NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS
AN INPUT TRANSLATOR FOR A
COMPUTER-AIDED DESIGN SYSTEM

by
Thomas H. Carson
June 1984

Thesis Advisor: Alan A. Ross

Approved for public release; distribution unlimited

84 09 28 046
An Input Translator for a Computer-Aided Design System

Thomas H. Carson

Naval Postgraduate School
Monterey, California 93943

June, 1984

Naval Postgraduate School
Monterey, California 93943

Approved for public release; distribution unlimited

ABSTRACT (Continued)

translate it into primitive list form. This form is used by the remainder of the system to select the best hardware and software components, as described in a set of realization libraries, to build the proposed controller.
An Input Translator for a Computer-Aided Design System

by

Thomas H. Carson
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1971

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

Author: ____________________

Approved by: ____________________

Thesis Advisor

Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences
The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment, which is a computer-aided design system. The Computer System Design Environment is used to design real-time controllers for a variety of purposes. The input translator will take an input, which has been developed in the prescribed language, CSTL, and with the aid of a partial syntax-directed editor, translate it into primitive list form. This form is used by the remainder of the system to select the best hardware and software components, as described in a set of realization libraries, to build the proposed controller.
# TABLE OF CONTENTS

I. INTRODUCTION AND BACKGROUND ...................................... 8  
   A. COMPUTER-AIDED DESIGN ........................................... 8  
   B. COMPUTER SYSTEM DESIGN ENVIRONMENT ............................. 9  
   C. PREVIOUS WORK .................................................... 10  
   D. SCOPE OF THIS THESIS ............................................. 12  

II. TRANSLATOR DESIGN .......................................................... 13  
    A. DESIGN REQUIREMENTS AND CONSTRAINTS ............................ 13  
    B. CSDL- THE INPUT LANGUAGE ........................................ 14  
    C. PARSER ALTERNATIVES .............................................. 17  
    D. PARSER STRUCTURE ................................................ 19  

III. TRANSLATOR IMPLEMENTATION .............................................. 21  
    A. LANGUAGE OF IMPLEMENTATION ...................................... 21  
    B. TRANSLATOR INPUT ................................................. 22  
    C. PRINCIPAL PROCEDURES AND DATA STRUCTURES ........................ 23  
    D. TRANSLATOR OUTPUT ............................................... 26  
       1. Primitive List .................................................. 26  
       2. Symbol Table ................................................... 27  
       3. Scratch Pad .................................................... 28  

IV. TESTING AND VALIDATION .................................................. 30  
    A. THEORY OF TESTING ............................................... 30  
    B. TEST RESULTS ..................................................... 31  

V. CONCLUSIONS AND RECOMMENDATIONS ...................................... 38  
    A. PROGRAM MAINTENANCE .............................................. 38  
    B. RECOMMENDATIONS FOR CSDE ........................................ 39  
    C. SUMMARY .......................................................... 40
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Computer System Design Environment</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>A Partial Example of CSDL</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>An Example of CSDL Syntax</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Parser Structure</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>A Sample of the Primitive List</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>IDENTIFICATION Section of the Input</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>DESIGN CRITERIA Section of the Input</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Primitive List Form of the DESIGN CRITERIA</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>ENVIRONMENT Section of the Input</td>
<td>33</td>
</tr>
<tr>
<td>4.5</td>
<td>Primitive List Form of the Input</td>
<td>34</td>
</tr>
<tr>
<td>4.6</td>
<td>PROCEDURES Section Input Example</td>
<td>35</td>
</tr>
<tr>
<td>4.7</td>
<td>Primitive List Form of the PROCEDURES Section</td>
<td>36</td>
</tr>
<tr>
<td>4.8</td>
<td>CONTINGENCY LIST Input Example</td>
<td>36</td>
</tr>
<tr>
<td>4.9</td>
<td>Primitive List Form of the CONTINGENCY LIST</td>
<td>37</td>
</tr>
</tbody>
</table>
I. INTRODUCTION AND BACKGROUND

A. COMPUTER-AIDED DESIGN

Computers, as design tools, are beginning to touch every facet of our lives. We can turn on our television sets to see an advertisement for an automobile with a computer being used to aerodynamically design the body style of the car. Architects are using computers to assist in modern drafting and architectural design techniques (Ref. 1). The potential for automating the design of many other products exists and research in all areas of computer-aided design (CAD) is continuing at a prolific rate.

While research in the areas of artificial intelligence may lead us someday to a computer which can, using natural language understanding, solve problems reserved for only humans today, current technology limits us to those problems where the computer relies on human expert input for a knowledge base. A knowledge base developed by human experts is used by the computer to derive a design much faster than a human and reduce the concern over the complexity of the process. The computer can maintain a large data base of components from which it can pick to satisfy a criteria as described by the designer. While not really creative, this system allows the mixing and matching, automatically, of components to produce the best combination available. The advantages to be gained are a decrease in the time it takes to complete the design process and error free results, while leaving the human designer to concentrate on the desired specification.
One of the most important features of such a system is its accessibility. The user interface must be one that meets the needs of the designer while remaining within the bounds of current technology. The interface must be user-friendly to the greatest possible extent. It is this particular portion of the problem which has given rise to the most debate and brought forth the widest range of possible solutions.

B. COMPUTER SYSTEM DESIGN ENVIRONMENT

The Computer System Design Environment (CSDE), under development at the Naval Postgraduate School in Monterey California, is one such computer-aided design system. It is based on the research contained in LtCol. Alan Ross’s doctoral dissertation [Ref. 2]. Ross’s work is an expansion and realization of the research conducted by M. W. Matelan [Ref. 3]. The first CSDE implementation is one in which real-time controllers (microprocessors) are designed based on a realization library (knowledge base) of current microprocessor technology. The system creates the problem statement in a syntax-directed editor, translates it into an intermediate form, selects a microprocessor realization from the library and generates the software and hardware component descriptions to implement the design. The components are used to select a processor volume or implementation. The volume checked to see if the timing constraints, set forth by the designer, can be achieved. If so, the monitor is generated and the output is formatted. The monitor produces the software and ancillary hardware to realize the correct strategy. If the timing constraints cannot be met, a new volume must be chosen and tested. The CSDE gives a designer the tools to derive the appropriate components that make up the controller, no matter what its task is to be.
Motivation and discussion of the CSDE are contained in [Refs. 2, 3].

C. PREVIOUS WORK

The modules that make up the Computer System Design Environment are depicted in figure 1.1. Matelan described a Control System Design Language (CSDL) as the input language for this system [Ref. 3]. Using CSDL with a syntax-directed editor keeps the input details at a high-level of abstraction while completely describing the proposed design.

To fulfill the input requirements of block 1 in figure 1.1, a syntax-directed editor was designed and partially implemented as a result of Lt. Barbara Sherlock's thesis at the Naval Postgraduate School in 1983 [Ref. 4]. Lt. Sherlock's editor receives a high level input description of the problem from the designer, formats it and passes it to the input translator. This form is a combination of Matelin's CSDL and ADA, the Department of Defense sponsored design language. This language, as the basis for input to and output from the editor, follows the concept that the problem statement should not require the designer to be proficient in the details of a high level programming language. The translator, as its name suggests, translates the output from the editor into an intermediate form acceptable to the follow on CSDE processes. Its design and implementation are the subjects of this thesis. The optimizer and functional mapper (Blocks 3 and 4, Figure 1.1) exist as Fortran programs in the CSDE. The optimizer requires an 80 column format for its input which is a primitive list or the set of functions that the controller will perform, in an almost assembly-like language format. It is developed from the contingency and procedures sections of the design input statement.
1. State Problem
   Define Algorithm

   Designer

   2. Translate to the Primitive Format.

   Translator

   3. Select a Realization Volume.

   Optimizer

   Volume to Try?

