation in it is canonical.

10. Parse

A Parse of a sentential form is the construction of its derivation.

11. Left-Right, Bottom-Up Parse

In a left-right, bottom-up parse, the leftmost simple phrase (the Handle) of the current sentential form is always reduced. Thus, the string to the right of this phrase always contains only terminals.

12. Rightmost Derivation

A rightmost derivation is one in which at each step the rightmost variable in the sentential form is replaced.

13. LR(k) Grammar

A Context-Free grammar is LR(k) if for any sentential form w=xuy the following holds: there is a unique form for w, and there is a rightmost derivation S=>*xUy=>xuy, where U is replaced by u at the last step, and U and u can be uniquely determined by scanning w from left-to-right up to a point at most k symbols beyond u.

B. SLR(k) PARSING ALGORITHM

This section describes Context-free LR(0) and SLR(1) grammar parsing as presented by reference 9.

1. Context-free Parsing

Let G be a Context-Free grammar, and suppose that the productions P are numbered 1,2,...,p. Let V be in the set VT U VN*. Then:

a. A left parse of V is a sequence of productions used in a leftmost derivation of v from S.

b. A right parse of v is the reverse of a sequence of productions used in a rightmost derivation of v from S in G.

Consider the grammar where the productions are numbered as shown:

a. E::=E+T--1

b. E::=T--2

c. T::=T*F--3

d. T::=F--4

e. F::=(E)--5

f. F::=a--6

The left parse of the sentence a*(a+a) is 23465124646. The right parse is 64642641532. The right most derivation from E is:

a. E=>E+T:1

b. =>E+T*F:3

c. =>E+T*a:6

d. =>E+F*a:4

f. =>E+a*a:6

g. =>T+a*a:2

h. =>F+a*a:4

i. =>a+a*a:6

Writing in reverse, the sequence of productions used in this derivation gives the right parse 64266431. In general a right parse for a string v in a grammar G is a sequence of productions which can be used to reduce v to the start symbol S.

The parsing proceeds using essentially a right parser cycling through all possible rightmost derivations, in reverse, that are consistent with the input. A move consists of scanning the string on top of the pushdown stack to see if there is a right side of a production that matches the symbols on the top of the stack. If so, a reduction is made, replacing these symbols by the symbol on the left side of the production.

If no reduction is possible, then the next input symbol is placed on the pushdown stack and another attempt is made to reduce the stack. This process continues until the grammar has been parsed or an error occurs.

2. LR(0) and SLR(1) Parsing

In reference 9, De Remer states that to construct an LR(0) parser for a Context-Free grammar G, configuration sets must be computed. A configuration set represents a state of the parse, that is, which parts of which productions may have been used to generate the input string to the point of the parse.

Each non-empty configuration set has one or more successors or configuration sets. In general, a configuration set S(i) has an s-successor for each symbol s in V that is preceded by a marker in one or more of S(i)'s configurations. A marker is a pointer to the next possible symbol to be read in the input list.

An s-successor state consists of a basis set combined with a closure set. The basis set consists of all configurations in S(i) having a marker before an s, but with the marker moved to follow the S(i). The closure set is defined recursively to be the largest set of configurations that can be derived from the basis set until a terminal symbol is reached.

An LR(0) parser for a grammar G is represented by the set of all configuration sets, where each set is a state of the parse with an accessing symbol, and a list of possible symbols which can be read next with an indication of whether to read the next symbol and go on to another state, or to reduce and go on to another state.

A Context-Free grammar is said to be SLR(1) if and only if each of the inadequate states of its successor states has mutually disjoint (simple) 1-symbol look-ahead sets which allow the parser to determine which reduction to make. An inadequate state is one in which the parser must look ahead k symbols in order to determine which reduction to make.

An SLR(1) parser for a grammar, therefore, is represented by the same set of configuration sets as is required for an LR(0) parser with the addition of the simple 1-look-ahead sets.

SLR(1) parsers make a decision to reduce based upon all the symbols in the parse stack plus one more from the input text. The parse is accomplished by restructuring the stack after each reduction and saving the state of the parse to allow the parser to know where it has been so that it can make the right decision for the next reduction.

C. SLR(1) SYNTAX ANALYSIS OF CMS-2RS

The SLR(1) Syntax Analyzer and parsing table generator produced by Woods and described in reference 11 was chosen to analyze the grammar because of its speed of execution.

In developing an SLR(1) grammar for CMS-2RS, it was necessary to write a complete initial description of the language and then take small sections of the grammar and analyze them until they were SLR(1). In that fashion, the grammar was successively built up and revised until a complete SLR(1) grammar was obtained. The execution time, using Wood's PL-360 based Syntax Analyzer, was on the order of 9 to 10 seconds of CPU time for about 200 productions on an IBM-360 Model 65 computer.

During the grammar analysis it became apparent that combining the same sub-strings of symbols on the right part of several productions into new productions greatly reduced the number of terminal transitions. This allowed a larger grammar to be handled for the same table sizes in Wood's Syntax Analyzer. The resultant grammar is thus highly optimized in terms of parsing table size. The limit of 255 productions in the Syntax Analyzer, however, did require that the revised grammar be split. The data declarations and header declarations were handled by separate grammars since they are blocked by easily recognizable beginning and ending delimiters. The implementation of this split and its implications are discussed in Section V.

V. DESIGN OF THE TWO-PASS CMS-2RS COMPILER

The purpose of this project is to take the first step toward a practical implementation of CMS-2RS. A two-pass compiler was selected taking into consideration main memory requirements and project modularity. Also, having two passes offers a potential for code optimization in the second pass. This approach facilitates the task of making a significant start on a useful compiler in a limited time by isolating the analysis and synthesis functions from the detailed code generation, storage assignment, and interface handling functions.

A. FIRST PASS

The first pass performs four basic functions. It scans the characters of the source program, builds the symbol table, parses program sentences, and generates an intermediate language representation of the source program.

1. Lexical Analyzer

The lexical analysis is performed by the procedure SCANNER. In writing the scanner's case statement, it was decided to use a different case for each delimiter (except for "**" and "¬=" which are handled with "*" and "¬" respectively). By doing this, there is no need for having a table of all the tokens in the language and the corresponding table lookup mechanism. SCANNER does all the symbol table lookup and enter operations required by the compiler. When an identifier is scanned, the symbol table is searched to determine if this identifier is a reserved word or if it is an already entered name. If not found, the new name is entered in the symbol table. From that point on each

identifier is represented by its index into the symbol table so that nc further search is ever needed.

As will be explained later, two independent analyzer procedures are used to parse the declarations and the dynamic statements. For this reason it became necessary to build a linkage between SCANNER and these two analyzers that would allcw the implementation of the necessary switching mechanisms. The linking functicn is performed by the procedure SCAN. SCAN is called by the analyzers when the next tcken is desired. SCAN calls SCANNER to obtain the next symbol in the input stream. When SCANNER returns certain reserved words, SCAN switches control to the appropriate analyzer. Fcr those tokens that appear in bcth grammars, SCAN allcws for two different internal representaticns. SCAN alsc facilitates the implementation of certain symbol table handling mechanisms as will be explained later.

2. Symbol Table Design

After all memory allocations tc variables used by the compiler are satisfied, the remainer of the assigned region is allocated to the symbol table. The symbol table is divided into two main parts: the identifier directcry and the constant table.

a. Block Structured Identifier Directory

The following method was chosen to implement the two level (global, local) block structure cf CMS-2 using hash addressing for table search. Every identifier is ccncatenated to a two character prefix which indicates the level of such identifier. Prefix "00" correspond tc global scope, while all other prefixes frcm "01" to "99" correspond to local scope. Therefore, every identifier declared in a SYS_DD is entered in the identifier directory with prefix "00". For example, variable "ABLE" would be known to the compiler as "00ABLE". A block counter is maintained and each time the reserved words SYS_PROC and SYS_PROC_R are

encountered this counter is incremented and the current prefix is correspondly altered. Thus, variables declared within LOC_DD's or AUTO_DD's are entered in the directory with the prefix corresponding to the sequential number of the SYS_PROC where these data designs appear. When SCANNER recognizes an identifier, it first looks for its presence in the identifier directory with the current prefix and, if not found, then with the global prefix ("00").

b. Hash Addressing

The chained hash addressing technique described by Gries [Ref. 12] is used to access the identifier directory. A hash table size of 1229 was chosen to minimize the number of collisions in medium size programs. This size can be changed to any desired prime number, requiring only one minor change in the procedure HASH. The hashing scheme uses the number of characters in the word, the second and third characters, and the last and next to the last characters as arguments. The hashing function is applied to the identifier after it has been concatenated to the proper prefix as previously discussed. Procedures RESERWRD and LOOKUP handle all the required symbol table searching. The list of reserved words is preloaded in the identifier directory by the procedure INITIALIZE. No prefix is concatenated with reserved words. In addition, no collisions occur when hashing reserved words, so RESERWRD needs only to compare the identifier scanned with one entry in the directory to find if it is a reserved word. Caution must be exercised if the size of the hash table is changed or if the language is revised to include new reserved words to insure that this property is preserved.

Procedure LOOKUP follows the basic flow chart found in page 222 of Ref. 12. LOOKUP adds the current block number prefix to the identifier and then hashes it and searches the identifier directory. If the identifier is not found then LOOKUP tries again using prefix "00." If still

nct fcund, the identifier is entered in the directory with the current prefix.

c. Identifier Directory Design

The identifier directory was designed taking in consideration the large variety of descriptors required to represent the wide vaiiety of conditions encountered in the language. Appendix D explains the descriptor formats in detail. The descriptors are stored in the constant table; hence, the identifier directory can have a simple and uniform structure. Each identifier directory entry uses 15 bytes distributed in four fields as shown below.

(1) Ten bytes are reserved in the identifier's name field to acccmodate a maximum of ten characters (eight characters from the scurce plus two in the prefix).

(2) One byte is used for the semantic ccde. The semantic code is an integer from 1 tq 148 that identifies the token recognized by SCANNER to procedure SCAN.

(3) Two bytes hold the chain field. This field is used in case of collisions to stcre a pointer connecting entries with the same value.

(4) Two bytes are used for the descriptor pointer field. This field contains a pointer to the constant table where the descriptor containing all the attributes of this identifier is stored.

d. Constant Table Design

The constant table is essentially a large array used to store constant values as they are recognized by the scanner and descriptors for identifiers as previously indicated. When procedure SCANNER scans a constant, it stores it in the constant table. This includes character ccnstants as well as numeric or bits constants. The first byte of every entry in the constant table is used to store the length of the entry , i.e., the number of bytes that

follow. In the case of numeric constants, the need arises to differentiate between the six different types of numeric constants used in the language. In order to do this, the low order five bits of the header byte are used to indicate the length of the entry as explained above. The remaining three high order bits are used to indicate the type of numeric constant according to the following code:

000 for octal integer
001 for octal fixed
010 for octal real
011 for decimal integer
100 for decimal fixed
101 for decimal real.

3. Syntax Analyzers

The syntactic analysis is performed by the procedures ANALYZE and BANALYZE. ANALYZE is the main grammar parser. It is called once by procedure MAIN and returns only when the source program has been completely parsed. ANALYZE calls SCAN each time a new token is to be read. When SCAN recognizes any of the reserved words SYS_DD, AUTO_DD, LOC_DD, PROCEDURE, or FUNCTION it calls BANALYZE which is the second parser. BANALYZE handles the declarations grammar. It also uses SCAN in the same manner as ANALYZE. When BANALYZE completes the parsing of a data design or a procedure or function declaration it returns to the place where it was called by SCAN, and SCAN passes the corresponding token to ANALYZE which continues execution. With this scheme, whole declarations are seen as a single terminal symbol by the main grammar. This concept, of course, can be extended to any number of parsers. For example, when a language is rather extensive (a classic example would be PL/I), it may become extremely difficult to develop a single grammar of a given type. This job is greatly simplified by dividing the task into a main grammar

and several component grammars. In addition, the most appropriate parser can be used for each component grammar. For example, a component grammar could handle expressions and use an operator precedence parser. Using the intermediate linkage concept between the scanner and the syntax analyzers as explained before, all that is needed is a special symbol to delimit each component grammar. These special symbols can be part of the syntax of the language or they can be added to the input stream by the compiler itself.

4. Backus-Naur Form (BNF)

Appendix A contains a complete listing, using Backus-Naur Form notation, of the SLR(1), context free grammar developed for the CMS-2RS language. The grammar is divided into a main grammar and a component grammar. The symbols <SYS DATA DECL>, <LOCAL DATA DECL>, <AUTO DATA DECL> and <SUB_ROUTINE DECL> are treated in the main grammar as terminal symbols. The component grammar handles these declarations. The mechanisms to implement this scheme are explained in Section 3 above.

5. Intermediate Language (IL)

Generation of the object program in an intermediate language form is one of the primary tasks of the first pass. Due to time limitations, the generation of intermediate language could not be implemented. An IL format that could be used in future work on this compiler is proposed in this section.

a. IL Format

Each entry in the IL table consist of three bytes. The high order seven bits of the first byte are used to represent the operation code. the low order remaining bit is used to indicate an indirect operation. A set of operation codes with their meanings is shown in Appendix E.

The codes shown should be sufficient to represent the entire CMS-2RS language. The second and third bytes are used for the operand field. In most cases this field will contain the address of the operand's entry in the identifier directory or the constant table.

6. First Pass Output

The first pass output should consist of two disk files, one for the symbol table and another for the intermediate language generated by the dynamic statements. Future work required to complete the first pass includes IL generation, and the writing of two routines, one to write the symbol table into a disk file and another to write an IL buffer into a second file.

B. SECOND PASS

The main task performed by the second pass will be the translation of intermediate language into machine language (ML). Routines must be developed to read the IL file and produce ML which executes on the target machine. This routines must insert the ML necessary to compute subscript values, perform data type conversion, map actual to formal parameters, and implement dynamic allocation. Some code optimization capability should also be included here.

Storage areas for variables and tables must be assigned in this pass. Routines must be provided that will process the symbol table file and perform static allocation or generate dynamic allocation mechanisms. This pass must allow for linkage with other program modules. Communication links must be established for external references. These links should be compatible with the requirements of the linking routines of the target machine's operating system.

VI. CONCLUSIONS

Several attemps have been made, with less than optimal results, to implement a universal language and compiler system. CMS-2 is a system of this type, but it has met with limited success when used to implement large, self-contained computer systems. To get around the language limitations, extensive use of the direct code option has been made. This practice defeats the purpose of a high level language and creates problems in system design and maintenance such as correctness of initial program logic and program segment integrity. If the potential of a high level language and compiler system are to be fully realized, there must be controls over which language constructs are allowed at each level of software. Furthermore, insertion of direct code should not be allowed. The CMS-2RS system is a step in that direction.

A. RESULTS

The most apparent ambiguities, redundancies, and limitations which exist in the CMS-2 program, data, and control structures have been identified. Proposed corrections to these deficiencies have been incorporated in CMS-2RS. An SLR(1) context free grammar has been defined for this language using BNF notation. This grammar is suitable for the construction of parsing tables using an automatic compiler generating system. Such a system, using Wood's SLR(1) ANALYZER, was employed to produce parsing tables for the first pass of the CMS-2RS compiler.

In order to support the structured programming concept in a modular library environment, changes where made to the external referencing mechanisms of CMS-2 so that external references are only allowed between different compile-time Systems. Header declaration statements were also revised to

prevent modifications of an Element's internal data allocation and control structure representation. This CMS-2 construct facilitates the violation of program segment integrity and allows careless program design and maintenance.

The concept of grammar segmentation was implemented in the CMS-2RS compiler. Data designs and dynamic statements were grouped into separate grammars which are parsed by different sections of the compiler. This concept is easily extendable to other sections of the grammar such as expressions and header statements.

B. FUTURE WORK

Several tasks need to be completed in order to fully implement the CMS-2RS system. The work required to complete the first pass includes the completion of intermediate language design and the writing of the semantic routines required for the translation of dynamic statements into intermediate language form. Also, header and user file declarations need to be implemented. This would be accomplished by adding the header and file declaration statements to the declarations' grammar and writing the correspondig semantic routines. File manipulation operators should be added to the main grammar.

The next step is to write the second pass. The second pass must handle a variety of functions including memory allocation, data type convertion, subscript calculation and checking, parameter mapping, and code generation and optimization.

Further development efforts should include studies of the desirability of adding list processing constructs to the CMS-2RS language.

APPENDIX A

BNF DESCRIPTION OF CMS-2FS

1. MAIN GRAMMAR

```
001  <SYS>  ::=  <SYS DECL  HEAD>  <SYS ELEM>  <END PHRASE>

002  <SYS DECL HEAD>  ::=  SYSTEM <ID>  $
003                     |  <SYS DECL HEAD>  <SYS ELEM>

004  <SYS ELEM>  ::=  <HEADER DECL>
005                |  <SYS DATA DECL>
006                |  <SYSPROC DECL HEAD>  <SYSPROC ELEM>  <END PHRASE>
007                |  <SYSPROCREN DECL HEAD>  <SYSPROCREN ELEM>  <END PHRASE>
008                |  EXTREF  <ID>  $

009  <SYSPROC  DECL HEAD>  ::=  SYS_PROC  <ID>  $
010                          |  <SYSPROC DECL HEAD>  <SYSPROC ELEM>

011  <SYSPROCREN DECL HEAD>  ::=  SYS_PROC_R  <ID>  $
012                            |  <SYSPROCREN DECL HEAD>  <SYSPROCREN ELEM>

013  <SYSPROC ELEM>  ::=  <SYSPROCREN ELEM>
014                   |  <LOCAL DATA DECL>

015  <SYSPROCREN ELEM>  ::=  <SUB-ROUTINE DECL>  <STM CLAUSE>
016                      |  <AUTO DATA DECL>

017  <END PHRASE>  ::=  END  $

018  <STM>  ::=  <LABEL DEFN>  <BASIC STM>
019          |  <BASIC STM>

020  <LABEL DEFN>  ::=  <LABEL>

021  <BASIC STM>  ::=  <SIMPLE STM>
022               |  <VARY CLAUSE>  <DO CLAUSE>
023               |  <CASE CLAUSE>  <CASE LIST>  <ELSE CASE>
024               |  <WHILE CLAUSE>  <DO CLAUSE>
025               |  <IF CLAUSE>  <THEN CLAUSE>
```

```
026  <SIMPLE STM>  ::=  <BEGIN HEAD>  <STM CLALSE>  END
027                   |  <SET CLAUSE>  TO  <EXPR>
028                   |  <SWAP CLAUSE>  <FOR CLALSE>
029                   |  <SEARCH CLAUSE>  <FOR CLAUSE>
030                   |  <INPUT CLAUSE>  <RECEPTACLE CLAUSE>
031                   |  <OLTPUT CLAUSE>  <SOURCE CLAUSE>
032                   |  <OUTPUT CLAUSE>
033                   |  <ENCCDE CLAUSE>  <SOURCE CLAUSE>
034                   |  <DECODE CLAUSE>  <RECEPTACLE CLAUSE>
035                   |  <PACK CLAUSE>  <SOURCE CLALSE>
036                   |  <UNPACK CLAUSE>  <RECEPTACLE CLAUSE>
037                   |  CALL  <PROCEDURE STM>
038                   |  <CCNTROL PHRASE>  <LABEL>
039                   |  RETURN  <DATA LNIT>
040                   |  <COBEGIN HEAD>  <STM CLALSE>  COEND
041                   | <RESERVE PHRASE> ) <SIMPLE STM>
042                   | <WAIT CLAUSE> )
043                   |  RETURN
044                   |  STCP

045  <VARY CLAUSE>  ::=  VARY  <LOOP CLAUSE>
046                   |  <VARY CLAUSE>  ,  <LCCP CLAUSE>

047  <LCCP CLAUSE>  ::=  <INITIAL CLALSE>  <TEST CLAUSE>
048                   |  <INITIAL CLAUSE>  <INCR CLAUSE>  <TEST CLAUSE>

049  <INITIAL CLAUSE>  ::=  <REAL VAR>  FROM  <SIGNED REAL PRI>

050  <INCR CLAUSE>  ::=  STEP  <SIGNEC REAL PRI>

051  <TEST CLAUSE>  ::=  THRU  <SIGNEC REAL PRI>

052  <CASE CLAUSE>  ::=  CASE  <DATA LNIT>

053  <CASE LIST>  ::=  OF  <INDEX CASE LIST>
054                 |  OF  <LABEL CASE LIST>

055  <INDEX CASE LIST>  ::=  <BASIC STM CLAUSE>
056                       |  <INDEX CASE LIST>  <BASIC STM CLAUSE>

057  <LABEL CASE LIST>  ::=  <LABEL CASE>
058                       |  <LABEL CASE LIST>  <LABEL CASE>

059  <LABEL CASE>  ::=  <CASE LABEL>  <BASIC STM CLAUSE>
060                  |  <CASE LABEL>  <LABEL CASE>

061  <CASE LABEL>  ::=  <REAL CONS>  :
062                  |  <STATLS CONS>  :
```

```
063  <ELSE CASE>  ::=  ELSE  <BASIC STM>

064  <EASIC STM CLAUSE>  ::=  <BASIC STM>  $

065  <STM CLAUSE>  ::=  <STM>  $

066  <WHILE CLAUSE>  ::=  WHILE  <BOOL EXPR>

067  <DO CLALSE>  ::=  DO  <SIMPLE STM>

068  <IF CLAUSE>  ::=  IF  <BOOL EXPR>

069  <THEN CLAUSE>  ::=  THEN  <STM>
070                    |  <TRUE PART>  <STM>

071  <TRUE PART>  ::=  THEN  <SIMPLE STM CLAUSE>
072                 |  THEN  <IF CLAUSE>  <TRUE PART>  <SIMPLE STM CLAUSE>

073  <SIMPLE STM CLAUSE>  ::=  <SIMPLE STM>  ELSE

075  <BEGIN HEAD>  ::=  BEGIN
075                  |  <BEGIN HEAD>  <STM CLALSE>

076  <COBEGIN HEAD>  ::=  COBEGIN
077                    |  <COBEGIN HEAD>  <STM CLAUSE>

078  <SET CLAUSE>  ::=  SET  <DATA UNIT>
079                  |  <SET CLAUSE>  <DATA UNIT CLAUSE>

080  <EXPR>  ::=  <REAL EXPR>
081            |  <BITS EXPR>
082            |  <CHAR EXPR>
083            |  <STATUS EXPR>
084            |  <BOOL EXPR>
085            |  <TABLE IDENTIFIER>

086  <REAL EXPR>  ::=  <REAL TERM>
087                 |  <REAL EXPR>  <ADDSUB OPER>  <REAL TERM>

083  <ADDSUB OPER>  ::=  +
089                    |  -

090  <REAL TERM>  ::=  <REAL SECN>
091                 |  <REAL TERM>  <MULDIV OPER>  <REAL SECN>
```

```
092  <MULDIV OPER>  ::=  *
093                     |  /
094                     |  REM

095  <REAL SECN>  ::=  <SIGNED REAL PRI>
096                 |  <REAL SECN>  **  <SIGNEC REAL PRI>

097  <SIGNED REAL PRI>  ::=  <REAL PRI>
098                       |  -  <REAL PRI>
099                       |  ABS  <REAL PRI>

100  <REAL PRI>  ::=  <REAL CONS>
101                |  <REAL DATA UNIT>
102                |  (  <REAL EXPR>  )
103                |  (  <REAL EXPR>  )  ..  <REAL CONS>
104                |  <REAL FUNC NAME>  <INPUT, FARAMS>  )

105  <REAL CCNS>  ::=  <REAL>
106                 |  <TAG>

107  <REAL DATA UNIT>  ::=  <REAL VAR>
108                      |  <REAL FIELD NAME>  <SUBSCRIPT CLAUSE>  )

109  <BITS EXPR>  ::=  <BITS FACTOR>
110                 |  <BITS EXPR>  <LCGICAL CFER>  <BITS FACTOR>

111  <LCGICAL OPER>  ::=  ANCL
112                    |  ORL

113  <BITS FACTCR>  ::=  <BITS SECN>
114                   |  <BITS FACTOR>  <SHIFT CFER>  <SIGNED REAL PRI>

115  <SHIFT CPER>  ::=  SHLL
116                  |  SHRL
117                  |  CIRSHLL
118                  |  CIRSHRL

119  <BITS SECN>  ::=  <BITS PRI>
120                 |  NCTL  <BITS PRI>

121  <BITS PRI>  ::=  <BITS CONS>
122                |  <BITS CATA UNIT>
123                |  (  <BITS EXPR>  )
124                |  <BITS FUNC NAME>  <INPUT PARAMS>  )

125  <BITS DATA UNIT>  ::=  <BITS VAR>
126                      |  <BITS FIELD NAME>  <SUBSCRIPT CLAUSE>  )
```

```
127  <CHAR EXPR>  ::=  <CHAR EXPR>  CAT  <CHAR FRI>
128                 |   <CHAR PRI>

129  <CHAR PRI>  ::=  <CHAR CCNS>
130                |  <CHAR CATA UNIT>
131                |  (  <CHAR EXPR>   )
132                |  <CHAR FUNC NAME>  <INPUT FARAMS>   )

133  <CHAR DATA UNIT>  ::=  <CHAR VAR>
134                      |  <CHAR FIELD NAME>  <SUESCRIPT CLAUSE>   )

135  <STATUS EXPR>  ::=  <STATUS CONS>
136                   |  <STATUS DATA UNIT>
137                   |  <STATUS FUNC NAME>  <INPUT PARAMS>   )

138  <STATUS DATA UNIT>  ::=  <STATUS VAR>
139                        |  <STATUS FIELD NAME>  <SUBSCRIPT CLAUSE>   )

140  <BOOL EXPR>  ::=  <BCCL SECN>
141               |    <BOOL EXPR>  <BCOL OPER>  <BOOL SECN>

142  <BCOL OPER>  ::=  AND
143               |    OR

144  <BOCL SECN>  ::=  <BCCL PRI>
145               |    NOT  <BOOL PRI>

146  <BCOL PRI>  ::=  <BCOL CCNS>
147               |   (  <BOOL EXPR>   )
148               |   <BOOL CATA UNIT>
149               |   <BCOL FUNC NAME>  <INPUT FARAMS>   )
150               |   <REL EXPR>

151  <BCOL DATA UNIT>  ::=  <BOCL VAR>
152                      |  <BOOL FIELD NAME>  <SUBSCRIPT CLAUSE>   )

153  <REL EXPR>  ::=  <REAL FRI>  <REL CPER>   <REAL PRI>
154               |   <BITS PRI>  <REL OPER>   <REAL PRI>
155               |   <CHAR PRI>  <REL CPER>   <CHAR PRI>
156               |   <STATUS EXPR>  <REL OPER>   <STATUS EXPR>

157  <REL OPER>  ::=  =
158               |   ¬=
159               |   <
160               |   >
161               |   <  =
162               |   >  =
```

```
163  <DATA UNIT>  ::=  :  <REAL DATA UNIT>
164                  |  :  <BITS DATA UNIT>
165                  |  :  <CHAR DATA UNIT>
166                  |  :  <BOOL DATA UNIT>
167                  |  :  <STATUS DATA UNIT>
168                  |  :  <TABLE IDENTIFIER>
169                  |  :  <SUBSCRIPT CLAUSE>  )

170  <DATA UNIT CLAUSE>  ::=  ,  <REAL DATA UNIT>.
171                         |  ,  <BITS DATA UNIT>
172                         |  ,  <CHAR DATA UNIT>
173                         |  ,  <STATUS DATA UNIT>
174                         |  ,  <BOOL DATA UNIT>
175                         |  ,  <TABLE IDENTIFIER>
176                         |  ,  <SUBSCRIPT CLAUSE>  )

177  <SUBSCRIPT CLAUSE>  ::=  <TABLE IDENTIFIER>  (  <SIGNED REAL PRI>
178                         |  <SUBSCRIPT CLAUSE>  ,  <SIGNED REAL PRI>

179  <TABLE IDENTIFIER>  ::=  <TABLE NAME>
180                         |  <SUB-TABLE NAME>
181                         |  <ITEM_AREA NAME>

182  <SWAP CLAUSE>  ::=  SWAP  <DATA UNIT>

183  <SEARCH CLAUSE>  ::=  SEARCH  :  <DATA STRUCTURE>

184  <DATA STRUCTURE>  ::=  <SUBSCRIPT CLAUSE>  )
185                      |  <FILE NAME>  (  <SIGNED REAL PRI>

186  <FOR CLAUSE>  ::=  FOR  <DATA UNIT>

187  <PROCEDURE STM>  ::=  <PROCEDURE NAME>  <ACTUAL PARAMETER LIST>  )
188                    |  <PROCEDURE NAME>

189  <ACTUAL PARAM LIST>  ::=  <INPUT PARAMS>  <OUTPUT PARAMS>  <LABEL PARAMS>
190                          |  <INPUT PARAMS>  <OUTPUT PARAMS>
191                          |  <OUTPUT PARAMS>  <LABEL PARAMS>
192                          |  <INPUT PARAMS>  <LABEL PARAMS>
193                          |  <INPUT PARAMS>
194                          |  <OUTPUT PARAMS>
195                          |  <LABEL PARAMS>

196  <INPUT PARAMS>  ::=  (  <EXPR>
197                    |  (  ,
198                    |  <INPUT PARAMS>  ,
199                    |  <INPUT PARAMS>  ;  <EXPR>
```

```
200  <OLTPUT PARAMS>  ::=  |  <DATA UNIT>
201                       |  |
202                       |  <OUTPUT PARAMS>  ,
203                       |  <OUTPLT PARAMS>  <CATA UNIT CLAUSE>

204  <LABEL PARAMS>  ::=  ||  <LABEL>
205                     |  <LABEL PARAMS>  ,  <LABEL>

206  <CCNTROL PHRASE>  ::=  RESUME
207                      |  RETURNTO
208                      |  GOTO

209  <RESERVE PHRASE>  ::=  RESERVE  (  <RESOURCE>
210                      |  <RESERVE PHRASE>  ,  <RESOURCE>

211  <RESOURCE>  ::=  <PROCEDURE STM>
212                |  <DATA STRUCTURE>
213                |  <CATA UNIT>

214  <WAIT CLAUSE>  ::=  WAIT  (  <PROCEDURE NAME>
215                   |  <WAIT CLAUSE>  ,  <PRCCEDURE NAME>
216  <INPUT CLAUSE>  ::=  INPUT  <FILE NAME>
217                    |  INPUT  <FILE NAME>  <FCRMAT NAME>

218  <CLTPUT CLAUSE>  ::=  CLTPUT  <FILE NAME>
219                     |  OUTPUT  <FILE NAME>  <FCRMAT NAME>

220  <ENCODE CLAUSE>  ::=  ENCODE  :  <CHAR DATA UNIT>  <FORMAT NAME>

221  <CECODE CLAUSE>  ::=  DECODE  :  <CHAR DATA UNIT>  <FORMAT NAME>

222  <PACK CLAUSE>  ::=  PACKIN  <DATA UNIT>

223  <LNPACK CLAUSE>  ::=  UNPCK  <DATA UNIT>

224  <RECEPTACLE CLAUSE>  ::=  INTO  <DATA UNIT>
225                        |'  <RECEPTACLE CLAUSE>  <DATA UNIT CLAUSE>

226  <SCURCE CLAUSE>  ::=  OUTOF  <DATA UNIT>
227                     |  <SOURCE CLAUSE>  <CATA UNIT CLAUSE>

228  <FILE NAME>  ::=  PRINT
229                 |  PNCH
230                 |  READ
231                 |  OCM
```

2. DECLARATIONS GRAMMAR

```
001  <DECL>  ::=  <SUB-ROUTINE DECL>
002             |  <DATA DECL>  <DATA ELEM>   END_DD  $

003  <SUB-ROUTINE DECL>  ::=  <PROCEDURE DECL>  $
004                        |  <PROCEDURE DECL>  <FPARM LIST>  )  $
005                        |  <FUNCTION CLAUSE>  )  $

006  <PROCEDURE DECL>  ::=  PROCEDURE  <ID>

007  <FPARM LIST>  ::=  <IN PARMS>  <OUT PARMS>  <LABEL PARMS>
008                  |  <IN PARMS>  <OUT PARMS>
009                  |  <IN PARMS>  <LABEL PARMS>
010                  |  <OUT PARMS>  <LABEL PARMS>
011                  |  <IN PARMS>
012                  |  <OUT PARMS>
013                  |  <LABEL PARMS>

014  <IN PARMS>  ::=  (  <VAR>
015                |  <IN PARMS>  ,  <VAR>

016  <OUT PARMS>  ::=  |  <VAR>
017                 |  <OUT PARMS>  ,  <VAR>

018  <LABEL PARMS>  ::=  ||  <ID>
019                   |  <LABEL PARMS>  ,  <ID>

020  <FUNCTION CLAUSE>  ::=  <FUNCTION HEAD>  (  <VAR>
021                       |  <FUNCTION CLAUSE>  ,  <VAR>

022  <FUNCTION HEAD>  ::=  FUNCTION  <DATA TYPE>  <ID>

023  <VAR>  ::=  <NUM VAR>
024           |  <BITS VAR>
025           |  <CHAR VAR>
026           |  <STAT VAR>
027           |  <BOOL VAR>
028           |  <TBL NAME>
029           |  <ITEM A. NAME>
030           |  <SUBTBL NAME>

031  <DATA DECL>  ::=  SYS_DD  <ID>  $
032                 |  LOC_DD  $
033                 |  AUTO_DD  $
034                 |  <DATA DECL>  <DATA ELEM>
```

```
035  <DATA ELEM>  ::=  <VRBL DECL HEAD>  <ITEM>  $
036                  |  <FORMAT DECL HEAD>  <DESCRGROUP>  )  $
037                  |  <TAG DECL HEAD>  <ID>  =  <REAL CONSTANT>  $
038                  |  <TABLE DECL HEAD>  <TABLE DECL>  END_TABLE  $

039  <VRBL DECL HEAD>  ::=  <VRBL HEAD>
040                      |  <VRBL DECL HEAD>  <ITEM>  ,

041  <VRBL HEAD>  ::=  VRBL  <DATA TYPE>

042  <DATA TYPE>  ::=  INTEGER  (  <REAL CONSTANT>  )
043                  |  FIXED  (  <REAL CONSTANT>  ,  <REAL CONSTANT>  )
044                  |  FLOAT  (  <REAL CONSTANT>  )
045                  |  BITS  (  <REAL CONSTANT>  )
046                  |  CHAR  (  <REAL CONSTANT>  )
047                  |  BOOLEAN
048                  |  <STATUS CLAUSE>  )

049  <STATUS CLAUSE>  ::=  STATUS  (  <STATUS CONSTANT>
050                     |  <STATUS CLAUSE>  ,  <STATUS CONSTANT>

051  <ITEM>  ::=  <ID>
052           |  <ID>  =  <CONSTANT>

053  <CONSTANT>  ::=  <NUMORTAG>
054               |  <BITS CONSTANT>
055               |  <CHAR CONSTANT>
056               |  <STATUS CONSTANT>
057               |  <BOOL CONSTANT>

058  <BOOL CONSTANT>  ::=  TRUE
059                     |  FALSE

060  <NUMORTAG>  ::=  <TAG>
061               |  <REAL CONSTANT>

062  <TAG DECL HEAD>  ::=  TAG
063                     |  <TAG DECL HEAD>  <ID>  =  <REAL CONSTANT>  ,

064  <FORMAT DECL HEAD>  ::=  FORMAT  <ID>  (  <CHAR CONSTANT>  ,
065                        |  <FORMAT DECL HEAD>  <DESCRGROUP>  ,

066  <DSCRLIST HD>  ::=  <REAL CONSTANT>  (

067  <DSCRLIST>  ::=  <DSCRLIST HD>  <DESCRIPTOR>
068               |  <DSCRLIST>  ,  <DESCRIPTOR>
```

```
069  <REPTN GROUP>  ::=  <DSCRLIST>  )

070  <DESCRGROUP>  ::=  <REPTN GROUP>
071                  |  <DESCRIPTOR>

072  <DESCRIPTOR>  ::=  <REAL CONSTANT>  <ID>  <REAL CONSTANT>
073                  |  <ID>  <REAL CONSTANT>
074                  |  <CHAR CONSTANT>
075                  |  <SLASH LIST>

076  <SLASH LIST>  ::=  /
077                  |  <SLASH LIST>  /

078  <TABLE DECL HEAD>  ::=  <TABLE CLAUSE>  )  $
079                      |  <TABLE DECL HEAD>  <TABLE DECL>

080  <TABLE CLAUSE>  ::=  <TABLE HEAD>  (  <NUMORTAG>
081                    |  INDIRCT  <TABLE HEAD>  (  <NUMORTAG>
082                    |  <TABLE CLAUSE>  ,  <NUMORTAG>

083  <TABLE HEAD>  ::=  TABLE  <ID>
084                  |  <TABLE HEAD>  ,  <ID>

085  <TABLE DECL>  ::=  <FIELD DECL>  $
086                  |  <ITEM_AREA CLAUSE>  $
087                  |  <SUBTABLE CLAUSE>  )  $

088  <FIELD DECL>  ::=  FIELD  <DATA TYPE>
089                  |  <FIELD DECL>  <FITEM>

090  <FITEM>  ::=  <ITEM>
091            |  <ID>  =  <REPETITION LIST>  <CONSTANT>  )

092  <REPETITION LIST>  ::=  (
093                      |  <NUMORTAG>  (
094                      |  <REPETITION LIST>  <CONSTANT>  ,

095  <ITEM_AREA CLAUSE>  ::=  ITEM_AREA  <ID>
096                       |  <ITEM_AREA CLAUSE>  ,  <ID>

097  <SUBTABLE CLAUSE>  ::=  SUBTABLE  <ID>  (  <NUMORTAG>
098                      |  <SUBTABLE CLAUSE>  ,  <NUMORTAG>
```

APPENDIX B

DESCRIPTION OF THE CMS-2RS LANGUAGE

The CMS-2RS language is described in the following sections using a modified metalinguistic notation which was originally developed by Backus and is described in reference 12. This modified notation uses the following basic symbols:

::=	A connective meaning "is defined to be".
|	A connective meaning "or defined to be".
< >	Delimiting brackets enclosing a metalinguistic variable.
{ }	Delimiting braces enclosing brackets meaning "more than one metalinguistic variable", where each variable is separated by a comma, dollar sign ($), or space as appropriate.

The sections below describe the syntax and semantics, and give examples for each structure or statement in the language. Program structure is described in Section 1. Section 2 describes the basic symbols and delimiters of the language that are formed from the CMS-2RS alphabet. Data structures and referencing conventions are described by Section 3, followed by control structures in Section 4. Section 5 describes Real, Bits, Status, Character and Boolean expressions. The header and debug statements were not implemented in the present version of CMS-2RS and are not included in this appendix. In the examples below, the notation "<metalinguistic variable>" is used when the Item has not yet been defined, but its inclusion is necessary for a complete description of the example. References to imposed limits on the numbers of elements allowed in the statements that follow are actually limitations imposed by CMS-2RS compiler and its table sizes.

1. Program Structure

a. System and System Elements

Syntax:

```
<SYSTEM>::=SYSTEM <IDENTIFIER>$
          |{<SYSTEM ELEMENT>}END$
<SYSTEM ELEMENT>::=<HEADER DECLARATION>
                  |<SYSTEM DATA DECLARATION>
                  |<SYSTEM PROCEDURE DECLARATION>
                  |<SYSTEM REENTRANT PROCEDURE
                       DECLARATION>
                  |EXTREF <IDENTIFIER>$
```

Semantics:

A System is a compile-time grouping of System Elements, headers, debug statements, and externally referenced (EXTREF) System Element identifiers. System Elements are either System Data, System Procedure, or System Reentrant Procedure declarations. The order in which Elements are listed is up to the programmer and specifies the desired order of compilation and linking. A header may only be included once at the beginning of a System. The total number of Elements allowed in a System is ninty-nine (99).

Examples: (see Section 1d.)

b. System Procedures and System Reentrant Procedures

Syntax:

```
<SYSTEM PROCEDURE DECLARATION>::=SYS_PROC <IDENTIFIER>$
                                      {<SYSTEM PROCEDURE
                                       ELEMENT>} END$
<SYSTEM REENTRANT PROCEDURE DECLARATION>::=SYS_PROC_R
                            <IDENTIFIER>$ {<SYSTEM REENTRANT
                            PROCEDURE ELEMENT>} END$
```

Semantics:

Both types of System Procedures are the basic building blocks of the CMS-2RS language. Procedures may reference each other only through their prime procedures and are separately compileable and executable. Reentrant procedures consist of separate sections of data and non-changeable code.

Examples: (see Section 1d.)

c. System Procedure Elements

Syntax:

```
<SYSTEM PROCEDURE ELEMENT>::=<LOCAL DATA DECLARATION>
                            |<AUTO DATA DECLARATION>
                            |<PROCEDURE SUB-ROUTINE
                                 DECLARATION>
                            |<FUNCTION SUB-ROUTINE
                                  DECLARATION>
<SYSTEM REENTRANT PROCEDURE ELEMENT>::=<AUTO DATA
                                                DECLARATION>
                                      |<PROCEDURE SUB-
                                        ROUTINE DECLARATION>
                                      |<FUNCTION SUB-
                                        ROUTINE DECLARATION>
```

Semantics:

The elements in both types of System Procedures may be ordered in any manner, but Auto Data declarations must precede the applicable sub-routine declaration. Auto Data declarations are dynamically allocated at execution time, with a separate copy provided to each calling sub-routine.

Examples: (see Section 1d.)

d. Procedure and Function Sub-routine Declarations

Syntax:

```
<PROCEDURE SUB-ROUTINE DECLARATION>::=PROCEDURE <IDENTI-
        FIER><INPUT PARAMETERS><OUTPUT PARAMETERS><LABEL
        PARAMETERS>)$<STATEMENT>$
<INPUT PARAMETERS>::=( {<VARIABLE>}
<OUTPUT PARAMETERS>::=| {<VARIABLE>}
<LABEL PARAMETERS>::=|| {<LABEL NAME>}
<FUNCTION SUB-ROUTINE DECLARATION>::=FUNCTION <DATA TYPE>
        <IDENTIFIER><INPUT PARAMETERS>)$<STATEMENT>$
```

Semantics:

A Procedure may have any one, two, three, or none of the members of the set of input variables, output variables, and label names as formal parameters. A Function,

however, must have at least one input parameter. All input and output parameters must have been previously declared in data declarations, but Function names are declared when the Function is defined. Both Procedure and Function sub-routine declarations must include one statement. There is no imposed limit on the number of sub-routine declarations allowed within a System Procedure or Reentrant Procedure.

Example:

```
SYSTEM ALPHA $
    <HEADER DECLARATION>
    <SYSTEM DATA DECLARATION>
    EXTREF A $
    SYS_PROC B $
        <LOCAL DATA DECLARATION>
        PROCEDURE E (IN|OUT||LABEL)$
            <STATEMENT>$
        <AUTO DATA DECLARATION>
        FUNCTION C (IN1)$
            <STATEMENT>$
    END$
    SYS_PROC_R D$
        <AUTO DATA DECLARATION>
        PROCEDURE D |OUT1||LABEL1)$
            <STATEMENT>$
        <AUTO DATA DECLARATION>
        FUNCTION E (IN2)$
            <STATEMENT>$
    END$
END$
```

e. Statements and Blocks

Syntax:

```
<STATEMENT>::=<LABEL NAME>:<BASIC STATEMENT>
             |<BASIC STATEMENT>
<BASIC STATEMENT>::=<SIMPLE STATEMENT>
                   |<VARY STATEMENT>
                   |<CASE STATEMENT>
                   |<WHILE STATEMENT>
```

```
                   |<IF STATEMENT>
<SIMPLE STATEMENT>::=BEGIN {<STATEMENT>} END
                    |COBEGIN {<STATEMENT>} COEND
                    |<SET STATEMENT>
                    |<SWAP STATEMENT>
                    |<SEARCH STATEMENT>
                    |<INPUT STATEMENT>
                    |<OUTPUT STATEMENT>
                    |<ENCODE STATEMENT>
                    |<DECODE STATEMENT>
                    |<PACK STATEMENT>
                    |<UNPCK STATEMENT>
                    |<PROCEDURE STATEMENT>
                    |<GOTO STATEMENT>
                    |<RETURNTO STATEMENT>
                    |<RESUME STATEMENT>
                    |<RESERVE STATEMENT>
                    |<WAIT STATEMENT>
                    |<RETURN STATEMENT>
                    |<STOP STATEMENT>
```

Semantics:

A statement directs some action or controls the execution of a group of statements. The group of statements may themselves be a statement, thus allowing nesting of statement groups. These goups are delimited by the reserved words BEGIN-END or COBEGIN-COEND and may be inter-nested to any level. BEGIN-END statements imply sequential processing and COBEGIN-COEND statements imply parallel processing. COBEGIN-COEND statements, however, may also be processed sequentially. All statements within COBEGIN-COEND blocks may only change-access mutually disjoint sets of data and control structures, that is, two statements cannot access the same data or control structure.

Example:

```
BEGIN
    <STATEMENT>
        .
        .
        .
    <STATEMENT>
    CCBEGIN
        <STATEMENT>
            .
            .
            .
        <STATEMENT>
    COEND$
    BEGIN
        <STATEMENT>
            .
            .
            .
        <STATEMENT>
    END$
        .
        .
        .
END$
```

2. Basic Symbols and Delimiters

The CMS-2RS alphabet consists of letters, digits and special symbols from the U.S. ASCII Standard Character Set. The following special symbols are elements of the alphabet:

+	(plus)	)	(right parenthesis)
-	(minus)	$	(dollar sign)
/	(slash)	,	(comma)
*	(asterisk)	'	(prime)
.	(decimal point)	|	(bar)
(	(left parenthesis)		(space)
<	(left bracket)	_	(underline)
>	(right bracket)	:	(colon)

=	(equal)	"	(quote)
¬	(nct)	&	(ampersand)
#	(pound)	%	(percent)

a. Identifiers and Reserved Words

Syntax:

```
<IDENTIFIER> ::=<LETTER>
               |<IDENTIFIER><DECIMAL DIGIT>
               |<IDENTIFIER><LETTER>
               |<IDENTIFIER>_
<LETTER>::=A|B|...|Z
<DECIMAL DIGIT>::=1|2|...|9|0
```

Semantics:

Identifiers are composed of sequences cf letters and digits of any length, and must begin with a letter. Only the first eight positions are used and hence twc different identifiers must be unique in the first eight positions. The blank space must be used as a delimiter between successive identifiers and may be used between delimiters and identifiers for clarity.