   YES

   NO

   Stop - Process Complete - Report Success or Failure.

   Optimizer

   4. Generate a Functional Realization

   Functional Mapper

   Functionally Feasible Realization?

   YES

   NO


   Timing Analyzer

   Feasible Monitor?

   YES

   NO

   6. Generate the Output Listing.

   Formatter

   Need one Realization or several?

   SEVERAL

   DNE

Figure 1.1 Computer System Design Environment.
prescribed this format when the CSDE was originally implemented by LtCol. Ross. Work is currently in progress to install both the design input and the realization library in a relational database, thereby updating the optimizing and mapping processes [Ref. 5]. But the basis of the input to these sections of the system, which is the primitive list, will remain the same.

D. SCOPE OF THIS THESIS

The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment. The translator must take the problem statement, a design for a real time microprocessor controller, and translate it into a list of primitives and a symbol table which can then be mapped across a realization library to determine the most feasible components. Appendix A is an example of a problem statement in the Computer System Design Language developed by Matelan. In addition, the translator must produce a timing table which can be used by the timing analyser to determine a feasible monitor to control the complete device [Ref. 2]. The timing table is developed using the contingency section of the problem statement which contains the timing requirements for the problem.

The next chapter of this thesis describes the design considerations for the translator and provides the detailed discussion of its requirements. The following chapters discuss the implementation and testing of the translator and the conclusions reached during the work. In addition, Appendix D includes the code for the translator with appropriate documentation.
II. TRANSLATOR DESIGN

A. DESIGN REQUIREMENTS AND CONSTRAINTS

As previously discussed, the input translator can be thought of as one of several modules in the CSDE system. In the CSDE hierarchy, it lies between the designer and the optimizer. If we consider, for a moment, the module as a black box, then we can better describe its function. The input to the module is a specification, written by the designer, for a real-time controller. While Lt. Sherlock, in her design of the input editor [Ref. 4], decided to produce a pseudo-ADA specification language as the output of the editor, the translator being designed by this author will use the Control System Design Language. If the ADA output was to be adapted, this language would have to be formalized and a grammar produced that is capable of being parsed. In addition, the pseudo-ADA provides no real advantage, as the specification must be parsed and the same output produced no matter what the language. Therefore, with the additional knowledge that the editor is the subject of a current thesis project which will return to the CSDL output, the decision was made to write the translator for that CSDL. A partial example of the CSDL description is contained in figure 2.1.

The output from the translator consists of a primitive list, a symbol table, and an application timing table. The primitive list is intermediate code which reflects the requirements of the input while the symbol table contains all input variables and their attributes. The application timing table contains the contingencies with their related tasks and all supplied timing values from the problem definition. This table is used during the timing analysis.
CONTINGENCY LIST

WHEN ALARM : 2MS, 50US DO ALERT;
EVERY 4MS : DO ENCODE;
WHEN DATA_READY : 1300US DO SERIALIZE;

PROCEDURES

FUNCTION DATA_READY:
  BINARY, 1;
  SENSE (BUFFER);
  IF BUFFER /= OLDBUF THEN DATA_READY := 1 END IF;
  EXIT DATA_READY;

Figure 2.1 A Partial Example of CSDL.

The requirements, then, exist for the input language, CSDL, to be analyzed to determine what method of translation is to be employed. In addition, the required output must be standardized among the system modules so the proper semantics can be developed for the translation process. Each of these issues will be discussed in detail in the following sections.

B. CSDL- THE INPUT LANGUAGE

A translator accepts a source program, written in a source language, and transforms it into an object program [Ref. 6]. A source language designed for use in a computer aided design system and utilized in CSDE is the Control System Design Language (CSDL), the origin of which was previously discussed. It is composed of an alphabet whose individual elements are called tokens and a grammar which expresses the rules governing the legal classes of token strings. The tokens can be further subdivided into terminals
and nonterminals. Terminals are the letters of the allowed alphabet while the nonterminals are representations of strings in the language which increase the expressive power. A partial example of the production rules for CSDL is contained in figure 2.2. The syntax for CSDL, which includes the alphabet and grammar, is contained in Appendix B.

\[
\text{<WHEN DO> ::= <QUALIFICATION> WHEN <NAME> \\
   <EPISODE TIMING> DO <TASK LIST>}
\]
\[
\text{<TASK LIST> ::= <NAME> / <TASK LIST> THEN <NAME>}
\]
\[
\text{<NAME> ::= *ID* / *ID* (EXPR LIST) / *ID* *NUMBER* : *NUMBER*}
\]

Figure 2.2 An Example of CSDL Syntax.

Different types of translating devices accept different languages classifications. To narrow the choice of translator designs, it must be determined which classification fits CSDL. Neither Matelan nor Ross, in their early work on CSDE, included this description [Refs. 2, 3]. So, a brief review of language classification will assist in this determination. Chomsky distinguished four general classes of grammars [Ref. 7]. Without turning this section into a text on language theory, these are, from the most general to the most specific: unrestricted, context-sensitive, context-free, and right-linear. Context-sensitive and context-free are subsets of the unrestricted class, while the right-linear and two other grammars related to the right-linear
grammars, left-linear and regular, are all subsets of the context-free grammars. These classes allow us to define sentence recognizing machines which form the basis for translators. CSDL falls into the category of context-free grammars. This is the set which, in its production rules, has any string of terminals and nonterminals on the right-hand side of the production while the left-hand side is restricted to nonterminals only. The classes of right-linear, left-linear, and regular grammars restrict the order and appearances of terminals and nonterminals on each side of the production rules and CSDL does not fall into one of these categories. Context-sensitive grammars allow terminals as well as one nonterminal on the left-hand side of the production rules and while CSDL does fit this category, the context-sensitive are a super-set of the context-free grammars, so this is not an issue when we try to develop the machines which can recognize CSDL.

Each of the phrase-structured grammar classes has an automaton associated with it. The right-linear grammars can be recognized and accepted by a finite-state automaton which consists of a finite set of states and a set of transitions between pairs of states. Each transition is associated with some terminal symbol. The context-sensitive grammars are recognized and accepted by a two-way, linear bounded automaton which is essentially a Turing machine whose tape cannot grow longer than the input string. And, finally, context-free grammars are recognized and accepted by a finite-state automaton controlling a push-down stack, with rules governing the operations on the stack [Ref. 6].

Matelan states that CSDL was created as a context-free grammar and inspection of the syntax contained in Appendix B confirms this [Ref. 3]. In order to recognize strings in the language and translate them into the prescribed primitive-list format a finite-state automaton with a push-down stack will be developed.
C. PARSER ALTERNATIVES

It was shown above that CSDL is a context-free grammar and a push-down automata will be required as the recognizer for strings in the language. Additional properties of CSDL must be investigated to further define the problem of parsing. In a context-free grammar each nonterminal can be expanded into some terminal string independently of its neighbors, and its expanded string essentially "pushes aside" its neighbors without interfering with their order in any way [Ref. 6]. But we do impose some ordering rule for the selection of the next nonterminal to replace, in a sentential form for a canonical derivation. The most common rules are left-most and right-most. In a left-most derivation, the left-most nonterminal in each sentential form is selected for the next replacement and in a right-most, the right-most nonterminal is selected. Most common programming languages are easily parsed from left to right, but with difficulty from right to left. Furthermore, algebraic operations are usually performed from left to right, by convention, so it is the order that will be considered. A top-down parse of some sentence, scanning from left-to-right through the string, corresponds to a left-most derivation while a bottom-up parse works from a given sentence upward toward the start symbol, in a left-to-right manner. [Ref. 6].

There are some rules governing the use of these two types of parsers which affect the choice of one for use in recognizing CSDL. Top-down recognition with a look-ahead of \( k \) symbols is only possible on a subset of the context-free grammars called \( LL(k) \) grammars. Although it is not obvious whether a grammar is \( LL(k) \), there is one property which is relevant in this discussion. An \( LL(k) \) grammar has no left recursive nonterminals, i.e., a nonterminal \( A \), such that \( A \rightarrow \)
Aw for some w, a string in the language [Ref. 6]. It can be quickly determined by examining Production 17 in Appendix B that CSDL is left-recursive. In fact, it is full of recursion. There are algorithms for removing left-recursion in grammars and for a small grammar that would be the choice. But CSDL has 190 production rules and removing the extensive recursion would increase the grammar size an unacceptable amount. So top-down parsing will be discarded as a possible parsing method.

A bottom-up LR(k) parser is the other major type of recognizer under consideration. A grammar is said to be LR(k) if, for every derivation, the production A => x can be inferred by scanning ux and at most the first k symbols of v in the following derivation step: uAv => uAx. The major advantage of this method is that an LR(k) parser can be constructed for any context-free grammar. This would eliminate the necessity to remove the left-recursion from CSDL.

There is one other major advantage in choosing a bottom-up LR(1) parser automaton. An automatic parser generator can construct, using a computer, the language specific tables that control the operation of the automaton. The LR package from Lawrence Livermore Laboratory [Refs. 8, 9] is such a system which constructs the tables. Having it available on the Vax 11/780 at the Naval Postgraduate School made the decision easy. It, also, has the advantages that the parsing routine is guaranteed to be correct, the CSDL grammar can be changed easily when necessary, and the resulting translator becomes simple and efficient. For the details as to how the package works see References 7 and 9.
D. PARSER STRUCTURE

The structure of the parser will follow the technique described by J.J. Myers in his book, *Composite Structured Design* [Ref. 10], and utilized in the design and implementation of an ADA pseudo-machine by Captain Alan Garlington in his thesis [Ref. 11]. The hierarchy for such a technique is depicted in figure 2.3. The top module, PARSE, provides FINDREDUCTION with the current state and current look-ahead symbol. FINDREDUCTION returns a production number if any

![Diagram of parser structure]

**Figure 2.3 Parser Structure.**
reduction exists. PARSE then calls DOREDUCTION and the sequence is repeated. If no reduction exists, PARSE calls FINDTRANSITION to see if any transitions exist. DOTRANSITION accomplishes the transition and the routine repeats until a final state is reached. If no transition exists, an error is detected and the routine either attempts to recover or halts depending on the severity of the error.

The parser maintains two stacks, one to store the next token and one to store the current state. When DOREDUCTION provides SEMANTIC with the production number, the components of the production have been placed on the stack and SEMANTIC can take the proper action. This action could include adding a symbol to the symbol table with its appropriate parameters, calling an error routine, or nothing. After completing the semantic actions the items on the stack are removed by DOREDUCTION and the proper token replaces them. Various auxiliary procedures and error routines will be necessary to complete the translator. These procedures will include the input and output routines required to read the designers input and place the created primitives and symbol table in proper format.

The translator, thus, will take a CSDL description of the desired controller, check for errors, parse the input, and produce the primitive list, the symbol table, and application timing table. The implementation of the translator is discussed in the following chapter.
III. TRANSLATOR IMPLEMENTATION

A. LANGUAGE OF IMPLEMENTATION

The Computer System Design Environment was originally implemented in Fortran, with the system maintaining its data base on formatted punch cards for use in a batch environment. Feasible alternatives exist and have been investigated in subsequent research. CSDE is now installed in the Vax 11/780 at the Naval Postgraduate School, with interactive computing available in a variety of programming languages. However, it is still maintained, mainly, in Fortran. It has been shown to be conceptually feasible to represent the data base requirements of the CSDE by a relational model [Ref. 5]. Future intentions are to realize this concept on a data base management system such as Oracle, which is available on the Vax machine. The importance of this is that format, such as column numbers and location of data, and interface compatibility lose their importance and the relations between the data and the representations of the relations become major concerns.

An additional property required for the implementation language is maintainability. CSDL will not be static, as previously discussed. As hardware technology changes, the realization library will have to be updated and, accordingly, CSDL will have to be modified. This will result in an update to the translator, reflecting, possibly, such changes as new primitives. A high level familiar programming language will ease the burden of maintenance for future users.
Pascal is a high level programming language which supports the above requirements and, therefore, was chosen over the possible alternatives. Pascal is familiar to most programmers and, in fact, is the "first language" taught to new Computer Science students at the Naval Postgraduate School. In addition, it is easily understood by programmers conversant with other block-structured languages.

The modularity available with Pascal's procedures, functions, and high level constructs will provide maintainability. Each major function in the parser will comprise a Pascal procedure, making the main body of the program simple. Also, a section to be modified or updated is self-contained, and can be separately compiled and debugged following changes.

Perhaps the most important reason for selecting Pascal is the expected improvements in the CSDE system database. When the libraries are placed in a relational database, and the designer's input must be mapped to a library in such a system, the data structures comprising the translator's outputs will require change. Pascal data structures are powerful and adaptable to a relational model, enabling this major modification to be completed without difficulty. Until such time as this occurs, the output will be of the form dictated by current system implementation. The details of this will follow.

B. TRANSLATOR INPUT

The input to the translator is the designer's requirements for a controller, written in the Control System Design Language, an example of which is contained in Appendix A. At this writing, a partial syntax-directed editor is under development which will create, based on the designer's ideas, the input in syntactically correct form. No
conceptual basis for the storage or file format of the input has been previously discussed, so several assumptions were made to enable production of the translator. Since the program is to be written in Pascal and reside as a member of the larger system, the CSDE, a simple file containing the CSDL problem description, which is read by the translator, will be utilized as the method of input. The file will be a text file with the only formatting restrictions being those imposed by the syntax of CSDL, the language being parsed. While this method of input will, currently, require some hands on system manipulation during run time, it is envisioned that a system macro can easily be developed at a later date to automate the process.

It is intended that the input be provided to the translator in syntactically correct form. However, as mentioned above, the means for this is not yet implemented. As example problem statements and test cases have been generated to exercise the translator, a requirement has developed for syntax error detection. This requirement can be eliminated and the ensuing code removed upon completion of the editor.

C. PRINCIPAL PROCEDURES AND DATA STRUCTURES

The parser was summarized at a high level of abstraction in Chapter 2 of this thesis. This section will point out the important procedures and data structures employed by the translator. The supporting functions which complete the translator are described as they appear in the program in Appendix D.