Examples:

```
ALPHA
SYS_PROC
BETA1
```

b. Real Constants

Syntax:

```
<REAL CCNSTANT>::=<OCTAL CONSTANT>
                 |<DECIMAL CONSTANT>
<OCTAL CONSTANT>::=&<OCTAL INTEGER>
                  |&<OCTAL FIXED-POINT>
                  |&<OCTAL FLOATING-PCINT>
<OCTAL INTEGER>::= <OCTAL DIGIT>
                 |<OCTAL INTEGER><OCTAL DIGIT>
<OCTAL DIGIT>::=0|1|2|...|6|7
<OCTAL FIXED-POINT>::=<OCTAL INTEGER>.
                     |.<OCTAL INTEGER>
                     |<OCTAL INTEGER>.<OCTAL INTEGER>
```

```
<OCTAL FLOATING-POINT>::=<OCTAL INTEGER>E<OCTAL EXPONENT>
                        |<OCTAL INTEGER>E<SIGNED OCTAL
                            EXPONENT>
                         |<OCTAL FIXED-POINT>E<OCTAL EXPONENT>
                         |<OCTAL FIXED-POINT>E<SIGNED OCTAL>
                             EXPONENT>
<OCTAL EXPONENT>::=<OCTAL DIGIT><OCTAL DIGIT>
<SIGNED OCTAL EXPONENT>::=+<OCTAL EXPONENT>
                         |-<OCTAL EXPONENT>
<EXPONENT>::=<DECIMAL DIGIT><DECIMAL DIGIT>
<DECIMAL CONSTANT> ::=<DECIMAL INTEGER>
                     |<DECIMAL FIXED-POINT>
                     |<DECIMAL FLOATING-POINT>
<DECIMAL INTEGER>::= <DECIMAL DIGIT>
<DECIMAL FIXED-POINT>::=<DECIMAL INTEGER>.
                       |.<DECIMAL INTEGER>
                       |<DECIMAL INTEGER>.<DECIMAL INTEGER>
<DECIMAL FLOATING-POINT>::=<DECIMAL INTEGER>E<DECIMAL
                              EXPONENT>
                          |<DECIMAL INTEGER>E<SIGNED
                               DECIMAL EXPONENT>
                          |<DECIMAL FIXED-POINT>E<DECIMAL
                              EXPONENT>
                          |<DECIMAL FIXED-POINT>E<SIGNED
                              DECIMAL EXPONENT>
<DECIMAL EXPONENT>::=<DECIMAL DIGIT><DECIMAL DIGIT>
<SIGNED DECIMAL EXPONENT>::=+<DECIMAL EXPONENT>
                           |-<DECIMAL EXPONENT>
```

Semantics:

The total number of digits and characters (., +, -, and E) allowed in a real constant is twenty-two (22).

Example:

1

1.0

.35

3.

2.5E+13

3E10

.2E-12

+1

-2.4

&-7E+12

&77

c. Bits Constant

Syntax:

```
<BITS CONSTANT>::=#<HEXADECIMAL DIGIT>
                 |<BITS CONSTANT><HEXADECIMAL DIGIT>
<HEXADECIMAL DIGIT>::=<DECIMAL DIGIT>|A|B|...|F
```

Semantics:

The Bits constant is a sequence of binary digits which is represented by a sequence of hexadecimal digits. Each digit represents four binary digits of ones or zeroes. The number of hexadecimal digits allowed in a Bits constant is twenty-two (22) including the pound sign (#).

Example:

#1234ABFE

d. Status Constant

Syntax:

```
<STATUS CONSTANT>::='<IDENTIFIER>'
```

Example:

'HOT'

'COLD'

'ALERT1'

e. String Constant

Syntax:

```
<STRING CONSTANT>::="<ALPHABET SYMBOL>"
                   |"<STRING CONSTANT><ALPHABET SYMBOL>"
```

```
<ALPHABET SYMBOL>::=<LETTER>
                   |<DECIMAL DIGIT>
                   |<SPECIAL SYMBOL>
<SPECIAL SYMBOL>::=+|-|/|*|.|)|(|$|,|'|space
                  |<|>|=|¬|&|_|:||
```

Semantics:

There is no imposed limit on the number of characters allowed in a string constant. A double quote ("""), however, is not allowed.

Example:

"STRING CONSTANT"

f. Boolean Constants

Syntax:

```
<BOOLEAN CONSTANT>::=TRUE
                    |FALSE
```

Semantics:

The internal value of TRUE is one (1) and FALSE is zero (0).

g. Comments

Syntax:

```
<COMMENT>::=%<ALPHABET SYMBOL>%
          |%<COMMENT><ALPHABET SYMBOL>%
```

Semantics:

The comment statement may be used to document a program. It may be inserted anywhere within the program or its statements but there must be at least one blank inserted if it follows an identifier.

Example:

SET: THIS IS AN ASSIGNMENT STATEMENT A TO B$

3. Data Structure

a. Data Declarations

Syntax:

```
<SYSTEM DATA DECLARATION>::=SYS_DD <IDENTIFIER>$
                                   {<DATA ELEMENT>} END$
<LOCAL DATA DECLARATION>::=LOC_DD <IDENTIFIER>$
                                {<DATA ELEMENT>} END$
```

```
<AUTO DATA DECLARATION>::=AUTO_DD <IDENTIFIER>$
                                    {<DATA ELEMENT>} END$
<DATA ELEMENT>::=<VARIABLE DECLARATION>
                 |<TABLE DECLARATION>
                 |<FORMAT DECLARATION>
                 |<TAG DECLARATION>
```

Semantics:

A data declaration defines the data identifiers and their attributes which apply to the dynamic statements that follow. There is no imposed limit on the number of data elements allowed within a data declaration or on the number of data declarations allowed within a System. System data declarations apply to the entire compile-time System and Local or Auto data declarations apply only to the System Procedure in which they are defined.

b. Variable Declarations

Syntax:

```
<VARIABLE DECLARATION>::=VRBL <DATA TYPE> <ITEM>  $
<DATA TYPE>::=INTEGER(<DECIMAL INTEGER>)
              |FIXED(<DECIMAL INTEGER>,<DECIMAL INTEGER>)
              |FLOAT(<DECIMAL INTEGER>)
              |BITS (<DECIMAL INTEGER>)
              |CHAR(<DECIMAL INTEGER>)
              |STATUS({<STATUS CONSTANT>})
              |BOOLEAN
<ITEM>::=<IDENTIFIER>
        |<IDENTIFIER>=<CONSTANT>
<CONSTANT>::=<REAL CONSTANT>
             |<BITS CONSTANT>
             |<CHARACTER CONSTANT>
             |<STATUS CONSTANT>
             |<BOOLEAN CONSTANT>
             |<TAG>
<TAG>::=<IDENTIFIER>
```

Semantics:

The variable is a one-dimensional data structure used to store data values for each data type. There is no im-

posed limit on the number of Items allowed in a variable declaration. The parenthesized integers in the data type specification determine the length of the Item. For types Integer, Fixed, Float and Bits the length is in binary digits. The actual length implemented, however, may vary depending upon the arithmetic and addressing characteristics of the target machine. There is no imposed limit on the number of character symbols allowed in a character variable.

Status variables do not have length in the usual sense but have a capacity based upon the number of status constants assigned them up to a maximum of 127.

Examples:

```
VRBL INTEGER(15)A,B1,C=123765,D3$
VRBL CHAR(34)C1="THIS IS A STRING?"
VRBL STATUS('COLD','WARM','HOT')WEATHER$
VRBL BOOLEAN ONTARGET=FALSE,HOSTILE=TRUE$
```

c. Tag Declaration

Syntax:

```
<TAG DECLARATION>::=TAG {<IDENTIFIER>=<REAL CONSTANT>} $
```

Semantics:

A tag is a name for a Real constant. Once declared, tags may be used in data declarations and dynamic statements, where the appropriate constant value will be substituted. There is no imposed limit on the number of identifiers defined in a tag declaration.

Example:

```
TAG A=3.5,B=1,C=3.2E-10$
```

d. Format Declaration

Syntax:

```
<FORMAT DECLARATION>::=FORMAT <IDENTIFIER>(<CARRIAGE
       CONTROL>,{<DESCRIPTOR GROUP>})$
                          Symbol:     Definition:
<CARRIAGE CONTROL>::=space            single space and print line.
                        |0            double space and print line.
                        |-            triple space and print line.
                        |1            page eject and print line.
                        |H            Same as 1 and cancel header.
```

```
<DESCRIPTOR GROUP>::=<DESCRIPTOR>
                    |m( <DESCRIPTOR> )
               Symbol: Definition:
<DESPRIPTOR>::=Iw.d    Fixed-point binary to fixed-point
                       decimal character string.
              |Fw.d    Floating-point binary tc fixed-point
                       decimal character string.
              |Ew.d    Floating-point binary tc floating
                       point decimal character string.
              |Ow.d    Fixed-point binary to fixed-point
                       octal character string.
              |Hw.d    Binary digits to hexadecimal
                       character string.
              |"STRING"Integer coded characters to
                       character symbol strings.
              |Aw      First w symbols of an alphanumeric
                       data unit are transferred as charac-
                       ters.
              |Lw      Last w symbols cf an alphanumeric
                       data unit are transferred as charac-
                       ters.
              |wX      Skip w characters of an input record
                       or space w characters in an output
                       record.
              |Tw      A position designator of tab for
                       buffer at character positicn w.
              |{/}     End of record, or number of records
                       to be skipped.  N+1 slashes cn input
                       causes n records to be skipped.
                       N+1 slashes on output causes n blank
                       records to be produced.
               |n<DESCRIPTOR>
```

Semantics:

The format declaration specifies to the compiler the desired conversion of data elements between internal and external forms. A format identifier is referenced by INPUT, OUTPUT, ENCODE and DECODE statements tc describe data con-

version requirements. In the format descriptor, w is an unsigned integer representing the maximum number of characters of a field in the external medium. Integer w includes the space for signs, radix points and exponent descriptions, but is limited in size to the width of the output medium line. The unsigned integer d represents the number of characters that appear to the right of the radix point in the output medium, hence d must be less than w. The unsigned integer n specifies the number of repetitions of a descriptor to be applied to consecutive output fields on a line. The unsigned integer m specifes the number of repetitions of a group of descriptors to be applied to consecutive output fields on a line. A space must follow a format data type symbol such as I, E, and H.

Examples:

```
FORMAT F100(" ",3E 10,F 14.7,4(I 6,F 6.2))$
FORMAT F200("1","STATUS:",A 4,///,"ACTION:",A 9)$
```

e. Table Declaration

Syntax:

```
<TABLE DECLARATION>::=<TABLE HEAD>{<TABLE ELEMENT>}END_TABLE $
<TABLE HEAD>::=TABLE{<IDENTIFIER>}({<DIMENSION>})$
              |INDIRECT TABLE{<IDENTIFIER>}({<DIMENSION>})$
<DIMENSION>::=<CONSTANT>
<TABLE ELEMENT>::=FIELD <DATA TYPE>{<FIELD ITEM>}$
                 |ITEM_AREA <IDENTIFIER>$
                 |SUB_TABLE <IDENTIFIER>({<DIMENSION>})$
<FIELD ITEM>::=<ITEM>
              |<IDENTIFIER>=({<PRESET>})
              |<IDENTIFIER>=m({<PRESET>})
<PRESET>::=<CONSTANT>
```

Semantics:

The Table declaration creates a multi-word data structure whose basic element is the Item. All Items in a Table contain the same number of words and Fields. An Item-area is a working area with a dimension of one, and the same Field characteristics of the parent Table. A Sub-table is a subset of contiguous Items in a Table, and thus is not a separate

data structure. The unsigned integer m indicates that m consecutive Fields are to be preset to the same value.

There is no imposed limit of the number of Table declaration blocks. The maximum number of Tables allowed in a Table declaration block is twenty (20). The maximum number of dimensions allowed in a Table is 128. The greatest value a single dimension can have is 65,536. The maximum number of Fields allowed in a Table Item is forty (40). There is no imposed limit on the number of Item-areas and Sub-tables that can be declared within a Table block.

To reference an Item within a Table, the Table name followed by the subscript list must be specified. To reference a Field within a Table Item, the Field name followed by the Table name and subscript list must be specified. To reference a Field or Item in a Sub-table is the same as for Tables. The dimensions must be within the defined subtable boundaries or an error will occur. To reference an Item-area, the Item-area name in parentheses must be specified. To reference a Field within an Item-area the Field name followed by the Item-area name in parentheses must be specified.

Example:

```
TABLE T1,T2,T3(10,20,5)$
    FIELD INTEGER(16)TF1=1000(-1)$
    FIELD BOOLEAN TF2$
    FIELD STATUS('DOWN','READY','AIRBORNE')AIRCRAFT$
    SUBTABLE ST1(500,1,1,1)$
    ITEM-AREA ITM1$
END_TABLE$
```

4. Control Structures

a. VARY Statement

Syntax:

```
<VARY STATEMENT>::=VARY <REAL VARIABLE> FROM <REAL PRIMARY>
                      THRU <REAL PRIMARY> DO <STATEMENT>$
                  |VARY <REAL VARIABLE> FROM <REAL PRIMARY>
                    STEP <REAL PRIMARY> THRU <REAL PRIMARY>
                      DO <STATEMENT>$
```

Semantics:

The VARY statement allows multiple loop indicies on the same level and multiple nesting of loops, with the upper limit on the number of indicies a function of target machine characteristics. Indicies must be declared in data declarations and retain their last value when a VARY loop is exited. The initial, step and test values are evaluated only once when the loop is entered but the loop index variable may be changed during execution.

The statement or statement block within a VARY loop must be executed once and the index may either be incremented or decremented until it reaches the test value. When all indicies on the same level have reached their test values the loop will be exited. A partial pass may be made through a loop by a RESUME statement. This is accomplished by a transfer of control to the increment and test step at the end of the loop.

The initial, step, and test value data elements must appear in the order listed, but the step value may be omitted in which case an implied increment of one is assumed. The Real data type for index variables and loop control values is always integer, therefore, fixed-point or floating-point values are not allowed.

Example:

```
VARY A FROM B STEP -1 THRU C DO
    BEGIN
    SET:D TO E+1 $
    VARY D FROM F THRU H,E FROM L THRU 20 DO
        BEGIN
        SET:G TO D-10 $
        SET:M TO E+9 $
        END$
    END$
```

b. CASE Statement

Syntax:

```
<CASE STATEMENT>::=CASE:<REAL DATA UNIT> OF <STATEMENT>$
                         ELSE <STATEMENT>$
                    |CASE:<REAL DATA UNIT> OF { {<REAL CONSTANT>:}
                         <STATEMENT>$} ELSE <STATEMENT>$
                    |CASE:<STATUS DATA UNIT> OF {{<STATUS
                         CONSTANT>:}<STATEMENT>$} ELSE
                         <STATEMENT>$
```

Semantics:

The CASE statement allows selection of a statement within a list of statements for processing depending upon the value of a data unit. This can be accomplished in twc ways. The index case allows declaration of a list of n statements with an implied numbering cf 0 to n-1. If the value of the data unit is from 0 to n-1 then the appropriate statement is executed, or the ELSE sase statement is executed. The label case method requires that each label be checked for a match with the data unit. If a match is found then the appropriate statement is executed, or the ELSE case statement is executed.

Example:

(1). Index Case

```
CASE:D OF
CALL PROC1$
CALL PROC2$
ELSE CALL PROC3$
```

(2). Label Case

```
CASE:A OF
2:3: SET: B TO C$
5:7:IF A=B THEN GOTO D$
ELSE SET CASE1 TO FALSE$
```

c. While Statement

Syntax:

```
<WHILE STATEMENT>::=WHILE <BOOLEAN EXPRESSION> DO <STATEMENT>$
```

Semantics:

The WHILE statement allows a statement cr statement

block to be executed in a loop until the value of the Boolean expression is FALSE. The Boolean expression is evaluated each time the loop is executed and if it is initially FALSE the loop will not be executed. The Boolean expression must be made FALSE during loop execution or the loop will execute infinitely.

Example:

```
WHILE A<5 DO
    BEGIN
        SET:B TO C+1$
        SET:A TO A+1$
    END$
```

d. IF Statement

Syntax:

```
<IF STATEMENT>::=IF <BOOLEAN EXPRESSION> THEN <STATEMENT>$
                |IF <BOOLEAN EXPRESSION> THEN <SIMPLE
                      STATEMENT>$ ELSE <STATEMENT>$
                |{IF <BOOLEAN EXPRESSION> THEN} <STATEMENT>$
                |{IF <BOOLEAN EXPRESSION> THEN} <SIMPLE
                      STATEMENT>$ {ELSE <SIMPLE STATEMENT>}
                      ELSE <STATEMENT>$
```

Semantics:

The IF statement allows nesting of IF THEN ELSE statements with no imposed limit. Only simple (not CASE, VARY, WHILE, RESERVE, or IF) statements are allowed within the nesting structure. Basic and simple statements, however, may be used for the last statement after the last ELSE. The ELSE statement is always matched with the nearest IF THEN phrase, thus eliminating the dangling ELSE problem.

Examples:

```
IF A=B THEN GOTO LABEL1$
IF A=B THEN IF C=D THEN SET:A TO D$
IF A=B THEN IF C=D THEN SET:A TO D ELSE SET:B TO C$
```

e. RESERVE and WAIT Statements

Syntax:

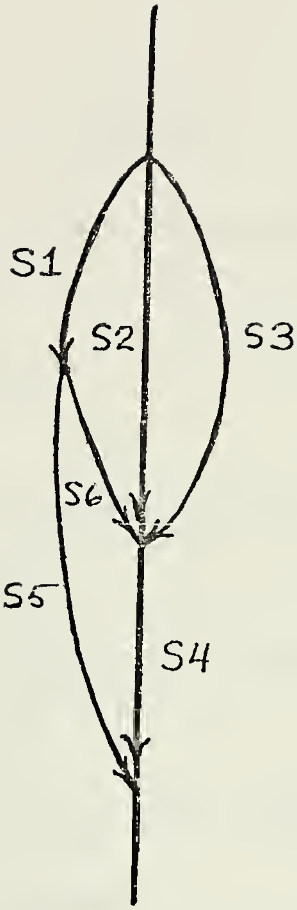
```
<RESERVE STATEMENT>::=RESERVE({<RESOURCE>})<SIMPLE
                           STATEMENT>$
```

```
<RESOURCE>::=<PROCEDURE STATEMENT>
           |:<DATA UNIT>
           |<DATA STRUCTURE>
<DATA STRUCTURE>::=<TABLE IDENTIFIER>({<REAL PRIMARY>})
                  |<FILE NAME>(<REAL PRIMARY>)
<TABLE IDENTIFIER>::=<TABLE NAME>
                    |<SUB-TABLE NAME>
                    |<ITEM-AREA NAME>
<WAIT STATEMENT>::=WAIT({<PROCEDURE NAME>})$
```

Semantics:

The RESERVE statement may be used to inform the monitor system of an impending entry into a critical section. The procedure statements, data units or data structures thus identified will not then be change-accessable by any other processes running in the same multi-programming or multi-processing environment. Upon leaving the RESERVE block, the resources will be freed for change-access by other processes. The WAIT statement has been added to allow synchronization of processes, that is, to specify that the controlling procedure is to suspend execution until one or more procedures have completed their execution.

```
Example:
COBEGIN
      BEGIN

      RESERVE(R1)
S1          BEGIN

            END$
      END$
      BEGIN

      RESERVE(R1)
            BEGIN
S2
            END$

      END$
      BEGIN
S3
      END$
      COBEGIN
         BEGIN
S4          WAIT(S2,S3,S6)$

         END$
         BEGIN
S5          WAIT(S1)$

         END$
         BEGIN
S6          WAIT(S1)$

         END$
      CCEND$
COEND$
```



f. SET Statement

Syntax:

```
<SET STATEMENT>::=SET: <DATA UNIT>  TO <EXPRESSION>$
<DATA UNIT>::=<REAL DATA UNIT>
             |<BITS DATA UNIT>
             |<BOOLEAN DATA UNIT>
             |<CHARACTER DATA UNIT>
             |<STATUS DATA UNIT>
             |<TABLE IDENTIFIER>
             |<TABLE IDENTIFIER>( <REAL PRIMARY> )
<REAL DATA UNIT>::=<REAL VARIABLE>
                  |<REAL FIELD NAME> <TABLE IDENTIFIER>
                       ( <REAL PRIMARY> )
<REAL VARIABLE>::=<INTEGER VARIABLE>
                 |<FIXED-POINT VARIABLE>
                 |<FLOATING-POINT VARIABLE>
<BITS DATA UNIT>::=<BITS VARIABLE>
                  |<BITS FIELD NAME> <TABLE IDENTIFIER>
                       ( <REAL PRIMARY> )
<BOOLEAN DATA UNIT>::=<BOOLEAN VARIABLE>
                     |<BOOLEAN FIELD NAME> <TABLE IDENTIFIER>
                          ( <REAL PRIMARY> )
<STATUS DATA UNIT>::=<STATUS VARIABLE>
                    |<STATUS FIELD NAME> <TABLE IDENTIFIER>
                         ( <REAL PRIMARY> )
<CHARACTER DATA UNIT>::=<CHARACTER VARIABLE>
                       |<CHARACTER FIELD NAME> <TABLE
                            IDENTIFIER>( <REAL PRIMARY> )
<EXPRESSION>::=<REAL EXPRESSION>
              |<BITS EXPRESSION>
              |<CHARACTER EXPRESSION>
              |<STATUS EXPRESSION>
              |<BOOLEAN EXPRESSION>
              |<TABLE IDENTIFIER>
```

Semantics:

The SET statement allows the assignment of evaluated expressions to a compatible data unit or Table structure.

All assignments must be type compatible and receptacle data unit lengths will determine whether the results are truncated at the least significant digits or right justified and filled in with zeroes. For Character assignments, the character symbols on the right of an expression will be truncated if the receptacle is smaller than the expression, otherwise, the result will be left justified and the remaining positions unchanged.

The Status assignment will transfer to the Status variable the integer value of the Status constant on the right.

Bits expression assignments follow these rules:

(1). If the length of the expression is greater than the data unit, then the result is right-justified in the data unit and truncated at the excess bits on the left.

(2). If the length of the expression is less than the data unit, then the result is right justified in the data unit and the unused bits are set to zero.

A multi-word Table-to-Table, Item-to-Item or single Item-to-Table assignment results in the transfer of all values from one element to another. Table-to-Table assignment implies that every word of one Table will be transferred to every corresponding word of another Table. Dimension compatibility is the responsibility of the programmer. The total number of words in each, however, must be the same. Item-to-Item assignments will also result in the transfer of all words in one Item to another Item. Assignment length compatibility is the responsibility of the programmer and excess words will be truncated. Item-to-Table assignment implies that every Table Item is replaced by the Item value.

Examples:

```
SET:A TO B$
SET:A,B,C TO D$
SET:WEATHER TO 'HOT'$
SET WEATHER,TEMPERATURE TO 'HOT'$
SET:A TO (B+C)*D$
```

```
        SET:(A)..5 TO C$
        SET:A TO (C)..2$
        SET:A TO C REM D$
        SET:A TO B ANDL C$
        SET:A TO FALSE$
        SET:C TO "CHARACTER STRING 11"$
        SET:A(1,1) to B(1,2)$
        SET:TABLE1 TO TABLE2$
        SET:TABLE2 TO ITEM1$
```

g. SWAP Statement

Syntax:

```
<SWAP STATEMENT>::=SWAP:<DATA UNIT> FOR:<DATA UNIT>$
```

Semantics:

The SWAP statement exchanges the values of two data units. Replacement rules are the same as those for the SET statement.

Example:

```
SWAP:C(1,2) FOR:D(2,1)$
```

h. SEARCH Statement

Syntax:

```
<SEARCH STATEMENT>::=SEARCH:<DATA STRUCTURE> FOR:<DATA UNIT>$
```

Semantics:

The SEARCH statement provides the capability to search a Table or File for an entry with same value as a data unit. If found, the index variables in the data structure's dimension list will be set at run-time to point to the value's location, otherwise they will be set to minus one. Both structures will be searched sequentially starting with the first dimension and varying it and adjacent dimensions upward from right to left to their maximum limit until the value is found or not found.

Example:

```
SEARCH:TABLE1(D1,D2) FOR:B(1,1,1)$
```

i. INPUT Statement

Syntax:

```
<INPUT STATEMENT>::=INPUT <FILE NAME> INTO:<DATA UNIT>$
                   |INPUT <FILE NAME> <FORMAT NAME>
                        INTO:<DATA UNIT>$
<FILE NAME>::=READ
             |OCM
             |<USER FILE NAME>
```

Semantics:

The INPUT statement causes a monitor routine to input data elements from a specified device or user declared file into a data unit or units. When the data unit is a whole Table the input fills the Table sequentially, word by word. If a format name is not referenced, the input character string is converted to the type and length of the receptacle. The user file name feature was not implemented in this version of CMS-2RS, but will allow data to be moved to and from internal and external devices.

Examples:

```
INPUT READ INTO:A,B,C$
INPUT USER1 FORM1 INTO:TABLE1$
```

j. OUTPUT Statement

Syntax:

```
<OUTPUT STATEMENT>::=OUTPUT <FILE NAME> <FORMAT NAME>$
                    |OUTPUT <FILE NAME> OUTOF:<DATA UNIT>$
                    |OUTPUT <FILE NAME> <FORMAT NAME> OUTOF:
                         <DATA UNIT>$
<FILE NAME>::=PRINT
             |PNCH
             |OCM
             |<USER FILE NAME>
```

Semantics:

The OUTPUT statement causes a monitor routine to output data or a character string to a specified device or user declared file from a data unit or units. When the data unit is a whole Table the OUTPUT statement empties the Table Items sequentially,word by word. If a format name is not referenced, the output character string is truncated at 22 positions per data unit.

Examples:

```
OUTPUT PRINT "THIS IS AN OUTPUT STRING"$
OUTPUT OCM OUTOF:A,B,C$
OUTPUT USER2 FORM2 OUTOF:TABLE4$
```

k. ENCODE and DECODE Statements

Syntax:

```
<ENCODE STATEMENT>::=ENCODE:<CHARACTER DATA UNIT> <FORMAT
                          NAME> OUTOF:<DATA UNIT>$
<DECODE STATEMENT>::=DECODE:<CHARACTER DATA UNIT> <FORMAT
                          NAME> INTO:<DATA UNIT>$
```

Semantics:

The ENCODE statement specifies that the source data unit(s) are to be converted to character strings and packed sequentially into the receptacle according to a specified format. The DECODE statement is the reverse of the ENCODE statement.

Examples:

```
ENCODE:CHARAC1 OUTOF:A,B,C$
DECODE:CHARAC1 INTO:A,B,C$
```

l. PACK and UNPCK Statements

Syntax:

```
<PACK STATEMENT>::=PACK:<DATA UNIT> OUTOF:<DATA UNIT>$
<UNPCK STATEMENT>::=UNPCK:<DATA UNIT> INTO:<DATA UNIT>$
```

Semantics:

The PACK statement transfers a bit string from a source list of data units into a receptacle data unit. They are stored consecutively from left-to-right and without spacing. The receptacle and source data units may be variable to Field, Field to variable, variable to variable, Field to Field, Table to Table, Table to variable, variable to Table, Field to Table, Table to Field, Item to Item, Table to Item, Item to Table, Item to Field and Field to Item. Assignment length compatibility is the responsibility of the programmer and excess words will be truncated.

Examples:

```
PACK:TABLE1  OUTOF:TABLE2$
PACK:A OUTOF:B,C,D$
```

m. Procedure Statement

Syntax:

```
<PROCEDURE STATEMENT>::=CALL <PROCEDURE NAME>
                            <INPUT PARAMETERS>
                            <OUTPUT PARAMETERS>
                            <LABEL PARAMETERS>)$
                       |CALL <PROCEDURE NAME>
<INPUT PARAMETERS>::=( ,
                    | <EXPRESSION> )
<OUTPUT PARAMETERS>::=|| ,
                     || <DATA UNIT>
<LABEL PARAMETERS>::=||<LABEL NAME>
```

Semantics:

The procedure CALL statement transfers control to a named procedure and maps any actual parameters to the corresponding formal parameters.

All procedure input and output parameters must have been previously declared. Parameters are normally mapped by passing values, thus resulting in a call by value. If the same data element is used for output in both actual and formal parameters, then the effect is call by result. If an actual parameter is omitted, or if the same data element is specified as both the actual and formal parameter then a call by address occurs. If call by address is desired and the data elements are not the same, then the CORAD operator must be used to pass the address of variables or subscripted variables. For Tables, the indirect declaration allows the Table address to be passed to the called procedure. By mixing any of the above conventions, calls by value-result or address-result may occur.

Examples:

(1). Value

```
TABLE T1,T2$
CALL A(T2| )$
```

```
        PROCEDURE A(T1| )$
```

(2). RESULT

```
        TABLE T1,T2$
        CALL A( |T2)$
        PROCEDURE A ( |T1)$
```

(3). Value-result

```
        VRBL A,B,C$
        CALL D(A|A)$
        PROCEDURE D(B|B)$
```

(4). Address (Tables)

```
        TABLE T2,T3$
        INDIRECT TABLE T1$
        CALL A (T2|T3)$
        PROCEDURE A(T2|T3)$

        CALL A(T2| )$
        CALL A(T3| )$
        PROCEDURE A(T1| )$
```

(5). Address-result

```
        TABLE T1,T2$
        CALL A(T1|T2)$
        PROCEDURE A(T1|T1)$
```

(6). Address (variables)

```
        VRBL A,B,C$
        CALL D(CORAD(A)| )$
        CALL D(CORAD(B)| )$
        PROCEDURE D(C| )$
```

n. GOTO Statement

Syntax:

```
<GOTO STATEMENT>::=GOTO <LABEL NAME>$
```

Semantics:

The GOTO statement transfers control to a labeled statement or statement block.

Examples:

```
GOTO LABEL1$
IF A=B THEN GOTO LABEL2$
```

o. RETURNTO Statement

Syntax:

```
<RETURNTO STATEMENT>::=RETURNTO <LABEL NAME>$
```

Semantics:

The RETURNTO statement transfers control back to one of the abnormal exit label parameters in a procedure call.

Example:

```
RETURNTO EXITLBL1$
```

o. RESUME Statement

Syntax:

```
<RESUME STATEMENT>::=RESUME <LABEL NAME>$
```

Semantics:

The RESUME statement specifies a transfer to the increment and test step within a VARY block.

Example:

```
IF A=B THEN RESUME VARY1$
```

p. RETURN Statement

Syntax:

```
<RETURN STATEMENT>::=RETURN
                    |RETURN:<DATA UNIT>
```

Semantics:

The RETURN statement transfers control from within a sub-routine to the point of call. The RETURN statement may be omitted if it is the last statement in a procedure. To return from a Function, the RETURN data unit statement must be used.

Example:

```
RETURN$
IF A=B THEN RETURN$
IF A<B THEN RETURN:A$
```

r. STOP Statement

Syntax:

```
<STOP STATEMENT>::=STOP
```

Semantics:

The STOP statement temporarily suspends program execution until an operator manually restarts the computer.

Example:

```
IF INTERRUPT(1) THEN STOP$
```

5. Expressions

a. Real Expressions

Syntax:

```
<REAL EXPRESSION>::=<REAL PRIMARY>
                    |<REAL EXPRESSION> <REAL OPERATOR>
                         <REAL PRIMARY>
                   Symbol:  Definition:    Priority:
<REAL OPERATOR>::=+         addition           4
                 |-         subtraction        4
                 |*         multiplication     3
                 |/         division           3
                 |REM       remainder          3
                 |**        exponentiation     2
                 |ABS       absolute value     1
                 |-         unary minus        1
<REAL PRIMARY>::=<REAL DATA UNIT>
                |<REAL CONSTANT>
                |(<REAL EXPRESSION>)
                |<REAL FUNCTION NAME> <INPUT PARAMETERS>)
                |(<REAL EXPRESSION>)..<REAL CONSTANT>
                |NUMBER(<BITS PRIMARY>)
                |CHARCODE(<CHARACTER PRIMARY>)
                |COUNT(<DATA UNIT>)
                |CORAD(<DATA UNIT>)
```

Semantics:

The Real expression allows mixed mode arithmetic with integer, fixed-point and floating-point operands. Evaluation is from left-to-right in the order of the priorities listed above. The REM operator provides the remainder after integer, fixed-point or floating-point division of two operands.

Examples:

```
SET:A TO B REM NUMBER(C)$
SET:D TO -B**C**E*CORAD(F)$
```

b. Bits Expressions

Syntax:

```
<BITS EXPRESSION>::=<BITS PRIMARY>
                  |<BITS EXPRESSION> <LOGICAL OPERATOR>
                       <BITS PRIMARY>
                  |<BITS EXPRESSION> <SHIFT OPERATOR>
                       <REAL PRIMARY>
                    Symbol:  Definition:        Priority:
<LOGICAL OPERATOR>::=NOTL    logical not           1
                    |ANDL    logical and           3
                    |ORL     logical or            3
<SHIFT OPERATOR>::=SHLL      shift left logical    2
                 |SHRL       shift right logical   2
                 |CIRSHLL    circular SHLL         2
                 |CIRSHRL    circular SHRL         2
<BITS PRIMARY>::=<BITS CONSTANT>
               |<BITS DATA UNIT>
               |(<BITS EXPRESSION>)
               |<BITS FUNCTION NAME> <INPUT PARAMETERS>)
               |BITSTRING<INPUT PARAMETERS>)
```

Semantics:

The Bits expressions allow logical operations on binary operands of equal or unequal length. The shift operators allow shifting of a Bits operand an integral amount either end-off or within the operand. All end-off shifts result in vacated bit positions being assigned the value of zero (0). The BITSTRING function has parameters: (real primary, real primary,data unit). The first parameter is the starting bit position in the third parameter. The second parameter is the number of bits to be extracted from the third parameter. The result is a substring of bits from the data unit.

Examples:

```
SET:A TO B ANDL C SHIL 3$
SET:D TO BITSTRING(4,10,E)$
```

c. Character Expressions

Syntax:

```
<CHARACTER EXPRESSION>::=<CHARACTER PRIMARY>
                          |<CHARACTER EXPRESSION> CAT
                               <CHARACTER PRIMARY>
<CHARACTER PRIMARY>::=<CHARACTER CONSTANT>
                      |<CHARACTER DATA UNIT>
                      |(<CHARACTER EXPRESSION>)
                      |<CHARACTER FUNCTION NAME><INPUT
                          PARAMETERS>)
                      |CODECHAR(<REAL PRIMARY>)
                      |SUBCHAR<INPUT PARAMETERS>)
```

Semantics:

The Character expression allows concatenation of successive character strings. The SUBCHAR function has paramters: (real primary, real primary, character primary). The first parameter is the starting character position in the third parameter. The second parameter is the number of characters to be extracted from the third patameter. The result is a substring of character symbols from the character primary.

e. Boolean Expression

Syntax:

```
<BOOLEAN EXPRESSION>::=<BOOLEAN PRIMARY>
                        |<BOOLEAN EXPRESSION> <BOOLEAN
                             OPERATOR> <BOOLEAN PRIMARY>
                      Symbol:    Definition:      Priority:
<BOOLEAN OPERATOR>::=NOT         complementation     2
                    |AND         conjunction         3
                    |OR          disjunction         3
<BOOLEAN PRIMARY>::=<BOOLEAN CONSTANT>
                   |<BOOLEAN DATA UNIT>
                   |(<BOOLEAN EXPRESSION>)
                   |<BOOLEAN FUNCTION NAME><INPUT
                       PARAMETERS>)
                   |<RELATIONAL EXPRESSION>
```

```
<RELATIONAL EXPRESSION>::=<REAL PRIMARY> <RELATIONAL OPERATOR>
                              <REAL PRIMARY>
                          |<BITS PRIMARY> <RELATIONAL
                              OPERATOR> <BITS PRIMARY>
                          |<CHARACTER PRIMARY> <RELATIONAL
                              OPERATOR> <CHARACTER PRIAMRY>
                          |<STATUS EXPRESSION> <RELATIONAL
                              OPERATOR> <STATUS EXPRESSION>
                         Symbol:  Definition:         Priority:
<RELATIONAL OPERATOR>::=<         less than                1
                        |>        greater than             1
                        |=        equal                    1
                        |¬=       not equal                1
                        |<=       less than or equal       1
                        |>=       greater than or equal    1
```

Semantics:

The Boolean expressions allow logical comparison of two or more operands for TRUE of FALSE conditions using Boolean operators. The relational expressions allow comparison of Real, Bits, Character or Status expressions.

Example:

```
IF A=B AND C=D THEN SET:A TO D$
```

d. Status Expressions

Syntax:

```
<STATUS EXPRESSION>::=<STATUS CONSTANT>
                      |<STATUS DATA UNIT>
                      |<STATUS FUNCTION NAME><INPUT PARAMETERS>)
```

Semantics:

The Status expression allows the use of Status constants, data units and functions in dynamic statements.

Example:

```
SET:WEATHER TO 'HOT'$
```

APPENDIX C

SAMPLE CMS-2RS PROGRAMS

```
SYSTEM ALPHA $ % THIS IS A DO-NOTHING PROGRAM TO SHOW SAMPLES OF DATA
                 DECLARATIONS AND DYNAMIC STATEMENTS  %
  SYS_DD D1 $
    VRBL STATUS('HERE','THERE','ANYWHERE') POS='HERE'$
    TAG F=5, G=21.3 $
    VRBL INTEGER(16) A=1, B, C=4 $
    VRBL FIXED(16,8) FX=44.8 $
    VRBL CHAR(4) CH="HELP" $
    VRBL BITS(32) BT=#FF $
    FORMAT F100(" ",3F 4.2, 3(4E 6)) $
    FORMAT E1("1", 3I 4,//,"YEA") $
    TABLE X,Y(2,3,6) $
      FIELD INTEGER(32) F1=12(1,2) $
      FIELD STATUS('ALERT','AIRBCRN',
      FIELD STATUS('ALERT','AIRBORN','COWN') AIRCRAFT $
      FIELD FIXED(6,2) F2=(1.2, 2.3, 3.4) $
      ITEM_AREA IA1, IA2 $
      SUBTABLE S1(4, 2, 2, 2)$
    END_TABLE $
  END_DD $


  SYS_DD S1 $
    VRBL INTEGER(32) A, C=1, D=40 $
    VRBL FLOAT(32) IN1, IN2, IN3, IN4$
    VRBL FIXED(32,16) OUT1, CUT2 $
    VRBL CHAR(10) CHAR1$
    FORMAT F200(" ", I 4)$
  END_DD $

  EXTREF S2 $

  SYS_PROC B $
    LCC_DD $
      VRBL INTEGER(16) A1, A2 $
    END_DD $

    PROCEDURE B(IN1,IN2|OUT1,CLT2||L1,L2) $
    VARY A FROM C STEP -1 THRU D DO
    BEGIN
```

```
      SET : OUT1 TO A $
      VARY A1 FROM 1 THRU 20, A2 FROM 2 THRU 10 DO
      BEGIN
        SET : OUT2 TO A1 $
        IF A1=A2 THEN RETURNTO L1 $
      END $
    END $ % END OF PROCEDURE "B" %


   PROCEDURE P2 $
   BEGIN
    IF A=C THEN IF IN1=IN2 THEN WHILE A<C DO RESERVE(OCM(20),:A,B)$
    IF A=C THEN IF IN1=IN2 THEN UNPCK:IN2 INTC: IN1,A$ELSE RETURN: C$
    ELSE CASE: C OF
         3: 4: ENCODE: CHAR1 F100 OUTOF: A,C$
         5:6:  DECODE: CHAR1 F100 INTC: A,C$
         ELSE OUTPUT PRINT F100 OUTOF: A,C$
  L1: INPUT READ F100 INTO: A,C$
    CALL B(A,C|OUT1, ||L1)$
   END $ % END OF PROCEDURE P2 %


   FUNCTION FIXED(32,16) FACT(IN3,IN4)$
   BEGIN
    SWAP : IN3 FOR : IN4 $
    RETURN : IN4 $
   END $ % END OF FACT %

  END $ % END OF SYS_PROC B %

END $ % END OF SYSTEM ALPHA %



SYSTEM BISEARCH $ % A BINARY SEARCH IS PERFORMED FOR AN IDENTIFIER
                    IN A TABLE VIA AN ALPHABETICALLY ORDERED DIRECTORY
                    CONTAINING FOR EACH ENTRY THE LENGTH (NO. OF
                    CHARACTERS) OF THE IDENTIFIER, THE ADDRESS OF THE
                    ACTUAL IDENTIFIER, AND A CODE NUMBER  %

  EXTREF IDTABLE $ % IDENTIFIER TABLE IS  A SYS_DD IN THE LIBRARY %

  SYS_DD D1   $
    TABLE DIRECTRY(1000) $
      FIELD INTEGER(8) LNGTH $
      FIELD INTEGER(16) ADDRS $
```

```
      FIELD INTEGER(16) COCE $
    END_TABLE $
    VRBL CHAR(20) IDENTBUF $ %IDENTIFIER BUFFER ,
    VRBL CHAR(20) IDENTBUF $ % IDENTIFIER BUFFER %
    VRBL INTEGER(8) IDLNGTH $
    VRBL INTEGER(16) INCEX, LCW, HIGH $
    VRBL BOOLEAN FOUND $
  ENC_DD $


  SYS_PROC MAIN $
    LCC_DD $
      FORMAT F1("1", X 10, I 5) $
      FORMAT F2(" ","IDENTIFIER NOT FCLND") $
    END_DD $

    PROCEDURE MAIN $
    BEGIN
      CALL READID $
      CALL SRCH(IDLNGTH,LOW,HIGH|FOUND,INDEX) $
      IF FOUNC THEN OUTPUT PRINT F1(COCE DIRECTRY(INDEX)) $
      ELSE CUTPUT PRINT F1 $
    END $
  END $ % END CF MAIN %


  SYS_PROC READID $
    LOC_DD $
      VRBL INTEGER(8) PTR = C $
      VRBL CHAR(1) CHTR = " " $
    END_DD $

    PROCEDURE REACID $
    BEGIN
      SET: IDENTBUF TO "                    " $
      INPUT READ ICENTBUF $
      SET: IDLNGTH TO 0 $
      SET: CHTR TO SUBCHAR(PTR,1,IDENTBUF) $
      WHILE CHTR¬=" " CO
      BEGIN
        SET: PTR TO PTR + 1 $
        SET: ICLNGTH TC IDLNGTH + 1 $
        SET: CHTR TO SUBCHAR(PTR,1,ICENTBUF) $
      END $
    END $
  END $ % END CF READIC %
```

```
  SYS_PROC SRCH $
    LOC_DD $
      VRBL INTEGER(16) L,LC,HI,I $
      VRBL BCCLEAN F $
      VRBL CHAR(20) IDB $
      VRBL INTEGER(8) R $
    END_DD $

    PROCEDURE SRCH(L,LO,HI|F,I) $
    BEGIN
  L1: SET: I TC (LO+HI)/2 $
      IF L=LNGTH DIRECTRY(I) THEN
      BEGIN
        SET: IDB TO IDENTRY IDTABLE(ACCRS DIRECTRY(I)) $
        CALL CCMPARE(IDENTBUF,IDB|R) $
      END $
      IF R=0 THEN SET: F TO TRUE $
      ELSE IF R=1 THEN SET: HI TO I $
      ELSE SET: LC TO I $
      IF NOT F THEN GOTO L1 $
    END $ % END CF SRCH %

    PROCEDURE COMPARE(IDENTBUF, IDB|R) *
    BEGIN
      IF IDENTBUF=IDB THEN SET: R TC 0 $
      ELSE IF IDENTBUF < IDB THEN SET: R TO 1 $
      ELSE SET: R TO 2 $
    END $ % END CF COMPARE %

  END $ % END CF SYS_PROC SRCH %

END $ % END OF BISEARCH %
```

APPENDIX D

DESCRIPTORS' PROTOCOL

Due to the variety of conditions that need be described, it becomes necessary to recognize several descriptor formats. These formats are detailed here to facilitate future work in completing this compiler. Unless otherwise specified, the header byte in every descriptor gives the number of bytes that follow. Fields are described sequentially as they appear in the descriptor.

1. Parameterless Procedure

A descriptor for a parameterless procedure consist of the header byte followed by a four-byte field to store the entry point address of the procedure.

2. Procedure With Parameters

A descriptor for a procedure with parameters consist of the header byte, a four-byte field to store the entry point address of the procedure, one byte to indicate the type and number of parameters, and a number of two-byte fields, one for each parameter, to store pointers to the parameters' entries in the identifier directory.

The two high order bits of the type-number byte indicate the type of parameters that follow according to the following code: 00 for input parameters, 01 for output parameters, and 10 for label parameters. The six low order bits remaining indicate number of parameters (64 maximum).

The type-number byte together with the pointer fields form a group. Up to three such groups can appear in the descriptor, one for each of the three types of parameters, INPUT, OUTPUT, and LABEL.

3. Function Name

A function name descriptor consist of the header byte and a four-byte field to store the entry point address of the procedure. The remainder of the descriptor varies in format among the five possible types of functions as follows:

a. Numeric Function

The entry point field is followed by a two-byte field for data type description. The two high order bits of the first byte indicate the type as follows:

00 for integer
01 for fixed
10 for float.

The next 14 bits of the field are used to store value of the data size parameter in data type declarations. In the case of type FIXED where two such parameters are used, the seven high order bits are used for the first parameter and the seven low order bits for the second.

The data type field is followed by a one-byte field to indicate the number of formal parameters, and a number of two-byte fields, one per formal parameter, to store pointers to the parameters' entries in the identifier directory,

b. Bits Function

Bits function descriptors are the same as numeric function descriptors except that all 16 bits of the data type field are used to store the value of the data size parameter.

c. Character Function

Character function descriptors are the same as bits function descriptors.

d. Status Function

The entry point field is followed by a one-byte field to indicate the present value of the status function, i.e., the number of the status constant assigned. This

field is followed by a number of two-byte fields, one per status constant, to store pointers to status constants in the identifier directory. A one-byte field to indicate the number of formal parameters and a number of two-byte fields for pointers to such parameters follow. At declaration time, the compiler associates an integer with each status constant in ascending order beginning with 1. This number is stored in the descriptor pointer field of the status constant's entry in the identifier directory. When a status variable is set to a status constant, the number assigned to that constant by the compiler is entered in the present value byte of the descriptor.

e. Boolean Function

Boolean function name descriptors need no data type field. Other than that, the descriptors are the same as those described above.

4. Tags

Tags do not need descriptors. Instead, the descriptor pointer field in the identifier directory has the pointer to the numeric value of the tag in the constant table.

5. Variables

Numeric, bits, and character variables have the same descriptor format. It consists of the header byte, two bytes for the data type field as described before, and a two-byte field for a pointer to the value in the constant table. This last field will have an unpredictable value unless the variable has been initiallized.

Status variables have the following descriptor format: header byte, one byte to indicate present value of the status variable, i.e., the number of the status constant assigned, and a number of two-byte fields, one per each status constant, to store pointers to status constants in the identifier directory (see status function above).

Boolean variables, like boolean functions, do not need descriptors. The descriptor pointer field is used to store the value of the variable; 1 for TRUE and 0 for FALSE.

6. Format Declaration Descriptor

Format declaration descriptors consist of the header byte, one byte for the carriage control field, and a sequence of format descriptor fields. There are five different types of format descriptor fields. The high order three bits of each field are used as a key to indicate the type of field as follows:

001 for numeric conversion format field
010 for character constant field
011 for slash list field
100 for repetition group head field
101 for repetition group tail field.

a. Numeric Conversion Format Field

This field is four bytes long. The first byte has the field key and the type conversion letter. This letter is represented by the low order five bits in the byte according to the following code:

01001 for I
00110 for F
00101 for E
10110 for O
01000 for H
00001 for A
10011 for L
00111 for X
00011 for T

The second byte has the value of the repetition factor "n" in the numeric conversion group (nIw.d). It can have any value between 1 and 255. The third byte has the value of the "w" parameter, and the fourth byte that of the "d" parameter. The "d" parameter can be zero.

b. Character Constant Field

The character constant field has three bytes, The first byte has the key field. The second and third bytes have a pointer to the character constant in the constant table.

c. Slash List Field

The slash list field has one byte. The three high order bits have the field key. The five low order bits give the number of slashes in the list.

d. Repetition Group Head Field

The repetition group head field has two bytes. The three high order bits have the field key. The remainder thirteen bits have the value of the group repetition factor m.

e. Repetition Group Tail Field

The repetition group tail field has one byte with the field key. All the descriptor fields bracketed by the repetition group head and tail fields will be repeated "m" times when implementing the format.

7. Tables

A table descriptor consist of the header byte, one byte for the number of dimensions field, a number of two-byte fields, one for each dimension, to store the value of the dimensions, and a two-byte field to store the pointer to the field pointer list in the constant table. The high order first bit of the number of dimensions field is used to indicate if the table is INDIRECT or not. This bit is set to "1" for INDIRECT TABLE declarations; it is reset to "0" otherwise. The remaining seven bits are used to store the number of dimensions (128 maximum). The field pointer list is a pseudo-descriptor consisting of a header byte and a number of two-byte fields, one per each field declared in the table declaration block, with pointers to the field name entries in the identifier directory.

8. Fields

Numeric, bits, and character field names have the same format. It consists of the header byte, twc bytes for the data type field, and a two-byte field for the pointer to the value in the constant table. If the field is initiallized at declaration time to a list of values, the follcwing additional fields are part of the descriptor: a two-byte field to store the number of values in the initiallization list, and a two-byte field to stcre the list repetition factor m. The last two bytes of every field descriptcr have a pcinter tc the parent table descriptor in the constant table.

Status field names have the same descriptor format as that described above except for the data type field. This field was previously described (see status function).

Boolean field names have the same descriptor fcrmat as those already described except for the absence of a data type field.

9. Item-areas

Item-areas do not need descriptors. The descriptor pointer field in the identifier directory has a pointer to the parent table descriptor which, in turn, has a pointer to the field pcinter list.

10. Subtables

A subtable descriptor ccnsists of the header byte, a number cf two-byte fields, one per dimension, to stcre their values, and a two-byte field for a pointer to the parent table descriptor.

APPENDIX E

OPERATION CODES FOR IL

Operation Code	Meaning
ADD	add operand
ADC	add constant in operand field
SUB	subtract operand
SBC	subtract constant in operand field
MUL	multiply times operand
MLC	multiply times constant in operand field
DIV	divide by operand
DVC	divide by constant in operand field
REM	remainder
TST	compare
EXP	exponentiation
ABS	absolute value
SCL	scale
CNT	count ones
NEG	unary minus
MOD	modulo
TST	compare
BLS	branch on <
BLQ	branch on <=
BEQ	branch on =
BNQ	branch on ¬=
BGQ	brach on >=
BGR	branch on >
BRF	branch on false
BRT	branch on true
BRU	unconditional branch
NOP	no operation
STP	stop
LDV	load value of operand
LDA	load address of operand
STV	store value of operand
STA	store address of operand

AND	logical and
IOR	inclusive logical or
XOR	exclusive logical or
NOT	logical complement
INX	subscript computation
XCH	exchange operands
CAT	string concatenation
INR	input from reader into operand
IRF	input from reader into operand with format
OTP	output to printer operand
OPF	output to printer operand with format
DEF	define label location
LNE	source line number
ESP	enter sysproc
LSP	leave sysproc
ESR	enter sysprocreen
LSR	leave sysprocreen
EPR	enter procedure
LPR	leave procedure
EFN	enter function
LFN	leave function
PRC	procedure call
FNC	function call
INC	increment
EVL	enter vary loop
LVL	leave vary loop
EWL	enter while loop
LWL	leave while loop
ECG	enter case group
LCG	leave case group
DCL	define case label
ERB	enter reserved block
LRB	leave reserved block
EBB	enter begin block
LBB	leave begin block
ECB	enter cobegin block

LCB	leave cobegin block
RVL	resume vary loop
RTL	retyrn to label
PCK	pack into data unit
UNP	unpack into data unit
ENC	encode into data unit
DEC	decode into data unit
SRC	search for data unit
SHL	shift left amount specified in operand
SHR	shift right amount specified in operand
CSL	circular shift left
CSR	circular shift right
INC	input from operator controlled medium
IOF	input from OCM with format
OTN	output to punch
ONF	output to punch with format
OTC	output to OCM
OOF	output to OCM with format
INU	input from user file
OTU	output to user file
BTI	convert binary to integer
ITB	convert integer to binary
ITC	convert integer to character
CTI	convert character to integer
ITX	convert integer to fixed
XTI	convert fixed to integer
ITL	convert integer to float
LTI	convert float to integer
XTL	convert fixed to float
LTX	convert float to fixed
OTD	convert octal to decimal
DTO	convert decimal to octal
XTB	convert fixed to binary
BTX	convert binary to fixed
LTB	convert float to binary
BTL	convert binary to float