The tables produced by the automatic parser generator, which control the operation of the parser, are placed in arrays. The array sizes were set in program constants and would be modified if a change to CSDL caused a modification
to the tables. In addition, the symbol table, the state and lookahead stacks, and the temporary and constant lists are all implemented as arrays of records. Each record contains such information as the type and precision of the variable or constant and a pointer to the next record in the list. With the continuous manipulation of these data structures, such as pops and pushes on the stacks as well as the requirement for access to each member of each list, it was determined that this implementation allowed the maximum degree of flexibility. Also, the size of each data structure was described by a program constant, in order to improve maintenance. A limit might change in a situation where a controller design required a large number of input and output signals or internal variables, exceeding the maximum allowed for a stack.

Four sections conceptually comprise the translator program. The first is the initialization sequence, comprised of the procedure INITIALIZE and supporting functions. This section sets the initial values for all program variables and initializes the temporary and constant lists as well as the input symbol table to null values. It also establishes the SYMTABLE, which is a list containing all reserved words in CSDL, each located by a pointer. This table is used in the program to check each input token to determine whether it is a reserved word, identifier, or operator. Additionally, INITIALIZE includes the procedures for sending the generated primitive list, symbol table, and scratch pad to output files.

The next section is the actual parsing routine, comprised of the procedures PARSE, FINDREDUCTION, DOREDUCTION, FINDTRANSITION, DOTRANSITION, and their supporting procedures and functions. This sequence of code, essentially, repeats itself, looking at each input token, retrieved by GETSYM, and attempts to move through the
required production until a final state is reached. The tokens are placed on the stack and if a reduction can be performed, control moves to the semantic portion of the program. If one cannot be done, the sequence finds the transition and continues movement through the production rule, getting the next input symbol, with the stack unchanged. If no transitions exist, an error is present in the input.

When a reduction number, which corresponds to a production number in the CSDL grammar, is found, the program sequences to the third main section which contains the semantic operations for the program. It includes the procedures SEMANTIC and SEMANTIC1. These are two large case statements which, for each production, do the proper stack operations and send the output information to the procedures in the initialization section to be formatted and filed. The semantic operations are called from within DOREDUCTION, but really comprise a separate module within the structure of the translator.

The last section of the translator is the set of procedures comprising the error handling routines. This includes procedures PRINTERRORS, RECOVER, ERROR, and PRINTLINEERRORS. The parsing routine attempts to recover, using the procedure of the same name, from an error while documenting it to the user. If the parser cannot recover, the program will halt and print the complete list, in a file, of the errors noted prior to the crash. This error handling sequence can be eliminated from the translator, as previously noted, on completion of a syntax directed editor, which would ensure error-free input.
D. Translator Output

1. Primitive List

The primary output of the translator is the primitive list, a sample of which is contained in figure 3.1. This list of primitives is similar to a set of macros with

| P | 1s. generated for:DATAA |
| P | 2s. proc (DATAA:1) |
| P | 3s. sensecond (PL3A:1) |
| P | 4s. eq (A01,PL3A,A01:8,1,8) |
| P | 5s. jmpf (A01,A01:8) |
| P | 6s. assign (DATAA,A01:1,8) |
| P | 7s. loc (A01:) |
| P | 8s. exitproc (DATAA:) |

Figure 3.1 A Sample of the Primitive List.

the operands and attributes corresponding to parameters. It is converted by the Optimizer module, in the CSDE, to the internal format required by succeeding modules [Ref. 2].

In the initial implementation of CSDE there was no front end, i.e., the modules for the designer input and translation to intermediate form did not exist. Therefore, the information required in each primitive was hand generated and formatted by column number so it could be inputed in batch form as a card image file. Because the information is still required in the same format, the output, containing the primitive list and called PRIMFILE, was set up in an identical manner. Each primitive is one line and spaces were added to emulate the blank columns on a punch card. The name of the primitive appears first (software primitives preceded by s., hardware by h.). This is followed by the operand list and selection list (if any) separated by a colon [Ref. 16].

26
The primitive list, also, contains the design criteria, preceded by d., which allows the designer to specify the order of consideration of the realization volumes. The generation of many realizations can also be requested, each of which is presented along with its chip count and power requirements. The final portion of the primitive list is the application timing table, with each line preceded by a beginning A. This table includes the timing constraints and associated requirements. Further details concerning the information present in each column in the sections of the primitive list are contained in Reference 2 and, therefore, will not be discussed here.

The primitive names, such as eq and mult, were chosen to correspond as closely as possible to the operation suggested, but the names in the realization volumes must be identical to the ones emitted by the translator for the CSDE to function. They are easily modified in the semantic portion of the program if required. However, new operations will dictate a modification to the CSDL grammar with additional productions and semantic rules.

2. Symbol Table

The concept of a symbol table, for use by succeeding modules in the CSDE, is a result of this thesis. With the format required in the primitive list, it is difficult for the modules to look up, during the mapping, the attributes such as type, precision, value, and technology for controller variables and constants. It is thought that if this information were available in an easily read form, it would increase the speed of the realization process and raise the level of system efficiency. The functional mapper could read the symbol table first, and generate memory requirements for the controller prior to addressing the operations.
A symbol table can take as many forms as there are formats for a file. The data structure used to hold the information is also arbitrary. At the time of this writing, no decision has been made as to the ideal implementation. So, the information will be dumped into a file in the same format as the primitive list and can later be changed. This information includes all variables with their initial value and precision, all required memory locations, and all constants. In addition, the input and output ports with the expected signal names, technology, and precision are included for possible use. The code to format the symbol table can either be written in a separate small routine to be installed as a system module, or could be added to the translator. Whichever the case, a decision in this matter may well come in time for the code to be included in the final version of the translator for this thesis, in which case it will be reflected in the example symbol table in the appendices.

3. Scratch Pad

A scratch pad file developed naturally as the translator was designed and implemented. Used initially for output, it significantly aided the debugging and verifying of the processes. Error routines, mentioned previously, send error diagnostics to this file. Traces of parser execution which were developed out of necessity as part of the debugging process also needed an output medium. Because of this, the scratch pad became a formal part of the translator. This file, TRANSLATE, is a text file, with no particular format, containing information which can be helpful to the user. If an error is detected in the input, the diagnostic, which traces the error, will appear here, with comments as to possible corrective action.
In addition, three toggles have been included which provide, if desired, 3 types of traces of program execution. TRACEPARSE will trace the parsing action, providing the transition and reduction numbers as the parser moves through the input. TRACETOK provides the input tokens, one at a time, as they are read. PRINTTABLE displays the controller symbol names with their attributes. These toggles are activated by including "-#" followed by a toggle name at the head of the input file. More than one toggle per execution can be utilized, but the ensuing report becomes difficult to comprehend.
IV. TESTING AND VALIDATION

A. THEORY OF TESTING

The theory of software testing is a difficult problem and the subject of extensive research. Preliminary reports from this research indicate varying effectiveness. Dijkstra says that debugging can only show the presence of errors, but never their absence [Ref. 12]. It is commonly agreed that program testing cannot assure program "correctness" except under special circumstances. But the debate over Dijkstra's statement continues because others have developed theorems which challenge his reasoning [Ref. 13].

The most important factor in testing is to have a well-understood goal for the testing process. In the case of the input translator, proving correctness was not at issue. The algorithm for the parser is well proven to be a correct one [Ref. 9], so the detection of bugs in the program was the goal. Methods of testing include top-down versus bottom-up, static versus dynamic, white box versus black box, and other less known systematic approaches. In bottom-up testing, the idea is to build the program with proven (bug free) components, while top-down begins with tests of the highest level using stubs to simulate the activity of the lower level modules. Static testing attempts to demonstrate the truth of an allegation, i.e., it roughly corresponds to bench-testing a power-driven device without applying power, and dynamic testing seeks to exercise a program in a controlled and systematic way. The white box testing approach is, knowing any part of the software system is present for some specific reason, then relating each piece of a software system to the requirement it fulfills. The black box method
is an extensive testing approach which attempts to demonstrate the presence of function by concentrating on the exterior specifications of the software system [Ref. 14].