LBD	length in binary digits
LBY	length in bytes
LWD	length in words

APPENDIX F

CMS-2RS COMPILER LISTING

```
BEGIN

 COMMENT  THE FOLLOWING CARDS WERE PUNCHED BY THE SLR(1) SYNTAX
          ANALYZER (SLR1ANAL) ;

COMMENT PARSING TABLES FOR MAIN GRAMMAR FOLLOWS;
 INTEGER NUMTERMINALS = 0108;
 INTEGER NUMNTS = 0090;
 INTEGER NUMSYMS = 0199;


 GLOBAL DATA SEGTABLE BASE R12;

 ARRAY 1938 BYTE VSTRING = (#C5X, #D9X, #D9X, #D6X, #D9X, #E2X, #E8X,
         #D4X, #40X, #6EX, #6DX, #4FX, #6DX, #4CX, #E2X, #E8X, #E2X,
         #6EX, #4CX, #E2X, #E8X, #E2X, #40X, #C4X, #C5X, #C3X, #D3X,
         #40X, #C8X, #C5X, #C1X, #C4X, #6EX, #4CX, #E2X, #E8X, #E2X,
         #4CX, #C5X, #D3X, #C5X, #D4X, #6EX, #4CX, #C5X, #D5X, #C4X,
         #4CX, #D7X, #C8X, #D9X, #C1X, #E2X, #C5X, #6EX, #E2X, #E8X,
         #E2X, #E3X, #C5X, #D4X, #4CX, #C9X, #C4X, #6EX, #5BX, #4CX,
         #C8X, #C5X, #C1X, #C4X, #C5X, #D9X, #4CX, #C4X, #C5X, #C3X,
         #D3X, #6EX, #4CX, #E2X, #E8X, #E2X, #4CX, #C4X, #C1X, #E3X,
         #C1X, #40X, #C4X, #C5X, #C3X, #D3X, #6EX, #4CX, #E2X, #E8X,
         #E2X, #D7X, #D9X, #D6X, #C3X, #40X, #C4X, #C5X, #C3X, #D3X,
         #40X, #C8X, #C5X, #C1X, #C4X, #6EX, #4CX, #E2X, #E8X, #E2X,
         #D7X, #D9X, #D6X, #C3X, #40X, #C5X, #D3X, #C5X, #D4X, #6EX,
         #4CX, #E2X, #E8X, #E2X, #D7X, #D9X, #D6X, #C3X, #D9X, #C5X,
         #D5X, #40X, #C4X, #C5X, #C3X, #D3X, #4CX, #C8X, #C5X, #C1X,
         #C4X, #6EX, #4CX, #E2X, #E8X, #E2X, #D7X, #D9X, #D6X, #C3X,
         #D9X, #C5X, #D5X, #40X, #C5X, #D3X, #C5X, #D4X, #6EX, #C5X,
         #E7X, #E3X, #D9X, #C5X, #C6X, #E2X, #E8X, #E2X, #6DX, #D7X,
         #D9X, #D6X, #C3X, #E2X, #E8X, #E2X, #6DX, #D7X, #D9X, #D6X,
         #C3X, #6DX, #D9X, #4CX, #D3X, #D6X, #C3X, #C1X, #D3X, #4CX,
         #C4X, #C1X, #E3X, #C1X, #40X, #C4X, #C5X, #C3X, #D3X, #6EX,
         #4CX, #E2X, #E4X, #C2X, #60X, #D9X, #D6X, #E4X, #E3X, #C9X,
         #D5X, #C5X, #40X, #C4X, #C5X, #C3X, #D3X, #6EX, #4CX, #E2X,
         #E3X, #D4X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX,
         #4CX, #C1X, #E4X, #E3X, #D6X, #40X, #C4X, #C1X, #E3X, #C1X,
         #40X, #C4X, #C5X, #C3X, #D3X, #6EX, #C5X, #D5X, #C4X, #4CX,
         #E2X, #E3X, #D4X, #6EX, #4CX, #D3X, #C1X, #C2X, #C5X, #D3X,
```

```
#40X, #C4X, #C5X, #C6X, #D5X, #6EX, #4CX, #C2X, #C1X, #E2X,
#C9X, #C3X, #40X, #E2X, #E3X, #D4X, #6EX, #4CX, #D3X, #C1X,
#C2X, #C5X, #D3X, #6EX, #4CX, #E2X, #C9X, #D4X, #D7X, #D3X,
#C5X, #40X, #E2X, #E3X, #D4X, #6EX, #4CX, #E5X, #C1X, #D9X,
#E8X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX,
#C4X, #D6X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX,
#4CX, #C3X, #C1X, #E2X, #C5X, #40X, #C3X, #D3X, #C1X, #E4X,
#E2X, #C5X, #6EX, #4CX, #C3X, #C1X, #E2X, #C5X, #40X, #D3X,
#C9X, #E2X, #E3X, #6EX, #4CX, #C5X, #D3X, #E2X, #C5X, #40X,
#C3X, #C1X, #E2X, #C5X, #6EX, #4CX, #E6X, #C8X, #C9X, #D3X,
#C5X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX,
#C9X, #C6X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX,
#4CX, #E3X, #C8X, #C5X, #D5X, #40X, #C3X, #D3X, #C1X, #E4X,
#E2X, #C5X, #6EX, #4CX, #C2X, #C5X, #C7X, #C9X, #D5X, #40X,
#C8X, #C5X, #C1X, #C4X, #6EX, #4CX, #E2X, #C5X, #E3X, #4CX,
#C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #E3X, #D6X, #4CX,
#C5X, #E7X, #D7X, #D9X, #6EX, #4CX, #E2X, #E6X, #C1X, #D7X,
#40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #C6X,
#D6X, #D9X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX,
#4CX, #E2X, #C5X, #C1X, #D9X, #C3X, #C8X, #40X, #C3X, #D3X,
#C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #C9X, #D5X, #D7X, #E4X,
#E3X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX,
#D9X, #C5X, #C3X, #C5X, #D7X, #E3X, #C1X, #C3X, #D3X, #C5X,
#4CX, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #D6X,
#E4X, #E3X, #D7X, #E4X, #E3X, #40X, #C3X, #D3X, #C1X, #E4X,
#E2X, #C5X, #6EX, #4CX, #E2X, #D6X, #E4X, #D9X, #C3X, #C5X,
#4CX, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #C5X,
#D5X, #C3X, #D6X, #C4X, #C5X, #40X, #C3X, #D3X, #C1X, #E4X,
#E2X, #C5X, #6EX, #4CX, #C4X, #C5X, #C3X, #D6X, #C4X, #C5X,
#4CX, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #D7X,
#C1X, #C3X, #D2X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X,
#6EX, #4CX, #E4X, #D5X, #D7X, #C1X, #C3X, #D2X, #40X, #C3X,
#D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #C3X, #C1X, #D3X, #D3X,
#4CX, #D7X, #D9X, #D6X, #C3X, #C5X, #C4X, #E4X, #D9X, #C5X,
#4CX, #E2X, #E3X, #D4X, #6EX, #4CX, #C3X, #D6X, #D5X, #E3X,
#D9X, #D6X, #D3X, #40X, #D7X, #C8X, #D9X, #C1X, #E2X, #C5X,
#6EX, #D9X, #C5X, #E3X, #E4X, #D9X, #D5X, #4CX, #C4X, #C1X,
#E3X, #C1X, #40X, #E4X, #D5X, #C9X, #E3X, #6EX, #4CX, #C3X,
#D6X, #C2X, #C5X, #C7X, #C9X, #D5X, #4CX, #C8X, #C5X, #C1X,
#C4X, #6EX, #C3X, #D6X, #C5X, #D5X, #C4X, #4CX, #D9X, #C5X,
#E2X, #C5X, #D9X, #E5X, #C5X, #40X, #D7X, #C8X, #D9X, #C1X,
#E2X, #C5X, #6EX, #5DX, #4CX, #E6X, #C1X, #C9X, #E3X, #40X,
#C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #E2X, #E3X, #D6X,
#D7X, #E5X, #C1X, #D9X, #E8X, #4CX, #D3X, #D6X, #D6X, #D7X,
#4CX, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #6BX, #4CX,
#C9X, #D5X, #C9X, #E3X, #C9X, #C1X, #D3X, #40X, #C3X, #D3X,
#C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #E3X, #C5X, #E2X, #E3X,
#4CX, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #C9X,
```

```
#D5X, #C3X, #D9X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X,
#6EX, #4CX, #D9X, #C5X, #C1X, #D3X, #4CX, #E5X, #C1X, #D9X,
#6EX, #C6X, #D9X, #D6X, #D4X, #4CX, #E2X, #C9X, #C7X, #D5X,
#C5X, #C4X, #40X, #D9X, #C5X, #C1X, #D3X, #40X, #D7X, #D9X,
#C9X, #6EX, #E2X, #E3X, #C5X, #D7X, #E3X, #C8X, #D9X, #E4X,
#C3X, #C1X, #E2X, #C5X, #D6X, #C6X, #4CX, #C9X, #D5X, #C4X,
#C5X, #E7X, #40X, #C3X, #C1X, #E2X, #C5X, #40X, #D3X, #C9X,
#E2X, #E3X, #6EX, #4CX, #D3X, #C1X, #C2X, #C5X, #D3X, #40X,
#C3X, #C1X, #E2X, #C5X, #40X, #D3X, #C9X, #E2X, #E3X, #6EX,
#4CX, #C2X, #C1X, #E2X, #C9X, #C3X, #4CX, #E2X, #E3X, #D4X,
#40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #D3X,
#C1X, #C2X, #C5X, #D3X, #40X, #C3X, #C1X, #E2X, #C5X, #6EX,
#4CX, #C3X, #C1X, #E2X, #C5X, #40X, #D3X, #C1X, #C2X, #C5X,
#D3X, #6EX, #4CX, #D9X, #C5X, #C1X, #D3X, #40X, #C3X, #D6X,
#D5X, #E2X, #6EX, #7AX, #4CX, #E2X, #E3X, #C1X, #E3X, #E4X,
#E2X, #40X, #C3X, #D6X, #D5X, #E2X, #6EX, #C5X, #D3X, #E2X,
#C5X, #E6X, #C8X, #C9X, #D3X, #C5X, #4CX, #C2X, #D6X, #D6X,
#D3X, #40X, #C5X, #E7X, #D7X, #D9X, #6EX, #C4X, #D6X, #C9X,
#C6X, #E3X, #C8X, #C5X, #D5X, #4CX, #E3X, #D9X, #E4X, #C5X,
#40X, #D7X, #C1X, #D9X, #E3X, #6EX, #4CX, #E2X, #C9X, #D4X,
#D7X, #D3X, #C5X, #40X, #E2X, #E3X, #D4X, #40X, #C3X, #C3X,
#C1X, #E4X, #E2X, #C5X, #6EX, #C2X, #C5X, #C7X, #C9X, #C5X,
#C3X, #D6X, #C2X, #C5X, #C7X, #C9X, #D5X, #E2X, #C5X, #E3X,
#4CX, #C4X, #C1X, #E3X, #C1X, #40X, #E4X, #D5X, #C9X, #E3X,
#40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #D9X,
#C5X, #C1X, #D3X, #40X, #C5X, #E7X, #D7X, #D9X, #6EX, #4CX,
#C2X, #C9X, #E3X, #E2X, #40X, #C5X, #E7X, #D7X, #D9X, #6EX,
#4CX, #C3X, #C8X, #C1X, #D9X, #40X, #C5X, #E7X, #D7X, #D9X,
#6EX, #4CX, #E2X, #E3X, #C1X, #E3X, #E4X, #E2X, #40X, #C5X,
#E7X, #D7X, #D9X, #6EX, #4CX, #E3X, #C1X, #C2X, #D3X, #C5X,
#40X, #C9X, #C4X, #C5X, #D5X, #E3X, #C9X, #C6X, #C9X, #C5X,
#D9X, #6EX, #4CX, #D9X, #C5X, #C1X, #D3X, #40X, #E3X, #C5X,
#D9X, #D4X, #6EX, #4CX, #C1X, #C4X, #C4X, #E2X, #E4X, #C2X,
#40X, #D6X, #D7X, #C5X, #D9X, #6EX, #4EX, #60X, #4CX, #D9X,
#C5X, #C1X, #D3X, #40X, #E2X, #C5X, #C3X, #D5X, #6EX, #4CX,
#D4X, #E4X, #D3X, #C4X, #C9X, #E5X, #4CX, #D6X, #D7X, #C5X,
#D9X, #6EX, #5CX, #61X, #D9X, #C5X, #D4X, #5CX, #5CX, #4CX,
#D9X, #C5X, #C1X, #D3X, #40X, #D7X, #D9X, #C9X, #6EX, #C1X,
#C2X, #E2X, #4CX, #D9X, #C5X, #C1X, #D3X, #40X, #C4X, #C1X,
#E3X, #C1X, #40X, #E4X, #D5X, #C9X, #E3X, #6EX, #4DX, #4BX,
#4BX, #4CX, #D9X, #C5X, #C1X, #D3X, #4CX, #C6X, #E4X, #D5X,
#C3X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #4CX, #C9X, #D5X,
#D7X, #E4X, #E3X, #40X, #D7X, #C1X, #D9X, #C1X, #D4X, #E2X,
#6EX, #4CX, #D9X, #C5X, #C1X, #D3X, #6EX, #4CX, #E3X, #C1X,
#C7X, #6EX, #4CX, #D9X, #C5X, #C1X, #D3X, #40X, #C6X, #C9X,
#C5X, #D3X, #C4X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #4CX,
#E2X, #E4X, #C2X, #E2X, #C3X, #D9X, #C9X, #D7X, #E3X, #4CX,
#C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX, #4CX, #C2X, #C9X,
```

```
#E3X, #E2X, #40X, #C6X, #C1X, #C3X, #E3X, #D6X, #D9X, #6EX,
#4CX, #D3X, #D6X, #C7X, #C9X, #C3X, #C1X, #D3X, #40X, #D6X,
#D7X, #C5X, #D9X, #6EX, #C1X, #D5X, #C4X, #D3X, #D6X, #C9X,
#D3X, #4CX, #C2X, #C9X, #E3X, #E2X, #4CX, #E2X, #C5X, #C3X,
#D5X, #6EX, #4CX, #E2X, #C8X, #C9X, #C6X, #E3X, #40X, #C6X,
#D7X, #C5X, #D9X, #6EX, #E2X, #C8X, #D3X, #D3X, #E2X, #C8X,
#D9X, #D3X, #C3X, #C9X, #D9X, #E2X, #C8X, #D3X, #D3X, #C3X,
#C9X, #D9X, #E2X, #C8X, #D9X, #D3X, #4CX, #C2X, #C9X, #E3X,
#E2X, #40X, #D7X, #D9X, #C9X, #6EX, #D5X, #D6X, #E3X, #C3X,
#4CX, #C2X, #C9X, #E3X, #E2X, #40X, #C3X, #D6X, #D5X, #E2X,
#6EX, #4CX, #C2X, #C9X, #E3X, #E2X, #4CX, #C4X, #C1X, #E3X,
#C1X, #40X, #E4X, #D5X, #C9X, #E3X, #6EX, #4CX, #C2X, #C9X,
#E3X, #E2X, #40X, #C6X, #E4X, #D5X, #C3X, #40X, #D5X, #C1X,
#D4X, #C5X, #6EX, #4CX, #C2X, #C9X, #E3X, #E2X, #40X, #E5X,
#C1X, #D9X, #6EX, #4CX, #C2X, #C9X, #E3X, #E2X, #40X, #C6X,
#C9X, #C5X, #D3X, #C4X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX,
#C3X, #C1X, #E3X, #4CX, #C3X, #C8X, #C1X, #D9X, #40X, #D7X,
#D9X, #C9X, #6EX, #4CX, #C3X, #C8X, #C1X, #D9X, #40X, #C3X,
#D6X, #D5X, #E2X, #6EX, #4CX, #C3X, #C8X, #C1X, #D9X, #40X,
#C4X, #C1X, #E3X, #C1X, #40X, #E4X, #D5X, #C9X, #E3X, #6EX,
#4CX, #C3X, #C8X, #C1X, #D9X, #40X, #C6X, #E4X, #D5X, #C3X,
#40X, #D5X, #C1X, #D4X, #C5X, #6EX, #4CX, #C3X, #C8X, #C1X,
#D9X, #40X, #E5X, #C1X, #D9X, #6EX, #4CX, #C3X, #C8X, #C1X,
#D9X, #40X, #C6X, #C9X, #C5X, #D3X, #C4X, #40X, #D5X, #C1X,
#D4X, #C5X, #6EX, #4CX, #E2X, #E3X, #C1X, #E3X, #E4X, #E2X,
#40X, #C4X, #C1X, #E3X, #C1X, #40X, #E4X, #D5X, #C9X, #E3X,
#6EX, #4CX, #E2X, #E3X, #C1X, #E3X, #E4X, #E2X, #40X, #C6X,
#E4X, #D5X, #C3X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #4CX,
#E2X, #E3X, #C1X, #E3X, #E4X, #E2X, #40X, #E5X, #C1X, #D9X,
#6EX, #4CX, #E2X, #E3X, #C1X, #E3X, #E4X, #E2X, #40X, #C6X,
#C9X, #C5X, #D3X, #C4X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX,
#4CX, #C2X, #D6X, #D6X, #D3X, #40X, #E2X, #C5X, #C3X, #D5X,
#6EX, #4CX, #C2X, #D6X, #D6X, #D3X, #4CX, #D6X, #D7X, #C5X,
#D9X, #6EX, #C1X, #D5X, #C4X, #D6X, #D9X, #4CX, #C2X, #D6X,
#D6X, #D3X, #40X, #D7X, #D9X, #C9X, #6EX, #D5X, #D6X, #E3X,
#4CX, #C2X, #D6X, #D6X, #D3X, #40X, #C3X, #D6X, #D5X, #E2X,
#6EX, #4CX, #C2X, #D6X, #D6X, #D3X, #40X, #C4X, #C1X, #E3X,
#C1X, #40X, #E4X, #D5X, #C9X, #E3X, #6EX, #4CX, #C2X, #C6X,
#D6X, #D3X, #40X, #C6X, #E4X, #D5X, #C3X, #40X, #D5X, #C1X,
#D4X, #C5X, #6EX, #4CX, #D9X, #C5X, #C3X, #40X, #C5X, #E7X,
#D7X, #D9X, #6EX, #4CX, #C2X, #D6X, #D6X, #D3X, #40X, #E5X,
#C1X, #D9X, #6EX, #4CX, #C2X, #D6X, #C6X, #D3X, #40X, #C6X,
#C9X, #C5X, #D3X, #C4X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX,
#4CX, #D9X, #C5X, #D3X, #40X, #D6X, #D7X, #C5X, #D9X, #6EX,
#7EX, #5FX, #7EX, #4CX, #6EX, #4CX, #E3X, #C1X, #C2X, #D3X,
#C5X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #4CX, #E2X, #E4X,
#C2X, #60X, #E3X, #C1X, #C2X, #D3X, #C5X, #40X, #D5X, #C1X,
#D4X, #C5X, #6EX, #4CX, #C9X, #E3X, #C5X, #D4X, #60X, #C1X,
```

```
        #D9X, #C5X, #C1X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #E2X,
        #E6X, #C1X, #D7X, #E2X, #C5X, #C1X, #D9X, #C3X, #C8X, #4CX,
        #C4X, #C1X, #E3X, #C1X, #40X, #E2X, #E3X, #D9X, #E4X, #C3X,
        #E3X, #E4X, #D9X, #C5X, #6EX, #4CX, #C6X, #C9X, #D3X, #C5X,
        #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #C6X, #D6X, #D9X, #4CX,
        #D7X, #D9X, #D6X, #C3X, #C5X, #C4X, #E4X, #D9X, #C5X, #40X,
        #D5X, #C1X, #D4X, #C5X, #6EX, #4CX, #C1X, #C3X, #E3X, #E4X,
        #C1X, #D3X, #40X, #D7X, #C1X, #D9X, #C1X, #D4X, #C5X, #E3X,
        #C5X, #D9X, #40X, #D3X, #C9X, #E2X, #E3X, #6EX, #4CX, #D6X,
        #E4X, #E3X, #D7X, #E4X, #E3X, #40X, #D7X, #C1X, #D9X, #C1X,
        #D4X, #E2X, #6EX, #4CX, #D3X, #C1X, #C2X, #C5X, #D3X, #40X,
        #D7X, #C1X, #D9X, #C1X, #D4X, #E2X, #6EX, #4FX, #4FX, #4FX,
        #D9X, #C5X, #E2X, #E4X, #D4X, #C5X, #D9X, #C5X, #E3X, #E4X,
        #D9X, #D5X, #E3X, #D6X, #C7X, #D6X, #E3X, #D6X, #D9X, #C5X,
        #E2X, #C5X, #D9X, #E5X, #C5X, #4CX, #D9X, #C5X, #E2X, #D6X,
        #E4X, #D9X, #C3X, #C5X, #6EX, #E6X, #C1X, #C9X, #E3X, #C9X,
        #D5X, #D7X, #E4X, #E3X, #4CX, #C6X, #D6X, #D9X, #D4X, #C1X,
        #E3X, #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #D6X, #E4X, #E3X,
        #D7X, #E4X, #E3X, #C5X, #D5X, #C3X, #D6X, #C4X, #C5X, #C4X,
        #C5X, #C3X, #D6X, #C4X, #C5X, #D7X, #C1X, #C3X, #D2X, #C9X,
        #D5X, #E4X, #D5X, #D7X, #C3X, #D2X, #C9X, #D5X, #E3X, #D6X,
        #D6X, #E4X, #E3X, #D6X, #C6X, #D7X, #D9X, #C9X, #D5X, #E3X,
        #D7X, #D5X, #C3X, #C8X, #D9X, #C5X, #C1X, #C4X, #D6X, #C3X,
        #D4X);

ARRAY 0199 INTEGER LOCLENGTH = (#0000000A, #00000283, #00000DC6,
        #00000F44, #00001041, #0000108D, #000013CF, #00002986,
        #00002B08, #00002D0A, #00002F91, #000033D2, #00003E50,
        #00003F43, #00004707, #00006AC2, #00009444, #00009DC6,
        #0000A4C5, #0000AA01, #0000AD84, #0000AE84, #0000B2C1,
        #0000BD8A, #0000C004, #0000C544, #0000C644, #0000C744,
        #0000C842, #0000DE81, #0000DECD, #0000E204, #0000E305,
        #0000E702, #0000E782, #0000E804, #0000F085, #0000F1C7,
        #0000F383, #00010EC1, #00010F01, #00011541, #00011581,
        #000115C3, #00011682, #00011983, #00011E41, #00011E82,
        #00011F10, #00012686, #00012805, #00012951, #000138C4,
        #000139C3, #00014044, #00014144, #00014247, #00014407,
        #00014844, #0001494B, #00015010, #0001540A, #00015691,
        #00015AC3, #00015ECB, #000164D0, #000168CA, #00016B51,
        #00017412, #0001788C, #00017B93, #000185C3, #00018682,
        #00018983, #00018A4B, #00019110, #0001978A, #00019A11,
        #0001A0C1, #0001A102, #0001A181, #0001A1C1, #0001A2CC,
        #0001A510, #0001A910, #0001AC04, #0001AE06, #0001B643,
        #0001B710, #0001C801, #0001C842, #0001C8C6, #0001CA48,
        #0001CC44, #0001CD47, #0001D184, #0001D285, #0001D3CD,
        #0001D706, #0001D886, #0001DA06, #0001DB86, #0001DD05,
        #0001DE44, #0001DF45, #0001E085, #0001E1C4, #0001E2C4,
        #0001E3C3, #00000345, #0000048F, #0000084A, #00000ACC,
```

```
        #00001793, #00001C4E, #00001FD6, #00002551, #0000384C,
        #00004005, #0000414C, #0000444B, #0CC048CC, #0CC04BCD,
        #0CCC4F0B, #000051CD, #0000550B, #000057CB, #00005A8E,
        #0CC05E0B, #000060CD, #000C640C, #00CC67CC, #00006A86,
        #00006C0D, #00006F4C, #0000724F, #0CCC76CE, #C0007993,
        #00007E4F, #0000820F, #000085CF, #0000898F, #00008D4D,
        #0000908F, #0000954F, #000C9910, #0CCC9E8B, #CC00A14E,
        #0000A610, #0000AA4D, #0000AF8D, #0CCCB310, #0C00B70D,
        #C000BA4D, #0000C111, #0000C8C1, #0000CD11, #0000D152,
        #0000D5CC, #0000D8CC, #0000DECB, #00CCE44B, #0000E90B,
        #0000EBD3, #0000F452, #0000F8CB, #000CFB8B, #0000FE4B,
        #0001010D, #00010452, #C0010 8CB, #0CC10B8D, #00010F4B,
        #0001120D, #0001170A, #C0011A50, #00C1230E, #0CC12D92,
        #0001320D, #0001354E, #00013A8B, #00C13C4C, #00C145CA,
        #0C014C10, #C0015B8A, #C0016CDC, #00C16F92, #0001804B,
        #0001830B, #0001870A, #00018C10, #0CC195CA, #C0C19E4A,
        #0C01AF9C, #0001B38B, #0001EE17, #0001C0CF, #0001C48E,
        #0001CF0A);

COMMENT  THE DPDA HAS 0197 READ STATES;

ARRAY 0197 SHORT INTEGER READSTART = (#0000S, #0002S, #0004S, #0C06S,
        #000CS, #000ES, #0010S, #0014S, #0017S, #0019S, #001BS,
        #0C1DS, #001FS, #0021S, #0023S, #009CS, #0092S, #0094S,
        #0096S, #0098S, #009AS, #0107S, #01CAS, #010CS, #010ES,
        #0110S, #017DS, #0180S, #0182S, #0184S, #0186S, #0188S,
        #018AS, #018CS, #018ES, #0190S, #0192S, #0194S, #0196S,
        #0203S, #0206S, #0209S, #020BS, #020DS, #027AS, #02E7S,
        #02E9S, #02EBS, #02EDS, #02F2S, #02F7S, #02F9S, #02FBS,
        #02FDS, #02FFS, #0301S, #0303S, #03C5S, #0372S, #0374S,
        #03E1S, #044ES, #04BBS, #04BDS, #052AS, #0538S, #053AS,
        #053CS, #053ES, #0540S, #0542S, #0544S, #0546S, #0548S,
        #054AS, #054ES, #055CS, #055ES, #05CBS, #05D5S, #05D7S,
        #05DAS, #05DCS, #05DFS, #064CS, #06B9S, #06BBS, #06BFS,
        #06C4S, #06C9S, #06CES, #06D3S, #06D5S, #06D7S, #06D9S,
        #06DBS, #06DFS, #06E3S, #06E7S, #06EES, #06EES, #06F6S,
        #06F8S, #06FAS, #06FDS, #070CS, #070AS, #070CS, #0779S,
        #07E6S, #07EAS, #07ECS, #0859S, #085BS, #085DS, #085FS,
        #0861S, #0864S, #0867S, #0869S, #086ES, #0871S, #0875S,
        #087AS, #087FS, #0881S, #0886S, #088CS, #0891S, #0898S,
        #089FS, #08A1S, #08A4S, #08A6S, #08AAS, #C8ADS, #C8AFS,
        #091CS, #091FS, #0921S, #0923S, #0926S, #0929S, #092BS,
        #092DS, #092FS, #0938S, #0941S, #094AS, #C9B7S, #09BBS,
        #09BFS, #09C3S, #09C6S, #09C9S, #09CCS, #09D3S, #09D5S,
        #09D7S, #09DDS, #09E3S, #09E8S, #09EES, #09EES, #09F1S,
        #09F4S, #09F7S, #09FAS, #09FDS, #0ACCS, #0A02S, #0AC4S,
        #0A71S, #0A7AS, #0A81S, #0A87S, #0A90S, #0A99S, #0AA2S,
        #0AA9S, #0AB2S, #0ABBS, #0AC4S, #0AC7S, #0AC9S, #0B36S,
```

```
        #0B38S, #0B46S, #0B48S, #0B51S, #0B53S, #0B59S, #0B5DS,
        #0B62S, #0B64S, #0B66S, #0E68S);

ARRAY 0197 BYTE RDNUM = (#01X, #01X, #01X, #C5X, #01X, #01X, #03X,
        #02X, #01X, #C1X, #01X, #01X, #01X, #C1X, #6DX, #01X, #01X,
        #01X, #01X, #01X, #6DX, #02X, #01X, #C1X, #01X, #6DX, #C2X,
        #01X, #01X, #01X, #01X, #01X, #01X, #01X, #01X, #01X, #01X,
        #01X, #6DX, #C2X, #02X, #01X, #01X, #6CX, #6DX, #01X, #01X,
        #01X, #04X, #04X, #01X, #01X, #01X, #C1X, #01X, #01X, #C1X,
        #6CX, #01X, #6DX, #6DX, #6DX, #01X, #6CX, #0DX, #01X, #01X,
        #01X, #01X, #01X, #01X, #01X, #01X, #C1X, #03X, #0CX, #01X,
        #6CX, #09X, #01X, #02X, #01X, #02X, #6DX, #6DX, #01X, #C3X,
        #04X, #04X, #04X, #04X, #01X, #01X, #C1X, #01X, #03X, #03X,
        #03X, #03X, #02X, #07X, #01X, #01X, #C2X, #02X, #09X, #01X,
        #6CX, #6DX, #03X, #01X, #6CX, #01X, #01X, #01X, #01X, #C2X,
        #02X, #C1X, #C4X, #02X, #03X, #04X, #C4X, #01X, #04X, #05X,
        #04X, #06X, #C6X, #01X, #02X, #01X, #C3X, #02X, #C1X, #6CX,
        #02X, #01X, #01X, #02X, #02X, #01X, #01X, #01X, #08X, #C8X,
        #08X, #6DX, #C3X, #03X, #03X, #02X, #02X, #02X, #06X, #01X,
        #01X, #05X, #05X, #04X, #02X, #02X, #C2X, #02X, #02X, #02X,
        #02X, #02X, #01X, #01X, #6CX, #08X, #C6X, #05X, #08X, #08X,
        #08X, #06X, #08X, #08X, #08X, #02X, #C1X, #6DX, #01X, #0CX,
        #01X, #08X, #01X, #05X, #03X, #04X, #01X, #01X, #01X, #C2X);

ARRAY 0145 BYTE SYMLIST = (#01X, #00X, #02X, #CCX, #C1X, #C0X, #05X,
        #06X, #07X, #08X, #09X, #00X, #03X, #CCX, #0DX, #00X, #0AX,
        #0BX, #0CX, #C0X, #0BX, #0CX, #00X, #C3X, #00X, #03X, #00X,
        #03X, #00X, #C4X, #00X, #04X, #C0X, #CDX, #00X, #00X, #C1X,
        #02X, #03X, #C4X, #05X, #06X, #07X, #C8X, #09X, #0AX, #0BX,
        #0CX, #0DX, #0EX, #0FX, #1CX, #11X, #12X, #13X, #14X, #15X,
        #16X, #17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX,
        #20X, #21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X,
        #2AX, #2BX, #2CX, #2DX, #2EX, #2FX, #3CX, #31X, #32X, #33X,
        #34X, #35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX,
        #3EX, #3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X,
        #48X, #49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X,
        #52X, #53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX,
        #5CX, #5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X,
        #66X, #67X, #68X, #69X, #6AX, #6BX, #6CX, #0DX);
CLCSE BASE;
GLCBAL DATA SEGT2 BASE R6;
  ARRAY 2778 BYTE CCNTSYMLIST = (
                                                        #00X, #04X,
        #00X, #04X, #00X, #04X, #0CX, #04X, #CCX, #00X, #01X, #02X,
        #03X, #04X, #05X, #06X, #07X, #08X, #C9X, #0AX, #0BX, #CCX,
        #0DX, #0EX, #CFX, #10X, #11X, #12X, #13X, #14X, #15X, #16X,
        #17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X,
        #21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX,
```

```
#2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X,
#35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX; #3EX,
#3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X,
#49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X,
#53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX,
#5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X,
#67X, #68X, #69X, #6AX, #6BX, #6CX, #16X, #21X, #0CX, #1CX,
#00X, #21X, #00X, #23X, #00X, #00X, #01X, #02X, #03X, #04X,
#05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX, #0EX,
#0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X, #18X,
#19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X, #22X,
#23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX, #2CX,
#2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X, #36X,
#37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX, #40X,
#41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X; #4AX,
#4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X, #54X,
#55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX, #5EX,
#5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X, #68X,
#69X, #6AX, #6BX, #6CX, #0FX, #16X, #0CX, #57X, #00X, #57X,
#00X, #67X, #00X, #68X, #0CX, #68X, #0CX, #67X, #00X, #68X,
#00X, #67X, #00X, #58X, #00X, #0EX, #0CX, #1DX, #00X, #0CX,
#01X, #02X, #03X, #04X, #05X, #06X, #07X, #08X, #09X, #0AX,
#0BX, #0CX, #0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #14X,
#15X, #16X, #17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX,
#1FX, #20X, #21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X,
#29X, #2AX, #2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X,
#33X, #34X, #35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX,
#3DX, #3EX, #3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X,
#47X, #48X, #49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X,
#51X, #52X, #53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX,
#5BX, #5CX, #5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X,
#65X, #66X, #67X, #68X, #69X, #6AX, #6BX, #6CX, #13X, #16X,
#0CX, #13X, #16X, #00X, #17X, #00X, #1DX, #00X, #00X, #01X,
#02X, #03X, #04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX,
#0CX, #0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X,
#16X, #17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX,
#20X, #21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X,
#2AX, #2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X,
#34X, #35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX,
#3EX, #3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X,
#48X, #49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X,
#52X, #53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX,
#5CX, #5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X,
#66X, #67X, #68X, #69X, #6AX, #6BX, #6CX, #00X, #01X, #02X,
#03X, #04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX,
#0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X,
#17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X,
#21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX,
```

```
#2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X,
#35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX,
#3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X,
#49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X,
#53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX,
#5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X,
#67X, #68X, #69X, #6AX, #6BX, #6CX, #1DX, #00X, #1DX, #00X,
#1DX, #00X, #69X, #6AX, #6BX, #6CX, #0CX, #69X, #6AX, #6BX,
#6CX, #00X, #1DX, #00X, #1DX, #00X, #1DX, #00X, #1DX, #0CX,
#2EX, #00X, #2EX, #00X, #17X, #00X, #0CX, #01X, #02X, #03X,
#04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX,
#0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X,
#18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X,
#22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX,
#2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X,
#36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX,
#40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X,
#4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X,
#54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX,
#5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X,
#68X, #69X, #6AX, #6BX, #6CX, #1FX, #0CX, #00X, #01X, #02X,
#03X, #04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX,
#0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X,
#17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X,
#21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX,
#2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X,
#35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX,
#3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X,
#49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X,
#53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX,
#5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X,
#67X, #68X, #69X, #6AX, #6BX, #6CX, #0CX, #01X, #02X, #03X,
#04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX,
#0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X,
#18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X,
#22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX,
#2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X,
#36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX,
#40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X,
#4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X,
#54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX,
#5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X,
#68X, #69X, #6AX, #6BX, #6CX, #00X, #01X, #02X, #03X, #04X,
#05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX, #0EX,
#0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X, #18X,
#19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X, #22X,
#23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX, #2CX,
#2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X, #36X,
```

```
#37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX, #40X,
#41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X, #4AX,
#4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X, #54X,
#55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX, #5EX,
#5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X, #68X,
#69X, #6AX, #6BX, #6CX, #0DX, #00X, #0CX, #01X, #02X, #03X,
#04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX,
#0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X,
#18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X,
#22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX,
#2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X,
#36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX,
#40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X,
#4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X,
#54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX,
#5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X,
#68X, #69X, #6AX, #6BX, #6CX, #17X, #33X, #3DX, #3EX, #42X,
#43X, #45X, #46X, #4CX, #4DX, #52X, #53X, #54X, #0CX, #1DX,
#00X, #16X, #00X, #1DX, #00X, #16X, #0CX, #1DX, #0CX, #16X,
#0CX, #16X, #00X, #16X, #00X, #16X, #0CX, #2EX, #59X, #5AX,
#00X, #17X, #33X, #3DX, #3EX, #42X, #43X, #45X, #46X, #4CX,
#4DX, #52X, #53X, #54X, #00X, #12X, #00X, #00X, #01X, #02X,
#03X, #04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX,
#0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X,
#17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X,
#21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX,
#2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X,
#35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX,
#3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X,
#49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X,
#53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX,
#5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X,
#67X, #68X, #69X, #6AX, #6BX, #6CX, #1DX, #52X, #53X, #54X,
#58X, #69X, #6AX, #6BX, #6CX, #00X, #5EX, #00X, #19X, #1AX,
#00X, #18X, #00X, #47X, #48X, #00X, #0CX, #01X, #02X, #03X,
#04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX,
#0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X,
#18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X,
#22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX,
#2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X,
#36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX,
#40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X,
#4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X,
#54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX,
#5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X,
#68X, #69X, #6AX, #6BX, #6CX, #00X, #01X, #02X, #03X, #04X,
#05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX, #0EX,
#0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X, #18X,
```

```
#19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X, #22X,
#23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX, #2CX,
#2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X, #36X,
#37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX, #40X,
#41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X, #4AX,
#4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X, #54X,
#55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX, #5EX,
#5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X, #68X,
#69X, #6AX, #6BX, #6CX, #2EX, #00X, #52X, #53X, #54X, #0CX,
#4EX, #4FX, #50X, #51X, #00X, #4EX, #4FX, #50X, #51X; #00X,
#4EX, #4FX, #50X, #51X, #00X, #4EX, #4FX, #50X, #51X, #0CX,
#2EX, #00X, #2EX, #00X, #2EX, #00X, #2EX, #00X, #52X; #53X,
#54X, #00X, #52X, #53X, #54X, #00X, #52X, #53X, #54X, #0CX,
#52X, #53X, #54X, #00X, #47X, #48X, #0CX, #52X, #53X, #54X,
#69X, #6AX, #6BX, #6CX, #0CX, #61X, #0CX, #61X, #00X, #42X,
#43X, #00X, #42X, #43X, #00X, #1DX, #52X, #53X, #54X, #58X,
#69X, #6AX, #6BX, #6CX, #00X, #58X, #0CX, #00X, #01X, #02X,
#03X, #04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX,
#0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X,
#17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X,
#21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX,
#2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X,
#35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX,
#3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X,
#49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X,
#53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX,
#5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X,
#67X, #68X, #69X, #6AX, #6BX, #6CX, #00X, #01X, #02X, #03X,
#04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX,
#0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X,
#18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X,
#22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX,
#2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X,
#36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX,
#40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X,
#4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X,
#54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX,
#5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X,
#68X, #69X, #6AX, #6BX, #6CX, #1EX, #31X, #32X, #00X, #04X,
#00X, #00X, #01X, #02X, #03X, #04X, #05X, #06X, #07X, #08X,
#09X, #0AX, #0BX, #0CX, #0DX, #0EX, #0FX, #10X, #11X, #12X,
#13X, #14X, #15X, #16X, #17X, #18X, #19X, #1AX, #1BX, #1CX,
#1DX, #1EX, #1FX, #20X, #21X, #22X, #23X, #24X, #25X, #26X,
#27X, #28X, #29X, #2AX, #2BX, #2CX, #2DX, #2EX, #2FX, #30X,
#31X, #32X, #33X, #34X, #35X, #36X, #37X, #38X, #39X, #3AX,
#3BX, #3CX, #3DX, #3EX, #3FX, #40X, #41X, #42X, #43X, #44X,
#45X, #46X, #47X, #48X, #49X, #4AX, #4BX, #4CX, #4DX, #4EX,
#4FX, #50X, #51X, #52X, #53X, #54X, #55X, #56X, #57X, #58X,
```

```
#59X, #5AX, #5BX, #5CX, #5DX, #5EX, #5FX, #60X, #61X, #62X,
#63X, #64X, #65X, #66X, #67X, #68X, #69X, #6AX, #6BX, #6CX,
#1DX, #00X, #1DX, #00X, #23X, #00X, #1FX, #00X, #27X, #28X,
#00X, #34X, #35X, #00X, #3FX, #00X, #4EX, #4FX, #50X, #51X,
#0CX, #47X, #48X, #00X, #29X, #2AX, #2BX, #00X, #36X, #37X,
#38X, #39X, #00X, #4EX, #4FX, #50X, #51X, #00X, #2CX, #CCX,
#4EX, #4FX, #50X, #51X, #00X, #2EX, #3BX, #3CX, #3DX, #3EX,
#0CX, #4EX, #4FX, #50X, #51X, #00X, #17X, #2EX, #30X, #31X,
#32X, #33X, #00X, #17X, #2EX, #30X, #31X, #32X, #33X, #CCX,
#2EX, #00X, #13X, #16X, #00X, #13X, #CCX, #16X, #59X, #5AX,
#00X, #16X, #5AX, #00X, #16X, #00X, #CCX, #01X, #02X, #03X,
#04X, #05X, #06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #CDX,
#0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X, #17X,
#18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X, #21X,
#22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX,
#2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X,
#36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX,
#4CX, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X,
#4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #5CX, #51X, #52X, #53X,
#54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX,
#5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X,
#68X, #69X, #6AX, #6BX, #6CX, #16X, #1DX, #00X, #03X, #CCX,
#2EX, #00X, #13X, #16X, #00X, #13X, #16X, #00X, #2EX, #CCX,
#2EX, #00X, #1AX, #00X, #17X, #28X, #2DX, #2EX, #30X, #31X,
#32X, #33X, #00X, #17X, #28X, #2DX, #2EX, #30X, #31X, #32X,
#33X, #00X, #17X, #28X, #2DX, #2EX, #30X, #31X, #32X, #33X,
#00X, #00X, #01X, #02X, #03X, #04X, #C5X, #06X, #07X, #08X,
#09X, #0AX, #0BX, #0CX, #0DX, #0EX, #CFX, #10X, #11X, #12X,
#13X, #14X, #15X, #16X, #17X, #18X, #19X, #1AX, #1BX, #1CX,
#1DX, #1EX, #1FX, #20X, #21X, #22X, #23X, #24X, #25X, #26X,
#27X, #28X, #29X, #2AX, #2BX, #2CX, #2DX, #2EX, #2FX, #3CX,
#31X, #32X, #33X, #34X, #35X, #36X, #37X, #38X, #39X, #3AX,
#3BX, #3CX, #3DX, #3EX, #3FX, #40X, #41X, #42X, #43X, #44X,
#45X, #46X, #47X, #48X, #49X, #4AX, #4BX, #4CX, #4DX, #4EX,
#4FX, #50X, #51X, #52X, #53X, #54X, #55X, #56X, #57X, #58X,
#59X, #5AX, #5BX, #5CX, #5DX, #5EX, #5FX, #60X, #61X, #62X,
#63X, #64X, #65X, #66X, #67X, #68X, #69X, #6AX, #6BX, #6CX,
#13X, #47X, #48X, #00X, #13X, #27X, #28X, #00X, #13X, #34X,
#35X, #00X, #13X, #3FX, #00X, #13X, #16X, #00X, #13X, #16X,
#00X, #17X, #2EX, #30X, #31X, #32X, #33X, #00X, #4EX, #00X,
#4EX, #00X, #2EX, #3BX, #3CX, #3DX, #3EX, #00X, #2EX, #4CX,
#41X, #42X, #43X, #00X, #1EX, #44X, #45X, #46X, #00X, #13X,
#16X, #00X, #13X, #16X, #00X, #13X, #16X, #00X, #13X, #16X,
#00X, #13X, #16X, #00X, #13X, #16X, #CCX, #13X, #16X, #0CX,
#13X, #16X, #00X, #61X, #00X, #61X, #CCX, #00X, #01X, #02X,
#03X, #04X, #05X, #06X, #07X, #08X, #C9X, #0AX, #0BX, #0CX,
#0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #14X, #15X, #16X,
#17X, #18X, #19X, #1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #20X,
```

```
#21X, #22X, #23X, #24X, #25X, #26X, #27X, #28X, #29X, #2AX,
#2BX, #2CX, #2DX, #2EX, #2FX, #30X, #31X, #32X, #33X, #34X,
#35X, #36X, #37X, #38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX,
#3FX, #40X, #41X, #42X, #43X, #44X, #45X, #46X, #47X, #48X,
#49X, #4AX, #4BX, #4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X,
#53X, #54X, #55X, #56X, #57X, #58X, #59X, #5AX, #5BX, #5CX,
#5DX, #5EX, #5FX, #60X, #61X, #62X, #63X, #64X, #65X, #66X,
#67X, #68X, #69X, #6AX, #6BX, #6CX, #17X, #28X, #2DX, #2EX,
#30X, #31X, #32X, #33X, #00X, #2EX, #3AX, #3BX, #3CX, #3DX,
#3EX, #00X, #2EX, #40X, #41X, #42X, #43X, #00X, #17X, #28X,
#2DX, #2EX, #30X, #31X, #32X, #33X, #CCX, #17X, #28X, #2DX,
#2EX, #30X, #31X, #32X, #33X, #00X, #17X, #28X, #2DX, #2EX,
#30X, #31X, #32X, #33X, #00X, #2EX, #3AX, #3BX, #3CX, #3DX,
#3EX, #00X, #17X, #28X, #2DX, #2EX, #30X, #31X, #32X, #33X,
#00X, #17X, #28X, #2DX, #2EX, #30X, #31X, #32X, #33X, #0CX,
#17X, #28X, #2DX, #2EX, #30X, #31X, #32X, #33X, #00X, #16X,
#5AX, #00X, #16X, #00X, #00X, #01X, #02X, #03X, #04X, #05X,
#06X, #07X, #08X, #09X, #0AX, #0BX, #0CX, #0DX, #0EX, #0FX,
#1CX, #11X, #12X, #13X, #14X, #15X, #16X, #17X, #18X, #19X,
#1AX, #1BX, #1CX, #1DX, #1EX, #1FX, #2CX, #21X, #22X, #23X,
#24X, #25X, #26X, #27X, #28X, #29X, #2AX, #2BX, #2CX, #2DX,
#2EX, #2FX, #30X, #31X, #32X, #33X, #34X, #35X, #36X, #37X,
#38X, #39X, #3AX, #3BX, #3CX, #3DX, #3EX, #3FX, #40X, #41X,
#42X, #43X, #44X, #45X, #46X, #47X, #48X, #49X, #4AX, #4BX,
#4CX, #4DX, #4EX, #4FX, #50X, #51X, #52X, #53X, #54X, #55X,
#56X, #57X, #58X, #59X, #5AX, #5BX, #5CX, #5DX, #5EX, #5FX,
#60X, #61X, #62X, #63X, #64X, #65X, #66X, #67X, #68X, #69X,
#6AX, #6BX, #6CX, #16X, #00X, #17X, #33X, #3DX, #3EX, #42X,
#43X, #45X, #46X, #4CX, #4DX, #52X, #53X, #54X, #00X, #03X,
#00X, #17X, #28X, #2DX, #2EX, #30X, #31X, #32X, #33X, #00X,
#2FX, #00X, #2EX, #40X, #41X, #42X, #43X, #00X, #29X, #2AX,
#2BX, #00X, #36X, #37X, #38X, #39X, #CCX, #2CX, #00X, #16X,
#00X, #13X, #00X, #31X, #32X, #00X);

ARRAY 0147 SHORT INTEGER STATELIST = (#0001S, #06FFS, #0004S, #06FFS,
     #02E8S, #06FFS, #0204S, #0205S, #00C8S, #0009S, #000AS,
     #06FFS, #000BS, #06FFS, #000CS, #06FFS, #020ES, #010ES,
     #0210S, #06FFS, #010ES, #0210S, #06FFS, #0010S, #06FFS,
     #0011S, #06FFS, #0012S, #06FFS, #02C2S, #06FFS, #0211S,
     #06FFS, #000CS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
     #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
     #06FFS, #06FFS, #06FFS, #0214S, #06FFS, #0023S, #03C4S,
     #06FFS, #06FFS, #022CS, #0029S, #06FFS, #06FFS, #06FFS,
     #06FFS, #06FFS, #002AS, #06FFS, #06FFS, #06FFS, #06FFS,
     #0128S, #06FFS, #012CS, #06FFS, #024AS, #024CS, #0C2DS,
     #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
     #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
     #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
```

```
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #002ES, #002FS, #06FFS,
          #06FFS, #06FFS, #06FFS, #02CES, #02CFS, #02D0S, #0036S,
          #0037S, #0030S, #06FFS, #0031S, #0032S, #0033S, #0034S,
          #0035S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #0CCCS, #06FFS, #0208S);
CLOSE BASE;
GLCBAL DATA SEGT3 BASE R7;
  ARRAY 1504 SHORT INTEGER C1STATELIST = (
                                    #06FFS, #02C9S, #06FFS, #020BS,
          #06FFS, #0241S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0023S, #03C4S,
          #06FFS, #06FFS, #022CS, #0029S, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #002AS, #06FFS, #06FFS, #06FFS, #06FFS,
          #012BS, #06FFS, #012CS, #06FFS, #024AS, #024CS, #002DS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #002ES, #002FS, #06FFS,
          #06FFS, #06FFS, #06FFS, #02CES, #02CFS, #02D0S, #0036S,
          #0037S, #0030S, #06FFS, #0031S, #0032S, #0033S, #0034S,
          #0035S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #0038S, #0139S, #06FFS, #013BS, #06FFS, #0139S, #06FFS,
          #013CS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #0214S, #06FFS, #0023S, #0304S, #06FFS,
          #06FFS, #022CS, #0029S, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #002AS, #06FFS, #06FFS, #06FFS, #06FFS, #012BS,
          #06FFS, #012CS, #06FFS, #024AS, #024CS, #002DS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #002ES, #002FS, #06FFS, #06FFS,
          #06FFS, #06FFS, #02CES, #02CFS, #02D0S, #0036S, #0037S,
          #0030S, #06FFS, #0031S, #0032S, #0033S, #0034S, #0035S,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #013FS,
          #0040S, #06FFS, #0041S, #06FFS, #0041S, #06FFS, #0043S,
          #06FFS, #0045S, #06FFS, #0045S, #06FFS, #0043S, #06FFS,
          #0045S, #06FFS, #0043S, #06FFS, #030CS, #06FFS, #0226S,
```

```
#06FFS, #004BS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #0214S, #06FFS, #0023S, #0304S,
#06FFS, #06FFS, #022CS, #0029S, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #002AS, #06FFS, #06FFS, #06FFS, #06FFS,
#012BS, #06FFS, #012CS, #06FFS, #024AS, #024CS, #002DS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #002ES, #002FS, #06FFS,
#06FFS, #06FFS, #06FFS, #02CES, #02CFS, #02D0S, #0036S,
#0037S, #0030S, #06FFS, #0031S, #0032S, #0033S, #0034S,
#0035S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#014DS, #004ES, #06FFS, #022AS, #004FS, #06FFS, #0051S,
#06FFS, #004BS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #026BS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0287S, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#0154S, #06FFS, #005BS, #0269S, #026AS, #005FS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0279S,
#005CS, #027DS, #0060S, #06FFS, #0281S, #005DS, #0285S,
#0061S, #005ES, #028AS, #0062S, #06FFS, #06FFS, #0153S,
#0292S, #0055S, #0297S, #0056S, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #026BS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #0287S, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #0154S, #06FFS, #005BS,
#0269S, #026AS, #005FS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #0279S, #005CS, #027DS, #0060S,
#06FFS, #0281S, #005DS, #0285S, #0061S, #005ES, #028AS,
#0062S, #06FFS, #06FFS, #0153S, #0292S, #0055S, #0297S,
#0056S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
```

```
#06FFS, #06FFS, #06FFS, #06FFS, #004BS, #C6FFS, #004BS,
#06FFS, #0064S, #06FFS, #02E4S, #02E5S, #02E6S, #02E7S,
#06FFS, #02E4S, #02E5S, #02E6S, #02E7S, #06FFS, #0067S,
#06FFS, #0068S, #06FFS, #004BS, #06FFS, #004BS, #06FFS,
#0069S, #06FFS, #006AS, #06FFS, #0051S, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
#06FFS, #0023S, #0304S, #06FFS, #06FFS, #022CS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#024AS, #024CS, #002DS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#002ES, #002FS, #06FFS, #06FFS, #06FFS, #06FFS, #02CES,
#02CFS, #02D0S, #0036S, #0037S, #003CS, #06FFS, #0031S,
#0032S, #0033S, #0034S, #0035S, #06FFS, #C6FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #016BS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#0023S, #0304S, #06FFS, #06FFS, #022CS, #0029S, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #002AS, #06FFS, #06FFS,
#0071S, #06FFS, #012BS, #06FFS, #012CS, #06FFS, #024AS,
#024CS, #002DS, #C6FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0269S, #026AS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #002ES,
#002FS, #06FFS, #06FFS, #06FFS, #06FFS, #02CES, #02CFS,
#02D0S, #0036S, #0037S, #003CS, #06FFS, #0031S, #0032S,
#0033S, #0034S, #0035S, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #0214S, #06FFS, #0023S, #C304S, #C6FFS,
#06FFS, #022CS, #0029S, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #002AS, #06FFS, #06FFS, #06FFS, #C6FFS, #012BS,
#06FFS, #012CS, #06FFS, #024AS, #024CS, #002DS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
```

```
#06FFS, #06FFS, #06FFS, #002ES, #002FS, #06FFS, #06FFS,
#06FFS, #06FFS, #02CES, #02CFS, #02D0S, #0036S, #0037S,
#0030S, #06FFS, #0031S, #0032S, #0033S, #0034S, #0035S,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0214S,
#06FFS, #0023S, #0304S, #06FFS, #06FFS, #022CS, #0029S,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #002AS, #06FFS,
#06FFS, #06FFS, #06FFS, #012BS, #06FFS, #012CS, #06FFS,
#024AS, #024CS, #002DS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#002ES, #002FS, #06FFS, #06FFS, #06FFS, #06FFS, #02CES,
#02CFS, #02D0S, #0036S, #0037S, #0030S, #06FFS, #0031S,
#0032S, #0033S, #0034S, #0035S, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #021AS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#026BS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#0287S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #0080S, #06FFS, #06FFS, #06FFS,
#06FFS, #0081S, #0154S, #06FFS, #005BS, #0269S, #026AS,
#005FS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#007ES, #0279S, #005CS, #027DS, #0060S, #06FFS, #0281S,
#005DS, #0285S, #0061S, #005ES, #028AS, #0062S, #06FFS,
#06FFS, #0153S, #0292S, #0055S, #0297S, #0056S, #06FFS,
#06FFS, #06FFS, #06FFS, #02B3S, #02B4S, #02B5S, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #026BS, #005FS, #027DS, #0060S, #0285S,
#0061S, #028AS, #0062S, #0297S, #0056S, #02B3S, #02B4S,
#02B5S, #06FFS, #004BS, #06FFS, #004CS, #06FFS, #004BS,
#06FFS, #0040S, #06FFS, #004BS, #06FFS, #0040S, #06FFS,
#0040S, #06FFS, #0040S, #06FFS, #004CS, #06FFS, #0188S,
#0089S, #008AS, #06FFS, #026BS, #005FS, #027DS, #0060S,
#0285S, #0061S, #028AS, #0062S, #0297S, #0056S, #02B3S,
#02B4S, #02B5S, #06FFS, #0228S, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#0023S, #0304S, #06FFS, #06FFS, #022CS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #024AS,
```

```
        #024CS, #002DS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #002ES,
        #002FS, #06FFS, #06FFS, #06FFS, #06FFS, #02CES, #02CFS,
        #02D0S, #0036S, #0037S, #0030S, #06FFS, #0031S, #0032S,
        #0033S, #0034S, #0035S, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #004BS, #02B3S, #02B4S, #02B5S, #030CS,
        #02E4S, #02E5S, #02E6S, #02E7S, #06FFS, #02D7S, #06FFS,
        #0092S, #0091S, #06FFS, #0093S, #06FFS, #028ES, #028FS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #026BS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #0287S, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0154S, #06FFS,
        #005BS, #0269S, #026AS, #005FS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #0279S, #005CS, #027DS,
        #0060S, #06FFS, #0281S, #005DS, #0285S, #0061S, #005ES,
        #028AS, #0062S, #06FFS, #06FFS, #06FFS, #0292S, #0055S,
        #0297S, #0056S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #026BS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #0287S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS);
CLCSE BASE;
GLOBAL DATA SEGT4 BASE R2;
  ARRAY 1272 SHORT INTEGER C2STATELIST = (
                        #06FFS, #0080S, #06FFS, #06FFS, #06FFS,
        #06FFS, #0081S, #0154S, #06FFS, #005BS, #0269S, #026AS,
        #005FS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #007ES, #0279S, #005CS, #027DS, #0060S, #06FFS, #0281S,
        #005DS, #0285S, #0061S, #005ES, #028AS, #0062S, #06FFS,
        #06FFS, #0153S, #0292S, #0055S, #0297S, #0056S, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
        #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
```

```
#06FFS, #06FFS, #0188S, #06FFS, #02B3S, #02B4S, #02B5S,
#06FFS, #029DS, #029ES, #0325S, #0326S, #06FFS, #029DS,
#029ES, #0325S, #0326S, #06FFS, #029DS, #029ES, #0325S,
#0326S, #06FFS, #029DS, #029ES, #0325S, #0326S, #06FFS,
#0188S, #06FFS, #0188S, #06FFS, #0188S, #06FFS, #0188S,
#06FFS, #02B3S, #02B4S, #02B5S, #06FFS, #02B3S, #02B4S,
#02B5S, #06FFS, #02B3S, #02B4S, #02B5S, #06FFS, #02B3S,
#02B4S, #02B5S, #06FFS, #028ES, #028FS, #06FFS, #02B3S,
#02B4S, #02B5S, #02E4S, #02E5S, #02E6S, #02E7S, #06FFS,
#02D9S, #06FFS, #02DBS, #06FFS, #0285S, #0061S, #06FFS,
#0285S, #0061S, #06FFS, #004BS, #02B3S, #02B4S, #02B5S,
#030CS, #02E4S, #02E5S, #02E6S, #02E7S, #06FFS, #02D6S,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #0023S, #0304S, #06FFS, #06FFS,
#022CS, #0029S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#002AS, #06FFS, #06FFS, #06FFS, #06FFS, #012BS, #06FFS,
#012CS, #06FFS, #024AS, #024CS, #002DS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #002ES, #002FS, #06FFS, #06FFS, #06FFS,
#06FFS, #02CES, #02CFS, #02D0S, #0036S, #0037S, #0030S,
#06FFS, #0031S, #0032S, #0033S, #0034S, #0035S, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#0023S, #0304S, #06FFS, #06FFS, #022CS, #0029S, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #002AS, #06FFS, #06FFS,
#06FFS, #06FFS, #012BS, #06FFS, #012CS, #06FFS, #024AS,
#024CS, #002DS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #002ES,
#002FS, #06FFS, #06FFS, #06FFS, #06FFS, #02CES, #02CFS,
#02D0S, #0036S, #0037S, #0030S, #06FFS, #0031S, #0032S,
#0033S, #0034S, #0035S, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #0071S, #0269S, #026AS, #06FFS, #0240S,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #0023S, #0304S, #06FFS, #06FFS,
#022CS, #0029S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
```

```
#002AS, #06FFS, #06FFS, #0071S, #06FFS, #012BS, #06FFS,
#012CS, #06FFS, #024AS, #024CS, #002DS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #0269S, #026AS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #002ES, #002FS, #06FFS, #06FFS, #06FFS,
#06FFS, #02CES, #02CFS, #02D0S, #0036S, #0037S, #0030S,
#06FFS, #0031S, #0032S, #0033S, #0034S, #0035S, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #023DS, #06FFS,
#023ES, #06FFS, #013CS, #06FFS, #0249S, #06FFS, #0258S,
#0259S, #06FFS, #026FS, #0270S, #06FFS, #00AES, #06FFS,
#029DS, #029ES, #0325S, #0326S, #06FFS, #028ES, #028FS,
#06FFS, #025CS, #025DS, #025ES, #06FFS, #0273S, #0274S,
#0275S, #0276S, #06FFS, #029DS, #029ES, #0325S, #0326S,
#06FFS, #00B1S, #06FFS, #029DS, #029ES, #0325S, #0326S,
#06FFS, #00B2S, #0279S, #005CS, #027DS, #0060S, #06FFS,
#029DS, #029ES, #0325S, #0326S, #06FFS, #026BS, #00B3S,
#005BS, #0269S, #026AS, #005FS, #06FFS, #026BS, #00B3S,
#005BS, #0269S, #026AS, #005FS, #06FFS, #00B4S, #06FFS,
#02B0S, #00B5S, #06FFS, #02BBS, #06FFS, #0329S, #0089S,
#008AS, #06FFS, #032BS, #008AS, #06FFS, #00BBS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #02C5S, #026BS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #0287S, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0080S, #06FFS,
#06FFS, #06FFS, #06FFS, #0081S, #0154S, #06FFS, #005BS,
#0269S, #026AS, #005FS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #007ES, #0279S, #005CS, #027DS, #0060S,
#06FFS, #0281S, #005DS, #0285S, #0061S, #005ES, #028AS,
#0062S, #06FFS, #06FFS, #0153S, #0292S, #0055S, #0297S,
#0056S, #06FFS, #06FFS, #06FFS, #06FFS, #02B3S, #02B4S,
#02B5S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #02C9S, #004BS, #06FFS,
#02CCS, #06FFS, #00B4S, #06FFS, #02A9S, #00B5S, #06FFS,
#02B8S, #00B5S, #06FFS, #00BCS, #06FFS, #00B4S, #06FFS,
#0091S, #06FFS, #026BS, #008CS, #0081S, #00B3S, #005BS,
#0269S, #026AS, #005FS, #06FFS, #026BS, #008CS, #0081S,
#00B3S, #005BS, #0269S, #026AS, #005FS, #06FFS, #026BS,
#0080S, #0081S, #00B3S, #005BS, #0269S, #026AS, #005FS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
```

```
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #026BS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #0287S, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0154S, #06FFS,
#005BS, #0269S, #026AS, #005FS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #0279S, #005CS, #027DS,
#0060S, #06FFS, #0281S, #005DS, #0285S, #0061S, #005ES,
#028AS, #0062S, #06FFS, #06FFS, #0153S, #0292S, #0055S,
#0297S, #0056S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0293S, #028ES,
#028FS, #06FFS, #032CS, #0258S, #0259S, #06FFS, #027BS,
#026FS, #0270S, #06FFS, #0283S, #00AES, #06FFS, #0295S,
#0329S, #06FFS, #0298S, #00B5S, #06FFS, #026BS, #00B3S,
#005BS, #0269S, #026AS, #005FS, #06FFS, #02A1S, #06FFS,
#02A2S, #06FFS, #00B2S, #0279S, #005CS, #027DS, #0060S,
#06FFS, #00BES, #0281S, #005DS, #0285S, #0061S, #06FFS,
#0287S, #005ES, #028AS, #0062S, #06FFS, #0268S, #0329S,
#06FFS, #027CS, #0329S, #06FFS, #0284S, #0329S, #06FFS,
#0289S, #0329S, #06FFS, #026CS, #00B5S, #06FFS, #027ES,
#00B5S, #06FFS, #0286S, #00B5S, #06FFS, #028BS, #00B5S,
#06FFS, #02DCS, #06FFS, #02DDS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #0214S, #06FFS,
#0023S, #0304S, #06FFS, #06FFS, #022CS, #0029S, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #002AS, #06FFS, #06FFS,
#06FFS, #06FFS, #012BS, #06FFS, #012CS, #06FFS, #024AS,
#024CS, #002DS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #002ES,
#002FS, #06FFS, #06FFS, #06FFS, #06FFS, #02CES, #02CFS,
#02D0S, #0036S, #0037S, #0030S, #06FFS, #0031S, #0032S,
#0033S, #0034S, #0035S, #06FFS, #06FFS, #06FFS, #06FFS,
#06FFS, #06FFS, #026BS, #0080S, #0081S, #00B3S, #005BS,
#0269S, #026AS, #005FS, #06FFS, #00B2S, #007ES, #0279S,
#005CS, #027DS, #0060S, #06FFS, #00BES, #0281S, #005DS,
#0285S, #0061S, #06FFS, #026BS, #008CS, #0081S, #00B3S,
#005BS, #0269S, #026AS, #005FS, #06FFS, #026BS, #0080S,
#0081S, #00B3S, #005BS, #0269S, #026AS, #005FS, #06FFS,
#026BS, #0080S, #0081S, #00B3S, #005BS, #0269S, #026AS,
#005FS, #06FFS, #00B2S, #007ES, #0279S, #005CS, #027DS,
```

```
          #0060S, #06FFS, #026BS, #0080S, #0081S, #C0B3S, #0C5BS,
          #0269S, #026AS, #005FS, #06FFS, #026ES, #0080S, #0081S,
          #00B3S, #005BS, #C269S, #026AS, #005FS, #06FFS, #026BS,
          #0080S, #0081S, #00B3S, #005BS, #0269S, #026AS, #005FS,
          #06FFS, #032BS, #CC8AS, #06FFS, #00BES, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
          #06FFS, #026BS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #0287S, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #008CS, #C6FFS, #06FFS,
          #06FFS, #06FFS, #0081S, #0154S, #06FFS, #005BS, #0269S,
          #026AS, #005FS, #C6FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #007ES, #0279S, #0C5CS, #027DS, #C060S, #06FFS,
          #0281S, #005DS, #0285S, #0061S, #005ES, #028AS, #0C62S,
          #06FFS, #06FFS, #0153S, #0292S, #0055S, #0297S, #0056S,
          #06FFS, #06FFS, #06FFS, #06FFS, #02B3S, #C2B4S, #02B5S,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #06FFS, #06FFS, #C6FFS, #06FFS,
          #06FFS, #06FFS, #06FFS, #00EBS, #06FFS, #026BS, #005FS,
          #027DS, #0060S, #0285S, #0C61S, #028AS, #0062S, #0297S,
          #0056S, #02B3S, #02B4S, #02B5S, #06FFS, #02CDS, #06FFS,
          #026BS, #0080S, #0081S, #00B3S, #005BS, #0269S, #026AS,
          #005FS, #06FFS, #00C4S, #06FFS, #00BES, #0281S, #0C5DS,
          #0285S, #0061S, #06FFS, #025CS, #025DS, #025ES, #06FFS,
          #0273S, #0274S, #0275S, #0276S, #06FFS, #00B1S, #06FFS,
          #00BBS, #06FFS, #02B9S, #06FFS, #0269S, #C26AS, #06FFS);

COMMENT  THE DPDA HAS 0232 REDUCE STATES;

ARRAY 0233 BYTE NUMTOPCP = (#00X, #02X, #02X, #01X, #00X, #C0X, #C2X,
          #02X, #02X, #02X, #01X, #02X, #01X, #CCX, #00X, #01X, #00X,
          #01X, #01X, #00X, #00X, #00X, #01X, #C2X, #01X, #01X, #C2X,
          #02X, #01X, #01X, #01X, #01X, #00X, #01X, #01X, #01X, #01X,
          #01X, #01X, #01X, #02X, #02X, #01X, #CCX, #00X, #01X, #02X,
          #01X, #02X, #02X, #01X, #01X, #01X, #01X, #01X, #00X, #C1X,
          #00X, #01X, #C1X, #01X, #01X, #01X, #C1X, #01X, #01X, #01X,
          #01X, #01X, #C1X, #01X, #01X, #03X, #C1X, #00X, #01X, #CCX,
          #01X, #01X, #01X, #00X, #00X, #00X, #00X, #00X, #00X, #CCX,
          #02X, #00X, #C0X, #00X, #02X, #00X, #0CX, #00X, #00X, #02X,
          #00X, #01X, #01X, #00X, #00X, #02X, #C4X, #02X, #0CX, #CCX,
          #00X, #02X, #00X, #02X, #00X, #00X, #0CX, #02X, #00X, #00X,
          #0CX, #00X, #00X, #01X, #00X, #C0X, #C2X, #02X, #00X, #02X,
          #02X, #00X, #00X, #00X, #02X, #02X, #CCX, #02X, #00X, #0CX,
          #02X, #00X, #02X, #00X, #02X, #00X, #CCX, #00X, #01X, #0CX,
          #02X, #00X, #02X, #00X, #00X, #02X, #C2X, #02X, #02X, #02X,
          #00X, #00X, #00X, #00X, #01X, #01X, #01X, #01X, #01X, #C1X,
```

```
        #01X, #C1X, #C2X, #01X, #01X, #01X, #C1X, #01X, #01X, #02X,
        #02X, #02X, #00X, #00X, #00X, #01X, #C2X, #01X, #03X, #C1X,
        #02X, #00X, #02X, #01X, #01X, #01X, #00X, #00X, #00X, #01X,
        #01X, #01X, #02X, #01X, #01X, #01X, #C1X, #01X, #02X, #00X,
        #00X, #00X, #02X, #02X, #00X, #00X, #CCX, #02X, #02X, #C1X,
        #02X, #01X, #02X, #03X, #03X, #01X, #C1X, #01X, #01X, #01X,
        #01X, #00X, #00X, #00X, #0CX, #02X);

ARRAY 0083 SHORT INTEGER REDUCESUCC = (#06FFS, #0002S, #0003S,
        #0003S, #0300S, #0300S, #03C0S, #03CCS, #C300S, #0C06S,
        #0006S, #0007S, #0007S, #0301S, #0301S, #0501S, #0501S,
        #050CS, #0503S, #0503S, #0114S, #05C4S, #0504S, #0504S,
        #0504S, #0504S, #0505S, #0505S, #05C5S, #C5C5S, #0505S,
        #0505S, #0505S, #0505S, #0505S, #0505S, #0505S, #0505S,
        #05C5S, #0505S, #C505S, #05C5S, #05C5S, #0505S, #0505S,
        #0015S, #0015S, #050ES, #050ES, #005CS, #0090S, #050FS,
        #0C16S, #003AS, #003AS, #0312S, #0312S, #0313S, #0313S,
        #0512S, #0512S, #016FS, #016FS, #0217S, #C511S, #0502S,
        #0017S, #0506S, #0507S, #0219S, #0219S, #0515S, #0515S,
        #0516S, #0119S, #C119S, #0126S, #0126S, #001AS, #001AS,
        #0508S, #0508S, #0508S);
CLCSE BASE;
GLCBAL DATA SEGT5 BASE R8;
  ARRAY 0150 SHORT INTEGER CONTREDUCESUCC = (
                                #0508S, #0508S, #0508S, #0518S,
        #0518S, #00ACS, #00ACS, #051DS, #051CS, #C0AFS, #00AFS,
        #00AFS, #051ES, #051ES, #0510S, #051CS, #0510S, #051FS,
        #051FS, #051FS, #C51FS, #051FS, #0513S, #C513S, #0520S,
        #0520S, #0519S, #0519S, #0CADS, #00ACS, #0523S, #0523S,
        #00B0S, #00B0S, #00B0S, #00E0S, #0271S, #C271S, #0524S,
        #0524S, #0524S, #0524S, #0525S, #0525S, #051AS, #051AS,
        #0526S, #0526S, #0526S, #0526S, #0527S, #C527S, #051BS,
        #051BS, #051BS, #0528S, #0528S, #0514S, #0514S, #0194S,
        #0194S, #0529S, #0529S, #052AS, #052AS, #C52AS, #052AS,
        #052AS, #052BS, #052BS, #0296S, #0296S, #C296S, #0296S,
        #052CS, #052CS, #C52CS, #052CS, #052CS, #052CS, #05CDS,
        #050DS, #050DS, #050DS, #05CDS, #05CDS, #C50DS, #0517S,
        #0517S, #0517S, #0517S, #0517S, #0517S, #0517S, #0522S,
        #0522S, #051CS, #051CS, #051CS, #001BS, #001CS, #052DS,
        #052DS, #0509S, #050CS, #05CCS, #0084S, #0084S, #0084S,
        #0C84S, #0084S, #0084S, #0084S, #0521S, #0521S, #0521S,
        #0521S, #052FS, #052FS, #052FS, #052FS, #C530S, #0530S,
        #0024S, #0024S, #0024S, #0027S, #0027S, #0531S, #0531S,
        #0531S, #0028S, #0028S, #001DS, #001DS, #0303S, #0303S,
        #001FS, #0020S, #0021S, #0022S, #05CAS, #050AS, #050BS,
        #05CBS, #052ES, #052ES, #052ES, #052ES, #0700S);

CCMMENT  THE DPDA HAS 0C49 LOOK AHEAD STATES;
```

```
ARRAY 0049 BYTE LASYMNUM = (#6EX, #71X, #73X, #79X, #79X, #82X, #79X,
         #79X, #79X, #79X, #79X, #79X, #90X, #93X, #7FX, #80X, #88X,
         #8AX, #7DX, #7DX, #78X, #84X, #84X, #84X, #84X, #84X, #A5X,
         #A6X, #A7X, #AAX, #B4X, #9AX, #A4X, #C3X, #C3X, #C3X, #92X,
         #C0X, #C0X, #C3X, #C3X, #B0X, #C3X, #C4X, #AEX, #A5X, #A6X,
         #AAX, #C3X);

ARRAY 0049 SHORT INTEGER SUCCSTATE = (#0203S, #02CAS, #02CCS, #C220S,
         #022BS, #024BS, #021ES, #021FS, #0221S, #0222S, #0223S,
         #0224S, #02BCS, #024DS, #0242S, #0244S, #C2D8S, #02DAS,
         #0235S, #0236S, #0215S, #0250S, #0251S, #0252S, #0253S,
         #0254S, #0256S, #026DS, #028CS, #025AS, #0277S, #0261S,
         #02AFS, #02C1S, #02C2S, #02C3S, #02AES, #029FS, #02A0S,
         #02BES, #02C0S, #02C6S, #02EFS, #02CAS, #0266S, #0257S,
         #026ES, #025BS, #02BDS);

ARRAY 0049 SHCRT INTEGER FAILSTATE = (#0005S, #000DS, #000FS, #CC1ES,
         #0025S, #003ES, #0042S, #0044S, #0046S, #0047S, #0048S,
         #0049S, #004AS, #004CS, #0052S, #0063S, #0065S, #0066S,
         #016CS, #006DS, #0073S, #0074S, #0075S, #0076S, #0077S,
         #0078S, #0079S, #007AS, #007BS, #007CS, #007DS, #007FS,
         #0082S, #0085S, #0086S, #0087S, #008ES, #009CS, #009DS,
         #00B6S, #00B7S, #01B8S, #0CB9S, #00BAS, #00BDS, #00BFS,
         #00C0S, #00C1S, #00C2S);

ARRAY C686 BYTE LATABLE = (
         COMMENT  <SYS DECL HEAD>;
         #07X, #C0X, #00X, #00X, #00X, #00X, #CEX, #00X, #00X, #CCX,
         #0CX, #00X, #C0X, #00X,

         COMMENT  <SYSPROC DECL HEAD>;
         #0CX, #38X, #00X, #00X, #00X, #00X, #0CX, #00X, #00X, #00X,
         #00X, #00X, #00X, #00X,

         COMMENT  <SYSPROCREN DECL HEAD>;
         #00X, #18X, #00X, #00X, #00X, #00X, #CCX, #00X, #00X, #00X,
         #00X, #00X, #00X, #00X,

         COMMENT  <SIMPLE STM>;
         #08X, #00X, #00X, #01X, #00X, #00X, #CCX, #00X, #00X, #0CX,
         #00X, #00X, #00X, #00X,

         COMMENT  <SIMPLE STM>;
         #08X, #00X, #00X, #01X, #00X, #00X, #00X, #00X, #00X, #00X,
         #00X, #00X, #00X, #00X,

         COMMENT  <BEGIN HEAD>;
```

```
#00X, #03X, #CCX, #10X, #AEX, #00X, #CCX, #0CX, #C0X, #0CX,
#06X, #1FX, #BEX, #00X,

COMMENT  <SIMPLE STM>;
#08X, #00X, #00X, #01X, #00X, #00X, #CCX, #00X, #00X, #0CX,
#0CX, #00X, #00X, #00X,

COMMENT  <SIMPLE STM>;
#08X, #00X, #C0X, #01X, #00X, #00X, #CCX, #00X, #00X, #00X,
#00X, #00X, #00X, #00X,

COMMENT  <SIMPLE STM>;
#08X, #00X, #00X, #01X, #00X, #00X, #CCX, #00X, #00X, #0CX,
#00X, #00X, #00X, #00X,

COMMENT  <SIMPLE STM>;
#08X, #00X, #00X, #01X, #00X, #00X, #0CX, #00X, #0CX, #CCX,
#0CX, #00X, #C0X, #00X,

COMMENT  <SIMPLE STM>;
#08X, #00X, #C0X, #01X, #00X, #00X, #0CX, #00X, #00X, #00X,
#00X, #00X, #C0X, #00X,

COMMENT  <SIMPLE STM>;
#08X, #00X, #00X, #01X, #00X, #00X, #CCX, #00X, #00X, #CCX,
#00X, #00X, #00X, #00X,

COMMENT  <PROCEDURE STM>;
#08X, #00X, #12X, #01X, #00X, #00X, #00X, #00X, #0CX, #CCX,
#0CX, #00X, #00X, #00X,

COMMENT  <COBEGIN HEAD>;
#00X, #02X, #DCX, #10X, #AEX, #00X, #CCX, #00X, #00X, #0CX,
#06X, #1FX, #BEX, #00X,

COMMENT  <WHILE CLAUSE>;
#00X, #00X, #00X, #00X, #4CX, #00X, #CCX, #0CX, #00X, #CCX,
#00X, #00X, #00X, #00X,

COMMENT  <IF CLAUSE>;
#00X, #00X, #00X, #00X, #10X, #00X, #0CX, #00X, #00X, #CCX,
#00X, #00X, #C0X, #00X,

COMMENT  <INPUT CLAUSE>;
#00X, #00X, #00X, #00X, #00X, #00X, #CCX, #00X, #00X, #00X,
#00X, #00X, #01X, #00X,

COMMENT  <OUTPUT CLAUSE>;
```

```
#08X, #00X, #00X, #01X, #00X, #00X, #CCX, #00X, #00X, #0CX,
#0CX, #00X, #00X, #80X,

CCMMENT  <CASE LIST>;
#0CX, #00X, #00X, #01X, #00X, #00X, #0CX, #00X, #00X, #CCX,
#00X, #00X, #C0X, #00X,

CCMMENT  <CASE LIST>;
#00X, #00X, #00X, #01X, #0CX, #00X, #0CX, #00X, #00X, #00X,
#00X, #00X, #00X, #00X,

COMMENT  <BASIC STM>;
#08X, #00X, #00X, #00X, #00X, #00X, #CCX, #0CX, #00X, #CCX,
#00X, #00X, #C0X, #00X,

COMMENT  <EXPR>;
#08X, #00X, #12X, #01X, #00X, #00X, #0CX, #00X, #0CX, #0CX,
#0CX, #60X, #00X, #00X,

COMMENT  <EXPR>;
#08X, #00X, #12X, #01X, #00X, #00X, #00X, #0CX, #00X, #00X,
#00X, #60X, #00X, #00X,

COMMENT  <EXPR>;
#08X, #00X, #12X, #01X, #00X, #00X, #CCX, #0CX, #0CX, #CCX,
#0CX, #60X, #00X, #00X,

COMMENT  <EXPR>;
#08X, #00X, #12X, #01X, #00X, #00X, #0CX, #00X, #0CX, #CCX,
#0CX, #60X, #00X, #00X,

COMMENT  <EXPR>;
#08X, #00X, #12X, #01X, #0CX, #00X, #0CX, #00X, #00X, #00X,
#00X, #60X, #00X, #00X,

COMMENT  <REAL EXPR>;
#08X, #00X, #12X, #01X, #01X, #80X, #CCX, #00X, #0CX, #CCX,
#0CX, #60X, #00X, #00X,

CCMMENT  <BITS EXPR>;
#08X, #00X, #12X, #01X, #00X, #00X, #0CX, #00X, #0CX, #CCX,
#00X, #60X, #00X, #00X,

COMMENT  <CHAR EXPR>;
#08X, #00X, #12X, #01X, #0CX, #00X, #CCX, #01X, #00X, #00X,
#00X, #60X, #00X, #00X,

COMMENT  <REAL TERM>;
```

```
#08X, #00X, #12X, #01X, #01X, #F0X, #CCX, #0CX, #0CX, #CCX,
#0CX, #60X, #00X, #00X,

COMMENT  <BITS SECN>;
#08X, #00X, #12X, #01X, #00X, #00X, #0FX, #C0X, #00X, #00X,
#00X, #60X, #00X, #00X,

COMMENT  <SIGNED REAL PRI>;
#08X, #00X, #12X, #61X, #41X, #F8X, #CFX, #C0X, #0CX, #0CX,
#00X, #60X, #00X, #00X,

COMMENT  <DATA UNIT CLAUSE>;
#08X, #01X, #12X, #01X, #00X, #00X, #CCX, #00X, #0CX, #CCX,
#0CX, #20X, #00), #C0X,

CCMMENT  <ACTUAL PARAMETER LIST>;
#0CX, #00X, #10X, #00X, #00X, #00X, #0CX, #00X, #00X, #00X,
#0CX, #Q0X, #C0X, #00X,

COMMENT  <ACTUAL PARAMETER LIST>;
#00X, #C0X, #10X, #00X, #00X, #00X, #CCX, #0CX, #00X, #0CX,
#00X, #00X, #00X, #00X,

COMMENT  <ACTUAL PARAMETER LIST>;
#00X, #00X, #10X, #00X, #00X, #00X, #CCX, #00X, #0CX, #CCX,
#0CX, #00X, #00X, #00X,

CCMMENT  <DATA UNIT>;
#08X, #Q1X, #12X, #09X, #00X, #00X, #CCX, #00X, #0CX, #0CX,
#01X, #20X, #C1X, #80X,

COMMENT  <REL CPER>;
#00X, #0CX, #C1X, #02X, #0CX, #02X, #FCX, #1EX, #FEX, #0CX,
#00X, #00X, #00X, #00X,

COMMENT  <REL OPER>;
#00X, #C0X, #01X, #02X, #00X, #02X, #FCX, #1EX, #FEX, #CCX,
#0CX, #00X, #C0X, #00X,

COMMENT  <ACTUAL PARAMETER LIST>;
#0CX, #00X, #10X, #00X, #0CX, #00X, #0CX, #00X, #C0X, #0CX,
#0CX, #QCX, #CCX, #00X,

CCMMENT  <ACTUAL PARAMETER LIST>;
#00X, #00X, #10X, #00X, #0CX, #00X, #CCX, #00X, #00X, #00X,
#00X, #00X, #00X, #00X,

COMMENT  <INPLT PARAMS>;
```

```
        #00X, #00X, #12X, #00X, #0CX, #00X, #CCX, #00X, #00X, #0CX,
        #00X, #60X, #00X, #00X,

        COMMENT  <ACTUAL PARAMETER LIST>;
        #00X, #00X, #10X, #00X, #00X, #00X, #00X, #00X, #00X, #CCX,
        #00X, #00X, #0CX, #00X,

        COMMENT  <OUTFUT PARAMS>;
        #0CX, #00X, #12X, #00X, #0CX, #00X, #0CX, #00X, #00X, #00X,
        #00X, #20X, #0CX, #00X,

        COMMENT  <REAL PRI>;
        #08X, #00X, #12X, #61X, #51X, #F8X, #CFX, #C0X, #01X, #83X,
        #C0X, #60X, #00X, #00X,

        COMMENT  <REAL EXPR>;
        #08X, #00X, #12X, #01X, #01X, #80X, #CCX, #00X, #00X, #0CX,
        #0CX, #60X, #C0X, #00X,

        COMMENT  <BITS EXPR>;
        #08X, #00X, #12X, #01X, #0CX, #C0X, #0CX, #00X, #00X, #0CX,
        #00X, #60X, #00X, #00X,

        COMMENT  <REAL TERM>;
        #08X, #00X, #12X, #01X, #01X, #F0X, #CCX, #00X, #0CX, #0CX,
        #00X, #60X, #00X, #00X,

        COMMENT  <ACTUAL PARAMETER LIST>;
        #00X, #00X, #10X, #00X, #00X, #C0X, #CCX, #00X, #0CX, #CCX,
        #0CX, #00X, #C0X, #00X);


COMMENT  THE DPDA HAS 0050 LOOKBACK STATES;

ARRAY 0050 BYTE LBSTART = (#00X, #03X, #05X, #C8X, #0CX, #12X, #17X,
        #19X, #1BX, #1EX, #20X, #23X, #26X, #28X, #33X, #35X, #37X,
        #40X, #43X, #46X, #4BX, #4FX, #51X, #53X, #5AX, #5DX, #6CX,
        #63X, #68X, #6FX, #71X, #73X, #7FX, #83X, #89X, #92X, #94X,
        #9DX, #A1X, #A9X, #AFX, #B3X, #B5X, #B7X, #BBX, #C2X, #C4X,
        #C7X, #C9X, #CDX);

ARRAY 0050 BYTE LBNUM = (#02X, #01X, #02X, #C3X, #05X, #04X, #01X,
        #01X, #02X, #01X, #02X, #02X, #01X, #CAX, #01X, #01X, #C8X,
        #02X, #02X, #C4X, #03X, #01X, #01X, #06X, #02X, #02X, #02X,
        #04X, #06X, #01X, #01X, #0BX, #03X, #C5X, #08X, #01X, #08X,
        #03X, #07X, #05X, #03X, #01X, #01X, #C3X, #06X, #01X, #C2X,
        #01X, #03X, #C1X);
```

```
ARRAY 0207 SHORT INTEGER LBSTATE = (#0005S, #000DS, #0000S, #C0C6S,
           #0000S, #010ES, #0119S, #0000S, #013CS, #013DS, #01ABS,
           #0000S, #0114S, #016BS, #013BS, #016CS, #016FS, #00C0S,
           #0139S, #014DS, #013CS, #01ABS, #0000S, #0015S, #0000S,
           #013CS, #0000S, #013FS, #0188S, #00CCS, #001BS, #0000S,
           #001DS, #0020S, #0000S, #001ES, #001FS, #0000S, #0023S,
           #0C00S, #0025S, #002AS, #002CS, #002ES, #0034S, #0035S,
           #0041S, #0043S, #0045S, #0089S, #00CCS, #0029S, #00C0S,
           #0050S, #0000S, #0091S, #0092S, #0093S, #00B0S, #00B1S,
           #00B4S, #00B5S, #00BCS, #0C00S, #013ES, #016CS, #0000S,
           #013BS, #006DS, #0000S, #0CC4S, #013BS, #006DS, #016FS,
           #0000S, #012BS, #012CS, #0154S, #0000S, #0018S, #0000S,
           #013CS, #0000S, #001AS, #0086S, #00E6S, #0042S, #0047S,
           #0049S, #0000S, #0154S, #00B3S, #00CCS, #0154S, #0CB2S,
           #0000S, #0154S, #00BES, #0000S, #00ACS, #013FS, #0188S,
           #01B8S, #0000S, #004BS, #0C40S, #00BAS, #013FS, #0188S,
           #01B8S, #0000S, #00ACS, #00C0S, #00AFS, #0000S, #0C80S,
           #0081S, #009BS, #012BS, #012CS, #0153S, #0194S, #013FS,
           #0154S, #0188S, #01B8S, #00C0S, #004BS, #0040S, #0CBAS,
           #0000S, #004AS, #0055S, #005BS, #005CS, #005DS, #0000S,
           #004BS, #0056S, #005FS, #0060S, #0061S, #0062S, #0040S,
           #00BAS, #0000S, #00ADS, #0000S, #007ES, #009ES, #0CADS,
           #00B2S, #012BS, #012CS, #0153S, #0194S, #0000S, #004BS,
           #0040S, #00BAS, #0000S, #0C9FS, #00AES, #00BES, #012BS,
           #012CS, #0153S, #0194S, #0000S, #004BS, #0067S, #0068S,
           #0C40S, #00BAS, #0000S, #004BS, #004CS, #00BAS, #0000S,
           #0194S, #0000S, #0153S, #000CS, #004BS, #0040S, #00BAS,
           #0000S, #0057S, #007FS, #0058S, #007CS, #0059S, #007BS,
           #0000S, #0064S, #0000S, #0030S, #0031S, #0000S, #004AS,
           #0000S, #004AS, #0085S, #0086S, #000CS, #004ES, #0C00S);

ARRAY 0207 SHORT INTEGER RESUMESTATE = (#0201S, #0206S, #0207S,
           #020DS, #0302S, #020FS, #0305S, #03CDS, #0245S, #0246S,
           #0246S, #0013S, #0212S, #023FS, #006ES, #006ES, #006ES,
           #0213S, #0243S, #0229S, #0314S, #0314S, #C215S, #0216S,
           #0218S, #0072S, #0018S, #021BS, #02C4S, #02C7S, #021CS,
           #021DS, #0306S, #C309S, #030BS, #03C7S, #0308S, #030AS,
           #0225S, #02D3S, #0227S, #0234S, #024ES, #02B6S, #02DES,
           #02DFS, #02BAS, #02E0S, #02E2S, #02C8S, #02D5S, #022DS,
           #022ES, #022FS, #0230S, #0233S, #0232S, #0231S, #0272S,
           #0260S, #02B1S, #02B2S, #00C3S, #025FS, #0237S, #0238S,
           #023BS, #0239S, #023AS, #023CS, #0267S, #0070S, #0070S,
           #0070S, #0264S, #030ES, #03CFS, #0095S, #0319S, #013DS,
           #01ABS, #0247S, #0248S, #024FS, #02CBS, #02CBS, #02E1S,
           #02E1S, #02E1S, #02E3S, #0C96S, #0096S, #0315S, #0097S,
           #0097S, #0316S, #0098S, #0098S, #0317S, #029CS, #0318S,
           #0318S, #0318S, #005AS, #0324S, #0320S, #0320S, #0255S,
```

```
          #0255S, #0255S, #008FS, #032DS, #031AS, #032FS, #031CS,
          #0262S, #0263S, #0299S, #0057S, #0057S, #0057S, #0057S,
          #031FS, #031FS, #031FS, #031FS, #0261S, #02A3S, #02AAS,
          #02AAS, #0265S, #0321S, #0099S, #00A1S, #00A2S, #00A3S,
          #00A4S, #008CS, #009AS, #00A5S, #00A6S, #00A7S, #00A8S,
          #0083S, #0083S, #008DS, #032ES, #031BS, #0278S, #029AS,
          #0277S, #0277S, #0058S, #0058S, #0058S, #0058S, #031ES,
          #02A4S, #02ABS, #02ABS, #027AS, #029BS, #027FS, #0280S,
          #0059S, #0059S, #0059S, #0059S, #031CS, #02A5S, #00A9S,
          #00AAS, #02ACS, #02ACS, #0282S, #02A7S, #02ADS, #02ADS,
          #0288S, #028DS, #028CS, #0291S, #0290S, #02A6S, #02AES,
          #02AES, #0294S, #009BS, #009BS, #009ES, #009ES, #009FS,
          #009FS, #00A0S, #02B7S, #02D4S, #031CS, #0311S, #008ES,
          #0322S, #0327S, #0323S, #0328S, #032AS, #0330S, #02D2S,
          #02D1S);

COMMENT  THE SYMBOLS ACCESSING THE STATES;

ARRAY 0197 BYTE SYMBEFOREREAD = (#00X, #01X, #6DX, #6EX, #C2X, #6FX,
          #71X, #73X, #07X, #08X, #09X, #03X, #0DX, #72X, #0BX, #74X,
          #03X, #03X, #03X, #76X, #77X, #7AX, #7CX, #7FX, #80X, #82X,
          #83X, #85X, #87X, #88X, #8AX, #8CX, #8DX, #8EX, #8FX, #1CX,
          #91X, #11X, #93X, #94X, #95X, #15X, #1BX, #20X, #22X, #26X,
          #55X, #56X, #60X, #62X, #63X, #64X, #65X, #66X, #5EX, #5FX,
          #16X, #21X, #7DX, #1CX, #23X, #A2X, #75X, #0FX, #16X, #57X,
          #89X, #67X, #8BX, #68X, #8BX, #89X, #8BX, #89X, #58X, #1DX,
          #75X, #13X, #16X, #16X, #97X, #17X, #A1X, #49X, #2EX, #4BX,
          #4DX, #AEX, #B6X, #B8X, #A8X, #30X, #3CX, #41X, #44X, #33X,
          #3EX, #43X, #46X, #A1X, #1DX, #C2X, #C2X, #1DX, #1DX, #2EX,
          #2EX, #1FX, #9BX, #9CX, #78X, #9FX, #A0X, #1EX, #80X, #79X,
          #A5X, #A6X, #A7X, #A8X, #A1X, #AAX, #B2X, #B8X, #ACX, #B6X,
          #3AX, #AEX, #28X, #2DX, #A9X, #B1X, #C3X, #B0X, #C4X, #C5X,
          #2EX, #59X, #5AX, #A9X, #B1X, #B1X, #C2X, #A9X, #99X, #1AX,
          #19X, #18X, #BCX, #A1X, #A5X, #A6X, #A7X, #B0X, #B1X, #C0X,
          #50X, #51X, #C0X, #C0X, #C0X, #B0X, #BCX, #B0X, #B0X, #B1X,
          #B1X, #B1X, #B1X, #B9X, #B9X, #A2X, #ABX, #B3X, #3FX, #ADX,
          #B5X, #2CX, #2EX, #2EX, #2EX, #16X, #C4X, #C5X, #16X, #C5X,
          #16X, #16X, #2EX, #13X, #2EX, #AAX, #B2X, #ACX, #C5X, #9AX,
          #2FX);

ARRAY 0049 BYTE SYMBEFORELA = (#6FX, #72X, #74X, #8AX, #11X, #75X,
          #89X, #8BX, #8BX, #89X, #8BX, #89X, #58X, #75X, #A1X, #A1X,
          #C2X, #C2X, #9BX, #9CX, #79X, #A5X, #A6X, #A7X, #A8X, #A1X,
          #AAX, #B2X, #B8X, #ACX, #B6X, #AEX, #A9X, #B0X, #C4X, #C5X,
          #A9X, #50X, #51X, #C4X, #C5X, #16X, #C5X, #16X, #13X, #AAX,
          #B2X, #ACX, #C5X);

CLOSE BASE;
```

```
ARRAY 18 INTEGER SEGTABLE = (
        #00000000,
        #00000794,
        #00000AB0,
        #00000C3A,
        #00000CFF,
        #0000186A,
        #00002F40,
        #0000302A,
        #000031FC,
        #0000322E,
        #00003290,
        #000032F2,
        #000035A0,
        #000035D2,
        #00003604,
        #000037A2,
        #00003940,
        #00003A05);
COMMENT END OF MAIN GRAMMAR PARSING TABLES. PARSING TABLES FCR SECOND
        GRAMMAR FOLLOWS;

INTEGER BNUMTERMINALS = 0047;
INTEGER BNUMNTS = 0037;
INTEGER BNUMSYMS = 0085;

 GLCBAL DATA BSEGTABLE BASE R11;

ARRAY 0794 BYTE BVSTRING = (#C5X, #D9X, #D9X, #D6X, #D9X, #E2X, #E8X,
        #D4X, #40X, #6EX, #6DX, #4FX, #6DX, #4CX, #C4X, #C5X, #C3X,
        #D3X, #6EX, #4CX, #E2X, #E4X, #C2X, #60X, #D9X, #D6X, #E4X,
        #E3X, #C9X, #D5X, #C5X, #40X, #C4X, #C5X, #C3X, #D3X, #6EX,
        #4CX, #C4X, #C1X, #E3X, #C1X, #40X, #C4X, #C5X, #C3X, #D3X,
        #6EX, #40X, #C4X, #C1X, #E3X, #C1X, #40X, #C5X, #D3X, #C5X,
        #D4X, #6EX, #C5X, #D5X, #C4X, #6DX. #C4X, #C4X, #5BX, #4CX,
        #D7X, #D9X, #D6X. #C3X, #C5X, #C4X, #E4X, #D9X, #C5X, #40X,
        #C4X, #C9X, #C3X, #D3X, #6EX, #4CX, #C6X, #D7X, #C1X, #D9X,
        #D4X, #40X, #D3X, #C9X, #E2X, #E3X, #6EX, #5DX, #4CX, #C6X,
        #E4X, #D5X, #C3X, #E3X, #C9X, #D6X, #D5X, #40X, #C3X, #D3X,
        #C1X, #E4X, #E2X, #C5X, #6EX, #D7X, #D9X, #D6X, #C3X, #C5X,
        #C4X, #E4X, #D9X, #C5X, #4CX, #C9X, #C4X, #6EX, #4CX, #C9X,
        #D5X, #40X, #D7X, #C1X, #D9X, #D4X, #E2X, #6EX, #4CX, #D6X,
        #E4X, #E3X, #40X, #D7X, #C1X, #D9X, #D4X, #E2X, #6EX, #4CX,
        #D3X, #C1X, #C2X, #C5X, #D3X, #40X, #C7X, #C1X, #D9X, #D4X,
        #E2X, #6EX, #4DX, #4CX, #E5X, #C1X, #D9X, #6EX, #6BX, #4FX,
        #4FX, #4FX, #4CX, #C6X, #E4X, #D5X, #C3X, #E3X, #C9X, #D6X,
```

```
#D5X, #40X, #C8X, #C5X, #C1X, #C4X, #6EX, #C6X, #E4X, #D5X,
#C3X, #E3X, #C9X, #D6X, #D5X, #4CX, #C4X, #C1X, #E3X, #C1X,
#4CX, #E3X, #E8X, #D7X, #C5X, #6EX, #4CX, #D5X, #E4X, #D4X,
#40X, #E5X, #C1X, #D9X, #6EX, #4CX, #C2X, #C9X, #E3X, #E2X,
#40X, #E5X, #C1X, #D9X, #6EX, #4CX, #C3X, #C8X, #C1X, #D9X,
#40X, #E5X, #C1X, #D9X, #6EX, #4CX, #E2X, #E3X, #C1X, #E3X,
#40X, #E5X, #C1X, #D9X, #6EX, #4CX, #C2X, #D6X, #D6X, #D3X,
#40X, #E5X, #C1X, #D9X, #6EX, #4CX, #E3X, #C2X, #D3X, #4CX,
#D5X, #C1X, #D4X, #C5X, #6EX, #4CX, #C9X, #E3X, #C5X, #D4X,
#40X, #C1X, #4BX, #40X, #D5X, #C1X, #D4X, #C5X, #6EX, #4CX,
#E2X, #E4X, #C2X, #E3X, #C2X, #D3X, #40X, #D5X, #C1X, #D4X,
#C5X, #6EX, #E2X, #E8X, #E2X, #6DX, #C4X, #C4X, #D3X, #C6X,
#C3X, #6DX, #C4X, #C4X, #C1X, #E4X, #E3X, #D6X, #6DX, #C4X,
#C4X, #4CX, #E5X, #D9X, #C2X, #D3X, #40X, #C4X, #C5X, #C3X,
#D3X, #40X, #C8X, #C5X, #C1X, #C4X, #6EX, #4CX, #C9X, #E3X,
#C5X, #D4X, #6EX, #4CX, #C6X, #D6X, #D9X, #D4X, #C1X, #E3X,
#40X, #C4X, #C5X, #C3X, #D3X, #40X, #C8X, #C5X, #C1X, #C4X,
#6EX, #4CX, #C4X, #C5X, #E2X, #C3X, #D9X, #C7X, #D9X, #D6X,
#E4X, #D7X, #6EX, #4CX, #E3X, #C1X, #C7X, #40X, #C4X, #C5X,
#C3X, #D3X, #40X, #C8X, #C5X, #C1X, #C4X, #6EX, #7EX, #4CX,
#D9X, #C5X, #C1X, #D3X, #40X, #C3X, #D6X, #D5X, #E2X, #E3X,
#C1X, #D5X, #E3X, #6EX, #4CX, #E3X, #C1X, #C2X, #D3X, #C5X,
#40X, #C4X, #C5X, #C3X, #D3X, #40X, #C8X, #C5X, #C1X, #C4X,
#6EX, #4CX, #E3X, #C1X, #C2X, #D3X, #C5X, #40X, #C4X, #C5X,
#C3X, #D3X, #6EX, #C5X, #D5X, #C4X, #6DX, #E3X, #C1X, #C2X,
#D3X, #C5X, #4CX, #E5X, #D9X, #C2X, #D3X, #40X, #C8X, #C5X,
#C1X, #C4X, #6EX, #E5X, #D9X, #C2X, #D3X, #C9X, #D5X, #E3X,
#C5X, #C7X, #C5X, #D9X, #C6X, #C9X, #E7X, #C5X, #C4X, #C6X,
#D3X, #D6X, #C1X, #E3X, #C2X, #C9X, #E3X, #E2X, #C3X, #C8X,
#C1X, #D9X, #C2X, #D6X, #D6X, #D3X, #C5X, #C1X, #D5X, #4CX,
#E2X, #E3X, #C1X, #E3X, #E4X, #E2X, #4CX, #C3X, #D3X, #C1X,
#E4X, #E2X, #C5X, #6EX, #E2X, #E3X, #C1X, #E3X, #E4X, #E2X,
#4CX, #E2X, #E3X, #C1X, #E3X, #E4X, #E2X, #40X, #C3X, #D6X,
#D5X, #E2X, #E3X, #C1X, #D5X, #E3X, #6EX, #4CX, #C3X, #D6X,
#D5X, #E2X, #E3X, #C1X, #D5X, #E3X, #6EX, #4CX, #D5X, #E4X,
#D4X, #D6X, #D9X, #E3X, #C1X, #C7X, #6EX, #4CX, #C2X, #C9X,
#E3X, #E2X, #40X, #C3X, #D6X, #D5X, #E2X, #E3X, #C1X, #D5X,
#E3X, #6EX, #4CX, #C3X, #C8X, #C1X, #D9X, #40X, #C3X, #D6X,
#D5X, #E2X, #E3X, #C1X, #D5X, #E3X, #6EX, #4CX, #C2X, #D6X,
#D6X, #D3X, #40X, #C3X, #D6X, #D5X, #E2X, #E3X, #C1X, #D5X,
#E3X, #6EX, #E3X, #D9X, #E4X, #C5X, #C6X, #C1X, #D3X, #E2X,
#C5X, #4CX, #E3X, #C1X, #C7X, #6EX, #E3X, #C1X, #C7X, #C6X,
#D6X, #D9X, #D4X, #C1X, #E3X, #4CX, #C4X, #E2X, #C3X, #D9X,
#D3X, #C9X, #E2X, #E3X, #4CX, #C8X, #C4X, #6EX, #4CX, #C4X,
#E2X, #C3X, #D9X, #D3X, #C9X, #E2X, #E3X, #6EX, #4CX, #C4X,
#C5X, #E2X, #C3X, #D9X, #C9X, #D7X, #E3X, #D6X, #D9X, #6EX,
#4CX, #D9X, #C5X, #D7X, #E3X, #D5X, #4CX, #C7X, #D9X, #C6X,
#E4X, #D7X, #6EX, #4CX, #E2X, #D3X, #C1X, #E2X, #C8X, #4CX,
```

```
         #D3X, #C9X, #E2X, #E3X, #6EX, #61X, #4CX, #E3X, #C1X, #C2X,
         #D3X, #C5X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X, #6EX,
         #4CX, #E3X, #C1X, #C2X, #D3X, #C5X, #40X, #C8X, #C5X, #C1X,
         #C4X, #6EX, #C9X, #D5X, #C4X, #C9X, #D9X, #C3X, #E3X, #E3X,
         #C1X, #C2X, #D3X, #C5X, #4CX, #C6X, #C9X, #C5X, #D3X, #C4X,
         #40X, #C4X, #C5X, #C3X, #D3X, #6EX, #4CX, #C9X, #E3X, #C5X,
         #D4X, #6DX, #C1X, #D9X, #C5X, #C1X, #4CX, #C3X, #D3X, #C1X,
         #E4X, #E2X, #C5X, #6EX, #4CX, #E2X, #E4X, #C2X, #E3X, #C1X,
         #C2X, #D3X, #C5X, #40X, #C3X, #D3X, #C1X, #E4X, #E2X, #C5X,
         #6EX, #C6X, #C9X, #C5X, #D3X, #C4X, #4CX, #C6X, #C9X, #E3X,
         #C5X, #D4X, #6EX, #4CX, #D9X, #C5X, #D7X, #C5X, #E3X, #C9X,
         #E3X, #C9X, #D6X, #D5X, #40X, #D3X, #C9X, #E2X, #E3X, #6EX,
         #C9X, #E3X, #C5X, #D4X, #6DX, #C1X, #D9X, #C5X, #C1X, #E2X,
         #E4X, #C2X, #E3X, #C1X, #C2X, #D3X, #C5X);

ARRAY 0085 INTEGER BLOCLENGTH=  (#0000000A, #CCCC0283, #0000CEC6,
         #00001041, #00001781, #00001C09, #00001E44, #000027C1,
         #00002941, #00002981, #000C29C2, #0CCC2EC8, #CCC032C9,
         #0000350A, #0000378A, #00003A0A, #0CCC3C8A, #CCC03F0A,
         #0CC0418E, #0CC0450D, #00004846, #00CC49C6, #00004B47,
         #00005DC1, #00005ECF, #000069C9, #0CCC6E04, #0CC06F07,
         #000070C5, #00C07205, #00007344, #00007444, #00007547,
         #00C07AC6, #00007C51, #000C858F, #0CCC894F, #000090C4,
         #000091C5, #00009305, #00009443, #0CCC9506, #CCC0A581,
         #0000AC47, #0000AE05, #0000BE05, #0000C249, #0000C488,
         #0CCC0346, #000004D2, #C000C94B, #0CCCCCCB, #CC001C90,
         #0000148C, #000017D1, #00001F4A, #0CCC21CB, #0CC0248D,
         #0CC02805, #00002A4F, #0C003C0B, #00C04D10, #00005106,
         #00005292, #00C057CC, #0000SACF, #0CCC61D1, #0CC0660C,
         #00006B4B, #0CC0770F, #0000808A, #00C0830A, #0CC08D0F,
         #0C00968D, #000099CA, #00009C4C, #00C09F4D, #0000A28C,
         #0000A5CE, #0000A94C, #0000AF4C, #00C0B252, #0CC0B6D1,
         #0C00BC47, #0000BE11);

 CCMMENT  THE DPDA HAS 0108 READ STATES;

ARRAY 0108 SHORT INTEGER BREADSTART=  (#0000S, #0002S, #0008S, #000AS,
         #0010S, #0015S, #0018S, #001AS, #001CS, #001ES, #0020S,
         #0022S, #002AS, #002CS, #002ES, #0033S, #0035S, #0C39S,
         #003BS, #003ES, #0C46S, #0049S, #004BS, #004DS, #0C4FS,
         #0053S, #0056S, #0058S, #0061S, #006AS, #006CS, #006ES,
         #0077S, #0079S, #0082S, #0084S, #0086S, #0088S, #008AS,
         #008CS, #008ES, #0091S, #0093S, #0095S, #C098S, #009AS,
         #009DS, #00A0S, #00A3S, #00A5S, #00A7S, #00ACS, #00AES,
         #00B0S, #00B3S, #00B6S, #00E9S, #00C1S, #00C3S, #00C5S,
         #00C7S, #00C9S, #00CCS, #0CCFS, #00D1S, #00D4S, #G0D6S,
         #0CD9S, #00DBS, #00E4S, #00E6S, #00EFS, #00F1S, #0CF3S,
         #0CF5S, #00F7S, #0CF9S, #0CFBS, #00FDS, #00FFS, #0107S,
```

```
          #0109S, #010ES, #0110S, #0112S, #0114S, #0116S, #0118S,
          #011AS, #011CS, #011FS, #0121S, #0123S, #0126S, #0128S,
          #012AS, #012CS, #012ES, #013CS, #0132S, #0135S, #013ES,
          #0141S, #0143S, #0145S, #014DS, #014FS, #0151S);

ARRAY 0108 BYTE BRDNUM = (#01X, #05X, #01X, #C5X, #04X, #02X, #01X,
          #01X, #01X, #01X, #01X, #07X, #01X, #01X, #04X, #01X, #C3X,
          #01X, #02X, #07X, #02X, #01X, #01X, #C1X, #03X, #02X, #01X,
          #08X, #08X, #01X, #01X, #08X, #01X, #C8X, #01X, #01X, #C1X,
          #01X, #01X, #01X, #02X, #01X, #01X, #02X, #01X, #02X, #02X,
          #02X, #01X, #01X, #04X, #01X, #01X, #C2X, #02X, #02X, #07X,
          #01X, #01X, #01X, #01X, #02X, #02X, #C1X, #02X, #01X, #C2X,
          #01X, #08X, #01X, #08X, #01X, #01X, #C1X, #01X, #01X, #01X,
          #01X, #01X, #07X, #01X, #04X, #01X, #C1X, #01X, #01X, #01X,
          #01X, #01X, #02X, #01X, #01X, #02X, #01X, #01X, #01X, #C1X,
          #01X, #01X, #02X, #08X, #02X, #01X, #01X, #07X, #01X, #01X,
          #02X);

ARRAY 0340 BYTE BSYMLIST = (#01X, #00X, #05X, #CBX, #14X, #15X, #16X,
          #00X, #01X, #00X, #1AX, #28X, #29X, #2BX, #2CX, #00X, #C3X,
          #07X, #09X, #0AX, #00X, #04X, #08X, #00X, #06X, #00X, #C3X,
          #00X, #C3X, #00X, #06X, #00X, #07X, #CCX, #1BX, #1CX, #1DX,
          #1EX, #1FX, #20X, #21X, #00X, #02X, #CCX, #06X, #00X, #C6X,
          #18X, #24X, #2AX, #00X, #06X, #00X, #2DX, #2EX, #2FX, #00X,
          #06X, #00X, #04X, #08X, #0CX, #1BX, #1CX, #1DX, #1EX, #1FX,
          #20X, #21X, #00X, #07X, #08X, #00X, #2CX, #00X, #06X, #00X,
          #04X, #00X, #C8X, #09X, #0AX, #00X, #C8X, #0AX, #00X, #08X,
          #00X, #0CX, #0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #00X,
          #0CX, #0DX, #0EX, #0FX, #10X, #11X, #12X, #13X, #00X, #C6X,
          #00X, #03X, #00X, #0CX, #0DX, #0EX, #0FX, #10X, #11X, #12X,
          #13X, #00X, #03X, #00X, #0CX, #0DX, #CEX, #0FX, #10X, #11X,
          #12X, #13X, #00X, #06X, #00X, #07X, #00X, #07X, #00X, #C7X,
          #00X, #C7X, #C0X, #07X, #00X, #04X, #C8X, #00X, #07X, #0CX,
          #03X, #00X, #03X, #08X, #00X, #17X, #CCX, #04X, #08X, #CCX,
          #04X, #08X, #00X, #06X, #07X, #00X, #18X, #00X, #2AX, #00X,
          #06X, #18X, #24X, #2AX, #00X, #17X, #CCX, #19X, #00X, #03X,
          #06X, #00X, #03X, #08X, #00X, #04X, #08X, #00X, #1BX, #1CX,
          #1DX, #1EX, #1FX, #20X, #21X, #00X, #06X, #00X, #06X, #00X,
          #07X, #00X, #03X, #00X, #18X, #27X, #CCX, #18X, #27X, #CCX,
          #06X, #00X, #07X, #08X, #00X, #03X, #00X, #08X, #0AX, #CCX,
          #08X, #00X, #CCX, #0DX, #0EX, #0FX, #1CX, #11X, #12X, #13X,
          #00X, #08X, #00X, #0CX, #0DX, #0EX, #CFX, #10X, #11X, #12X,
          #13X, #00X, #06X, #00X, #18X, #00X, #18X, #00X, #18X, #00X,
          #18X, #00X, #18X, #00X, #22X, #00X, #22X, #00X, #18X, #22X,
          #23X, #24X, #25X, #26X, #27X, #00X, #03X, #00X, #06X, #18X,
          #24X, #2AX, #00X, #18X, #00X, #06X, #00X, #18X, #00X, #03X,
          #00X, #17X, #00X, #06X, #00X, #03X, #CCX, #18X, #27X, #CCX,
          #07X, #00X, #24X, #00X, #18X, #27X, #00X, #08X, #00X, #C4X,
```

```
          #00X, #08X, #COX, #04X, #0CX, #04X, #CCX, #04X, #00X, #03X,
          #08X, #00X, #07X, #18X, #22X, #23X, #24X, #25X, #26X, #27X,
          #0CX, #18X, #27X, #00X, #08X, #00X, #18X, #00X, #18X, #22X,
          #23X, #24X, #25X, #26X, #27X, #00X, #C7X, #0CX, #04X, #00X,
          #04X, #08X, #00X);

ARRAY 0340 SHORT INTEGER BSTATELIST = (#0001S, #06FFS, #0009S, #0C0BS,
          #0006S, #0007S, #0008S, #06FFS, #0263S, #06FFS, #0013S,
          #023ES, #0011S, #0015S, #0016S, #06FFS, #0203S, #001BS,
          #001CS, #001DS, #06FFS, #001ES, #001FS, #C6FFS, #0020S,
          #06FFS, #0220S, #06FFS, #0221S, #06FFS, #0206S, #06FFS,
          #0021S, #06FFS, #0023S, #0C24S, #0025S, #0026S, #0C27S,
          #022FS, #0029S, #06FFS, #002AS, #06FFS, #C304S, #06FFS,
          #0030S, #002FS, #024AS, #024CS, #06FFS, #0033S, #06FFS,
          #0038S, #0039S, #C03AS, #06FFS, #003BS, #06FFS, #003CS,
          #003DS, #06FFS, #0023S, #0024S, #0025S, #C026S, #0C27S,
          #022FS, #0029S, #06FFS, #003ES, #003FS, #06FFS, #0016S,
          #06FFS, #0253S, #06FFS, #0041S, #06FFS, #C044S, #001CS,
          #001DS, #06FFS, #0046S, #001CS, #06FFS, #0047S, #06FFS,
          #0217S, #0218S, #0219S, #021AS, #021BS, #021CS, #021DS,
          #021ES, #06FFS, #0217S, #0218S, #0219S, #C21AS, #021BS,
          #021CS, #021DS, #021ES, #06FFS, #0212S, #06FFS, #0205S,
          #06FFS, #0217S, #0218S, #0219S, #021AS, #C21BS, #021CS,
          #021DS, #021ES, #06FFS, #021FS, #06FFS, #0217S, #0218S,
          #0219S, #021AS, #021BS, #021CS, #021DS, #C21ES, #06FFS,
          #0216S, #06FFS, #0048S, #06FFS, #0049S, #06FFS, #004AS,
          #06FFS, #004BS, #06FFS, #004CS, #06FFS, #0230S, #004DS,
          #06FFS, #004ES, #06FFS, #0202S, #06FFS, #0223S, #0228S,
          #06FFS, #004FS, #06FFS, #0050S, #0241S, #C6FFS, #0245S,
          #0051S, #06FFS, #0052S, #0242S, #06FFS, #0249S, #06FFS,
          #024DS, #06FFS, #0030S, #0053S, #024AS, #C24CS, #06FFS,
          #0054S, #06FFS, #0055S, #06FFS, #0255S, #C30AS, #06FFS,
          #0256S, #0057S, #C6FFS, #0058S, #0059S, #06FFS, #0023S,
          #0024S, #0025S, #0026S, #0027S, #022FS, #0029S, #06FFS,
          #025FS, #06FFS, #005AS, #06FFS, #005BS, #C6FFS, #024ES,
          #06FFS, #023DS, #023CS, #06FFS, #023CS, #023CS, #06FFS,
          #0254S, #06FFS, #C05CS, #003FS, #06FFS, #0204S, #06FFS,
          #0046S, #001DS, #06FFS, #0047S, #06FFS, #C217S, #0218S,
          #0219S, #021AS, #021BS, #021CS, #021DS, #021ES, #06FFS,
          #0047S, #06FFS, #0217S, #0218S, #0219S, #021AS, #021BS,
          #021CS, #021DS, #021FS, #06FFS, #0213S, #C6FFS, #C05ES,
          #06FFS, #005FS, #06FFS, #0060S, #06FFS, #0061S, #06FFS,
          #0062S, #06FFS, #0232S, #06FFS, #0231S, #06FFS, #023DS,
          #0238S, #0236S, #0237S, #023AS, #023BS, #C23CS, #06FFS,
          #0224S, #06FFS, #C030S, #0053S, #024AS, #024CS, #06FFS,
          #0248S, #06FFS, #0052S, #06FFS, #0063S, #06FFS, #0226S,
          #06FFS, #0064S, #06FFS, #0260S, #06FFS, #0257S, #06FFS,
          #023DS, #023CS, #C6FFS, #0065S, #06FFS, #0066S, #06FFS,
```

```
        #023DS, #023CS, #06FFS, #0047S, #06FFS, #022AS, #06FFS,
        #0067S, #06FFS, #022CS, #06FFS, #022DS, #06FFS, #022ES,
        #06FFS, #0225S, #023FS, #06FFS, #025CS, #023DS, #0238S,
        #0236S, #0237S, #023AS, #023BS, #023CS, #06FFS, #023DS,
        #023CS, #06FFS, #0240S, #06FFS, #006AS, #06FFS, #023DS,
        #0238S, #0236S, #0237S, #023AS, #023BS, #023CS, #06FFS,
        #025DS, #06FFS, #022BS, #06FFS, #025ES, #025ES, #06FFS);

 COMMENT  THE DPDA HAS 0099 REDUCE STATES;

ARRAY 0100 BYTE BNUMTOPCP = (#00X, #00X, #03X, #01X, #03X, #02X, #01X,
        #02X, #01X, #01X, #01X, #00X, #00X, #00X, #01X, #02X, #01X,
        #02X, #01X, #02X, #02X, #02X, #02X, #00X, #00X, #00X, #00X,
        #00X, #00X, #00X, #00X, #02X, #01X, #01X, #01X, #02X, #03X,
        #04X, #03X, #00X, #02X, #01X, #03X, #05X, #03X, #03X, #03X,
        #00X, #01X, #02X, #02X, #00X, #02X, #00X, #00X, #00X, #00X,
        #00X, #00X, #00X, #00X, #00X, #00X, #04X, #04X, #02X, #01X,
        #01X, #02X, #01X, #00X, #00X, #02X, #01X, #00X, #00X, #00X,
        #01X, #02X, #01X, #02X, #03X, #02X, #01X, #02X, #01X, #01X,
        #02X, #01X, #01X, #00X, #04X, #00X, #01X, #02X, #01X, #02X,
        #03X, #02X, #02X);

ARRAY 0100 SHORT INTEGER BREDUCESUCC = (#B3FCS, #0002S, #0002S,
        #0201S, #0201S, #0201S, #0004S, #0017S, #0017S, #0017S,
        #0017S, #0017S, #0017S, #0017S, #0301S, #0301S, #0500S,
        #0500S, #0501S, #0501S, #0005S, #0005S, #000AS, #0502S,
        #0502S, #0502S, #0502S, #0502S, #0502S, #0502S, #0502S,
        #0003S, #0003S, #0003S, #0003S, #0300S, #0300S, #0300S,
        #0300S, #000DS, #000DS, #0227S, #0503S, #0503S, #0503S,
        #0503S, #0503S, #0503S, #0503S, #0028S, #0028S, #0504S,
        #0504S, #0505S, #0505S, #0505S, #0505S, #0505S, #0239S,
        #0239S, #0506S, #0506S, #000FS, #000FS, #000ES, #000ES,
        #0032S, #002ES, #002ES, #0246S, #002DS, #002DS, #0507S,
        #0507S, #0507S, #0507S, #0305S, #0305S, #0010S, #0010S,
        #0012S, #0012S, #0012S, #0508S, #0508S, #0306S, #0306S,
        #0306S, #0035S, #0035S, #0259S, #0259S, #0068S, #0068S,
        #0068S, #0036S, #0036S, #0037S, #0037S, #0700S);

 COMMENT  THE DPDA HAS 0013 LOOK AHEAD STATES;

ARRAY 0013 BYTE BLASYMNUM = (#32X, #35X, #35X, #35X, #3EX, #4BX, #42X,
        #35X, #35X, #35X, #3EX, #35X, #46X);

ARRAY 0013 SHORT INTEGER BSUCCSTATE = (#0222S, #020BS, #020CS, #020DS,
        #0233S, #024BS, #024FS, #02C8S, #02C9S, #020AS, #0233S,
        #0207S, #0235S);

ARRAY 0013 SHORT INTEGER BFAILSTATE = (#000CS, #0018S, #0019S, #001AS,
```

```
          #002CS, #0031S, #0034S, #0042S, #0043S, #0045S, #0056S,
          #0C5DS, #0069S);

ARRAY 0078 BYTE BLATABLE = (
          COMMENT  <DATA DECL>;
          #00X, #00X, #00X, #20X, #0CX, #D8X,

          COMMENT  <FPARM LIST>;
          #08X, #00X, #C0X, #00X, #0CX, #00X,

          COMMENT  <FPARM LIST>;
          #08X, #00X, #00X, #00X, #00X, #C0X,

          COMMENT  <FPARM LIST>;
          #08X, #00X, #00X, #00X, #00X, #00X,

          COMMENT  <ITEM>;
          #12X, #80X, #00X, #00X, #0CX, #00X,

          COMMENT  <DESCRIPTOR>;
          #08X, #80X, #C0X, #00X, #00X, #C0X,

          COMMENT  <TABLE DECL HEAD>;
          #00X, #00X, #C0X, #00X, #00X, #07X,

          COMMENT  <FPARM LIST>;
          #08X, #00X, #CCX, #00X, #0CX, #00X,

          COMMENT  <FPARM LIST>;
          #08X, #00X, #C0X, #00X, #0CX, #00X,

          COMMENT  <FPARM LIST>;
          #08X, #00X, #00), #00X, #0CX, #00X,

          COMMENT  <ITEM>;
          #12X, #80X, #C0X, #00X, #0CX, #00X,

          COMMENT  <FPARM LIST>;
          #08X, #00X, #C0X, #00X, #0CX, #00X,

          COMMENT  <CONSTANT>;
          #1AX, #80X, #00X, #00X, #00X, #00X);


 COMMENT  THE DPDA HAS 0009 LOOKBACK STATES;

ARRAY 0009 BYTE BLBSTART = (#00X, #02X, #06X, #CCX, #CFX, #11X, #13X,
```

```
        #1AX, #1DX);

ARRAY CC09 BYTE BLBNUM = (#01X, #03X, #05X, #C2X, #01X, #01X, #06X,
        #02X, #01X);

ARRAY 0031 SHORT INTEGER BLESTATE = (#0004S, #0C00S, #0004S, #C01ES,
        #0019S, #0000S, #001BS, #001CS, #001FS, #0021S, #0044S,
        #0000S, #000BS, #0013S, #000CS, #00CDS, #0000S, #0068S,
        #00C0S, #003DS, #003ES, #0C59S, #005CS, #0064S, #0065S,
        #0000S, #000ES, #0032S, #000CS, #00C3S, #000CS);

ARRAY CC31 SHORT INTEGER BRESUMESTATE = (#0302S, #0307S, #0303S,
        #0308S, #0309S, #030BS, #02CES, #021CS, #0215S, #0214S,
        #020FS, #0211S, #0022S, #0229S, #0258S, #002BS, #025AS,
        #006BS, #0234S, #0252S, #025CS, #0262S, #0251S, #030CS,
        #0261S, #0235S, #0247S, #0243S, #0244S, #0014S, #0C40S);

 CCMMENT  THE SYMBOLS ACCESSING THE STATES;

ARRAY 0108 BYTE CSYMBEFCREREAD = (#00X, #01X, #30X, #32X, #34X, #36X,
        #14X, #15X, #16X, #05X, #3BX, #03X, #33X, #3DX, #3FX, #41X,
        #42X, #29X, #4EX, #1AX, #4FX, #2BX, #2CX, #35X, #37X, #38X,
        #39X, #07X, #09X, #0AX, #04X, #08X, #06X, #07X, #3CX, #1BX,
        #1CX, #1DX, #1EX, #1FX, #45X, #21X, #C2X, #3EX, #06X, #40X,
        #4AX, #18X, #06X, #4DX, #49X, #06X, #43X, #5CX, #51X, #52X,
        #2DX, #2EX, #2FX, #06X, #04X, #08X, #07X, #08X, #4FX, #04X,
        #38X, #39X, #08X, #39X, #08X, #08X, #C7X, #07X, #07X, #07X,
        #07X, #08X, #07X, #17X, #04X, #08X, #06X, #18X, #17X, #19X,
        #06X, #08X, #04X, #08X, #06X, #07X, #07X, #39X, #18X, #18X,
        #18X, #18X, #18X, #18X, #17X, #07X, #24X, #08X, #54X, #47X,
        #18X, #46X);

ARRAY 0013 BYTE DSYMBEFORELA = (#33X, #37X, #38X, #39X, #06X, #4DX,
        #43X, #38X, #39X, #39X, #06X, #39X, #47X);

 CLOSE BASE;

 ARRAY 18 INTEGER BSEGTABLE = (
        #00000000,
        #0000031C,
        #0CCCC470,
        #00000548,
        #000005B4,
        #0CCCC708,
        #000C09B0,
        #00000A14,
        #0CCCCADC,
        #00000AEA,
```

```
                #00000B04,
                #00000B1E,
                #00000B6C,
                #00000B75,
                #00000B7E,
                #00000BBC,
                #00000BFA,
                #00000C66);
    COMMENT  END OF CARDS PUNCHED BY THE SLR(1) SYNTAX ANALYZER;


  COMMENT DECLARATIONS USED BY THE ANALYZER;
    INTEGER TOKEN=1; COMMENT TOKEN IS POINTER PROVIDED BY THE SCANNER. IT
                             HAS THE VALUE OF THE INDEX IN THE TERMINAL
                             SYMBOL LIST OF THE LAST SYMBOL SCANNED;
    INTEGER BTOKEN; COMMENT SAME AS ABOVE BUT FOR DATA DECL. GRAMMAR;
    INTEGER PRODNUM; COMMENT PRODUCTION NUMBER USED TO IDENTIFY PRODUCTION
                             WHEN CALLING SYNTHESIZE;
    INTEGER BPRODNUM; COMMENT SAME AS ABOVE BUT FOR BSYNTHESIZE;
    INTEGER SP=0; COMMENT POINTER TO LAST SYMBOL IN STATESTACK;
    INTEGER BSP; COMMENT SAME AS ABOVE BUT FOR BANALYZE;
    INTEGER LOCATION; COMMENT INDEX TO VSTRING OF SYMBOL;
    INTEGER LENGTH; COMMENT NUMBER OF BYTES IN SYMBOL;
    INTEGER VPT; COMMENT INDEX SAME AS TOKEN USED WHEN CALLING "FIND";
    INTEGER SEGBASE; COMMENT BASE ADDRESS OF PARSING TABLES;
    INTEGER BSEGBASE; COMMENT BASE ADDRESS OF PARSING TABLES OF DATA
                      DECLARATION GRAMMAR;
    INTEGER LATABSIZE; COMMENT LOOK AHEAD TABLE SIZE;
    INTEGER BLATABSIZE; COMMENT SAME AS ABOVE BUT FOR BANALYZE;
    ARRAY 64 BYTE BCD; COMMENT BUFFER USED TO STORE CURRENT SYMBOL;
    ARRAY 8 BYTE CONBUF; COMMENT BUFFER USED IN PRINTIME,PRINTSUMMARY
                                 AND PRINTDATE ROUTINES;
    ARRAY 3 INTEGER TIME; COMMENT SAME AS ABOVE;


   INTEGER MASK = #000000FF;
   INTEGER MASK7 = #00000007;
   INTEGER MASKFFFF = #0000FFFF;
   INTEGER BLANKMASK = #40404040;
   INTEGER MASK1 = #00000001;
   INTEGER MASKFF00 = #0000FF00;

   COMMENT  DECLARE USEFUL LITERALS TO SIMPLIFY
            ACCESSING THE PARSING TABLES;

     EQUATE AVSTRING        SYN 0;
     EQUATE ALOCLENGTH      SYN 4;
```

```
      EQUATE AREADSTART      SYN 8;
      EQUATE ARDNUM          SYN 12;
      EQUATE ASYMLIST        SYN 16;
      EQUATE ASTATELIST      SYN 20;
      EQUATE ANUMTOPOP       SYN 24;
      EQUATE AREDUCESUCC     SYN 28;
      EQUATE ALASYMNUM       SYN 32;
      EQUATE ASUCCSTATE      SYN 36;
      EQUATE AFAILSTATE      SYN 40;
      EQUATE ALATABLE        SYN 44;
      EQUATE ALBSTART        SYN 48;
      EQUATE ALBNUM          SYN 52;
      EQUATE ALBSTATE        SYN 56;
      EQUATE ARESUMSTATE     SYN 60;
      EQUATE ASYMBEFOREREAD  SYN 64;
      EQUATE BSYMBEFORELA    SYN 68;
    BYTE TRUE=#FFX, FALSE=#00X;


COMMENT  SCANNER DECLARATIONS;

      BYTE IDISLBL=#00X; COMMENT WHEN IDISLBL IS SET IDENTS ARE TREATED AS
                                                                LABELS;
    BYTE PCAL=#00X; COMMENT WHEN PCALL IS SET THE IDENT THAT FOLLOWS IS
                      A PROCEDURE NAME;
    BYTE ENDIT=#00X; COMMENT WHEN ENDIT IS SET SCANNER'S OUTPUT IS
                      ALWAYS TOKEN=1;
    BYTE BENDIT=#00X; COMMENT BENDIT IS SET BY "END_CD", "PROCEDURE" AND
                       "FUNCTION". WHEN BENDIT IS SET "$" SETS ENDIT.
                       BOTH ENDIT AND BENDIT ARE RESET UPON RETURN FROM
                       BANALYZE. ;
    BYTE LISTFLAG=#FFX; COMMENT LISTING PRODUCED IF LISTFLAG IS SET;
    ARRAY 80 BYTE CBUF; COMMENT  INPUT CARD BUFFER;
    ARRAY 132 BYTE BLANK=132(" ");  COMMENT  BLANK STRING USED TO CLEAN
                                             THE WRITE BUFFER;