Just these brief examples illustrate the fact that software testing is not a well-defined science. The primary reason for this is that the concern for software reliability is relatively new. Only in the past decade has any notable effort been expended in understanding how a program can be proven correct or demonstrated to be reliable.

In selecting possible approaches to establish the reliability of the input translator, a combination of methods was determined to be the best. As previously noted, program correctness is not the objective. Also, the individual modules were completely debugged as they were built. So the problem reduces to ensuring the function between the input and output is such that the correct output is realized for each possible input. This is a black box methodology, but, since the program is to be exercised as a whole, the concept of dynamic testing also applies. The following section discusses the results of this testing and validation.

B. TEST RESULTS

Functional testing, a form of dynamic testing, involves the testing of a system over each of the different possible classes of input, the testing of each function implemented by the system, and the generation of test output in each of the possible output classes [Ref. 15]. This is the methodology that was employed in the testing of the input translator.

The CSDL input is divided into 5 parts which are IDENTIFICATION, DESIGN CRITERIA, ENVIRONMENT, PROCEDURES, and CONTINGENCIES. Each of these sections was examined in detail to determine the finite set of permutations in structure and content, as set forth by the CSDL grammar.
These possibilities formed the basis for test cases to be used in exercising the input translator. Due to its complexity, the PROCEDURES section received the most attention, but each part is discussed, in terms of the test results, below.

The IDENTIFICATION section, an example of which is contained in figure 4.1, consists of 3 character strings

```
IDENTIFICATION
  DESIGNER: "Thomas H. Carson"
  DATE: "10 April 1984"
  PROJECT: "Start Malfunction Controller"
```

Figure 4.1 IDENTIFICATION Section of the Input.

which make up a portion of the documentation for the system. The strings are not parsed by the input translator, just simply read and ignored. This section is optional, as are they all, and the parser performs no other action on it.

The DESIGN CRITERIA, an example of which is contained in figure 4.2, allows the user to specify the metric and number of monitors and volumes to be employed in the mapping process, the next module in the CSDE system. The metric is one of 3 choices, all character strings, while the monitors

```
DESIGN CRITERIA
  METRIC FIRST;
  VOLUMES 1;
  MONITORS 1;
```

Figure 4.2 DESIGN CRITERIA Section of the Input.
and volumes are integer values. The parser reads each of these and checks for correctness in terms of value. It then reformats the information and places it in a special portion of the primitive list. The possible alternatives were exercised and the corresponding correct output was generated, an example of which is contained in figure 4.3.

The ENVIRONMENT section, an example of which is contained in figure 4.4, contains the variable declarations for the input. It has 4 parts which are: procedure declarations, input signals, output signals, and duplex signals. While each of the above is optional, the controller will have to sense at least one input and emit one output to have some function. Each declaration can have its name, structure, precision, initial value, or the technology associated with it. The type of declaration decides which and how many of the attributes each variable will have. For the internal
program variables, in the ARITHMETIC section, the translator generates a system variable primitive and for each of the signals it produces the associated software primitive. Again, while the translator is parsing the structure of the input, the real work done is reformatting the names and their associated attributes into the appropriate primitive. No errors could be detected in exercising the program over this portion of the input. An example of the output generated by the above example is contained in figure 4.5.

The PROCEDURES section contains the functions and tasks which establish the purpose of the controller being realized. The differences between functions and tasks are: functions are allowed only one basic statement and return a value while tasks allow multiple statements and perform a job. The key to both is the basic statement which is one of several types seen in most programming languages. The alternatives are: if-then, while-do, for loop, assignment, data input, data output, perform task, and wait. The only ones that might not be familiar are the "perform task" and "wait". "Perform task" allows for nested procedures and the "wait" statement causes the program to suspend itself for a prescribed period of time. An example of a task is contained in figure 4.6.

\begin{verbatim}
4s.inputport (ENT,TTL:8)
5s.inputport (FIRE SENSE,TTL:1)
6s.inputport (OIL PRES,TTL:8)
7s.outputport(FIRE EXT,TTL:1)
8s.var (STAGFLG:1,0)
\end{verbatim}

Figure 4.5 Primitive List Form of the Input.
### Figure 4.6 PROCEDURES Section Input Example.

This section is where the parser really does its work. It must parse each statement in the procedure and generate an appropriate section of primitives, including temporaries, assembly-language-like software primitives and labels, which fulfill the intent of the statement. Each of the basic statements was exercised through the translator without any error detection, but because of the increased complexity of this section, 100% reliability cannot be confirmed. To do so would require a technique such as path testing. This requires that every logical path through a program be tested at least once. Another possibility is to construct test data which causes each branch in the program to be traversed [Ref. 15]. Algorithms for such testing exist, but the problems with each are the excessive time, CPU service, and output verification required. Therefore, complete path or branch testing was not attempted. However, the author's confidence in the correctness, after the testing that was conducted, is 100%. An output produced from the above input example is contained in figure 4.7.

The CONTINGENCY LIST section, an example of which is contained in figure 4.8, sets up the flow and timing for the controller by establishing how often each procedure should be executed. There are 4 types of statements allowed in this section: when-do, at time, simple do, and the every.
Figure 4.7 Primitive List Form of the PROCEDURES Section.

While the parsing action for these statements, basically just a reformating routine, is simple, problems developed in
determining what the format in the primitive list should be. For consistancy, each was treated the same with blanks left in the columns in the "simple do", "every", and "at time" statements where a contingency name appears in the "when-do" statement. Examination of the example output in figure 4.9 and its comparison with the input example above will clarify this point. Alternative entries, such as the word "each", to replace the blanks are under consideration. The decision, based on the requirements of the functional mapper, will not occur prior to the submission of this thesis. Therefore, it is not possible to establish complete
correctness in this section. The program does act according to its given requirements but the possibility exists for these to change and, at that time, the section will have to be reverified.

Only the primitive list has been discussed above as program output. The translator also generates a symbol table and a scratch pad. Since the symbol table is unformatted, testing established only that the required information was present. The scratch pad is not a functional member of the CSDE and, therefore, was not tested. The error-checking routines within the translator were tested in so far as they were used in creating correct input file for the testing described above. This was considered adequate due to the impending completion of a syntax-directed editor for composing the CSDL input for the translator.

The input translator was built bottom-up in modular form. This methodology and the use of the automatically generated driver for the parsing routines were significant reasons for the relatively error-free results obtained during the testing of the translator as a whole.

Figure 4.9 Primitive List Form of the CONTINGENCY LIST.
V. CONCLUSIONS AND RECOMMENDATIONS

A. PROGRAM MAINTENANCE

The use of the automatic parser generator in providing the control tables for the translator allows the maximum degree of flexibility for program maintenance. The resulting main program is modular and space efficient, with anticipated changes, such as new emissions from the semantic routines, easily included in the system.