    ARRAY 132 BYTE WBUF=132(" ");  COMMENT WRITE BUFFER;
    SHORT INTEGER CARDCOUNT=1S;  COMMENT  NUMBER OF CARDS READ BY GETCHAR;
    LONG REAL CONWORK;  COMMENT  DOUBLE WORD BUFFER USED TO CONVERT TO
                                 DECIMAL;
    SHORT INTEGER CP=10; COMMENT  CP IS POINTER TO NEXT CHARACTER TO
                                 BE SCANNED;
    BYTE CHAR;  COMMENT  SINGLE CHARACTER BUFFER;
    SHORT INTEGER CLASS;  COMMENT  CHARACTER CLASS RECEPTACLE;
    ARRAY 22 BYTE SBUF;   COMMENT  BUFFER USED BY SCANNER TO ASSEMBLE
                                   SYMBOLS;
    SHORT INTEGER SPTR;  COMMENT  INDEX IN SBUF OF LAST CHARACTER INSERTED
  ; SHORT INTEGER XR;  COMMENT  ERROR NUMBER WHEN CALLING THE ERROR
                               PROCEDURE;
```

```
  SHORT INTEGER ERRCOUNT=0; COMMENT ERROR COUNTER;

  ARRAY 250 BYTE CHARCLASS=(75(#1AX),                  #08X,#0BX,#12X,#14X,
#19X,#01X,10(#1AX),#10X,#0DX,#13X,#18X,#09X,#15X,#16X,9(#1AX),#11X,#0CX,
#05X,#CAX,11(#1AX),#1BX,#06X,#07X,#0FX,#17X,#0EX,65(#1AX),6(#04X),
3(#C5X),7(#1AX),9(#05X),8(#1AX),8(#05X),6(#1AX),8(#02X),2(#03X));

  INTEGER RTRNTOANAL; COMMENT RECEPTACLE FOR RETURN ADDRESS WHEN ANALYZE
                      CALLS SCAN;
  INTEGER BANALBASE, SCANBASE; COMMENT ENTRY POINT RECEPTACLES USED WHEN
                                       SCAN CALLS BANALYZE;
COMMENT  SYMBOL TABLE AND CONSTANT TABLE STRUCTURE DECLARATIONS;

  INTEGER STBASE, STLENGTH, CTBASE, STLIMIT, SEMBASE, STCHAINBASE;
COMMENT STBASE IS BASE ADDRESS OF SYMBOL TABLE, IT ALSO IS BASE ADDRESS
OF IDENTIFIER NAMES FIELD IN SYM. TBL. CTBASE IS BASE ADDRESS OF
CONSTANT TABLE. SEMBASE IS BASE ADDRESS OF SEMANTICS FIELD. STCHAIN IS
BASE ADDRESS OF FIELD WITH POINTERS TO NEXT AVAILABLE SYMBOL TABLE SLOT
IN CASE OF COLLISION;
  INTEGER CTLIMIT; COMMENT ABSOLUTE ADDRESS LIMIT OF CONSTANT TABLE;
  INTEGER DESCRPTRBASE; COMMENT BASE ADDRESS OF POINTER TO DESCRIPTOR
                        FIELD IN SYMBOL TABLE;
  SHORT INTEGER DPTR=0S; COMMENT INDEX TO DESCRBUF OF NEXT BYTE TO BE
                         LOADED;
  SHORT INTEGER TYPLGTH; COMMENT TEMPORARY LOCATION USED BY DATA
                         DECLARATION SEMANTIC ROUTINES;
  SHORT INTEGER DATABUF; COMMENT DATA TYPE BUFFER;
  SHORT INTEGER CTR; COMMENT COUNTER USED IN SEVERAL PLACES IN SEMANTIC
                     ROUTINES;
  BYTE TYPE; COMMENT TYPE IS USED TO CHECK FOR COMPATIBILITY DURING
             ASSIGNMENTS;
  SHORT INTEGER TBLDESCRB; COMMENT TABLE DESCRIPTOR BASE ADDRESS IS
                           STORED HERE IN TABLE DECLARATION SEMANTIC
                           ROUTINES    ;
  SHORT INTEGER FLBPTR=0S; COMMENT INDEX TO FLDPTRSBUF OF NEXT BYTE
                           TO BE LOADED;
  BYTE FIRSTVAL=#00X; COMMENT FLAG USED IN REPETITION LIST PROCESSING
                      ROUTINES TO IDENTIFY FIRST VALUE IN LIST   ;
  ARRAY 80 BYTE FLDPTRSBUF; COMMENT BUFFER TO STORE POINTERS TO FIELD
                            DECLARATION ENTRIES IN THE  IDENTIFIER
                            DIRECTORY DURING TABLE DECLARATION BLOCK
                            PROCESSING  ;
  ARRAY 20 SHORT INTEGER TBLIST; COMMENT BUFFER TO STORE TABLE NAMES
                                 DURING TABLE HEAD PROCESSING;
  SHORT INTEGER TBLISTPTR=0; COMMENT INDEX INTO TBLIST;

  SHORT INTEGER BLKCTR=0;  COMMENT  BLOCK LEVEL COUNTER USED AS A PREFIX
                                    TO IDENTIFIERS TO INTRODUCE BLOCK
```

```
                                                        STRUCTURE INTO SYMBOL TABLE SCHEME;
SHORT INTEGER HTPTR; COMMENT  INDEX INTO HASHTABLE PRODUCED BY HASHING
                              SCHEME;
SHORT INTEGER STPTR;  COMMENT  SYMBOL TABLE INDEX;
SHORT INTEGER CTPTR =0; COMMENT CONSTANT TABLE INDEX;
SHORT INTEGER SEM;  COMMENT SEMANTICS RECEPTACLE;
SHORT INTEGER POINTER;  COMMENT  POINTER INTO SYMBOL TABLE OR CONSTANT
                                 TABLE ASSOCIATED WITH EACH CONSTANT
                                 OR IDENTIFIER;
ARRAY 11 BYTE TEMPREFIX; COMMENT TEMPORARY BUFFER USED TO SET SBUF
                         BEFORE HASHING RESERVED WORDS;
ARRAY 10 BYTE PREFIX; COMMENT TEMPORARY BUFFER USED TO SET SBUF
                      BEFORE HASHING IDENTIFIERS;
SHORT INTEGER POINTFREE=_1; COMMENT POINTER TO NEXT AVAILABLE SLOT
                            IN SYMBOL TABLE;

GLOBAL DATA SEGN002 BASE R9;
ARRAY 1229 SHORT INTEGER HASHTBL=1229(_1); COMMENT HASH TABLE;
CLOSE BASE;

GLOBAL DATA SEGN003  BASE R10;
ARRAY 150 SHORT INTEGER POINTERSTACK=150(0); COMMENT STACK USED BY
      ANALYZE TO PASS POINTERS TO SYNTHESIZE;
ARRAY 150 SHORT INTEGER BPOINTERSTACK=150(0); COMMENT SAME AS ABOVE
                                              BUT FOR BANALYZE;
ARRAY 256 BYTE DESCRBUFFER=256(#40X); COMMENT BUFFER TO BUILD UP
      DESCRIPTORS BEFORE LOADING THEM IN CONSTANT TABLE;
CLOSE BASE;

COMMENT BASE ADDRESS RECEPTACLES DECLARED TO FACILITATE USE OF ABOVE
DECLARED ARRAYS;
INTEGER PTRSTACK=#00000000;
INTEGER BPTRSTACK=#0000012C;
INTEGER DESCRBUF=#00000258;

  FUNCTION CLR(1,#1500);
FUNCTION SC(1C,#FB00);
FUNCTION SETZONE(8,#96F0);  COMMENT  FUNCTION TO SET ZONE TO 1111;
EXTERNAL PROCEDURE GETCORE(R14); NULL;


PROCEDURE FIND(R4);
   BEGIN ARRAY 4 INTEGER SAVEREGS;
       STM(R1,R4,SAVEREGS);
       MVC(63,BCD,BLANK);
       R6 := R6 - R6; R1 := VPT SHLL 2;
       R3 := ALOCLENGTH; R2 := SEGBASE + SEGTABLE(R3) + R1;
```

```
          R3 := B2; R1 := R3 SHRL 6;
          LCCATICN := R1; R1 := R3 AND #3F;
          LENGTH := R1;  R2 := LENGTH - 1;
          FCR R1 := 0 STEP 1 UNTIL R2 DC
          BEGIN
             R3 := LOCATION + R1; IC(R6,VSTRING(R3));
             STC(R6,BCD(R1));
          END;
       LM(R1,R4,SAVEREGS);
       ENC; CCMMENT END OF FIND;

    PROCEDURE BFIND(R4); COMMENT SAME AS FIND BUT FCR SECCNC GRAMMAR;
       BEGIN ARRAY 4 INTEGER SAVEREGS;
          STM(R1,R4,SAVEREGS);
          MVC(63,BCD,BLANK);
          R6 := R6 - R6; R1 := VPT SHLL 2;
          R3 := ALOCLENGTH; R2 :=BSEGBASE +BSEGTABLE(R3) + R1;
          R3 := B2; R1 := R3 SHRL 6;
          LCCATION := R1; R1 := R3 AND #3F;
          LENGTH := R1;  R2 := LENGTH - 1;
          FOR R1 := 0 STEP 1 LNTIL R2 DC
          BEGIN
             R3:=LCCATION+R1; IC(R6,BVSTRING(R3));
             STC(R6,BCD(R1));
          END;
       LM(R1,R4,SAVEREGS);
       ENC; COMMENT END OF FIND;

   PRCCECLRE PRINTIME(R6);
       BEGIN INTEGER SAVE6;
          SAVE6 := R6;
          UNPK(7,3,CONBUF,TIME(0));
          MVC(1,WBUF(18),CONBUF(1));  MVC(0,WBUF(2C),":");
          MVC(1,WBUF(21),CCNBUF(3));
          R6 := SAVE6;
       ENC;      COMMENT END OF PRINTIME;

   PRCCECLRE PRINTDATE(R6);
       BEGIN INTEGER SAVE6, SAVE15;
          SAVE6 := R6; SAVE15 := R15;
          R1 := 2; SVC(11);
          R15 := SAVE15; R0 := RO OR  #F;
          TIME(C) := R0;
          TIME(4) := R1; UNPK(7,3,CONBUF,TIME(4));
          MVC(1,WBUF(9),CONBUF(3));
          MVC(0,WBUF(11),".")    ; MVC(2,WBUF(12),CCNBUF(5));
          R6 := SAVE6;
       ENC;   COMMENT END OF PRINTDATE;
```

```
PROCEDURE PRINTSUMMARY(R4);
COMMENT  THIS PROCEDURE SHOULD SUBTRACT CURRENT TIME OF DAY
         WITH THAT SAVED IN PROCEDURE INITIALIZE AND STORE
         THE RESULT IN WBUF STARTING  IN COLUMN 19;
   BEGIN INTEGER SAVE4;
         INTEGER SAVE15;
         SAVE15 := R15;
         SAVE4 := R4;
      R1 := 2; SVC(11);
         R15 := SAVE15;
         R0 := R0 OR #F; TIME(8) := R0;
         SD(3,3,TIME(8),TIME(0));
         UNPK(7,3,CONBUF,TIME(8)); MVC(1,WBUF(21),CONBUF(3));
         MVC(0,WBUF(23),".")    ; MVC(1,WBUF(24),CONBUF(5));
         MVC(6,WBUF(28),"SECONDS");
         MVC(17,WBUF,"TIME IN EXECUTION:"); R0 := @WBUF; WRITE;
         MVC(40,WBUF,BLANK);
         R4 := SAVE4;
   END; COMMENT END OF PRINTSUMMARY;

  PROCEDURE HASH(R4); COMMENT THIS HASHING SCHEME USES FIVE ARGUMENTS:
                       THE WORD LENGTH, THE SECOND AND THIRD AND THE
                      LAST TWO CHARACTERS OF THE  IDENTIFIER;
   BEGIN ARRAY 6 INTEGER SAVEREGS; STM(R0,R5,SAVEREGS);
   R4:=SPTR; R1:=R4+1;
   IC(R0,SBUF(1)); R0:=R0 AND #3F SHLL 18;
   IC(R2,SBUF(2)); R2:=R2 AND #3F SHLL 12;
   IC(R3,SBUF(R4)); R3:=R3 AND #3F; R5:=R4-1;
   IC(R4,SBUF(R5)); R4:=R4 AND #3F SHLL 6;
   R1:=R1 SHLL 24 OR R0 OR R2 OR R4 OR R3;
   R0:=0; R1:=R1/1229;
   HTPTR:=R0; COMMENT R0 CONTAINS THE REMAINDER;
   LM(R0,R5,SAVEREGS);
  END; COMMENT END OF HASH;

  PROCEDURE ERROR(R4); COMMENT THIS PROCEDURE HAS THE COMPLETE SET OF
                        ERROR MESSAGES. MESSAGE IS SELECTED BY LOADING
                        ERROR NUMBER INTO XR BEFORE AN ERROR CALL;
   BEGIN ARRAY 5 INTEGER SAVEREGS;STM(R0,R4,SAVEREGS);
   R0:=ERRCOUNT+1; ERRCOUNT:=R0;
COMMENT IF LISTING FLAG IS OFF, FORCE PRINTING OF CARD BUFFER;
   IF ¬ LISTFLAG THEN
   BEGIN
     R0:=CARDCOUNT+1; CARDCOUNT:=R0;
     CVD(R0,CONWORK); UNPK(3,7,WBUF(15),CONWORK);
     SETZONE(WBUF(18)); MVC(79,WBUF(22),CBUF);
     R0:=@WBUF; WRITE; MVC(131,WBUF,BLANK);
```

```
  END;
  RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK);
  MVC(8,WBUF,"*** ERROR");
  R1:=CP+22; R2:=@WBUF(R1);
  MVC(0,B2,"|");
  RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK);
  R2:=XR;
  CASE R2 OF
  BEGIN
    MVC(37,WBUF,"ILLEGAL IDENTIFIER. IT WILL BE IGNORED");
    MVC(36,WBUF,"ILLEGAL CHARACTER. IT WILL BE IGNORED");
    BEGIN
      MVC(15,WBUF,"STACK OVERFLOW.");
      RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK); GOTO EXIT;
    END;
    MVC(19,WBUF,"ILLEGAL SYMBOL PAIR:");
    BEGIN
      MVC(24,WBUF,"PROGRAM ENDS PREMATURELY.");
      RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK); GOTO EXIT;
    END;
    MVC(79,WBUF,CBUF);
    MVC(47,WBUF,"ILLEGAL HEXADECIMAL CONSTANT. IT WILL BE IGNORED");
    BEGIN
      MVC(47,WBUF,"SYMBOL TABLE OVERFLOW. REMAINING STEPS CANCELLED");
      RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK); GOTO EXIT;
    END;
  MVC(44,WBUF,"ILLEGAL LABEL DECLARATION. IT WILL BE IGNORED");
  MVC(43,WBUF,"ILLEGAL TAG DECLARATION. IT WILL BE IGNORED.");
  MVC(49,WBUF,"ILLEGAL STATUS CONSTANT IDENT. IT WILL BE IGNORED.");
  BEGIN
    MVC(50,WBUF,"CONSTANT TABLE OVERFLOW. REMAINING STEPS CANCELLED.")
    RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK); GOTO EXIT;
  END;
  MVC(26,WBUF,"ILLEGAL DATA SPECIFICATION.");
  MVC(41,WBUF,"INCOMPATIBLE DATA TYPES IN INITIALIZATION.");
  MVC(35,WBUF,"IMPROPER CARRIAGE CONTROL CHARACTER.");
  MVC(35,WBUF,"ILLEGAL FORMAT DESCRIPTOR PARAMETER.");
  END; COMMENT END ERROR CASE STMT;
  RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK);
  LM(RO,R4,SAVEREGS);
END; COMMENT END OF PROCEDURE ERROR;

PROCEDURE FUTIL(R4); COMMENT IT LOADS VALUE IN RO IN IN SEMANTIC FIELD
                     OF IDENT WITH POINTER ON TOP OF STACK. ALSO
                     LOADS DATABUF IN DESCRBUF;
BEGIN
  R1:=BSP SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R3,B2);
```

```
   R2:=SEMBASE+R3; STC(R0,B2);
   R2:=R10+DESCRBUF+5; R1:=DATABUF;
   R3:=R1 SHRL 8; STC(R3,B2);
   R2:=R2+1; STC(R1,B2);
   R0:=7; DPTR:=R0;
END; COMMENT END OF FUTIL;

PROCEDURE DATASIZE(R4); COMMENT THIS PROCEDURE TAKES INDEX TO BPTRSTAC
                        FROM R1 AND RETURNS THE INTEGER VALUE IN THE
                        CONSTANT TABLE CONVERTED TO BINARY FORM;
BEGIN
   INTEGER SAVE4; SAVE4:=R4;
   R1:=R1 SHLL 1;
   R2:=R10+BPTRSTACK+R1; LH(R3,B2); COMMENT R3 HAS CTPTR;
   R2:=CTBASE+R3; IC(R4,B2); COMMENT R4 HAS HEADER BYTE;
   R5:=R4 AND #FF; R5:=R5 SHRL 5;
   IF R5¬=3 THEN COMMENT NUMERIC CONSTANT IS NOT A DECIMAL INTEGER AS
              IT SHOULD;
   BEGIN
     R0:=13; XR:=R0; ERROR;
   END ELSE
   BEGIN
     R5:=R4 AND #1F; COMMENT R5 HAS LENGTH;
     R2:=R2+1; IC(R1,B2); R1:=R1 AND #F;
     R0:=R0-R0;
     FOR R3:=2 STEP 1 UNTIL R5 DO
     BEGIN
       R2:=R2+1; IC(R4,B2); R4:=R4 AND #F;
       R1:=R1 * 10S + R4;
     END;
   END;
   R4:=SAVE4;
END; COMMENT END OF DATASIZE;

PROCEDURE SETHEADER(R4); COMMENT USED BY PRODUCTIONS 7 THRU 13 TO LOAD
                         LENGTH OF DESCRIPTOR (NOT COUNTING HEADER
                         ITSELF) INTO HEADER BYTE NOW THAT ALL PARMS
                         HAVE BEEN ENTERED;
BEGIN R1:=DPTR-1; R2:=R10+DESCRBUF; STC(R1,B2); END;

PROCEDURE NWD(R4); COMMENT "N", "W" AND "D" PARTS OF NUMERIC
                   CONVERTION FORMAT. R1 INPUTS POINTER TO REAL
                   CONSTANT. THIS PROCEDURE CHECKS THAT REAL CONSTANT
                   IS EITHER INTEGER OR FIXED. IF INTEGER, BINARY
                   FORM IS RETURNED IN R1 AND R0 IS SET TO ZERO. IF
                   FIXED, INTEGER PART IN BINARY FORM IS RETURNED IN
                   R1 AND FRACTIONAL PART IN R0;
BEGIN INTEGER TEMP; ARRAY 3 INTEGER SAVEREGS;
```

```
  STM(R2,R4,SAVEREGS);
  R2:=CTBASE+R1; IC(R3,B2);
  R5:=R3 SHRL 5; R3:=R3 AND #1F; R5:=R5 AND #3;
  IF R5=3 THEN COMMENT INTEGER CONST.;
  BEGIN
    R2:=R2+1; IC(R1,B2); R1:=R1 AND #F;
    R0:=R0-R0;
    FOR R6:=2 STEP 1 UNTIL R3 DO
    BEGIN
      R2:=R2+1; IC(R5,B2); R5:=R5 AND #F;
      R1:=R1*10S+R5;
    END;
    R0:=R0-R0;
  END ELSE
  BEGIN
    IF R5=4 THEN COMMENT FIXED CONST. ;
    BEGIN
      R2:=R2+1; R4:=1; IC(R5,B2); R5:=R5 AND #FF; R1:=R5 AND #F;
      IF R5=#4B THEN R1:=R1-R1
      ELSE BEGIN
        R2:=R2+1; R4:=R4+1; IC(R5,B2);
        R5:=R5 AND #FF; R0:=R0-R0;
        WHILE R5¬=#4B DO
        BEGIN
          R5:=R5 AND #F; R1:=R1*10S+R5;
          R2:=R2+1; R4:=R4+1; IC(R5,B2); R5:=R5 AND #FF;
      END;
    END;
      TEMP:=R1; R2:=R2+1; R4:=R4+2;
        IC(R1,B2); R1:=R1 AND #F;
      R0:=R0-R0;
      FOR  R6:=R4 STEP 1 UNTIL R3 DO
      BEGIN
        R2:=R2+1; IC(R5,B2); R5:=R5 AND #F;
        R1:=R1*10S+R5;
      END;
      R0:=R1;
      R1:=TEMP;
    END ELSE BEGIN R0:=16; XR:=R0; ERROR; END;
  END;
  LM(R2,R4,SAVEREGS);
END; COMMENT END OF NWD;

PROCEDURE CHECKID(R4); COMMENT R1 HAS POINTER TO IDENT IN IDENT
                       DIRECTORY. CHECKID CHECKS THAT IDENT IS ONE OF
                       9 LEGAL LETTER PARAMETERS (I,F,E,O,H,A,L,X,T)
                       AND RETURNS CORRESPONDING CODE IN R1. IF ILLEGA
                       L, THEN R1 IS -1, ELSE R1 HAS THE LETTER;
```

```
BEGIN INTEGER SAVE4; SAVE4:=R4;
  R1:=R1*10S;
  R4:=STBASE+R1+2;
  IC(R1,B4); R1:=R1 AND #1F;
  R2:=R1 SHRL 4; R3:=R1 AND #F;
  R2:=R2+1;
  CASE R2 OF
  BEGIN
    BEGIN COMMENT IF 5TH BIT IS ZERC;
      CASE R3 OF
      BEGIN
        NULL; R1:=_1; NULL; R1:=_1; NULL; NULL; NULL; NULL; NULL;
        R1:=_1; R1:=_1; R1:=_1; R1:=_1; R1:=_1; R1:=_1;
      END;
    END;
    BEGIN COMMENT IF 5TH BIT IS ONE;
      CASE R3 OF
      BEGIN
        R1:=_1; R1:=_1; NULL; R1:=_1; R1:=_1; NULL; R1:=_1; R1:=_1;
        R1:=_1; R1:=_1; R1:=_1; R1:=_1; R1:=_1; R1:=_1; R1:=_1;
      END;
    END;
  ENC;
  R4:=SAVE4;
END; COMMENT END OF CHECKID;

FRCCECURE SETVRBLDSCR(R4); COMMENT IT LOADS "4" IN HEACER BYTE,
                           DATABUF IN NEXT TWC BYTES, AND NEXT TWO
                           BYTES WITH POINTER CN TOP CF STACK;
BEGIN
  R1:=DATABUF; R2:=DESCRBUF+R10;
  R3:=4; STC(R3,B2); R2:=R2+1;
  R3:=R1 SHRL 8; STC(R3,B2); R2:=R2+1;
  STC(R1,B2); R2:=R2+1;
  R1:=BSP SHLL 1; R5:=R10+BPTRSTACK+R1;
  LH(R1,B5); R3:=R1 SHRL 8;
  STC(R3,B2); R2:=R2+1;
  STC(R1,B2);
  RC:=5; DPTR:=R0;
ENC;

FRCCEDURE ENTER(R5); COMMENT ENTER INSERTS THE ICENTIFIER IN SBUF
                     INTO A SLOT IN THE SYMBCL TABLE AT INDEX LCADED
                     IN R3;
BEGIN ARRAY 4 INTEGER SAVEREGS; STM(R2,R5,SAVEREGS);
  R3:=R3*10;
  R4:=STBASE+R3;
  MVC(9,B4,SBUF);
```

```
   LM(R2,R5,SAVEREGS);
END; COMMENT END OF ENTER;

PROCEDURE COMPARE(R4);COMMENT COMPARE TESTS IF IDENTIFIER IN SBUF IS
                      EQUAL TO ENTRY IN SYMBOL TABLE AT INDEX STPTR;
BEGIN ARRAY 3 INTEGER SAVEREGS; STM(R2,R4,SAVEREGS);
   R3:=STPTR;
   R3:=R3*10;
   R4:=STBASE+R3;
   CLC(9,SBUF,B4);
   LM(R2,R4,SAVEREGS);
END; COMMENT END OF COMPARE;

PROCEDURE RESERWRD(R4); COMMENT RESERWRD IS USED BY SCANNER TO FIND IF
                        IDENTIFIER IS A RESERVED WORD. IF WORD IS
                        FOUND ITS SEMANTICS IS LOADED INTO SEM ELSE SEM
                         IS ZERO;
BEGIN ARRAY 5 INTEGER SAVEREGS; STM(R0,R4,SAVEREGS);
   MVC(10,TEMPREFIX,"0         ");
   MVC(9,TEMPREFIX(1),SBUF);
   MVC(10,SBUF,TEMPREFIX);
   R0:=SPTR+1; IF R0>10 THEN R0:=10;
   SPTR:=R0;
   HASH;
   MVC(9,TEMPREFIX,SBUF(1));
   MVC(9,SBUF,TEMPREFIX);
   R0:=SPTR-1; SPTR:=R0;
   R1:=HTPTR; R1:=R1 SHLL 1;
   R3:=R9+R1; LH(R2,B3);
   IF R2=_1 THEN BEGIN R0:=0; SEM:=R0; END
   ELSE BEGIN
     STPTR:=R2; COMPARE;
     IF ¬= THEN BEGIN R0:=0; SEM:=R0; END ELSE
     BEGIN R3:=SEMBASE+R2; IC(R0,B3); R0:=R0 AND #FF; SEM:=R0; END;
   END;
   LM(R0,R4,SAVEREGS);
END; COMMENT END OF RESERWRD;


PROCEDURE LOOKUP(R4); COMMENT LOOKUP TAKES THE IDENTIFIER IN SBUF AND
                      ADDS THE CURRENT BLOCK NUMBER PREFIX TO IT. THEN
                      HASHES IT AND LOOKS FOR IT IN THE SYMBOL TABLE.
                      IF THERE, IT RETURNS ITS INDEX AND SEMANTICS.
                      ELSE IT TRIES AGAIN WITH PREFIX "00". IF NOT
                      THERE EITHER THEN ENTERS IT IN THE SYMBOL TABLE
                      WITH CURRENT BLOCK NUMBER PREFIX AND RETURNS ITS
                      INDEX AND IDENTIFIER SEMANTICS;
BEGIN  ARRAY 5 INTEGER SAVEREGS; STM(R0,R4,SAVEREGS);
```

```
    MVC(7,PREFIX(2),BLANK);
    MVC(7,PREFIX(2),SBUF);
    MVC(9,SBUF,PREFIX);
    R0:=SPTR+2;
    IF R0>10 THEN R0:=10;
    SPTR:=R0;
HASH1: HASH;
    R1:=HTPTR; R1:=R1 SHLL 1;
    R3:=R9+R1; LH(R2,B3);
IF1: IF R2¬=_1 THEN COMMENT COMPARE SBUF WITH IDENTRY UNTIL IDENTIFIER
                   IS FOUND OR STCHAIN IS ZERO;
    BEGIN
      STPTR:=R2; COMPARE;
      IF = THEN COMMENT IDENT IS FOUND;
        BEGIN
        POINTER:=R2; R3:=SEMBASE+R2;
        IC(R0,B3); R0:=R0 AND #FF;
        SEM:=R0; GOTO LOOKEXIT;
      END
      ELSE BEGIN R3:=R2 SHLL 1;
             R4:=STCHAINBASE+R3; LH(R2,B4); GOTO IF1; END;
    END
    ELSE BEGIN COMMENT IDENT IS NOT IN SYMBOL TABLE WITH THIS PREFIX;
      CLC(1,SBUF,"00");
      IF ¬= THEN BEGIN COMMENT TRY WITH "00";
          MVC(1,SBUF,"00");
          GOTO HASH1;
      END
      ELSE COMMENT THIS PREFIX IS "00" SO IDENT IS NOT IN SYMBOL TABLE;
      BEGIN COMMENT LETS ENTER THE IDENT WITH CURRENT PREFIX;
        MVC(1,SBUF,PREFIX);
        HASH; R1:=HTPTR; R1:=R1 SHLL 1;
        R4:=R9+R1;
IF2: LH(R2,B4);
        IF R2=_1 THEN
        BEGIN
          R3:=POINTFREE+1; R0:=STLIMIT; IF R3>=R0 THEN BEGIN
                                        R0:=8; XR:=R0; ERROR; END;
          POINTFREE:=R3; POINTER:=R3; ENTER; STH(R3,B4);
          R2:=R3 SHLL 1; R4:= STCHAINBASE+R2; R0:=_1; STH(R0,B4);
          R0:=0; SEM:=R0;
        END
        ELSE BEGIN
          R2:=R2 SHLL 1; R4:=STCHAINBASE+R2; GOTO IF2;
        END;
      END;
    END;
LOOKEXIT: LM(R0,R4,SAVEREGS);
```

```
END; CCMMENT END OF LCOKUP;

PRCCEDURE GETCHAR(R4);  COMMENT  GETCHAR LOACS CBUF(CP) INTC CHAR AND
ITS CLASS INTC CLASS. IT THEN INCREMENTS CP ANC IF NEECEC, REACS NEW
CARD AND LISTS IT IF LISTFLAG IS SET;

BEGIN  ARRAY 5 INTEGER SAVEREGS; STM(RO,R4,SAVEREGS);
   R3:=CP;
   IC(R1,CBUF(R3)); STC(R1,CHAR); R1:=R1 AND #FF;
   IC(R2,CHARCLASS(R1)); R2:=R2 AND #FF; CLASS:=R2;
   R3:=R3+1; CP:=R3;
   IF R3>79 THEN
   BEGIN
      RO:=@CBUF; READ; COMMENT READ A CARD INTC CARD BUFFER;
      IF > THEN  COMMENT CCNDITICN CODE = 2 INCICATES EOF;
      BEGIN
   MVC(79,CBUF,BLANK);
         MVC(10,CBUF(10),"ECF EOF EOF");
         SET(ENDIT);
      END;
      RO:=CARDCCUNT+1; CARDCOUNT:=RO; CCMMENT UPCATE CARCCOUNT;
      IF LISTFLAG THEN CCMMENT IF LISTFLAG IS SET THEN LIST CARCCCUNT
                                 AND CARC BUFFER;
      BEGIN
         CVD(RO,CONWORK); UNPK(3,7,WBUF(15),CCNWCRK);
         SETZCNE(WBUF(18)); MVC(79,WBUF(22),CBUF);
         RO:=@WBUF; WRITE; MVC(131,WBUF,BLANK);
      ENC;
      RO:=10; CP:=RO; COMMENT  CHARACTER PTR PCINTS AT COLUNM 11 CF NEW
                               CARD;
   ENC;
   LM(RO,R4,SAVEREGS);
END; COMMENT END OF GETCHAR;

PRCCEDURE CCNCAT(R4);  CCMMENT CONCAT ASSEMBLES CHARACTERS AS THEY
ARE SCANNED INTO THE SYMBOL BUFFER SBUF;
BEGIN  ARRAY 3 INTEGER SAVEREGS;  STM(R2,R4,SAVEREGS);
   R2:=SPTR+1; SPTR:=R2;
   IF R2<22 THEN
   BEGIN  R3:=@SBUF(R2);  MVC(0,B3,CHAR); ENC;
   LM(R2,R4,SAVEREGS);
ENC; COMMENT END OF CONCAT;

GLCBAL PROCEDURE SCANNER(R3); COMMENT SCANNER HAS 27 CASES AS DETAILED
                              BELOW;
```

```
  BEGIN
    ARRAY  6 INTEGER SAVEREGS; STM(RC,R5,SAVEREGS);
COMMENT SYMECL EUFFER SETUP;
    RO:=_1; SPTR:=RO; MVC(21,SBUF,BLANK);
COMMENT SKIP OVER BLANKS;
    IC(R1,CHAR); R1:=R1 AND #FF;
    WHILE R1=#40 DO BEGIN GETCHAR; IC(R1,CHAR); R1:=R1 ANC #FF; ENC;
    R2:=CLASS;
STARTSCAN:CASE R2 OF
BEGIN
  BEGIN CCMMENT CASE 1 HANDLES OCTAL CCNSTANTS;
    GETCHAR; R2:=CLASS;
    WHILE R2=2 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
    IF R2=8 THEN CCNCAT CCMMENT DECIMAL POINT;
    ELSE BEGIN
      IC(R1,CHAR); R1:=R1 AND #FF;
      IF R1=#C5 THEN GCTC L1 COMMENT FLOATIN PCINT CONSTANT;
      ELSE CCMMENT LOAD CONSTANT IN TAELE AND (C TC EXIT;
      EEGIN
        R5:=SPTR+1; R3:=CTPTR; PCINTER:=R3;
        R4:=R5+R3+CTBASE; RO:=CTLIMIT; IF R4>=RO THEN
        BEGIN RO:=12; XR:=RO; ERROR; END;
        R4:=CTBASE+R3; STC(R5,B4);
        R4:=R4+1; R5:=R5-1;
        EX(R5,MVC(0,B4,SBUF));
        R3:=R3+R5+2; CTPTR:=R3;
        RO:=1; SEM := RO; GOTO EXIT1;
      END;
    END;
    GETCHAR; R2:=CLASS;
    WHILE R2=2 CO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
    IC(R1,CHAR); R1:=R1 AND #FF;
    IF R1=#C5 THEN GOTO L1 CCMMENT FLOATING PCINT CONSTANT;
    ELSE COMMENT      LCAD CCNSTANT IN TABLE ANC GC TO EXIT;
      EEGIN
        R5:=SPTR+1; R3:=CTPTR; POINTER:=R3;
        R4:=R5+R3+CTBASE; RO:=CTLIMIT; IF R4>=RO THEN
        BEGIN RO:=12; XR:=RO; ERROR; END;
        R5:=R5 CR #20;
        R4:=CTBASE+R3; STC(R5,B4);
            R5:=R5 AND #1F;
        R4:=R4+1; R5:=R5-1;
        EX(R5,MVC(0,B4,SBUF));
        R3:=R3+R5+2; CTPTR:=R3;
        RO:=1; SEM := RO; GCTC EXIT1;
      END;
L1: CCNCAT; GETCHAR; R2:=CLASS;
    IF R2=20 OR R2=21 THEN CCMMENT SIGNEC EXPCNENT;
```

```
    BEGIN
      CONCAT; GETCHAR; R2:=CLASS;
    WHILE R2=2 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
    END ELSE WHILE R2=2 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
  COMMENT LOAD CONSTANT IN TABLE ;
        R5:=SPTR+1; R3:=CTPTR; POINTER:=R3;
        R4:=R5+R3+CTBASE; R0:=CTLIMIT; IF R4>=R0 THEN
        BEGIN R0:=12; XR:=R0; ERROR; END;
        R5:=R5 OR #40;
        R4:=CTBASE+R3; STC(R5,B4);
            R5:=R5 AND #1F;
        R4:=R4+1; R5:=R5-1;
        EX(R5,MVC(0,B4,SBUF));
        R3:=R3+R5+2; CTPTR:=R3;
        R0:=1; SEM := R0;
EXIT1:
  END; COMMENT END CASE 1;

  BEGIN COMMENT  CASE 2 HANDLES DECIMAL CONSTANTS;
    CONCAT; GETCHAR; R2:=CLASS;
    WHILE R2=2 OR R2=3 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
    IF R2=8 THEN CONCAT COMMENT DECIMAL POINT;
    ELSE BEGIN
      IC(R1,CHAR); R1:=R1 AND #FF;
      IF R1=#C5 THEN GOTO L2 COMMENT FLOATIN POINT CONSTANT;
      ELSE COMMENT LOAD CONSTANT IN TABLE AND GO TO EXIT;
      BEGIN
        R5:=SPTR+1; R3:=CTPTR; POINTER:=R3;
        R4:=R5+R3+CTBASE; R0:=CTLIMIT; IF R4>=R0 THEN
        BEGIN R0:=12; XR:=R0; ERROR; END;
        R5:=R5 OR #60;
        R4:=CTBASE+R3; STC(R5,B4);
            R5:=R5 AND #1F;
        R4:=R4+1; R5:=R5-1;
        EX(R5,MVC(0,B4,SBUF));
        R3:=R3+R5+2; CTPTR:=R3;
        R0:=1; SEM := R0; GOTO EXIT2;
      END;
    END;
    GETCHAR; R2:=CLASS;
    WHILE R2=2 OR R2=3 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
    IC(R1,CHAR); R1:=R1 AND #FF;
    IF R1=#C5 THEN GOTO L2 COMMENT FLOATING POINT CONSTANT;
    ELSE COMMENT      LOAD CONSTANT IN TABLE AND GO TO EXIT;
      BEGIN
        R5:=SPTR+1; R3:=CTPTR; POINTER:=R3;
        R4:=R5+R3+CTBASE; R0:=CTLIMIT; IF R4>=R0 THEN
        BEGIN R0:=12; XR:=R0; ERROR; END;
```

```
        R5:=R5 OR #80;
        R4:=CTBASE+R3; STC(R5,B4);
            R5:=R5 AND #1F;
        R4:=R4+1; R5:=R5-1;
        EX(R5,MVC(0,B4,SBUF));
        R3:=R3+R5+2; CTPTR:=R3;
        R0:=1; SEM := R0; GOTO EXIT2;
      END;
L2: CONCAT; GETCHAR; R2:=CLASS;
    IF R2=20 OR R2=21 THEN COMMENT SIGNED EXPONENT;
    BEGIN
      CONCAT; GETCHAR; R2:=CLASS;
    WHILE R2=2 OR R2=3 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
    END ELSE WHILE R2=2 OR R2=3 DO BEGIN CONCAT; GETCHAR; R2:=CLASS;END;
  COMMENT LOAD CONSTANT IN TABLE ;
        R5:=SPTR+1; R3:=CTPTR; POINTER:=R3;
        R4:=R5+R3+CTBASE; R0:=CTLIMIT; IF R4>=R0 THEN
        BEGIN RC:=12; XR:=R0; ERROR; END;
        R5:=R5 OR #A0;
        R4:=CTBASE+R3; STC(R5,B4);
            R5:=R5 AND #1F;
        R4:=R4+1; R5:=R5-1;
        EX(R5,MVC(0,B4,SBUF));
        R3:=R3+R5+2; CTPTR:=R3;
        R0:=1; SEM := R0;
EXIT2:
  END; COMMENT  END CASE 2;

  BEGIN COMMENT CASE 3 IS JUST ANOTHER ENTRY TO CASE 2 DUE TO THE
                SEPARATION OF DIGITS INTO CLASS 2 SET (OCTAL DIGITS)
                AND CLASS 3 (8,9);
    R2:=2; GOTO STARTSCAN;
  END; COMMENT END OF CASE 3;

  BEGIN COMMENT CASE 4 HANDLES IDENTIFIERS;
    CONCAT; GETCHAR; R2:=CLASS;
    WHILE R2<6 AND R2>1 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
    IF R2=27 THEN COMMENT  IT CAN BE A RESERVED WORD AND ":", OR IT CAN
                  BE A LABEL DECLARATION;
    BEGIN
      RESERWRD; R0:=SEM;
      IF R0=0 THEN COMMENT IT IS NOT A RESERVED WORD SO IT MUST BE A
                   LEGAL OR ILLEGAL LABEL DECLARATION;
      BEGIN
          LOOKUP; R0:=SEM;
          IF R0=0 OR R0=7 THEN COMMENT IT IS A  LEGAL LABEL DECL. ;
            BEGIN R1:=SEMBASE+POINTER; R2:=6; STC(R2,B1); SEM:=R2; END
            ELSE BEGIN COMMENT ILLEGAL LABEL;
```

```
       RO:=9; XR:=RO; ERROR; GETCHAR; R2:=CLASS;
       RO:=_1; SPTR:=RO; MVC(21,SBUF,BLANK); GOTO STARTSCAN;
       END;
   GETCHAR;
  END;
 END ELSE COMMENT LOOKUP SYMBOL TABLE FOR RESERVED WORD;
 BEGIN
  RESERWRD; RO:=SEM;
  IF RO=0 THEN COMMENT IT IS NOT A RESERVED WORD;
  BEGIN
   LOOKUP; RO:=SEM;
   IF RO=0 THEN COMMENT A NEW ENTRY;
    BEGIN R1:=SEMBASE+POINTER; R2:=7; STC(R2,B1); SEM:=R2; END;
  END;
 END;
 END; COMMENT END OF CASE 4;

 BEGIN COMMENT CASE 5 IS JUST ANOTHER ENTRY FOR CASE 4 DUE TO
               SPLITTING OF LETTERS INTO CLASS 4 (HEX NUMBERS) AND
               CLASS 5 (G TO Z AND _);
  R2:=4; GOTO STARTSCAN;
 END; COMMENT END OF CASE 5;

 BEGIN COMMENT  CASE 6 HANDLES HEXADECIMAL CONSTANTS;
  GETCHAR; R2:=CLASS;
  WHILE R2<5 AND R2>1 DO BEGIN CONCAT; GETCHAR; R2:=CLASS; END;
COMMENT NOW ENTER HEX CONSTANT INTO CONSTANT TABLE AND OUTPUT SEMANTICS;
   R5:=SPTR+1;
  IF R5=0 OR R2=5 THEN
  BEGIN
   R5:=7; XR:=R5; ERROR; RO:=_1; SPTR:=RO; MVC(21,SBUF,BLANK);
   GOTO STARTSCAN;
  END
   ELSE BEGIN
    R3:=CTPTR; POINTER:=R3;
    R4:=R5+R3+CTBASE; RO:=CTLIMIT; IF R4>=RO THEN
    BEGIN RO:=12; XR:=RO; ERROR; END;
    R4:=CTBASE+R3; STC(R5,B4);
    R4:=R4+1; R5:=R5-1;
  EX(R5,MVC(0,B4,SBUF));
    R3:=R3+R5+2; CTPTR:=R3;
  RO:=2; SEM:=RO;
   END;
 END; COMMENT  END OF CASE 6;

 BEGIN COMMENT CASE 7 HANDLES "@" ;
  GETCHAR; RO:=30; SEM:=RO;
 END; COMMENT END CASE 7;
```

```
BEGIN COMMENT CASE 8 HANDLES ".."_SCALE SYMBOL;
  GETCHAR; R2:=CLASS;
  IF R2¬=8 THEN COMMENT "." IS DECIMAL POINT OF NUMERIC CONSTANT;
  BEGIN
    R2:=2; GOTO STARTSCAN;
  END;
  GETCHAR;
  RO:=11; SEM:=RO;
END; COMMENT END CASE 8;

BEGIN COMMENT CASE 9 HANDLES "¬" AND "¬=";
  GETCHAR; R2:=CLASS;
  IF R2=23 THEN COMMENT OUTPUT ¬= SEMANTICS;
  BEGIN RO:=26; SEM:=RO;
  CONCAT; GETCHAR;
  END
  ELSE  COMMENT OUTPUT ¬ SEMANTICS;
  BEGIN
    RC:=22; SEM:=RO;
  END;
END; COMMENT END OF CASE 9;

BEGIN COMMENT CASE 10 HANDLES ">";
  GETCHAR; RO:=27; SEM:=RO;
END; COMMENT END CASE 10;

BEGIN COMMENT CASE 11 HANDLES "<";
  GETCHAR; RO:=12; SEM:=RC;
END; COMMENT END CASE 11;

BEGIN COMMENT CASE 12 HANDLES COMMENTS;
  GETCHAR; R2:=CLASS;
  WHILE R2¬=12 DO BEGIN GETCHAR; R2:=CLASS; END;
  GETCHAR; R2:=CLASS; GOTO STARTSCAN;
END; COMMENT END OF CASE 12;

BEGIN COMMENT CASE 13 HANDLES "*" AND"**";
  GETCHAR; R2:=CLASS;
  IF R2=13 THEN COMMENT OUTPUT ** SEMANTICS;
  BEGIN
    RO:=19; SEM:=RO;
    GETCHAR;
  END
  ELSE COMMENT OUTPUT * SEMANTICS;
  BEGIN
    RO:=18; SEM:=RO;
  END;
```

```
END; COMMENT END OF CASE 13;

BEGIN COMMENT CASE 14 HANDLES STRING CONSTANTS;
  GETCHAR; R2:=CLASS;
R3:=CTPTR; POINTER:=R3; R1:=CTBASE+R3+1; R5:=R5-R5;
  WHILE R2¬=14 DO COMMENT LOAD CHARACTER STRING INTO CONSTANT TABLE;
  BEGIN
    IF R1>= CTLIMIT THEN
      BEGIN RC:=12; XR:=R0; ERROR; END;
  IC(R0,CHAR); STC(R0,B1);                   R1:=R1+1; R5:=R5+1;
    GETCHAR; R2:=CLASS;
  END;
R4:=CTBASE+R3; STC(R5,B4); R4:=R3+R5+1; CTPTR:=R4;
  GETCHAR; R0:=3; SEM:=R0;
END; COMMENT END CASE 14;

BEGIN CCMMENT CASE 15 HANDLES STATUS CONSTANTS;
  GETCHAR; R2:=CLASS;
  IF R2¬=4 AND R2¬=5 THEN COMMENT ILLEGAL INITIAL CHARACTER;
  BEGIN
    R0:=1; XR:=RC; ERRCR;
    WHILE R2¬=15 DO BEGIN GETCHAR; R2:=CLASS; END;
    GETCHAR; R2:=CLASS;  GCTO STARTSCAN;
  END
  ELSE
    WHILE R2<6 AND R2>1 DC BEGIN CCNCAT; GETCHAR; R2:=CLASS; END;
    IF R2¬=15 THEN COMMENT ILLEGAL SYMBOL;
    BEGIN
      RC:=1; XR:=R0; ERROR;
      WHILE R2¬=15 DO BEGIN GETCHAR; R2:=CLASS; END;
      GETCHAR; R2:=CLASS;
      R0:=_1; SPTR:=R0; MVC(21,SBUF,BLANK);
      GOTO STARTSCAN;
    END
    ELSE CCMMENT OUTPUT STATUS CONSTANT SEMANTICS;
  BEGIN
  LCCKUP; R0:=SEM;
  IF RC=0 THEN CCMMENT NEW ENTRY;
  BEGIN
    R1:=SEMBASE+POINTER; R2:=4;
    STC(R2,B1); SEM:=R2;
  END ELSE
  BEGIN
    IF R0 ¬= 4 THEN CCMMENT ILLEGAL STATUS CCNSTANT IDENTIFIER;
    BEGIN R0:=11; XR:=R0; ERROR; END ELSE SEM:=RC;
  END;
END;
  GETCHAR;
```

```
END; COMMENT END OF CASE 15;

BEGIN COMMENT CASE 16 HANDLES "$";
  GETCHAR; RO:=17; SEM:=RO;
END; COMMENT  END CASE 16;

BEGIN COMMENT CASE 17 HANDLES ",";
  GETCHAR; RO:=25; SEM:=RO;
END; COMMENT END OF CASE 17;

BEGIN COMMENT CASE 18 HANDLES "(";
  GETCHAR; RO:=13; SEM:=RO;
END; COMMENT END OF CASE 18;

BEGIN COMMENT CASE 19 HANDLES ")";
  GETCHAR; RO:=20; SEM:=RO;
END; COMMENT END OF CASE 19;

BEGIN COMMENT CASE 20 HANDLES "+";
  GETCHAR; RO:=14; SEM:=RO;
END; COMMENT END OF CASE 20;

BEGIN COMMENT CASE 21 HANDLES "-";
  GETCHAR; RO:=23; SEM:=RO;
END; COMMENT END OF CASE 21;

BEGIN COMMENT CASE 22 HANDLES "/";
  GETCHAR; RO:=24; SEM:=RO;
END; COMMENT END OF CASE 22;

BEGIN COMMENT CASE 23 HANDLES "=";
  GETCHAR; RO:=29; SEM:=RO;
END; COMMENT END OF CASE 23;

BEGIN COMMENT CASE 24 HANDLES SEMICOLON;
  GETCHAR; RO:=21; SEM:=RO;
END; COMMENT END OF CASE 24;

BEGIN COMMENT CASE 25 HANDLES "|" AND "||";
  GETCHAR; R2:=CLASS;
  IF R2=25 THEN COMMENT OUTPUT "||" SEMANTICS;
  BEGIN
    RO:=16; SEM:=RO;
    GETCHAR;
  END
  ELSE COMMENT OUTPUT "|" SEMANTICS;
    BEGIN RO:=15; SEM:=RO; END;
END; COMMENT END OF CASE 25;
```

```
   BEGIN COMMENT CASE 26 HANDLES ILLEGAL CHARACTERS;
     IC(R1,CHAR); R1:=R1 AND #FF;
     IF R1=#4C THEN
     BEGIN
       WHILE R1=#40 DO BEGIN GETCHAR; IC(R1,CHAR); R1:=R1 AND #FF; END;
       R2:=CLASS; GOTO STARTSCAN;
     END ELSE
     BEGIN
     R0:=2; XR:=R0; ERRCR; GETCHAR; R2:=CLASS; GCTC STARTSCAN;
     END;
     END; CCMMENT END OF CASE 26;

   BEGIN COMMENT CASE 27 HANDLES ":" WHEN USED CTHER THAN WITH ICENTS
                 TO FORM LABELS;
     GETCHAR; RC:=28; SEM:=R0;
   END ; COMMENT END OF CASE 27;

 END; CCMMENT END OF CASE ELCCK;
   LM(RC,R5,SAVEREGS);
   END; CCMMENT END OF SCANNER;


 GLOEAL PROCEDURE SCAN(R4); COMMENT THIS PROCEDURE CCNTRCLS THE OUTPUT
                            OF THE SCANNER  AND THE INTERPHASE BETWEEN
                            BOTH GRAMMARS;
 BEGIN
   ARRAY 4 INTEGER SAVER; STM(R0,R3,SAVER);
 IF ENDIT THEN BEGIN R1:=1; TOKEN:=R1; ETCKEN:=R1; END
 ELSE BEGIN
       SCANNER;
   R2:=SEM;
   CASE R2 OF
   BEGIN
     BEGIN R1:=0049; TOKEN:=R1; R1:=024; BTOKEN:=R1; END; COMMENT "1-
                                                          NUMERIC CONST.";
     BEGIN R1:=0C59; TOKEN:=R1; R1:=035; BTOKEN:=R1; END; COMMENT "2-
                                                            BITS CCNST." ;
     BEGIN R1:=0064; TOKEN:=R1; R1:=036; BTOKEN:=R1; END; COMMENT "3-
                                                       CHARACTER CCNST.";
     BEGIN R1:=0030; TOKEN:=R1; R1:=034; BTOKEN:=R1; END; COMMENT "4-
                                                          STATUS CONST.";
     BEGIN R1:=0074; TOKEN:=R1; END; CCMMENT "5-BCCLEAN CCNSTANT";
     BEGIN R1:=0014; TOKEN:=R1; END; COMMENT "6- LABEL";
     BEGIN COMMENT "7- IDENT" ;
        IF ENDIT THEN BEGIN R1:=1; TOKEN:=R1; ETCKEN:=R1; END
            ELSE BEGIN
                 IF PCAL THEN BEGIN
```

```
                                    R1:=C098; TOKEN:=R1; RESET(PCAL);
                                END ELSE
            IF IDISLBL THEN BEGIN R1:=0014; TCKEN:=R1; RESET(ICISLBL);
                            END ELSE
        BEGIN R1:=0003; TCKEN:=R1; R1:=006; BTCKEN:=R1; END;
      END;
   END;
   BEGIN R1:=0050; TOKEN:=R1; R1:=039; BTOKEN:=R1; END; COMMENT "8-
                                                             TAG";
   NULL; NULL;
   BEGIN R1:=0047; TOKEN:=R1; END; COMMENT "11- ..";
   BEGIN R1:=0080; TOKEN:=R1; END; COMMENT "12- <";
   BEGIN R1:=0046; TOKEN:=RI; R1:=007; BTOKEN:=R1; END; COMMENT "13-(";
   BEGIN R1:=CC39; TOKEN:=R1; END; COMMENT "14- +";
   BEGIN R1:=0089; TOKEN:=R1; R1:=0C9; BTOKEN:=R1; END; COMMENT "15-|";
   BEGIN R1:=0090; TOKEN:=R1; R1:=010; BTOKEN:=R1; END; COMMENT "16-||";
   BEGIN COMMENT "17- $";
     IF BENDIT THEN BEGIN R1:=003; BTOKEN:=R1; SET(ENDIT); END
     ELSE BEGIN R1:=0004; TCKEN:=R1; R1:=003; BTOKEN:=R1; END;
   END;
   BEGIN R1:=0041; TOKEN:=R1; END; COMMENT "18- *";
   BEGIN R1:=0044; TOKEN:=R1; END; COMMENT "19- **";
   BEGIN R1:=CC19; TOKEN:=R1; R1:=004; BTOKEN:=R1; END; COMMENT "20-)";
NULL; NULL;
   BEGIN R1:=CC4C; TOKEN:=R1; END; COMMENT "23. -";
   BEGIN R1:=0042; TOKEN:=R1; R1:=042; BTOKEN:=R1; END; COMMENT "24-/";
   BEGIN R1:=0022; TOKEN:=R1; R1:=008; BTOKEN:=R1; END; COMMENT "25-,";
   BEGIN R1:=CC79; TOKEN:=R1; END; COMMENT "26- ¬=";
   BEGIN R1:=0081; TOKEN:=R1; END; COMMENT "27- >";
   BEGIN R1:=0029; TOKEN:=R1; END; COMMENT "28- :";
   BEGIN R1:=0078; TOKEN:=R1; R1:=023; BTOKEN:=R1; END; COMMENT "29-=";
NULL;
   BEGIN R1:=0023; TOKEN:=R1; R1:=012; BTOKEN:=R1; END; COMMENT "31-
                                                             NUMERIC VAR";
   BEGIN R1:=0048; TOKEN:=R1; END; COMMENT "32- NUM FUNCTION NAME";
   BEGIN R1:=0051; TOKEN:=R1; R1:=048; BTOKEN:=R1; END; COMMENT "33-
                                                             NUM FLD NAME";
   BEGIN R1:=0061; TOKEN:=R1; R1:=013; BTOKEN:=R1; END; COMMENT "34-
                                                             BITS VAR";
   BEGIN R1:=0060; TOKEN:=R1; END; COMMENT "35- BITS FUNCTION NAME";
   BEGIN R1:=0062; TOKEN:=R1; R1:=049; BTOKEN:=R1; END; COMMENT "36-
                                                             BITS FLD NAME";
   BEGIN R1:=0066; TOKEN:=R1; R1:=014; BTOKEN:=R1; END; COMMENT "37-
                                                             CHAR VAR";
   BEGIN R1:=0065; TOKEN:=R1; END; COMMENT "38- CHAR FUNCTION NAME";
   BEGIN R1:=0067; TOKEN:=R1; R1:=050; BTOKEN:=R1; END; COMMENT "39-
                                                             CHAR FLD NAME";
   BEGIN R1:=0069; TOKEN:=R1; R1:=015; BTOKEN:=R1; END; COMMENT "40-
```

```
                                                          STAT VAR";
    BEGIN R1:=0068; TOKEN:=R1; END; COMMENT "41- STAT FUNCTION NAME";
    BEGIN R1:=0070; TOKEN:=R1; R1:=051; BTOKEN:=R1; END; COMMENT "42-
                                                          STAT FLD NAME";
    BEGIN R1:=0076; TOKEN:=R1; R1:=016; BTOKEN:=R1; END; COMMENT "43-
                                                          BOOLEAN VAR";
    BEGIN R1:=0075; TOKEN:=R1; END; COMMENT "44- BOOL FUNCTION NAME";
    BEGIN R1:=0077; TOKEN:=R1; R1:=052; BTOKEN:=R1; END; COMMENT "45-
                                                          BOOL FLD NAME";
    BEGIN R1:=0082; TOKEN:=R1; R1:=017; BTOKEN:=R1; END; COMMENT "46-
                                                          TABLE NAME";
    BEGIN R1:=0088; TOKEN:=R1; END; COMMENT "47- PROCEDURE NAME";
    BEGIN R1:=0097; TOKEN:=R1; END; COMMENT "48- FORMAT NAME";
    BEGIN R1:=0084; TOKEN:=R1; R1:=018; BTOKEN:=R1; END; COMMENT "49-
                                                        ITEM_AREA NAME";
    BEGIN R1:=0083; TOKEN:=R1; R1:=019; BTOKEN:=R1; END; COMMENT "50-
                                                          SUBTABLE NAME";
NULL; NULL; NULL; NULL;
    BEGIN R1:=0045; TOKEN:=R1; END; COMMENT "55- ABS";
    BEGIN COMMENT "56- AUTO_DD";
      R1:=022; BTOKEN:=R1;
      R15:=BANALBASE;
      BALR(R4,R15);
      R15:=SCANBASE;
      RESET(ENDIT); RESET(BENDIT);
      R1:=0012; TOKEN:=R1; COMMENT TOKEN FOR <AUTO DATA DECL> ;
      R4:=RTRNTOANAL;
    END;
    BEGIN R1:=0071; TOKEN:=R1; END; COMMENT "57- AND";
    BEGIN R1:=0052; TOKEN:=R1; END; COMMENT "58- ANDL";
    BEGIN R1:=0036; TOKEN:=R1; END; COMMENT "59- BEGIN";
    BEGIN R1:=030; BTOKEN:=R1; END; COMMENT "60- BITS";
    BEGIN R1:=0060; TOKEN:=R1; END; COMMENT "61- BITSTRING";
    BEGIN R1:=032; BTOKEN:=R1; END; COMMENT "62- BOOLEAN";
    BEGIN R1:=0016; TOKEN:=R1; SET(PCAL); END; COMMENT "63- CALL";
    BEGIN R1:=0027; TOKEN:=R1; END; COMMENT "64- CASE";
    BEGIN R1:=0063; TOKEN:=R1; END; COMMENT "65- CAT";
    BEGIN R1:=0056; TOKEN:=R1; END; COMMENT "66- CIRSHLL";
    BEGIN R1:=0057; TOKEN:=R1; END; COMMENT "67- CIRSHRL";
    BEGIN R1:=0048; TOKEN:=R1; END; COMMENT "68- CHARCODE";
    BEGIN R1:=0065; TOKEN:=R1; END; COMMENT "69- CODECHAR";
    BEGIN R1:=0100; TOKEN:=R1; END; COMMENT "70- DECODE";
    BEGIN R1:=0033; TOKEN:=R1; END; COMMENT "71- DC";
    BEGIN R1:=0031; TOKEN:=R1; END; COMMENT "72- ELSE";
    BEGIN R1:=0099; TOKEN:=R1; END; COMMENT "73- ENCODE";
    BEGIN R1:=0013; TOKEN:=R1; END; COMMENT "74- END";
    BEGIN R1:=002; BTOKEN:=R1; SET(BENDIT); END; COMMENT "75- END_DD";
    BEGIN R1:=025; BTOKEN:=R1; END; COMMENT "76- END_TABLE";
```

```
    BEGIN R1:=0007; TOKEN:=R1; END; COMMENT "77- EXTREF";
    BEGIN R1:=045; BTOKEN:=R1; END; COMMENT "78- FIELD";
    BEGIN R1:=028; BTOKEN:=R1; END; COMMENT "79- FIXED";
    BEGIN R1:=029; BTOKEN:=R1; END; COMMENT "80- FLOAT";
    BEGIN R1:=0087; TOKEN:=R1; END; COMMENT "81- FOR";
    BEGIN R1:=041; BTOKEN:=R1; END; COMMENT "82- FORMAT";
    BEGIN R1:=0024; TOKEN:=R1; END; COMMENT "83- FROM";
  NULL;
    BEGIN COMMENT "85- FUNCTION";
      R1:=011; BTOKEN:=R1;
      SET(BENDIT);
      R15:=BANALBASE;
      BALR(R4,R15);
      R15:=SCANBASE;
      RESET(BENDIT); RESET(ENDIT);
      R1:=0011; TOKEN:=R1; COMMENT TOKEN FOR <SUB_ROUTINE DECL> ;
      R4:=RTRNTOANAL;
    END;
    BEGIN R1:=0093; TOKEN:=R1; SET(IDISLBL); END; COMMENT "86- GOTO";
    BEGIN R1:=0034; TOKEN:=R1; END; COMMENT "87- IF";
    BEGIN R1:=0096; TOKEN:=R1; END; COMMENT "88- INPUT";
    BEGIN R1:=027; BTOKEN:=R1; END; COMMENT "89- INTEGER";
    BEGIN R1:=0103; TOKEN:=R1; END; COMMENT "90- INTO";
    BEGIN R1:=043; BTOKEN:=R1; END; COMMENT "91- INDIRCT";
    BEGIN R1:=046; BTOKEN:=R1; END; COMMENT "92- ITEM_AREA";
    BEGIN COMMENT "93- LOC_DD";
      R1:=021; BTOKEN:=R1;
      R15:=BANALBASE;
      BALR(R4,R15);
      R15:=SCANBASE;
      RESET(ENDIT); RESET(BENDIT);
      R1:=0010; TOKEN:=R1; COMMENT TOKEN FOR <LOCAL DATA DECL> ;
      R4:=RTRNTOANAL;
    END;
    BEGIN R1:=0073; TOKEN:=R1; END; COMMENT "94- NOT";
    BEGIN R1:=0058; TOKEN:=R1; END; COMMENT "95- NCTL";
    BEGIN R1:=0048; TOKEN:=R1; END; COMMENT "96- NUMBER";
    BEGIN R1:=0108; TOKEN:=R1; END; COMMENT "97- OCM";
    BEGIN R1:=0072; TOKEN:=R1; END; COMMENT "98- CR";
    BEGIN R1:=0053; TOKEN:=R1; END; COMMENT "99- CRL";
    BEGIN R1:=0028; TOKEN:=R1; END; COMMENT "100- CF";
    BEGIN R1:=0104; TOKEN:=R1; END; COMMENT "101- CUTOF";
    BEGIN R1:=0098; TOKEN:=R1; END; COMMENT "102- OUTPUT";
    BEGIN R1:=032; BTOKEN:=R1; END; COMMENT "103- OVERLAY";
    BEGIN R1:=0101; TOKEN:=R1; END; COMMENT "104- PACKIN";
    BEGIN R1:=0105; TOKEN:=R1; END; COMMENT "105- PRINT";
    BEGIN COMMENT "106- PROCEDURE";
      R1:=005; BTOKEN:=R1;
```

```
     SET(BENDIT);
     R15:=BANALBASE;
     BALR(R4,R15);
     R15:=SCANBASE;
     RESET(BENDIT); RESET(ENDIT);
     R1:=0011; TOKEN:=R1; CCMMENT TCKEN FOR <SUB_ROUTINE DECL> ;
     R4:=RTRNTCANAL;
   ENC;
   BEGIN R1:=0106; TOKEN:=R1; ENC; COMMENT "1C7- PNCH";
   BEGIN R1:=0107; TOKEN:=R1; ENC; CCMMENT "1C8- REAC";
   BEGIN R1:=0043; TOKEN:=R1; END; COMMENT "1C9- REM";
   BEGIN R1:=0091; TOKEN:=R1; SET(IDISLBL); ENC; COMMENT "110- RESUME";
   BEGIN R1:=0C17; TOKEN:=R1; END; CCMMENT "111- RETURN";
BEGIN R1:=0092; TOKEN:=R1; SET(IDISLBL); END; CCMMENT "112-RETURNTC";
   BEGIN R1:=0086; TOKEN:=R1; ENC; CCMMENT "113- SEARCH";
 NULL;
   BEGIN R1:=0054; TOKEN:=R1; ENC; CCMMENT "115- SHLL";
   BEGIN R1:=0C55; TOKEN:=R1; END; CCMMENT "116- SHRL";
   BEGIN R1:=0025; TOKEN:=R1; END; CCMMENT "117- STEP";
   BEGIN R1:=0065; TOKEN:=R1; ENC; COMMENT "118- SUBCHAR";
   BEGIN R1:=0002; TOKEN:=R1; ENC; CCMMENT "119- SYSTEM";
   BEGIN COMMENT  "120- SYS_DD";
     MVC(1,PREFIX(0),"00");
     R1:=02C; BTOKEN:=R1;
     R15:=BANALBASE;
     BALR(R4,R15);
     R15:=SCANBASE;
     RESET(ENDIT); RESET(BENDIT);
     R1:=0006; TOKEN:=R1; CCMMENT TCKEN FOR <SYS CATA DECL> ;
     R4:=RTRNTOANAL;
   ENC;
   BEGIN COMMENT  "121- SYS_PROC";
     R1:=BLKCTR+1; BLKCTR:=R1;
     CVD(R1,CCNWORK);
     UNPK(3,7,TEMPREFIX(0),CCNWCRK); SETZONE(TEMPREFIX(3));
     MVC(1,PREFIX(0),TEMPREFIX(2));
     R1:=0008; TCKEN := R1;
   END;
   BEGIN COMMENT "122- SYS_PROC_R";
     R1:=BLKCTR+1; BLKCTR:=R1;
     CVD(R1,CONWORK);
     UNPK(3,7,TEMPREFIX(0),CONWORK); SETZONE(TEMPREFIX(3));
     MVC(1,PREFIX(0),TEMPREFIX(2));
     R1:=00C9; TOKEN:=R1;
   ENC;
   BEGIN R1:=047; BTOKEN:=R1; END; CCMMENT "123- SUBTABLE";
   BEGIN R1:=031; BTOKEN:=R1; END; CCMMENT "124- CHAR";
   BEGIN R1:=033; BTOKEN:=R1; ENC; CCMMENT "125- STATUS";
```

```
    BEGIN R1:=0038; TOKEN:=R1; END; COMMENT "126- SET";
    BEGIN R1:=0085; TOKEN:=R1; END; COMMENT "127- SWAP";
    BEGIN R1:=040; BTOKEN:=R1; END; COMMENT "128- TAG";
    BEGIN R1:=044; BTOKEN:=R1; END; COMMENT "129- TABLE";
    BEGIN R1:=0015; TOKEN:=R1; END; COMMENT "130- TO";
    BEGIN R1:=0026; TOKEN:=R1; END; COMMENT "131- THRU";
    BEGIN R1:=0035; TOKEN:=R1; END; COMMENT "132- THEN";
    BEGIN R1:=0102; TOKEN:=R1; END; COMMENT "133- UNPCK";
    BEGIN R1:=0021; TOKEN:=R1; END; COMMENT "134- VARY";
    BEGIN R1:=026; BTOKEN:=R1; END; COMMENT "135- VRBL";
    BEGIN R1:=0032; TOKEN:=R1; END; COMMENT "136- WHILE";
    BEGIN                      R1:=038; BTOKEN:=R1; END; COMMENT "137-
                                                                  FALSE";
    BEGIN                      R1:=037; BTOKEN:=R1; END; COMMENT "138-
                                                              TRUE";
    BEGIN R1:=0037; TOKEN:=R1; END; COMMENT "139- COBEGIN";
    BEGIN R1:=0018; TOKEN:=R1; END; COMMENT "140- COEND";
  NULL;
    BEGIN R1:=0094; TOKEN:=R1; END; COMMENT "142- RESERVE";
     BEGIN R1:=0095; TOKEN:=R1; END; COMMENT "143-WAIT";
  BEGIN R1:=0052; TOKEN:=R1; END; COMMENT "144- CCRAD";
  BEGIN R1:=C051; TOKEN:=R1; END; COMMENT "145- COUNT";
  BEGIN R1:=0020; TOKEN:=R1; END; COMMENT "146- STOP";
  BEGIN R1:=0075; TOKEN:=R1; END; COMMENT "147- INTERRUPT";
  BEGIN R1:=0075; TOKEN:=R1; END; COMMENT "148- SUBSCRPT";
  END; COMMENT END OF CASE STATEMENT;
END;
  LM(RO,R3,SAVER);
END; COMMENT END OF SCAN;


  PROCEDURE SYNTHESIZE(R4); COMMENT IT HANDLES THE SEMANTICS OF THE
                            MAIN GRAMMAR;
  BEGIN ARRAY 5 INTEGER SAVEREGS; STM(RO,R4,SAVEREGS);
    R2:=PRODNUM;
CVD(R2,CONWORK); UNPK(3,7,WBUF(12),CONWORK); SETZONE(WBUF(15));
MVC(7,WBUF,"PROD NO.");
RO:=@WBUF; WRITE; MVC(25,WBUF,BLANK);
  COMMENT CASE R2 OF
          BEGIN
           236 CASES (ONE PER PRODUCTION PLUS ONE MORE)
          END;
    LM(RO,R4,SAVEREGS);
  END; COMMENT END OF SYNTHESIZE;

  GLOBAL PROCEDURE BSYNTHTWO(R4); COMMENT CONTINUATION OF BSYNTHESIZE
                                         (CASES 64 THRU 97);
  BEGIN ARRAY 7 INTEGER SAVEREGS; STM(RO,R6,SAVEREGS)         ;
```

```
   R2:=BPROCNUM;
STARTBSYN:
   R2:=R2-63;
   CASE R2 OF
   BEGIN
 BEGIN COMMENT 64- <FORMAT DECL HEAD>::=FORMAT <ID> (<CHAR CONSTANT>,;
   R1:=BSP-3;  R1:=R1 SHLL 1;
   R2:=R10+BPTRSTACK+R1;
   LH(R1,B2);  R3:=SEMBASE+R1;
   R0:=48;  STC(R0,B3);
   R2:=R2-2; STH(R1,B2);
   R2:=R2+6; LH(R1,B2);
   R2:=CTBASE+R1;  IC(R1,B2);  R1:=R1 AND #FF;
   IF R1¬=#1 THEN BEGIN R0:=15; XR:=R0;  ERROR; END
   ELSE BEGIN
     R2:=R2+1;  IC(R1,B2);
     R2:=R10+DESCRBUF; STC(R1,B2);
     R0:=1; DPTR:=R0;
   END;
 END;

   NULL;
 BEGIN COMMENT 66.-<DSCRLIST HD>::=<REAL CONST>   (      ;
   R1:=BSP-1; R1:=R1 SHLL 1;
   R2:=R10+BPTRSTACK+R1; LH(R1,B2);
   NWD; R3:=#8000 OR R1;
   R1:=DPTR; R2:=R10+DESCRBUF+R1;
   R4:=R3 SHRL 8; STC(R4,B2);
   R2:=R2+1; STC(R3,B2);
   R1:=R1+2; DPTR:=R1;
 END;
 NULL; NULL;


 BEGIN COMMENT  69.-<REPTN GROUP>::=<DSCRLIST>   )                ;
   R1:=DPTR; R2:=R10+DESCRBUF+R1;
   R3:=#A0; STC(R3,B2);
   R1:=R1+1; DPTR:=R1;
 END;

 NULL; NULL;
 BEGIN COMMENT 72.-<DESCRIPTOR>::=<REAL CONST> <ID> <RELA CONST>   ;
   R1:=BSP-1; R1:=R1 SHLL 1;
   R2:=R10+BPTRSTACK+R1;
   LH(R1,B2); R1:=R1 AND #FFFF;    CHECKID;
   IF R1=_1 THEN BEGIN R0:=16; XR:=R0; ERROR; END
   ELSE BEGIN
     R0:=DPTR; R2:=R10+DESCRBUF+R0;
```

```
    R3:=#20 OR R1; STC(R3,B2);
    R2:=R2+1; R1:=BSP-2;
    R1:=R1 SHLL 1; R3:=R10+BPTRSTACK+R1;
    LH(R1,B3); NWD;
    STC(R1,B2); R2:=R2+1;
    R3:=R3+4; LH(R1,B3);
    NWD; STC(R1,B2); R2:=R2+1;
    STC(R0,B2); R0:=DPTR+4; DPTR:=R0;
    ENC;
  ENC;

  BEGIN COMMENT 73.- <DESCRIPTOR>::=<ID> <REAL CCNSTANT>    ;
    R1:=BSP-1; R1:=R1 SHLL 1;
    R2:=R10+BPTRSTACK+R1;
    LH(R1,B2); R1:=R1 AND #FFFF;     CHECKID;
    IF R1=_1 THEN BEGIN R0:=16; XR:=R0; ERROR; END
    ELSE BEGIN
      R0:=DPTR; R2:=R10+DESCRBUF+R0;
      R3:=#20 OR R1; STC(R3,B2); R2:=R2+1;
      R3:=#1; STC(R3,B2); R2:=R2+1;
      R1:=BSP SHLL 1; R4:=R1C+BPTRSTACK+R1;
      LH(R1,B4); NWD; STC(R1,B2); R2:=R2+1;
      STC(R0,B2); R0:=DPTR+4; DPTR:=RC;
    ENC;
  END;

  BEGIN COMMENT 74-<DESCRIPTOR>::=<CHAR CONSTANT>     ;
    R1:=BSP SHLL 1; R0:=DPTR;
    R2:=R10+BPTRSTACK+R1;
    LH(R3,B2);
    R2:=R10+DESCRBUF+R0;
    R4:=#40; STC(R4,B2);
    R2:=R2+1; R4:=R3 SHRL 8;
    STC(R4,B2); R2:=R2+1;
    STC(R3,B2); R0:=R0+3;
    DPTR:=R0;
  END;

  BEGIN CCMMENT 75.-<DESCRIPTOR>::=<SLASH LIST>       ;
    R1:=DPTR; RC:=CTR;
    R2:=R10+DESCRBUF+R1;
    R3:=#60 CR R0;
    STC(R3,B2); R1:=R1+1; DPTR:=R1;
  ENC;

  BEGIN COMMENT 76-<SLASH LIST>::=/     ;
    R0:=1; CTR:=R0;
  END;
```

```
BEGIN COMMENT 77-<SLASH LIST>::=<SLASH LIST>/     ;
  R0:=CTR+1; CTR:=R0;
END;

BEGIN COMMENT 78.-<TABLE DECL HEAD>::=<TABLE CLAUSE>  )  $     ;
   R1:=CTR AND #7F;
   R2:=R10+DESCRBUF; STC(R1,B2);
   R0:=CTPTR; TBLDESCRB:=R0; R4:=CTBASE+R0;
   R1:=DPTR+2; STC(R1,B4);          R4:=R4+1;
   R1:=R1-1; EX(R1,MVC(0,B4,B2));
   R0:=R0+R1+1; CTPTR:=R0;
   R1:=TBLISTPTR;
   FOR R5:=0 STEP 2 UNTIL R1 DO
   BEGIN
      R2:=@TBLIST+R5; LH(R3,B2);
      R3:=R3 SHLL 1; R2:=DESCRPTRBASE+R3;
      STH(R0,B2);
   END;
END;

   NULL;

BEGIN COMMENT 80.-<TABLE CLAUSE>::=<TABLE HEAD>  (  <NUMORTAG>    ;