The most obvious portion of the system which will undergo modification is the CSDL grammar. Matelan states there is no capability in CSDL for notational extensibility, nor should there be [Ref. 16]. He points out it is doubtful that the average user can design extensions that will cause less harm than good and the provision of a good macro capability and extendable function/task libraries are preferred. This author disagrees. The advantage of using the parser generator is that a new set of tables can be produced for a modified CSDL with virtually no disturbance to the translator. The changes to the grammar must be consistent with the rules governing LL(1) grammars, but such modifications as the inclusion of new tokens or reserved words is within the capability of advanced program language students. In addition, to keep CSDL static, using the function/task libraries to realize new primitive operations as they become technologically feasible, only results in a less efficient controller. It could lead, in the worst case, to the inability to utilize the latest hardware technology available for controller design. This was most certainly not the intent.
One particular change to CSDL needs immediate attention. Many controllers require an analog input or output signal. Matelan, in his work, makes the assumption that analog information must be converted to a digital signal before it reaches the interface [Ref. 16]. If we have the ability to modify CSDL, this is no longer a complex issue. One possibility is to add a new input/output signal type with the CSDE calling for an A/D-D/A converter when this type is mapped to a library.

B. RECOMMENDATIONS FOR CSDE

The flow of information between modules in CSDE, prior to the impending completion of the designer and translator modules, is consistent. Since the system was implemented in Fortran and resides on one machine, there is no problem. But with the completion of the translator, the subject of this thesis, and the designer module having the same time schedule, the problem of interfaces becomes significant. The translator is written in Pascal and, while the output is a simple text file, the information flow will not be as it should. In addition, the designer module is being written in the "C" programming language which will generate, possibly, further interface problems. It is therefore recommended that the CSDE system be incorporated, as quickly as possible, into a data base management system such as Oracle. This was previously discussed in Chapter 3 of this thesis and the design of the data base was the subject of earlier research [Ref. 5]. This research pointed out that this redesign of CSDE should help streamline the operation and eliminate any complex programming schemes that were built out of necessity in earlier work. This concept will eliminate, completely, the interface problems, as the information will be input to and accessible from the database as
it passes from one module to the next. An overall improvement in system documentation, efficiency, and usability should result while, at the same time, allowing each module to maintain its individuality and function.

C. SUMMARY

In completing the design and implementation of the input translator, the objective of this thesis has been accomplished. The translator has been tested and fulfills the function required of the module in the Computer System Design Environment [Ref. 2]. The additional features added to the translator include an option to monitor the execution sequence in a variety of tracing modes and extensive error checking. While not considered integral parts of the program, these features were useful in its design and testing. As the CSDE evolves, these features might become superfluous, at which time they could be eliminated. But the information provided by the features should be considered prior to that decision.

The design methodology employed in the production of the translator was not innovative, but it allowed the program to be simple and straightforward. The use of the automatic parser generator was a time saver and proved to be a tool which will allow for the ease of future program maintenance as no other could. In one sense, we could say, while generating a module for a computer aided design system, a computer aided programming tool was central in the development.

The translator will reside in the CSDE on the VAX 11/780 VMS operating system. It is a Pascal program with the source and object code available under the filename "CSDL". The user instructions have been fully covered in the previous sections of this thesis.
APPENDIX A
TRANSLATOR INPUT EXAMPLE

IDENTIFICATION

DESIGNER: "Alan Ross"
DATE: "12-23-83"
PROJECT: "Dial Process Control Application"

DESIGN CRITERIA

METRIC FIRST;
VOLUMES 8;
MONITORS 8;

ENVIRONMENT

INPUT: CONSI,8,TTL; CONST,8,TTL; FLGA,1,TTL; P(NA,8,TTL;
FLGB,1,TTL; PINB,8,TTL; END INPUT;

OUTPUT: VA,8,TTL; VB,8,TTL; END OUTPUT;

ARITHMETIC: KCA,8; KCB,8; CNT_B,9; ITIA,8; ITI3,8; AINT,8;
TDA,8; TDB,3; BINT,8; VSA,8; VSB,8; BDII|F,8;
PSA,8; PSB,8; CONPTT,8; EA,8; EB,9; KPIA,8;
EA1,8; EA2,8; EB1,8; EB2,8; KPIB,8;
END ARITHMETIC;

PROCEDURES

FUNCTION DATA_A:
  BINARY,1;
  SENSE (FL3A);
  IF FLGA = 1 THEN DATA_A := 1; END IF;
END DATA_A;

FUNCTION DATA_B:
  BINARY,1;
SENSE (FL3B):
IF FLGB = 1 THEN DATA_B := 1; END IF;
END DATA_B;

FUNCTION BCNT:
BINARY, 1;
IF CNT_B >= 4 THEN BCNT := 1; END IF;
END BCNT;

TASK AFIX:
ARITHMETIC: ADIFF, 8; END ARITHMETIC;
SENSE (PINA);
EA := PINA*KCA - PSA;
ADIFF := (3*EA - 4*EA1 + EA2)*5;
AINT := AINT + EA/KC;
VA := VSA + KCA*(EA + ITIA*AINT + TDA*ADIFF);
ISSUE (VA);
DATA_A := 0;
EA2 := EA1;
EA1 := EA;
END AFIX;

TASK B_CALC:
SENSE (PINB);
EB := PINB*KCB - PSB;
BDIFF := (3*EB - 4*EB1 + EB2)*10;
BINT := BINT + EB/KCB;
CNTB := CNTB + 1;
DATA_B := 0;
END B_CALC;

TASK BFIX;
CNTB := 0;
VB := VSB + KCB*(EB + ITIB*BINT + TDB*BDIFF);
ISSUE (VB);
END BFIX;
FUNCTION CONFLG:
    BINARY, 1;
    SENSE (CONFIN);
    IF CONFIN > 0 THEN CONFLG := 0; END IF;
END CONFLG;

TASK CHGCON;
    SENSE (CONST);
    IF CONPTT = 1 THEN KCA := CONST; END IF;
    IF CONPTT = 2 THEN ITIA := 1/CONST; END IF;
    IF CONPTT = 3 THEN TDA := CONST; END IF;
    IF CONPTT = 4 THEN VSA := CONST; END IF;
    IF CONPTT = 5 THEN PSA := CONST; END IF;
    IF CONPTT = 6 THEN AINT := CONST; END IF;
    IF CONPTT = 7 THEN KCB := CONST; END IF;
    IF CONPTT = 8 THEN ITIB := 1/CONST; END IF;
    IF CONPTT = 9 THEN TDB := CONST; END IF;
    IF CONPTT = 10 THEN VSB := CONST; END IF;
    IF CONPTT = 11 THEN PSB := CONST; END IF;
    IF CONPTT = 12 THEN BINT := CONST; END IF;
END CHGCON;

CONTINGENCY LIST

    WHEN DATA_A :100MS DO AFIX;
    WHEN DATA_B :50MS DO B_CALC;
    WHEN BCNT :100MS DO BFIX;
    WHEN CONFLG DO CHGCON;
## APPENDIX B

**FORMAL SYNTAX OF CSDL**

<table>
<thead>
<tr>
<th>TERMINALS</th>
<th>NONTERMINALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (</td>
<td>84. &lt;AOP&gt;</td>
</tr>
<tr>
<td>2. )</td>
<td>85. &lt;ARITHMETIC BODY&gt;</td>
</tr>
<tr>
<td>3. *</td>
<td>86. &lt;ARITHMETIC DEC&gt;</td>
</tr>
<tr>
<td>4. **</td>
<td>87. &lt;ARITHMETIC SPEC&gt;</td>
</tr>
<tr>
<td>5. <em>ID</em></td>
<td>88. &lt;ASSIGNMENT STMT&gt;</td>
</tr>
<tr>
<td>6. <em>NUMBER</em></td>
<td>89. &lt;AT TIME&gt;</td>
</tr>
<tr>
<td>7. <em>STRING</em></td>
<td>90. &lt;B1&gt;</td>
</tr>
<tr>
<td>8. +</td>
<td>91. &lt;B2&gt;</td>
</tr>
<tr>
<td>9. -</td>
<td>92. &lt;BASIC STMT&gt;</td>
</tr>
<tr>
<td>10. -</td>
<td>93. &lt;BINARY BODY&gt;</td>
</tr>
<tr>
<td>11. -</td>
<td>94. &lt;BINARY DEC&gt;</td>
</tr>
<tr>
<td>12. /</td>
<td>95. &lt;BINARY PRECISION&gt;</td>
</tr>
<tr>
<td>13. /=</td>
<td>96. &lt;BINARY SPEC&gt;</td>
</tr>
<tr>
<td>14. :</td>
<td>97. &lt;CHARACTER REP LIST&gt;</td>
</tr>
<tr>
<td>15. :=</td>
<td>98. &lt;CHARACTER REP&gt;</td>
</tr>
<tr>
<td>16. ;</td>
<td>99. &lt;CODE DEC LIST&gt;</td>
</tr>
<tr>
<td>17. &lt;</td>
<td>100. &lt;CODE DEC&gt;</td>
</tr>
<tr>
<td>18. &lt;=</td>
<td>101. &lt;CODE ID&gt;</td>
</tr>
<tr>
<td>19. =</td>
<td>102. &lt;CODE SPEC&gt;</td>
</tr>
<tr>
<td>20. ==</td>
<td>103. &lt;CODE VAR SPEC&gt;</td>
</tr>
<tr>
<td>21. =&gt;</td>
<td>104. &lt;CONTINGENCY DEF&gt;</td>
</tr>
<tr>
<td>22. &gt;</td>
<td>105. &lt;CONTINGENCY LIST&gt;</td>
</tr>
<tr>
<td>23. &gt;=</td>
<td>106. &lt;CONTROL SYSTEM DESIGN&gt;</td>
</tr>
<tr>
<td>24. AND</td>
<td>107. &lt;DATA INPUT&gt;</td>
</tr>
<tr>
<td>25. ARITHMETIC</td>
<td>108. &lt;DATA OUTPUT&gt;</td>
</tr>
</tbody>
</table>
26. ASCII6
27. ASCII7
28. AT
29. BCD
30. BINARY
31. CODE
32. CONTINGENCY
33. COST
34. CRITERIA
35. DATE
36. DESIGN
37. DESIGNER
38. DO
39. DUPLEX
40. EBCDIC
41. ECL
42. END
43. ENVIRONMENT
44. EVERY
45. FIRST
46. FOR
47. FROM
48. FUNCTION
49. H
50. IDENTIFICATION
51. IF
52. ITL
53. IN
54. INPUT
55. ISSUE
56. LIST
57. M
58. METRIC
59. MONITORS
60. MS
62. <DEC GP>
63. <DEC>
64. <DECIMAL PRECISION>
65. <DESIGN CRITERIA>
66. <DUPLEX SPEC>
67. <ENVIRONMENT SECTION>
68. <EPISODE TIMING>
69. <EVERY>
70. <EXPR LIST>
71. <EXPRESSION>
72. <EXP_2>
73. <EXP_3>
74. <EXP_4>
75. <FACTOR>
76. <FOR HEAD>
77. <FOR LOOP>
78. <FORMAL PARMLIST>
79. <FUNCTION>
80. <FUNCTION_HE1D>
81. <ID LIST>
82. <ID SECTION>
83. <IF HEAD>
84. <IF THEN>
85. <INITIAL VALUE>
86. <INPUT SPEC>
87. <Labeled stmt>
88. <LEFT PART LIST>
89. <LIST BODY>
90. <MAX LOOP COUNT>
91. <METRIC>
92. <MOP>
93. <NAME>
94. <NUM>
95. <NUMBER LIST>
96. <OUTPUT SPEC>
61. NOT
62. NS
63. OR
64. OUTPUT
65. POWER
66. PROCEDURES
67. PROJECT
68. S
69. SENSE
70. TASK
71. TERM
72. THEN
73. TO
74. TTL
75. UNTIL
76. US
77. VARIABLES
78. VOLUMES
79. WAIT
80. WHEN
81. WHILE
82. {
83. }
84. <PERFORM TASK>
85. <PERIOD>
86. <PI>
87. <PRIMARY>
88. <PROC DEC GP>
89. <PROC DEC>
90. <PROC GP>
91. <PROC SECTION>
92. <PROC>
93. <QUALIFICATION>
94. <RANK>
95. <RELATION>
96. <RELATIONAL GP>
97. <ROB>
98. <SIMPLE DO>
99. <SIMPLE EXP>
100. <STM GP>
101. <STM>
102. <STRUCTURE>
103. <SYSTEM GOAL SYMBOL>
104. <TASK LIST>
105. <TASK>
106. <TASK_HEAD>
107. <TECHNOLOGY>
108. <TERM>
109. <TIME MEASURE>
110. <TIME>
111. <TIMED BLOCK>
112. <TIMED BLOCK_HEAD>
113. <TRANSMISSION BODY>
114. <TRANSMISSION DEC>
115. <WAIT UNTIL>
116. <WAIT>
117. <WAIT_HEAD>
118. <WHEN DO>
THE PRODUCTIONS

1. <SYSTEM GOAL SYMBOL> ::= END <CONTROL SYSTEM DESIGN> END

2. <AOP> ::= / -

4. <MOP> ::= *

5. //

6. <RELATIONAL OP> ::= <

7. / <=

8. / =

9. / >

10. / >=

11. / /=

12. <PRIMARY> ::= *NUMBER*

13. / *STRING*

14. / <NAME>

15. / ( <EXPRESSION> )

16. <FACTOR> ::= <PRIMARY>

17. / <FACTOR> ** <PRIMARY>

18. <TERM> ::= <FACTOR>

19. / <TERM> <MOP> <FACTOR>

20. <SIMPLE EXP> ::= <TERM>

21. / <AOP> <TERM>

22. / NOT TERM

23. / <SIMPLE EXP> <AOP> <TERM>

24. <RELATION> ::= <SIMPLE EXP>

25. / <SIMPLE EXP> <RELATIONAL OP> <SIMPLE EXP>
26. <EXP_4> ::= <RELATION>
27. / <EXP_4> AND <RELATION>
28. <EXP_3> ::= <EXP_4>
29. / <EXP_3> OR <EXP_4>
30. <EXP_2> ::= <EXP_3>
31. / <EXP_2> => <EXP_3>
32. <EXPRESSION> ::= <EXP_2>
33. / <EXPRESSION> == <EXP_2>
34. <EXPR LIST> ::= <EXPRESSION>
35. / <EXPR LIST>, <EXPRESSION>
36. <IF THEN> ::= <IF HEAD> THEN <STMT GP> END IF
37. <IF HEAD> ::= IF <EXPRESSION>
38. <WHILE DO> ::= <WHILE HEAD> DO <STMT GP> END WHILE
39. <WHILE HEAD> ::= <WHILE> <EXPRESSION> : <MAX LOOP COUNT>
40. <WHILE> ::= WHILE
41. <FOR LOOP> ::= <FOR HEAD> DO <STMT GP> END FOR
42. <FOR HEAD> ::= FOR *ID* FROM <EXPRESSION> TO
        <EXPRESSION> : <MAX LOOP COUNT>
43. <PERFORM TASK> ::= *ID*
44. / *ID* ( <EXPR LIST> : <ID LIST> )
45. <MAX LOOP COUNT> ::= *NUMBER*
46. <LEAVE PART LIST> ::= <NAME> :=
47. / <LEAVE PART LIST> <NAME> :=
48. <ASSIGNMENT STMT> ::= <LEAVE PART LIST> <EXPRESSION>
49. <DATA INPJT> ::= SENSE (<NAME>)
51. \(<DATA\ OUTPUT> ::= ISSUE ( <NAME> )\)