   R1:=BSP; DATASIZE;
   R2:=R10+DESCRBUF+1;
   R3:=R1 SHRL 8; STC(R3,B2);
   R2:=R2+1; STC(R1,B2);
   R0:=3; DPTR:=R0;
   R0:=1; CTR:=R0;
END;

BEGIN COMMENT 81.-<TABLE CLAUSE>::=INDIRCT <TABLE HEAD>  (  <NUMORTAG>
;  R1:=BSP; DATASIZE;
   R2:=R10+DESCRBUF; R3:=#80;
   STC(R3,B2);     R2:=R2+1;
   R3:=R1 SHRL 8; STC(R3,B2);
   R2:=R2+1; STC(R1,B2);
   R0:=3; DPTR:=R0;
   R0:=1; CTR:=R0;
END;

BEGIN COMMENT 82.-<TABLE CLAUSE>::=<TABLE CLAUSE>  ,  <NUMORTAG>   ;
   R1:=BSP; DATASIZE;
   R0:=DPTR; R2:=R10+DESCRBUF+R0;
   R3:=R1 SHRL 8; STC(R3,B2);
   R2:=R2+1; STC(R1,B2);
   R0:=R0+2; DPTR:=R0;
```

```
  R0:=CTR+1; CTR:=R0;
END;

BEGIN COMMENT 83.-<TABLE HEAD>::=TABLE <ID>    ;
  R1:=BSP SHLL 1; R2:=R10+BPTRSTACK+R1;
  LH(R1,B2); R2:=SEMBASE+R1;
  R0:=46; STC(R0,B2);
  R2:=@TBLIST; STH(R1,B2);
  R0:=2; TBLISTPTR:=R0;
END;

BEGIN COMMENT 84.-<TABLE HEAD>::=<TABLE HEAD>  ,  <ID>   ;
  R1:=BSP SHLL 1; R2:=R10+BPTRSTACK+R1;
  LH(R1,B2); R2:=SEMBASE+R1;
  R0:=46; STC(R0,B2);
  R0:=TBLISTPTR; R2:=@TBLIST+R0;
  STH(R1,B2); R0:=R0+2;
  TBLISTPTR:=R0;
END;

  NULL; NULL;

BEGIN COMMENT 87.-<TABLE DECL>::=<SUBTABLE CLAUSE>  )  $  ;
  R1:=BSP-2; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R1,B2);
  R1:=R1 SHLL 1;
  R2:=DESCRPTRBASE+R1; R0:=CTPTR;
  STH(R0,B2); R1:=TBLDESCRB;
  R3:=DPTR; R2:=R10+DESCRBUF+R3;
  STH(R1,B2); R4:=CTBASE+R0;
  R5:=R3+2; STC(R5,B4);
  R2:=R10+DESCRBUF; R3:=R3+1;
  R4:=R4+1; EX(R3,MVC(0,B4,B2));
END;

BEGIN COMMENT 88.- <FIELD DECL>::=FIELD <DATA TYPE>      ;
  R1:=BSP SHLL 1; R2:=R10+BPTRSTACK+R1;
  LH(R3,B2); R2:=R2-2; STH(R3,B2);
END;

BEGIN COMMENT 89.-<FIELD DECL>::=<FIELD DECL>  <FITEM>    ;
  R1:=BSP-1; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R3,B2);
  R3:=R3 AND #FF;
  CASE R3 OF
  BEGIN
    R0:=33; R0:=36; R0:=39; R0:=42; R0:=45;
  END;
```

```
   R2:=R2+2; LH(R4,B2);
   R1:=FLBPTR; R2:=@FLCPTRSBUF+R1;
   R5:=R4 SHRL 8; STC(R5,B2);
   R2:=R2+1; STC(R4,B2);
   R1:=R1+2; FLBPTR:=R1;
   R2:=SEMBASE+R4; STC(R0,B2);
   R4:=R4 SHLL 1; R2:=DESCRPTRBASE+R4;
   RC:=CTPTR; STH(R0,B2);
   R1:=DPTR-1; R2:=R10+DESCRBUF;
   R4:=CTBASE+R0;
   EX(R1,MVC(0,B4,B2));
   R0:=R0+R1+1; CTPTR:=R0;
   RC:=R0-RC; DPTR:=R0;
END;

BEGIN COMMENT 90.-<FITEM>::=<ITEM>      ;
   R1:=DPTR; R2:=R10+DESCRBUF+R1;
   R3:=TBLDESCRB; R4:=R3 SHRL 8;
   STC(R4,B2); R2:=R2+1;
   STC(R3,B2); R1:=R1+2;
   DPTR:=R1; R2:=R2-R1; IC(R3,B2);
   R3:=R3+2; STC(R3,B2);
END;

BEGIN COMMENT 91.-<FITEM>::=<ID> = <REPETITICN LIST> <CCNST> )   ;
   R0:=CTR+1; R1:=DPTR-1;
   R2:=R10+DESCRBUF; STC(R1,B2);
   R1:=R1-5; R2:=R10+DESCRBUF+R1;
   R3:=R0 SHRL 8; STC(R3,B2);
   R2:=R2+1; STC(R0,B2);
END;

BEGIN COMMENT 92.-<REPETITION LIST>::=  (        ;
   R1:=BSP-3; R1:=R1 SHLL 1;
   R2:=R10+BPTRSTACK+R1; LH(R1,B2);
   R1:=R1 AND #FF;
   IF R1<4 THEN
   BEGIN
     R1:=DATABUF; R2:=R10+DESCRBUF+1;
     R3:=R1 SHRL 8; STC(R3,B2);
   R2:=R2+1;
   STC(R1,B2); R2:=R2+3;
   RC:=5;
   ENC ELSE
   IF R1=5 THEN BEGIN R2:=R10+DESCRBUF+3; R0:=3; END;
   R2:=R2+2; R3:=1;
   R1:=R1 SHRL 8;
   STC(R1,B2); R2:=R2+1;
```

```
  STC(R3,B2); R2:=R2+1;
  R3:=TBLDESCRB; R1:=R3 SHRL 8;
  STC(R1,B2); R2:=R2+1;
  STC(R3,B2);  R0:=R0+6;
  DPTR:=R0; SET(FIRSTVAL);
END;

BEGIN COMMENT 93.-<REPETITION LIST>::=<NUMCRTAG>  (      ;
  R1:=BSP-4; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R1,B2);
  R1:=R1 AND #FF;
  IF R1<4 THEN
  BEGIN
    R1:=DATABUF; R2:=R10+DESCRBUF+1;
    R3:=R1 SHRL 8; STC(R3,B2); R2:=R2+1;
    STC(R1,B2); R2:=R2+3; R0:=5;
  END ELSE
  IF R1=5 THEN BEGIN R2:=R10+DESCRBUF+3; R0:=3; END;
  DPTR:=R0; R2:=R2+4;
  R3:=TBLDESCRB; R1:=R3 SHRL 8;
  STC(R1,B2); R2:=R2+1;
  STC(R3,B2); R1:=BSP-1;
  DATASIZE; R0:=DPTR;
  R2:=R10+DESCRBUF+R0+2; R3:=R1 SHRL 8;
  STC(R3,B2); R2:=R2+1;
  STC(R1,B2); R0:=R0+6;
  DPTR:=R0; SET(FIRSTVAL);
END;

BEGIN COMMENT 94.- <REPETITION LIST>::=<REPETITICN LIST><CONST>  ,  ;
  IF FIRSTVAL THEN
  BEGIN
    R1:=BSP-1; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R1,B2);
  R0:=DPTR-8; R2:=R10+DESCRBUF+R0; R3:=R1 SHRL 8;
  STC(R3,B2); R2:=R2+1;
  STC(R1,B2); R0:=R0-R0;
  CTR:=R0; RESET(FIRSTVAL);
  END;
  R0:=CTR+1; CTR:=R0;
END;

BEGIN COMMENT 95.-<ITEM_AREA CLAUSE>::=ITEM_AREA <ID>     ;
  R1:=BSP SHLL 1; R2:=R10+BPTRSTACK+R1;
  LH(R1,B2); R2:=SEMBASE+R1;
  R0:=49; STC(R0,B2);
  R1:=R1 SHLL 1; R2:=DESCRPTRBASE+R1;
  R3:=TBLDESCRB; STH(R3,B2);
```

```
   END;

   BEGIN COMMENT 96.-<ITEM_AREA CLAUSE>::=<ITEM_AREA CLAUSE> , <ID>    ;
     R2:=95; GOTO STARTBSYN;
   END;

   BEGIN COMMENT 97.-<SUBTABLE CLAUSE>::=SUBTABLE <ID> ( <NUMORTAG>    ;
     R1:=BSP-2; R1:=R1 SHLL 1;
     R2:=R10+BPTRSTACK+R1; LH(R1,B2);
     R3:=SEMBASE+R1; R0:=50; STC(R0,B3);
     R2:=R2-2; STH(R1,B2);
     R1:=BSP; DATASIZE;
     R3:=R1 SHRL 8;
     R2:=R10+DESCRBUF; STC(R3,B2);
     R2:=R2+1; STC(R1,B2);
     R0:=2; DPTR:=R0;
   END;

   BEGIN COMMENT 98.-<SUBTABLE CLAUSE>::=<SUBTABLE CLAUSE> , <NUMORTAG> ;
     R1:=BSP; DATASIZE;
     R0:=DPTR; R2:=R10+DESCRBUF+R0;
     R3:=R1 SHRL 8; STC(R3,B2);
     R2:=R2+1; STC(R1,B2);
     R0:=R0+2; DPTR:=R0;
   END;

     NULL;
     END; COMMENT END OF CASE STATEMENT;
   LM(R0,R6,SAVEREGS);
   END; COMMENT END OF BSYNTHTWO;

 GLOBAL PROCEDURE BSYNTHESIZE(R4); COMMENT IT HANDLES THE SEMANTICS
                                              OF THE DATA DECLARATIONS;
   BEGIN
     ARRAY 7 INTEGER SAVEREGS; STM(R0,R6,SAVEREGS);
     R2:=BPRODNUM;
 CVD(R2,CONWORK); UNPK(3,7,WBUF(12),CONWORK); SETZONE(WBUF(15));
 MVC(8,WBUF,"BPROD NO.");
 R0:=@WBUF; WRITE; MVC(25,WBUF,BLANK); WRITE;
     IF R2<64 THEN
     BEGIN
  STARTBSYN:
     CASE R2 OF
     BEGIN
     NULL; NULL;
   BEGIN COMMENT 3.- <SUB_ROUTINE DECL>::=<PROCEDURE DECL> $ ;
     R1:=BSP-1; R1:=R1 SHLL 1;
     R2:=R10+BPTRSTACK+R1;
```

```
     LH(R3,B2);
     R3:=R3 SHLL 1;
     R2:=DESCRPTRBASE+R3;
     R4:=CTPTR;
     STH(R4,B2);
     R1:=4;
     R2:=CTBASE+R4;
     STC(R1,B2);
COMMENT HERE ADD IL TO BRANCH TO UNSPECIFIED LOCATION (AROUND PROCEDURE
BODY) AND TO LOAD NEXT INSTRUCTION ADDRESS (ENTRY POINT TO PROCEDURE)
AT LOCATION CTBASE+CTPTR+1 (4 BYTES);
     R4:=R4+5; CTPTR:=R4;
  END;

  BEGIN COMMENT 4.- <SUB_ROUTINE DECL>::=<PROC DECL> <FPARM LIST> ) $ ;
     R1:=BSP-3; R1:=R1 SHLL 1;
     R2:=R10+BPTRSTACK+R1; LH(R3,B2);
     R3:=R3 SHLL 1;
     R2:=DESCRPTRBASE+R3;
     R4:=CTPTR; STH(R4,B2);
     R1:=CTBASE+R4;
     R2:=DPTR-1; R3:=R10+DESCRBUF;
     EX(R2,MVC(0,B1,B3));
  COMMENT HERE GENERATE IL TO BRANCH TO UNSPECIFIED LOCATION (AROUND
  PROCEDURE BODY) AND TO LOAD NEXT INSTRUCTION ADDRESS (ENTRY POINT TO
  PROCEDURE) AT LOCATION  CTBASE+CTPTR+1 (4 BYTES) ;
     R4:=R4+R2+1; CTPTR:=R4;
     R0:=R0-RC; DPTR:=R0;
  END;

  BEGIN COMMENT 05.- <SUB_ROUTINE DECL>::=<FUNCTION CLAUSE>  )  $  ;
     R1:=TYPLGTH; R3:=CTR;
     R2:=R10+DESCRBUF+R1; STC(R3,B2);
     RC:=DPTR-1;
     R2:=R10+DESCRBUF; STC(R0,B2);
     R1:=BSP-2; R1:=R1 SHLL 1;
     R2:=R10+BPTRSTACK+R1; LH(R3,B2); R3:=R3 SHLL 1;
     R2:=DESCRPTRBASE+R3;
     R4:=CTPTR; STH(R4,B2);
     R1:=CTBASE+R4;
     R2:=R10+DESCRBUF;
     EX(RC,MVC(0,B1,B2));
     R0:=R0-RC; DPTR:=R0 ;
  END;

  BEGIN COMMENT 6.- <PROCEDURE DECL>::=PROCEDURE <ID> ;
     R1:=BSP; R1:=R1 SHLL 1;
     R2:=R10+BPTRSTACK+R1;
```

```
   LH(R3,B2);
   R2:=R2-2; STH(R3,B2); COMMENT PREPARING STACK TO SAVE POINTER UPON
                                 REDUCTION;
   R2:=SEMBASE+R3;
   R0:=47; STC(R0,B2);
END;

SETHEADER; COMMENT  7.- <FPARM LIST>::=<IN PARMS> <OUT PARMS> <LABEL>;
SETHEADER; COMMENT  8.- <FPARM LIST>::=<IN PARMS> <OUT PARMS>        ;
SETHEADER; COMMENT  9.- <FPARM LIST>::=<IN PARMS> <LABEL PARMS>      ;
SETHEADER; COMMENT 10.- <FPARM LIST>::=<OUT PARMS> <LABEL PARMS>     ;
SETHEADER; COMMENT 11.- <FPARM LIST>::=<IN PARMS>                    ;
SETHEADER; COMMENT 12.- <FPARM LIST>::=<OUT PARMS>                   ;
SETHEADER; COMMENT 13.- <FPARM LIST>::=<LABEL PARMS>                 ;

BEGIN  COMMENT 14.- <IN PARMS>::= (  <VAR>  ;
   RC:=1; R2:=R10+DESCRBUF+5; STC(R0,B2);
   R1:=BSP; R1:=R1 SHLL 1;
   R4:=R10+EPTRSTACK+R1;
   LH(R3,B4);
   R2:=R2+1; STH(R3,B2);
   R0:=8; DPTR:=R0;
END;

BEGIN COMMENT 15.- <IN PARMS>::=<INPARMS>  ,  <VAR>  ;
   R2:=R10+DESCRBUF +5;
   IC(R0,B2); R0:=R0+1; STC(R0,B2);
   R1:=BSP; R1:=R1 SHLL 1;
   R2:=R10+BPTRSTACK+R1; LH(R3,B2);
   R4:=DPTR; R2:=R10+DESCRBUF+R4;
   STH(R3,B2);
   R4:=R4+2; DPTR:=R4;
END;

BEGIN COMMENT 16.- <OUT PARMS>::= |  <VAR>   ;
   R1:=DPTR;
   IF R1=0 THEN R1:=5; COMMENT NO INPUT PARMS;
   R0:=#41; R2:=R10+DESCRBUF+R1;
   STC(R0,B2); TYPLGTH:=R1;
   R2:=R2+1;
   R3:=BSP; R3:=R3 SHLL 1;
   R5:=R10+EPTRSTACK+R3; LH(R4,B5);
   R5:=R4 SHRL 8; STC(R5,B2);
   R2:=R2+1; STC(R4,B2);
   R1:=R1+3; DPTR:=R1;
END;

BEGIN COMMENT 17.- <OUT PARMS>::=<OUT PARMS>  ,  <VAR>   ;
```

```
    R1:=TYPLGTH; R2:=R10+DESCRBUF+R1;
    IC(R0,B2); R0:=R0+1; STC(R0,B2);
    R3:=BSP; R3:=R3 SHLL 1;
    R5:=R10+BPTRSTACK+R3; LH(R4,B5);
    R1:=DPTR; R2:=R10+DESCRBUF+R1;
    R5:=R4 SHRL 8; STC(R5,B2);
    R2:=R2+1; STC(R4,B2);
    R1:=R1+2; DPTR:=R1;
  END;

  BEGIN COMMENT 18.- <LABEL PARMS>::= || <ID>  ;
    R1:=DPTR;
    IF R1=0 THEN R1:=5; COMMENT NO INPUT OR OUTPUT PARMS,;
    R0:=#81; R2:=R10+DESCRBUF+R1;
    STC(R0,B2); TYPLGTH:=R1;
    R3:=BSP; R3:=R3 SHLL 1;
    R5:=R10+BPTRSTACK+R3; LH(R4,B5);
    R2:=R2+1; R5:=R4 SHRL 8;
    STC(R5,B2); R2:=R2+1; STC(R4,B2);
    R1:=R1+3; DPTR:=R1;
  END;

  BEGIN COMMENT 19.- <LABEL PARMS>::=<LABEL PARMS>  ,  <ID>  ;
    R1:=TYPLGTH; R2:=R10+DESCRBUF+R1;
    IC(R0,B2); R0:=R0+1; STC(R0,B2);
    R3:=BSP; R3:=R3 SHLL 1;
    R5:=R10+BPTRSTACK+R3;  LH(R4,B5);
    R1:=DPTR; R2:=R10+DESCRBUF+R1;
    R5:=R4 SHRL 8; STC(R5,B2);
    R2:=R2+1; STC(R4,B2);
    R1:=R1+2; DPTR:=R1;
  END;

  BEGIN COMMENT 20.- <FUNCTION CLAUSE>::=<FUNCTION HEAD> ( <VAR>  ;
    R1:=BSP SHLL 1;
    R2:=R10+BPTRSTACK+R1;
    LH(R3,B2);
    R0:=DPTR; TYPLGTH:=R0; R0:=R0+1;
    R2:=R10+DESCRBUF+R0;
    R4:=R3 SHRL 8; STC(R4,B2);
    R2:=R2+1; STC(R3,B2);
    R0:=R0+2; DPTR:=R0;
    R0:=1; CTR:=R0;
  END;

  BEGIN COMMENT 21.- <FUNCTION CLAUSE>::=<FUNCTION CLAUSE> , <VAR>  ;
    R1:=BSP SHLL 1;
    R2:=R10+BPTRSTACK+R1;
```

```
    LH(R3,B2);
    RO:=DPTR;
    R2:=R10+DESCRBUF+RO;
    R4:=R3 SHRL 8; STC(R4,B2);
    R2:=R2+1; STC(R3,B2);
    RO:=RO+2; DPTR:=RO;
    RO:=CTR+1; CTR:=RO;
END;

BEGIN COMMENT  22.- <FUNCTION HEAD>::=FUNCTION <DATA TYPE> <ID> ;
    R1:=BSP-1; R1:=R1 SHLL 1;
    R2:=R10+BPTRSTACK+R1; LH(R3,B2); R3:=R3 AND #FFFF;
    CASE R3 OF
    BEGIN
        BEGIN COMMENT CASE 1.- NUMERIC FUNCTION ;
            RO:=32; FUTIL;
        END;
        BEGIN COMMENT CASE 2.- BITS FUNCTION ;
            RO:=35; FUTIL;
        END;
        BEGIN COMMENT CASE 3.- CHARACTER FUNCTION ;
            RO:=38; FUTIL;
        END;
        BEGIN COMMENT CASE 4.- STATUS FUNCTION ;
            R1:=BSP SHLL 1;
            R2:=R10+BPTRSTACK+R1; LH(R3,B2);
            RO:=41; R2:=SEMBASE+R3; STC(RO,B2);
            R2:=R10+DESCRBUF; IC(R1,B2); R1:=R1 AND #FF;
            FOR R4:=R1 STEP _1 UNTIL 1 DO
            BEGIN
                R2:=R10+DESCRBUF+R4; IC(R3,B2);
                R2:=R2+4; STC(R3,B2);
        END;
            R1:=DPTR+4; DPTR:=R1;
        END;
        BEGIN COMMENT CASE 5.- BOOLEAN FUNCTION;
            R1:=BSP SHLL 1; R2:=R10+BPTRSTACK+R1;
            LH(R3,B2);
            RO:=44; R2:=SEMBASE+R3;
            STC(RO,B2); RO:=5; DPTR:=RO;
        END;
    END; COMMENT END OF CASE STMT;
    R1:=BSP SHLL 1; R2:=R10+BPTRSTACK+R1;
    LH(R3,B2);
    R2:=R2-4; STH(R3,B2);
END; COMMENT END OF CASE 22;

    NULL; NULL; NULL; NULL; NULL; NULL; NULL; NULL;
```

```
  NULL; NULL; NULL; NULL;

  BEGIN COMMENT 35.- <DATA ELEM>::=<VRBL DECL HEAD> <ITEM>  $    ;
  R1:=BSP-2; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R3,B2); R3:=R3 AND #FF;
  CASE R3 OF
  BEGIN
    R0:=31; R0:=34; R0:=37; R0:=40; R0:=43;
  END;
  R2:=R2+2; LH(R4,B2);
  R2:=SEMBASE+R4; STC(R0,B2);
  IF R3<5 THEN COMMENT FOR OTHER THAN BOOL VAR, LOAD DESCRIPTOR IN
                                CONSTANT TABLE;
  BEGIN
    R4:=R4 SHLL 1; R2:=DESCRPTRBASE+R4;
    R0:=CTPTR; STH(R0,B2);
    R1:=DPTR-1; R2:=R10+DESCRBUF;
    R4:=CTBASE+R0;
    EX(R1,MVC(0,B4,B2));
  R0:=R0+R1+1; CTPTR:=R0;
    R0:=R0-R0; DPTR:=R0;
  END;
END;

BEGIN COMMENT 36.-<DATA ELEM>::=<FORMAT DECL HEAD><DESCRGROUP>)$   ;
  R1:=BSP-3; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1;
  LH(R1,B2); R1:=R1 SHLL 1;
  R2:=DESCRPTRBASE+R1; R0:=CTPTR;
  STH(R0,B2); R2:=CTBASE+R0;
  R1:=DPTR  ; STC(R1,B2);
  R2:=R2+1; R4:=R10+DESCRBUF;
  R1:=R1-1;
  EX(R1,MVC(0,B2,B4));
  R1:=DPTR; R0:=R0+R1; CTPTR:=R0;
  R0:=R0-R0; DPTR:=R0;
END;

BEGIN COMMENT 37.- <TAG DECL HEAD>::=<TAG DECL HEAD> <ID> = <REAL CONS
                                                           TANT> $  ;
  R1:=BSP -1; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R3,B2);
  R2:=R2-4; LH(R4,B2);
  R2:=SEMBASE+R4; R0:=8; STC(R0,B2);
  R4:=R4 SHLL 1; R2:=DESCRPTRBASE+R4;
  STH(R3,B2);
END;
```

```
BEGIN COMMENT 38.-<DATA ELEM>::=<TABLE DECL HEAD> <TABLE DECL> END_TAB
                                                                   LE  $ ;
  R1:=CTPTR; R0:=TBLDESCRB;
  R2:=CTBASE+R0; IC(R3,B2);
R3:=R3 AND #FF;
  R2:=R2+R3-1;
  R4:=R1 SHRL 8; STC(R4,B2);
  R2:=R2+1; STC(R1,B2);
  R2:=CTBASE+R1; R4:=aFLDPTRSBUF;
  R1:=FLBPTR  ; STC(R1,B2);
  R2:=R2+1; R1:=R1-1;
  EX(R1,MVC(0,B2,B4));
  R0:=CTPTR+R1+2; CTPTR:=R0;
END;

  NULL;
BEGIN COMMENT 40.- <VRBL DECL HEAD>::=<VRBL DECL HEAD> <ITEM> ,  ;
  R2:=35; GOTO STARTBSYN;
END;

BEGIN COMMENT 41.- <VRBL HEAD>::=VRBL <DATA TYPE>    ;
  R1:=BSP SHLL 1;
  R2:=R10+BPTRSTACK+R1;
  LH(R3,B2); R2:=R2-2; STH(R3,B2);
END;

BEGIN COMMENT 42.- <DATA TYPE>::=INTEGER  (  <REAL CONSTANT>  )  ;
  R1:=BSP-1;
  DATASIZE; R1:=R1 AND #3FFF;
  DATABUF:=R1; R1:=BSP-3; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; R0:=1; STH(R0,B2);
END;

BEGIN COMMENT 43.-<DATA TYPE>::=FIXED ( <REAL CONST> , <REAL CONST> );
  R1:=BSP-1;
  DATASIZE; R1:=R1 SHLL 7;
  R6:=#4000 OR R1;
  R1:=BSP-3;
  DATASIZE; R1:=R1 AND #7F;
  R6:=R6 OR R1;
  DATABUF:=R6;
  R1:=BSP-5; R1:=R1 SHLL 1;
  R2:=R10 + BPTRSTACK + R1; R0:=1;
  STH(R0,B2);
END;

BEGIN COMMENT 44.- <DATA TYPE>::=FLOAT  (  <REAL CONST>  )   ;
  R1:=BSP-1;
```

```
  DATASIZE;     R1:=R1 AND #3FFF;
  R6:=#8000 OR R1;
  DATABUF:=R6;
  R1:=BSP-3; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; R0:=1;
  STH(R0,B2);
END;

BEGIN COMMENT 45.- <DATA TYPE>::= BITS  (  <REAL CONST>  )    ;
  R1:=BSP-1;
  DATASIZE;
  DATABUF:=R1;
  R1:=BSP-3; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; R0:=2;
  STH(R0,B2);
END;

BEGIN COMMENT 46.- <DATA TYPE>::=CHAR  (  <REAL CONST>  )   ;
  R1:=BSP-1;
  DATASIZE;
  DATABUF:=R1;
  R1:=BSP-3; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; R0:=3;
  STH(R0,B2);
END;
BEGIN COMMENT 47.- <DATA TYPE>::=BOOLEAN       ;
  R1:=BSP SHLL 1;
  R2:=R10+BPTRSTACK+R1; R0:=5;
  STH(R0,B2);
END;

BEGIN COMMENT 48.- <DATA TYPE>::=<STATUS CLAUSE>  )   ;
  R1:=CPTR-1; R2:=R10+DESCRBUF;
  STC(R1,B2);
  R1:=BSP-1; R1:=R1 SHLL 1;
  R2:=R10+BPTRSTACK+R1; R0:=4;
  STH(R0,B2);
END;

BEGIN COMMENT 49.-<STATUS CLAUSE>::=STATUS  (  <STATUS CONST>   ;
  R1:=BSP SHLL 1;
  R2:=R10+BPTRSTACK+R1; LH(R3,B2);
  R4:=R3 SHLL 1;
  R2:=DESCRPTRBASE+R4; R0:=1;
  STH(R0,B2);
  R2:=R10+DESCRBUF+2; STH(R3,B2);
  R0:=4; DPTR:=R0;
  R0:=1; CTR:=R0;
```

```
    END;

    BEGIN COMMENT 50.-<STATUS CLAUSE>::=<STATUS CLAUSE> , <STATUS CONST> ;
      R1:=BSP SHLL 1;
      R2:=R10+EPTRSTACK+R1; LH(R3,B2);
      R4:=R3 SHLL 1;
      R2:=DESCRPTRBASE+R4; R0:=CTR+1; CTR:=R0;
      STH(R0,B2);
      R1:=DPTR; R2:=R10+DESCRBUF+R1;
      STH(R3,B2);
      R1:=R1+2; DPTR:=R1;
    END;

    BEGIN COMMENT 51.- <ITEM>::=<ID>            ;
      R1:=BSP-1; R1:=R1 SHLL 1;
      R2:=R10+EPTRSTACK+R1; LH(R3,B2); R3:=R3 AND #FF;
L1:   CASE R3 OF
      BEGIN
        BEGIN COMMENT NUMERIC CONSTANT;
          R1:=4; R2:=R10+DESCRBUF; STC(R1,B2); R2:=R2+1;
          R1:=DATABUF; R3:=R1 SHRL 8;
          STC(R3,B2); R2:=R2+1;
          STC(R1,B2);
          R0:=5; DPTR:=R0;
        END;
        BEGIN COMMENT BITS CONSTANT;
          R3:=1; GOTO L1;
        END;
        BEGIN COMMENT CHAR CONSTANT;
          R3:=1; GOTO L1;
        END;
        NULL; COMMENT STATUS CONSTANT;
        NULL; COMMENT BOOL CONSTANT;
      END;
    END; COMMENT END OF CASE 51;

    BEGIN COMMENT 52.- <ITEM>::=<ID>  =  <CONSTANT>        ;
      R1:=BSP-3; R1:=R1 SHLL 1;
      R2:=R10+EPTRSTACK+R1; LH(R3,B2); R3:=R3 AND #FF;
      IC(R4,TYPE); R4:=R4 AND #FF;
      CLR(R3,R4);
      IF ¬= THEN BEGIN R0:=14; XR:=R0; ERROR; END
      ELSE BEGIN
        CASE R3 OF
        BEGIN
          BEGIN COMMENT CASE 1, NUMERIC CONSTANT;
            R1:=DATABUF AND #FFFF SHRL 14;
            R2:=R2+6; LH(R3,B2);
```

```
    R4:=CTBASE+R3; IC(RO,B4);
    RO:=RO AND #FF SHRL 5;
    R5:=R1+3;
    IF R1¬=RO AND R5¬=RO THEN CCMMENT ILLEGAL ASSIGNMENT;
    BEGIN RO:=14; XR:=RO; ERROR; END
    ELSE SETVRBLCSCR;
   END;
   SETVRBLCSCR; COMMENT CASE 2, BITS CONSTANT;
   SETVRBLDSCR; COMMENT CASE 3, CHAR CONSTANT;
   BEGIN COMMENT CASE 4, STATUS CONSTANT;
    R2:=R2+6; LH(R1,B2); R3:=R1 SHLL 1;
  R1:=R1 AND #FFFF;
    R4:=DESCRPTRBASE+R3; LH(R5,B4);
    R3:=R5 SHLL 1; R4:=R10+DESCRBUF+R3; LH(R3,B4);
  R3:=R3 AND #FFFF;
    CLR(R1,R3);
    IF ¬= THEN BEGIN RO:=14; XR:=RO; ERRCR; END
    ELSE BEGIN
     R4:=R10+DESCRBUF+1; STC(R5,B4);
    END;
   END;
   BEGIN CCMMENT CASE 5, BOOL CCNSTANT;
    R2:=R2+6; LH(R1,B2);
    R2:=R2-4; LH(R3,B2); R3:=R3 SHLL 1;
    R2:=DESCRPTRBASE+R3; STH(R1,B2);
   END;
  END; COMMENT END OF CASE STMT;
 END;
END; COMMENT END OF CASE 52.  ;

BEGIN COMMENT 53.- <CCNSTANT>::=<NUMCRTAG>   ;
 R1:=1; STC(R1,TYPE);
END;

BEGIN COMMENT 54.- <CCNSTANT>::=<BITS CCNSTANT>  ;
 R1:=2; STC(R1,TYPE);
END;

BEGIN COMMENT 55.- <CONSTANT>::=<CHAR CONSTANT>   ;
 R1:=3; STC(R1,TYPE);
END;

BEGIN CCMMENT 56.- <CCNSTANT>::=<STATUS CONSTANT>   ;
 R1:=4; STC(R1,TYPE);
END;

BEGIN CCMMENT 57.- <CCNSTANT>::=<BCCL CONSTANT>   ;
 R1:=5; STC(R1,TYPE);
```

```
END;

BEGIN COMMENT 58.- <BOOL CONSTANT>::=TRUE      ;
  R1:=BSP SHLL 1;
  R2:=R10+BPTRSTACK+R1;
  R3:=1; STH(R3,B2);
END;

BEGIN COMMENT 59.- <BOOL CONSTANT>::=FALSE     ;
  R1:=BSP SHLL 1;
  R2:=R10+BPTRSTACK+R1;
  R3:=R3-R3; STH(R3,B2);
END;

BEGIN COMMENT 60.- <NUMORTAG>::=<TAG>        ;
  R1:=BSP SHLL 1;
  R2:=R10+BPTRSTACK+R1;
  LH(R3,B2); R3:=R3 SHLL 1;
  R4:=DESCRPTBASE+R3; LH(R3,B4);
  STH(R3,B2);
END;

  NULL; NULL;

BEGIN COMMENT 63.- <TAG DECL HEAD>::=<TAG DECL HEAD> <ID> = <REAL CONS
                                                           TANT> ,  ;
  R2:=37; GOTO STARTBSYN;
END;


  END; COMMENT END OF CASE STMT;
END ELSE BSYNTHTWO;
  LM(R0,R6,SAVEREGS);
END; COMMENT END OF BSYNTHESIZE;


GLOBAL PROCEDURE ANALYZE(R4); COMMENT THIS IS THE SLR(1) PARSER FOR THE
                              MAIN GRAMMAR (DYNAMIC STATEMENTS);
   BEGIN ARRAY 8 INTEGER SAVEREGS;
     INTEGER NEXTSYMBOL;
         ARRAY 150 INTEGER STATESTACK = 150(0);
         INTEGER STATENUM =0, LASYMBOL = 0;

         PROCEDURE PUSHANDREAD(R4);
            BEGIN INTEGER SAVE4;
               SAVE4 := R4; R1 := SP;
               IF R1 < 150 THEN
               BEGIN
```

```
               R1 := R1 + 1; SP := R1;
            R1:=R1 SHLL 1;
            R4:=POINTER;
            R2:=R10+PTRSTACK+R1;
            STH(R4,B2);
            END ELSE
            BEGIN
               R4:=3; XR:=R4; ERRCR;
            END;
            R1 := TCKEN; NEXTSYMBCL := R1;
            R4:=@ANALYZE;
            R4 := R4 + #7C;
            RTRNTOANAL:=R4;
            SCAN;
            R4:=SAVE4;
         END;  COMMENT  END PUSHANDREAD;

INTEGER CYCLECNT=0; STM(RO,R7,SAVEREGS);
      WHILE TRUE DO
      BEGIN
         R1 := CYCLECNT + 1; CYCLECNT := R1;
         R1 := SP SHLL 2; R2 := STATESTACK(R1) SHRL 8 + 1;
         CASE R2 OF
         BEGIN
            COMMENT  CASE 1, READ VIA LINEAR SEARCH;
            BEGIN
               PUSHANDREAD; R1 := STATENUM SHLL 1;
               R2 := AREADSTART;
               R4 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B4);
               R1 := R1 SHRL 1; R5 := ARDNUM;
               R4 := SEGBASE + SEGTABLE(R5) + R1;
               IC(R3,B4); R3 := R3 AND MASK;
               R3 := R3 + R2; R4 := NEXTSYMBCL;
               R5 := ASYMLIST;
               R6 := SEGBASE + SEGTABLE(R5) + R2; IC(R5,B6);
               R5 := R5 AND MASK;
               WHILE R4 ¬= R5 AND R2 < R3 DO
               BEGIN
                  R2 := R2 + 1; R6 := R6 + 1; IC(R5,B6);
               END;
               R2 := R2 SHLL 1;
               R4 := ASTATELIST;
               R3 := SEGBASE + SEGTABLE(R4) + R2;
               LH(R1,B3); R1 := R1 AND MASKFFFF;
               R2 := SP SHLL 2; STATESTACK(R2) := R1;
               R1 := R1 AND MASK; STATENUM := R1;
            END;
            COMMENT  CASE 2, READ VIA AN ARRAY ACCESSS;
```

```
BEGIN
   PUSHANDREAD; R1 := STATENUM SHLL 1;
   R2 := AREADSTART;
   R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
   R2 := R2 + NEXTSYMBOL SHLL 1; R1 := ASTATELIST;
   R3 := SEGBASE + SEGTABLE(R1) + R2; LH(R1,B3);
   R1 := R1 AND MASKFFFF;
   R2 := SP SHLL 2; STATESTACK(R2) := R1;
   R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT  CASE 3, REDUCE;
BEGIN
   R1 := STATENUM; PRODNUM := R1; SYNTHESIZE;
   R1 := STATENUM; R2 := ANUMTOPCP;
   R3 := SEGBASE + SEGTABLE(R2) + R1;
   IC(R2,B3);
   R2 := R2 AND MASK; R3 := SP - R2; SP := R3;
   R1 := R1 SHLL 1; R2 := AREDUCESUCC;
   R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
   R1 := SP SHLL 2; STATESTACK(R1) := R2;
   R2 := R2 AND MASK; STATENUM := R2;
END;
COMMENT  CASE 4, LOOK AHEAD (ORDINARY);
BEGIN
   R1 := TOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
   R5 := STATENUM * LATABSIZE + R2;
   R3 := R1 AND MASK7; R4 := 7 - R3;
   R3 := ALATABLE;
   R6 := SEGBASE + SEGTABLE(R3) + R5; IC(R1,B6);
   R1 := R1 AND MASK SHRL R4 AND MASK1;
   IF R1 = 1 THEN
   BEGIN
      R1 := STATENUM SHLL 1;
      R2 := ASUCCSTATE;
      R3 := SEGBASE + SEGTABLE(R2) + R1;
      LH(R2,B3);
   END ELSE
   BEGIN
      R1 := STATENUM SHLL 1;
      R2 := AFAILSTATE;
      R3 := SEGBASE + SEGTABLE(R2) + R1;
      LH(R2,B3);
   END;
   R2 := R2 AND MASKFFFF; R1 := SP SHLL 2;
   STATESTACK(R1) := R2; R2 := R2 AND MASK;
   STATENUM := R2;
END;
COMMENT  CASE 5, LOOK AHEAD (FOR A PRODUCTION
```

```
              WITH AN EMPTY RIGHT PART);
BEGIN
   R1 := TOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
   R5 := STATENUM * LATABSIZE + R2;
   R3 := R1 AND MASK7; R4 := 7 - R3;
   R3 := ALATABLE;
   R4 := SEGBASE + SEGTABLE(R3) + R5; IC(R1,B4);
   R1 := R1 AND MASK SHRL R4 AND MASK1;
   IF R1 = 1 THEN
   BEGIN
      R1 := SP SHLL 2;
      R2 := STATESTACK(R1) SHRL 8;
      WHILE R2 = 3 OR R2 = 4 DO
      BEGIN
         R3 := STATESTACK(R1) AND MASK SHLL 1;
         R4 := AFAILSTATE;
         R5 := SEGBASE + SEGTABLE(R4) + R3;
         LH(R4,B5);
         STATESTACK(R1) := R4;
         R2 := R4 SHRL 8;
      END;
      R1 := SP + 1; SP := R1;
      R1 := STATENUM SHLL 1;
      R2 := ASUCCSTATE;
      R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
   END ELSE
   BEGIN
      R1 := STATENUM SHLL 1;
      R2 := AFAILSTATE;
      R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
   END;
   R1 := SP SHLL 2; STATESTACK(R1) := R2;
   R2 := R2 AND MASK; STATENUM := R2;
END;
COMMENT  CASE 6, LOOK BACK;
BEGIN
   R1 := SP - 1 SHLL 2; R2 := STATENUM;
   R3 := ALBSTART;
   R4 := SEGBASE + SEGTABLE(R3) + R2; IC(R3,B4);
   R3 := R3 AND MASK;
   R4 := ALBNUM;
   R5 := SEGBASE + SEGTABLE(R4) + R2; IC(R4,B5);
   R4 := R4 AND MASK + R3;
   R3 := R3 SHLL 1; R5 := ALBSTATE;
   R7 := SEGBASE + SEGTABLE(R5) + R3;
   LH(R5,B7);
   R6 := STATESTACK(R1); R3 := R3 SHRL 1;
   WHILE R6 ¬= R5 AND R3 < R4 DO
```

```
  BEGIN
    R3 := R3 + 1 SHLL 1; R5 := ALBSTATE;
  R7 := SEGBASE + SEGTABLE(R5) + R3; LH(R5,B7);
    R3 := R3 SHRL 1;
  END;
  R3 := R3 SHLL 1; R1 := ARESUMSTATE;
  R7 := SEGBASE + SEGTABLE(R1) + R3; LH(R1,B7);
  R2 := SP SHLL 2; STATESTACK(R2) := R1;
  R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT  CASE 7, ERROR;
BEGIN INTEGER PREVERRCYCLE = #FFFFFFFF;
  R0 := @WBUF; WRITE;
  R1 := CYCLECNT - 2; CYCLECNT := R1;
  IF R1 ¬= PREVERRCYCLE THEN
  BEGIN
    PREVERRCYCLE := R1; R1 := SP - 1 SHLL 2;
    R2 := STATESTACK(R1);
    IF R2 < 512 THEN
    BEGIN
      R2 := R2 AND MASK;
      R3 := ASYMBEFCREREAD;
      R4 := SEGBASE + SEGTABLE(R3) + R2;
      IC(R3,B4);
    END ELSE
    BEGIN
      R2 := STATENUM; R3 := ESYMBEFCRELA;
      R4 := SEGBASE + SEGTABLE(R3) + R2; IC(R3,B4);
    END;
    R3 := R3 AND MASK; R2 := 33;
    FOR R7 := 0 STEP 1 UNTIL 1 DO
    BEGIN
     VPT := R3; FIND; R1 := LENGTH - 1;
     FOR R6 := 0 STEP 1 UNTIL R1 DO
     BEGIN
       IC(R5,BCD(R6)); STC(R5,WBUF(R2));
       R2 := R2 + 1;
     END;
     R2 := R2 + 2; R3 := STATENLM;
     IF R3 = 255 THEN R3 := NEXTSYMBOL
     ELSE R3 := LASYMBOL;
     R3 := R3 AND MASK;
    END;
    R4:=4; XR:=R4; ERROR;
    MVC(131,WBUF,BLANK);
    MVC(17,WBUF,"PARTIAL PARSE IS: ");
    R0 := @WBUF; WRITE; MVC(17,WBUF,BLANK);
    R2 := SP - 1 SHLL 2;
```

```
FCR R1 := 8 STEP 4 UNTIL R2 CO
BEGIN
   R3 := STATESTACK(R1);
   IF R3 < 512 THEN
   BEGIN
      R3 := R3 AND MASK;
      R4 := ASYMBEFOREREAD;
      R5 := SEGBASE+SEGTABLE(R4)+R3;
      IC(R4,B5);
   END ELSE
   BEGIN
      R3 := R3 AND MASK;
      R4 := BSYMBEFORELA;
      R5 := SEGBASE+SEGTABLE(R4)+R3;
      IC(R4,B5);
   END;
   R4 := R4 AND MASK; VPT := R4; FIND;
   MVC(63,WBUF(4),BCD); WRITE;
   MVC(63,WBUF(4),BLANK);
  END;
END;
R1 := NEXTSYMBOL;
IF R1 = 1 THEN
BEGIN
   R4:=5; XR:=R4; ERRCR;
END ELSE
BEGIN
   VPT := R1; FIND; R1 := LENGTH - 1;
   MVC(16,WBUF,"THE INPUT SYMBOL,");
   MVC(63,WBUF(20),BCD);
   MVC(16,WBUF(60)," WILL BE IGNORED:");
   RO := @WBUF;
   WRITE; MVC(131,WBUF,BLANK);
END;
R1 := STATENUM;
IF R1 = 255 THEN
BEGIN
COMMENT  ERROR OCCURRED IN A READ STATE;
   R1 := SP - 1; SP := R1; R1 := R1 SHLL 2;
   R2 := STATESTACK(R1) AND MASK;
   STATENUM := R2;
END ELSE
BEGIN
COMMENT  ERROR OCCURRED IN A LOOK-AHEAD STATE;
   R4:=@ANALYZE;
   R4 :=R4 + #6B2;
   RTRNTOANAL:=R4;
   SCAN;  COMMENT  SKIP THE NEXT SYMBCL;
```

```
                      R1 := R1 SHLL 1;
                   R2 := ASUCCSTATE;
                   R4 := SEGBASE + SEGTABLE(R2) + R1;
                      LH(R2,B4); R2 := R2 AND MASKFF00;
                      LH(R3,B4); R3 := R3 AND MASK;
                      R4 := ANUMTOPOP; R5 := SEGBASE+SEGTABLE(R4)+R3;
                      IC(R4,B5); R4 := R4 AND MASK;
                      IF R2 = 512 ANC R4 ¬= 255 THEN
                         R1 := STATENUM OR #COCCC300
                      ELSE R1 := STATENUM OR #00000400;
                      R2 := SP SHLL 2; STATESTACK(R2) := R1;
                   END;
                END;   CCMMENT  ENC OF CASE 7;
                BEGIN COMMENT CASE 8. EXIT;
                   R1 := 1; TOKEN := R1; R1 := C;
                   SP := R1; STATESTACK(R1) := R1;
                   BPTRSTACK(R1):=R1;
                   STATENUM := R1; GCTC XXX;
                END;
             END;  COMMENT  END OF CASE(STATETYPE);
          END; CCMMENT END OF WHILE BLOCK;
XXX:      LM(RC,R7,SAVEREGS);
     END; CCMMENT END OF ANALYZE;


GLCBAL PROCEDURE BANALYZE(R4); COMMENT THIS IS THE SLR(1) PARSER ECR
                               THE  CATA DECLARATIONS GRAMMAR;
  BEGIN ARRAY 8 INTEGER SAVEREGS;
    INTEGER NEXTSYMBOL;
    ARRAY 150 INTEGER STATESTACK=150(0);
    INTEGER STATENUM=0, LASYMBOL=0;
    BYTE FTFLAG=#FF; COMMENT FIRST TIME PUSHANCREAD IS CALLED FTFLAG IS
                              SET;
    INTEGER CYCLECNT=0;

    FRCCEDURE PUSHANDREAC(R4);
    BECIN ARRAY 4 INTEGER SAVER; STM(R1,R4,SAVER);
      R1:=BSP;
      IF R1<150 THEN BEGIN R1:=R1+1; BSP:=R1;
                R1:=R1 SHLL 1;
                R2:=R10+EPTRSTACK+R1;
                R4:=PCINTER;
                STH(R4,B2);
                     ENC
      ELSE BEGIN R4:=3; XR:=R4; ERROR; ENC;
      IF FTFLAG THEN
      BEGIN R1:=1; NEXTSYMBCL:=R1; ENC
      ELSE BEGIN R1:=BTCKEN; NEXTSYMBCL:=R1; SCAN; END;
```

```
  LM(R1,R4,SAVER);
 END; COMMENT END OF PUSHANDREAD;

STM(R0,R7,SAVEREGS);
 SET(FTFLAG);
RC:=R0-R0; BSP:=R0;
WHILE TRUE DO
BEGIN
 R1:=CYCLECNT+1; CYCLECNT:=R1;
 R1:=BSP SHLL 2; R2:=STATESTACK(R1) SHBL 8 + 1;
 CASE R2 OF
 BEGIN
  COMMENT CASE 1. READ VIA LINEAR SEARCH;
  BEGIN
   PUSHANDREAD; RESET(FTFLAG); R1:=STATENUM SHLL 1;
        R2 := AREADSTART;
   R4:=BSEGBASE+BSEGTABLE(R2)+R1; LH(R2,B4);
        R1 := R1 SHRL 1; R5 := ARDNUM;
   R4:=BSEGBASE+BSEGTABLE(R5)+R1;
        IC(R3,B4); R3 := R3 AND MASK;
        R3 := R3 + R2; R4 := NEXTSYMBOL;
        R5 := ASYMLIST;
   R6:=BSEGBASE+BSEGTABLE(R5)+R2; IC(R5,B6);
        R5 := R5 AND MASK;
        WHILE R4 ¬= R5 AND R2 < R3 DO
        BEGIN
          R2 := R2 + 1; R6 := R6 + 1; IC(R5,B6);
        END;
        R2 := R2 SHLL 1;
        R4 := ASTATELIST;
   R3:=BSEGBASE+BSEGTABLE(R4)+R2;
        LH(R1,B3); R1 := R1 AND MASKFFFF;
        R2 :=BSP SHLL 2; STATESTACK(R2) := R1;
        R1 := R1 AND MASK; STATENUM := R1;
       END;
       COMMENT  CASE 2, READ VIA AN ARRAY ACCESSS;
       BEGIN
        PUSHANDREAD; R1 := STATENUM SHLL 1;
        R2 := AREADSTART;
        R3 :=BSEGBASE +BSEGTABLE(R2) + R1; LH(R2,B3);
        R2 := R2 + NEXTSYMBOL SHLL 1; R1 := ASTATELIST;
        R3 :=BSEGBASE +BSEGTABLE(R1) + R2; LH(R1,B3);
        R1 := R1 AND MASKFFFF;
        R2 :=BSP SHLL 2; STATESTACK(R2) := R1;
        R1 := R1 AND MASK; STATENUM := R1;
       END;
       COMMENT  CASE 3, REDUCE;
       BEGIN
```

```
    R1  := STATENUM;BPRODNUM := R1;BSYNTHESIZE;
    R1  := STATENUM; R2 := ANUMTOFOP;
    R3  :=BSEGBASE +BSEGTABLE(R2) + R1;
    IC(R2,B3);
    R2  := R2 AND MASK; R3 :=BSP - R2;BSP := R3;
    R1  := R1 SHLL 1; R2 := AREDUCESUCC;
    R3  :=BSEGBASE +BSEGTABLE(R2) + R1; LH(R2,B3);
    R1  :=BSP SHLL 2; STATESTACK(R1) := R2;
    R2  := R2 AND MASK; STATENUM := R2;
END;
COMMENT  CASE 4, LOOK AHEAD (ORDINARY);
BEGIN
    R1  :=BTOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
    R5  := STATENUM *BLATABSIZE + R2;
    R3  := R1 AND MASK7; R4 := 7 - R3;
    R3  := ALATABLE;
    R6  :=BSEGBASE +BSEGTABLE(R3) + R5; IC(R1,B6);
    R1  := R1 AND MASK SHRL R4 AND MASK1;
    IF R1 = 1 THEN
    BEGIN
        R1  := STATENUM SHLL 1;
        R2  := ASUCCSTATE;
        R3  :=BSEGBASE +BSEGTABLE(R2) + R1;
        LH(R2,B3);
    END ELSE
    BEGIN
        R1  := STATENUM SHLL 1;
        R2  := AFAILSTATE;
        R3  :=BSEGBASE +BSEGTABLE(R2) + R1;
        LH(R2,B3);
    END;
    R2 := R2 AND MASKFFFF; R1  :=BSP SHLL 2;
    STATESTACK(R1) := R2; R2 := R2 AND MASK;
    STATENUM := R2;
END;
COMMENT  CASE 5, LOOK AHEAD (FOR A PRODUCTION
         WITH AN EMPTY RIGHT PART);
BEGIN
    R1  :=BTOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
    R5  := STATENUM *BLATABSIZE + R2;
    R3  := R1 AND MASK7; R4 := 7 - R3;
    R3  := ALATABLE;
    R4  :=BSEGBASE +BSEGTABLE(R3) + R5; IC(R1,B4);
    R1  := R1 AND MASK SHRL R4 AND MASK1;
    IF R1 = 1 THEN
    BEGIN
        R1  :=BSP SHLL 2;
        R2  := STATESTACK(R1) SHRL 8;
```

```
      WHILE R2 = 3 OR R2 = 4 DO
      BEGIN
         R3 := STATESTACK(R1) AND MASK SHLL 1;
         R4 := AFAILSTATE;
         R5 :=BSEGBASE +BSEGTABLE(R4) + R3;
         LH(R4,B5);
         STATESTACK(R1) := R4;
         R2 := R4 SHRL 8;
      END;
      R1 :=BSP + 1;BSP := R1;
      R1 := STATENUM SHLL 1;
      R2 := ASUCCSTATE;
      R3 :=BSEGBASE +BSEGTABLE(R2) + R1; LH(R2,B3);
   END ELSE
   BEGIN
      R1 := STATENUM SHLL 1;
      R2 := AFAILSTATE;
      R3 :=BSEGBASE +BSEGTABLE(R2) + R1; LH(R2,B3);
   END;
   R1 :=BSP SHLL 2; STATESTACK(R1) := R2;
   R2 := R2 AND MASK; STATENUM := R2;
END;
COMMENT  CASE 6, LOCK BACK;
BEGIN
   R1 :=BSP - 1 SHLL 2; R2 := STATENUM;
   R3 := ALBSTART;
   R4 :=BSEGBASE +BSEGTABLE(R3) + R2; IC(R3,B4);
   R3 := R3 AND MASK;
   R4 := ALBNUM;
   R5 :=BSEGBASE +BSEGTABLE(R4) + R2; IC(R4,B5);
   R4 := R4 AND MASK + R3;
   R3 := R3 SHLL 1; R5 := ALBSTATE;
   R7 :=BSEGBASE +BSEGTABLE(R5) + R3;
   LH(R5,B7);
   R6 := STATESTACK(R1); R3 := R3 SHRL 1;
   WHILE R6 ¬= R5 AND R3 < R4 DO
   BEGIN
      R3 := R3 + 1 SHLL 1; R5 := ALBSTATE;
   R7 :=BSEGBASE +BSEGTABLE(R5) + R3; LH(R5,B7);
      R3 := R3 SHRL 1;
   END;
   R3 := R3 SHLL 1; R1 := ARESUMSTATE;
   R7 :=BSEGBASE +BSEGTABLE(R1) + R3; LH(R1,B7);
   R2 :=BSP SHLL 2; STATESTACK(R2) := R1;
   R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT  CASE 7, ERROR;
BEGIN INTEGER PREVERRCYCLE = #FFFFFFFF;
```

```
RO := @WBUF; WRITE;
R1 := CYCLECNT - 2; CYCLECNT := R1;
IF R1 ¬= PREVERRCYCLE THEN
BEGIN
   PREVERRCYCLE := R1; R1 :=BSP - 1 SHLL 2;
   R2 := STATESTACK(R1);
   IF R2 < 512 THEN
   BEGIN
      R2 := R2 AND MASK;
      R3 := ASYMBEFOREREAD;
      R4 :=BSEGBASE +BSEGTABLE(R3) + R2;
      IC(R3,B4);
   END ELSE
   BEGIN
      R2 := STATENUM; R3 := ESYMBEFORELA;
      R4 :=BSEGBASE +BSEGTABLE(R3) + R2; IC(R3,B4);
   END;
   R3 := R3 AND MASK; R2 := 33;
   FOR R7 := 0 STEP 1 UNTIL 1 DO
   BEGIN
     VPT:=R3; BFIND; R1:=LENGTH-1;
     FOR R6 := 0 STEP 1 UNTIL R1 DO
     BEGIN
       IC(R5,BCD(R6)); STC(R5,WBUF(R2));
       R2 := R2 + 1;
     END;
     R2 := R2 + 2; R3 := STATENUM;
     IF R3 = 255 THEN R3 := NEXTSYMBOL
     ELSE R3 := LASYMBOL;
     R3 := R3 AND MASK;
   END;
   R4:=4; XR:=R4; ERROR;
   MVC(131,WBUF,BLANK);
   MVC(17,WBUF,"PARTIAL PARSE IS: ");
   RO := @WBUF; WRITE; MVC(17,WBUF,BLANK);
   R2 :=BSP - 1 SHLL 2;
   FOR R1 := 8 STEP 4 UNTIL R2 DO
   BEGIN
      R3 := STATESTACK(R1);
      IF R3 < 512 THEN
      BEGIN
         R3 := R3 AND MASK;
         R4 := ASYMBEFOREREAD;
         R5 := BSEGBASE + BSEGTABLE(R4) + R3;
         IC(R4,B5);
      END ELSE
      BEGIN
         R3 := R3 AND MASK;
```

```
            R4 := BSYMBEFORELA;
            R5:=BSEGBASE + BSEGTABLE(R4) + R3;
            IC(R4,B5);
         END;
         R4 := R4 AND MASK; VPT := R4; BFIND;
         MVC(63,WBUF(4),BCD); WRITE;
         MVC(63,WBUF(4),BLANK);
      END;
   END;
   R1 := NEXTSYMBOL;
   IF R1 = 1 THEN
   BEGIN
      R4:=5; XR:=R4; ERROR;
   END ELSE
   BEGIN
      VPT:=R1; BFIND; R1:=LENGTH-1;
      MVC(16,WBUF,"THE INPUT SYMBOL,");
      MVC(63,WBUF(20),BCD);
      MVC(16,WBUF(60)," WILL BE IGNORED:");
      R0 := @WBUF;
      WRITE; MVC(131,WBUF,BLANK);
   END;
   R1 := STATENUM;
   IF R1 = 255 THEN
   BEGIN
   COMMENT  ERROR OCCURRED IN A READ STATE;
      R1 :=BSP - 1;BSP := R1; R1 := R1 SHLL 2;
      R2 := STATESTACK(R1) AND MASK;
      STATENUM := R2;
   END ELSE
   BEGIN
   COMMENT  ERROR OCCURRED IN A LOOK-AHEAD STATE;
      SCAN;  COMMENT  SKIP THE NEXT SYMBOL;
      R1 := R1 SHLL 1;
   R2 := ASUCCSTATE;
   R4 :=BSEGBASE +BSEGTABLE(R2) + R1;
      LH(R2,B4); R2 := R2 AND MASKFF00;
      LH(R3,B4); R3 := R3 AND MASK;
      R4:=ANUMTOPOP; R5:=BSEGBASE+BSEGTABLE(R4)+R3;
      IC(R4,B5); R4 := R4 AND MASK;
      IF R2 = 512 AND R4 ¬= 255 THEN
         R1 := STATENUM OR #0000C300
      ELSE R1 := STATENUM OR #0000C400;
      R2 :=BSP SHLL 2; STATESTACK(R2) := R1;
   END;
END;  COMMENT  END OF CASE 7;
BEGIN COMMENT CASE 8. EXIT;
   R1:=1; BTOKEN:=R1; R1:=0;
```

```
                    BSP := R1; STATESTACK(R1) := R1;
                     PTRSTACK(R1):=R1;
                     STATENUM := R1; GOTO XXX;
                  END;
               END;  COMMENT  END OF CASE(STATETYPE);
            END; COMMENT END OF WHILE BLOCK;
XXX:        LM(R0,R7,SAVEREGS);
    END; COMMENT END OF BANALYZE;


GLOBAL PROCEDURE INITTWO(R4); COMMENT CONTINUATION OF INITIALLIZE;
 BEGIN ARRAY 5 INTEGER SAVEREGS; STM(R0,R4,SAVEREGS);
 COMMENT PRELOADING OF RESERVED WORDS INTO SYMBOL TABLE, AND INITIAL-
 LIZATION OF SEMAN AND STCHAIN;
 COMMENT FIRST INITIALIZATION OF SEMAN AND STCHAIN (TO -1) ;
 R2:=SEMBASE-1; COMMENT R2 WILL HAVE ADDRESS OF SEMAN(0 TO 93);
 R3:=STCHAINBASE-2; COMMENT R3 WILL HAVE ADDRESS OF STCHAIN(0 TO 93);
   R0:=-1;
   FOR R1:=55 STEP 1 UNTIL 148 DO
 BEGIN COMMENT R1 HAS SEMANTIC VALUE;
   R2:=R2+1; STC(R1,B2); R3:=R3+2; STH(R0,B3);
 END;

 COMMENT INITIALIZATION OF IDENTRY AND HASHTBL;
 R4:=R9; R1:=STBASE;
 MVC(9,B1, "ABS        "); R0:=0; R2:=R4+1332; STH(R0,B2);
 R1:=R1 +10; MVC(9,B1,"AUTO_DD   "); R0:=1; R2:=R4+940 ; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"AND        "); R0:=02; R2:=R4+2040; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"ANDL       "); R0:=03; R2:=R4+0168; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"BEGIN      "); R0:=04; R2:=R4+1022; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"BITS       "); R0:=05; R2:=R4+0000; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"BITSTRING "); R0:=06; R2:=R4+1982; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"BOOLEAN    "); R0:=07; R2:=R4+2162; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"CALL       "); R0:=08; R2:=R4+1944; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"CASE       "); R0:=09; R2:=R4+1378; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"CAT        "); R0:=10; R2:=R4+1856; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"CIRSHLL    "); R0:=11; R2:=R4+1936; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"CIRSHRL    "); R0:=12; R2:=R4+0246; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"CHARCODE   "); R0:=13; R2:=R4+1902; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"CODECHAR   "); R0:=14; R2:=R4+0720; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"DECODE     "); R0:=15; R2:=R4+2092; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"DO         "); R0:=16; R2:=R4+0188; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"ELSE       "); R0:=17; R2:=R4+0364; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"ENCODE     "); R0:=18; R2:=R4+1166; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"END        "); R0:=19; R2:=R4+0060; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"END_DD     "); R0:=20; R2:=R4+1164; STH(R0,B2);
 R1:=R1+10; MVC(9,B1,"END_TABLE "); R0:=21; R2:=R4+1450; STH(R0,B2);
```

```
R1:=R1+10;  MVC(9,B1,"EXTREF    ");  R0:=22;  R2:=R4+1272;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FIELD     ");  R0:=23;  R2:=R4+1102;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FIXED     ");  R0:=24;  R2:=R4+1768;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FLOAT     ");  R0:=25;  R2:=R4+2124;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FOR       ");  R0:=26;  R2:=R4+1782;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FORMAT    ");  R0:=27;  R2:=R4+2394;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FROM      ");  R0:=28;  R2:=R4+2042;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FLDOVERLAY");  R0:=29;  R2:=R4+1046;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"FUNCTION  ");  R0:=30;  R2:=R4+2306;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"GOTO      ");  R0:=31;  R2:=R4+1990;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"IF        ");  R0:=32;  R2:=R4+1210;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"INPUT     ");  R0:=33;  R2:=R4+0610;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"INTEGER   ");  R0:=34;  R2:=R4+2086;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"INTO      ");  R0:=35;  R2:=R4+0182;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"INDIRCT   ");  R0:=36;  R2:=R4+1850;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"ITEM_AREA ");  R0:=37;  R2:=R4+1748;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"LOC_DD    ");  R0:=38;  R2:=R4+2426;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"NOT       ");  R0:=39;  R2:=R4+0522;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"NOTL      ");  R0:=40;  R2:=R4+2428;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"NUMBER    ");  R0:=41;  R2:=R4+0768;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"OCM       ");  R0:=42;  R2:=R4+0458;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"OR        ");  R0:=43;  R2:=R4+0958;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"ORL       ");  R0:=44;  R2:=R4+1604;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"OF        ");  R0:=45;  R2:=R4+0126;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"OUTOF     ");  R0:=46;  R2:=R4+0908;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"OUTPUT    ");  R0:=47;  R2:=R4+0574;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"OVERLAY   ");  R0:=48;  R2:=R4+2112;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"PACKIN    ");  R0:=49;  R2:=R4+1148;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"PRINT     ");  R0:=50;  R2:=R4+2406;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"PROCEDURE ");  R0:=51;  R2:=R4+1496;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"PNCH      ");  R0:=52;  R2:=R4+1418;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"READ      ");  R0:=53;  R2:=R4+1824;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"REM       ");  R0:=54;  R2:=R4+2094;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"RESUME    ");  R0:=55;  R2:=R4+2348;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"RETURN    ");  R0:=56;  R2:=R4+0562;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"RETURNTO  ");  R0:=57;  R2:=R4+2392;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SEARCH    ");  R0:=58;  R2:=R4+1868;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SELECT    ");  R0:=59;  R2:=R4+1922;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SHLL      ");  R0:=60;  R2:=R4+0928;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SHRL      ");  R0:=61;  R2:=R4+1696;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"STEP      ");  R0:=62;  R2:=R4+1566;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SUBCHAR   ");  R0:=63;  R2:=R4+0240;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SYSTEM    ");  R0:=64;  R2:=R4+1282;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SYS_DD    ");  R0:=65;  R2:=R4+1122;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SYS_PROC  ");  R0:=66;  R2:=R4+1514;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SYS_PROC_R");  R0:=67;  R2:=R4+0134;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"SUBTABLE  ");  R0:=68;  R2:=R4+0320;  STH(R0,B2);
R1:=R1+10;  MVC(9,B1,"CHAR      ");  R0:=69;  R2:=R4+0462;  STH(R0,B2);
```

```
   R1:=R1+10;  MVC(9,B1,"STATUS    "); R0:=70;  R2:=R4+1188;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"SET       "); R0:=71;  R2:=R4+1356;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"SWAP      "); R0:=72;  R2:=R4+1050;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"TAG       "); R0:=73;  R2:=R4+0708;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"TABLE     "); R0:=74;  R2:=R4+1098;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"TC        "); R0:=75;  R2:=R4+2330;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"THRU      "); R0:=76;  R2:=R4+0006;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"THEN      "); R0:=77;  R2:=R4+2332;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"UNPCK     "); R0:=78;  R2:=R4+1422;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"VARY      "); R0:=79;  R2:=R4+0672;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"VRBL      "); R0:=80;  R2:=R4+0112;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"WHILE     "); R0:=81;  R2:=R4+1652;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"FALSE     "); R0:=82;  R2:=R4+1396;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"TRUE      "); R0:=83;  R2:=R4+0510;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"CCBEGIN   "); R0:=84;  R2:=R4+1462;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"COEND     "); R0:=85;  R2:=R4+2416;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"FREE      "); R0:=86;  R2:=R4+2294;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"RESERVE   "); R0:=87;  R2:=R4+2340;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"WAIT      "); R0:=88;  R2:=R4+1806;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"CCRAD     "); R0:=89;  R2:=R4+2314;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"COUNT     "); R0:=90;  R2:=R4+0020;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"STOP      "); R0:=91;  R2:=R4+1284;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"INTERRUPT "); R0:=92;  R2:=R4+0042;  STH(R0,B2);
   R1:=R1+10;  MVC(9,B1,"SUBSCRPT  "); R0:=93;  R2:=R4+0892;  STH(R0,B2);
     FCINTFREE:=R0;
   LM(R0,R4,SAVEREGS);
   END; COMMENT END OF INITTWO;

   GLOBAL PROCEDURE INITIALIZE(R4); COMMENT IT SETS UP THE
                          CONDITIONS NEEDED TO START EXECUTION OF THE
                          COMPILER;
   BEGIN  ARRAY  5 INTEGER SAVEREGS; STM(R0,R4,SAVEREGS);

        R0 := @WBUF;
        MVC(43,WBUF,"   CMS-2RS COMPILER _  VERSION OF JUNE 1973.");
        WRITE; MVC(72,WBUF,BLANK); WRITE; WRITE; PRINTDATE;
        PRINTIME; MVC(8,WBUF,"TODAY IS ");
        MVC(0,WBUF(16),"@");
     R0:=@WBUF;
        WRITE; MVC(131,WBUF,BLANK); WRITE; WRITE;

 COMMENT  INITIAL SETUP BEFORE CALLING GETCHAR FOR THE FIRST TIME. CP
          AND CARDCOUNT ARE INITIALLIZED IN DECLARATIONS;

   R0:=@CBUF; READ;
   IF LISTFLAG THEN
   BEGIN
     R0:=CARDCOUNT; CVD(R0,CONWORK); UNPK(3,7,WBUF(15),CONWORK);
```

```
      SETZONE(WBUF(18)); MVC(79,WBUF(22),CBUF);
      R0:=@WBUF; WRITE; MVC(131,WBUF,BLANK);
    END;
  GETCHAR;

      SEGBASE := R12; COMMENT  SAVE BEGINNING ADDRESS OF PARSING TABLES
                       FOR MAIN GRAMMAR;
      BSEGBASE:=R11; COMMENT SAVE BEGINNING ADDRESS OF PARSING TABLES FOR
                            DATA DECLARATIONS GRAMMAR;
         R1 := NUMTERMINALS + 1 SHRL 3;
         R2 := NUMTERMINALS + 1 AND #7;
         IF R2 > 0 THEN R1 := R1 + 1;
         LATABSIZE := R1;
       MVC(2,VSTRING(10),"EOF");

      R1:=BNUMTERMINALS + 1 SHRL 3;
      R2:=BNUMTERMINALS + 1 AND #7;
      IF R2 > 0 THEN R1:=R1+1;
      BLATABSIZE:=R1;
      MVC(2,BVSTRING(10),"EOF");


  COMMENT SETTING-UP OF SYMBOL TABLE MEMORY ALLOCATION, BASE ADDRESSES
  AND LIMITS;
    GETCORE; COMMENT GETCORE RETURNS BASE ADDRESS AND LENGTH OF MEMORY
                     ALLOCATED TO SYMBOL TABLE;
    STBASE:=R1; COMMENT BASE ADDRESS OF SYMBOL TABLE;
    STLENGTH:=R0;
    R0:=R0 SHRL 1; COMMENT R0 IS LENGTH OF IDENT DIRECTORY SECTION;
    R2:=R0+R1; CTBASE := R2; COMMENT CONSTANT TABLE BASE ADDRESS;
    R2:=R2+R0;
    CTLIMIT:=R2   ; COMMENT CONST. TBL. ABSOLUTE LIMIT;
    R1:=R0; R0:=R0-R0; R1:=R1/15; COMMENT 15 BYTES PER ENTRY. ;
    STLIMIT:=R1; COMMENT R1 HAS THE NUMBER OF ENTRIES POSSIBLE;
    R3:=R1*10; COMMENT CHUNCK ALLOCATED TO IDENTRY;
    R0:=STBASE+R3; SEMBASE:=R0; COMMENT SEMANTIC FIELD BASE ADDRESS;
    R0:=SEMBASE+R1;
    R0:=R0 AND #FFFFFFFE+2; COMMENT THIS DONE TO GUARANTEE R3 HAS EVEN
                            VALUE AND DO NOT OVERLAP PREVIOUS ARRAY;
    STCHAINBASE:=R0; COMMENT STCHAIN FIELD BASE ADDRESS;
    R3:=R1 SHLL 1; COMMENT CHUNK ALLOCATED TO STCHAIN;
    R0:=R0+R3;
    DESCRPTRBASE:=R0; COMMENT BASE ADDRESS OF DESCRIPTOR POINTER FIELD;

      INITTWO;
      MVC(9,PREFIX,"00       ");
      R1:=@BANALYZE; BANALBASE:=R1;
      R1:=@SCAN; SCANBASE:=R1;
```

```
     LM(RC,R4,SAVEREGS);
   END; COMMENT END OF INITIALIZE;

   PROCEDURE MAIN(R4);
     BEGIN
     INITIALIZE;
     ANALYZE;
     PRINTSUMMARY;
   ENC; CCMMENT END OF MAIN;

   MAIN;
 EXIT:
   MVC(17,WBUF,"END CF CCMPILATION"); RO:=@WBUF; WRITE;

 ENC.
```

LIST OF REFERENCES

1. Fleet Computer Programming Center, Pacific, Compiler Monitor System-2 (CMS-2) User's Reference Manual, M-5012, v. I, II, and III.

2. UNIVAC, Defence Systems Division, CMS-2 User's Reference Manual.

3. UNIVAC, Defense Systems Division, CMS-2 Study Guide, PX6346.

4. Dijkstra, E. W., "Notes on Structured Programming," Technische Hogescool Eindhoven, 1969.

5. Dijkstra, E. W., "Structured Multiprogramming in Software Engineering Techniques," NATO Science Committee, p. 88-93, 1969.

6. Bohm, C., and Jacopini, G., "Flow Diagrams, Turing Machines and Languages With Only Two Formation Rules," Communications of the ACM, v. 9, p. 366-371, May 1966.

7. Bauer, H. R., and others, Algol W Language Description, Stanford University, Computer Science Department, 1969.

8. Hansen, P. B., "Structured Multiprogramming," Communications of the ACM, v. 15, p. 574-578, July 1972.

9. DeRemer, F. L., "Simple LR(k) Grammars," Communications of the ACM, v. 14, p. 453-460, July 1971.

10. Hopcroft, J. E., and Ullman, J. D., Formal Languages and their Relation to Automata, Addison-Wesley, 1969.

11. Woods, R. A., "A PL-360 Based Compiler Generating System," Naval Postgraduate School Thesis, December 1972.

12. Gries, D., Compiler Construction for Digital Computers, Wiley, 1971.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor G. A. Kildall, Code 53Kd Naval Postgraduate School Monterey, California 93940	1
4.	Chairman, Computer Science Group, Code 72 Naval Postgraduate School Monterey, California 93940	1
5.	LTJG R. H. Brubaker, Code 53Bh Naval Postgraduate School Monterey, California 93940	1
6.	LCDR V. C. Secades HSL-31, NAS Imperial Beach San Diego, California 92154	1
7.	LT D. C. Rummler U.S.S. Long Beach, CGN-9 F.P.O. San Francisco, California 96601	1
8.	MR. M. A. Lamendola, Code 5200 Naval Electronics Laboratory Center San Diego, California 92152	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

IGINATING ACTIVITY (Corporate author) Naval Postgraduate School	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

PORT TITLE

Steps Toward a Revised Compiler-Monitor System (CMS-2)

SCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; June 1973

THOR(S) (First name, middle initial, last name)

Vincent C. Secades
David C. Rummler

PORT DATE June 1973	7a. TOTAL NO. OF PAGES 223	7b. NO. OF REFS 12
ONTRACT OR GRANT NO. ROJECT NO.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

ISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

UPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

BSTRACT

This paper describes a proposal for a revised Compiler Monitor System II (CMS-2). Primary emphasis is placed on design improvements to the CMS-2 compiler and language. Changes to the Monitor and Librarian which support the above improvements are discussed or implied where appropriate. A new concept is proposed, called multi-level programming, which allows the system designer to define the levels of language constructs which are appropriate for the various types of program modules in a large self-contained software system. The approach taken is to design a language and compiler-monitor system (CMS-2RS) which will facilitate the multi-level programming concept and the top-down programming method of software engineering in a production library environment.

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
S-2						
mpiler Generator System						
lti-Level Programming						
ogramming Language						
R(1) Grammar						
ructured Programming						

145222
Thesis
S4062 Secades
c.1 Steps toward a revised
compiler-monitor system
II (CMS-2).