52. \(<TIME\ MEASURE> ::= H\)
53. \(/ M\)
54. \(/ S\)
55. \(/ NS\)
56. \(/ US\)
57. \(/ NS\)

58. \(<PERIOD> ::= \text{NUMBER} \ <TIME\ MEASURE>\)

59. \(<TIME> ::= <PERIOD>\)
60. \(/ <TIME> <PERIOD>\)

61. \(<TIMED\ BLOCK> ::= <TIMED\ BLOCK\ HEAD> \ DO <STM GP> \ END IN\)

62. \(<TIMED\ BLOCK\ HEAD> ::= \text{IN} <PERIOD>\)

63. \(<WAIT> ::= \text{WAIT} <PERIOD>\)
64. \(/ \text{WAIT} <EXPRESSION> : <PERIOD>\)

65. \(<WAIT\ UNTIL> ::= <WAIT\ HEAD> <EXPRESSION> : <PERIOD>\)

66. \(<WAIT\ HEAD> ::= \text{WAIT} \ UNTIL\)

67. \(<BASIC\ STMT> ::= <IF\ THEN>\)
68. \(/ \text{WHILE\ DO}\)
69. \(/ \text{FOR\ LOOP}\)
70. \(/ \text{PERFORM\ TASK}\)
71. \(/ \text{ASSIGNMENT\ STMT}\)
72. \(/ \text{DATA\ INPUT}\)
73. \(/ \text{DATA\ OUTPUT}\)
74. \(/ \text{TIMED\ BLOCK}\)
75. \(/ \text{WAIT}\)
76. \(/ \text{WAIT\ UNTIL}\)

77. \(<LABELED\ STMT> ::= \text{*ID*} : <BASIC\ STMT>\)

78. \(<STM> ::= <BASIC\ STMT>\)
79. / <LABELED STMT>

80. <STMT GP> ::= <STMT> ;
81. / <STMT GP> <STMT> ;

82. <PROC DEC> ::= <BINARY SPEC>
83. / <ARITHMETIC SPEC>
84. / <CODE SPEC>
85. / <CODE VAR SPEC>

86. <INPUT SPEC> ::= INPUT : <TRANSMISSION BODY> END INPUT

87. <OUTPUT SPEC> ::= OUTPUT : <TRANSMISSION BODY>
                           END OUTPUT

88. <DEC> ::= <PROC DEC>
89. / <INPUT SPEC>
90. / <OUTPUT SPEC>
91. / <DUPLEX SPEC>

92. <DEC GP> ::= <DEC> ;
93. / <DEC GP> <DEC> ;

94. <DUPLEX SPEC> ::= DUPLEX <TRANSMISSION BODY> END DUPLEX

95. <BINARY SPEC> ::= BINARY : <BINARY BODY> END BINARY

96. <ARITHMETIC SPEC> ::= ARITHMETIC : <ARITHMETIC BODY>
                                               END ARITHMETIC

97. <TRANSMISSION BODY> ::= <TRANSMISSION DEC> ;
98. / <TRANSMISSION BODY> <TRANSMISSION DEC> ;

99. <TRANSMISSION DEC> ::= *ID*, <BINARY PRECISION>,
                           <TECHNOLOGY>

100. <BINARY BODY> ::= <BINARY DEC> ;
101. / <BINARY BODY> <BINARY DEC> ;

102. <BINARY DEC> ::= *ID* <STRUCTURE>, <BINARY PRECISION>
                           <INITIAL VALUE>

50
103. `<ARITHMETIC BODY>` ::= `<ARITHMETIC DEC>` ;
104.  
  / `<ARITHMETIC BODY>` `<ARITHMETIC DEC>` ;
105. `<ARITHMETIC DEC>` ::= `*ID*` `<STRUCTURE>`,
  `<DECIMAL PRECISION> <INITIAL VALUE>`
106. `<STRUCTURE>` ::= 
107.  
  / ( `<NUMBER LIST> ` )
108. `<NUMBER LIST>` ::= `*NUMBER*`
109.  
  / `<NUMBER LIST>`, `*NUMBER*`
110. `<BINARY PRECISION>` ::= `*NUMBER*`
111. `<DECIMAL PRECISION>` ::= `*NUMBER*`
112. `<INITIAL VALUE>` ::= 
113.  
  / , `*NUMBER*`
114. `<TECHNOLOGY>` ::= `TTL`
115.  
  / `ECL`
116.  
  / `ITL`
117. `<CODE VAR SPEC>` ::= `CODE VARIABLES` : `<CODE DEC LIST>`
  END CODE VARIABLES
118. `<CODE DEC LIST>` ::= `<CODE DEC>` ;
119.  
  / `<CODE DEC LIST>` `<CODE DEC>` ;
120. `<CODE DEC>` ::= `*ID*` : `<CODE ID>`
121. `<CODE SPEC>` ::= `CODE : `*ID*`, `<BINARY PRECISION>` ;
  `<CHARACTER REP LIST>` END CODE
122. `<CHARACTER REP LIST>` ::= `<CHARACTER REP>` ;
123.  
  / `<CHARACTER REP LIST>` `<CHARACTER REP>` ;
124. `<CHARACTER REP>` ::= `*ID*` : `*NUMBER*`
125. `<CODE ID>` ::= `*ID*`
126.  
  / `ASCII6`
127.  / ASCII7
128.  / EBCIDIC
129.  / BCD
130.  <ID LIST> ::= 
131.  / *ID* 
132.  / <ID LIST>, *ID* 
133.  <NAME> ::= *ID* 
134.  / *ID* ( <EXPR LIST> ) 
135.  / *ID* { *NUMBER* : *NUMBER* } 
136.  <FORMAL PARAM LIST> ::= 
137.  / ( <ID LIST> : <ID LIST> ) 
138.  <PROC> ::= <TASK> 
139.  / <FUNCTION> 
140.  <TASK> ::= <TASK HEAD> ; <ZOPT PROC DEC GP> <STMT GP> 
    END *ID* 
141.  <ZOPT PROC DEC GP> ::= 
142.  / <PROC DEC GP> 
143.  <PROC DEC GP> ::= <PROC DEC> ; 
144.  / <PROC DEC GP> <PROC DEC> ; 
145.  <TASK HEAD> ::= TASK *ID* <FORMAL PARAM LIST> 
146.  <FUNCTION> ::= <FUNCTION HEAD> ; <ZOPT PROC DEC GP> 
    <STMT> END *ID* 
147.  <FUNCTION HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST> : 
    BINARY, <BINARY PRECISION> <INITIAL VALUE> 
148.  / FUNCTION *ID* <FORMAL PARAM LIST> : 
    ARITHMETIC, <DECIMAL PRECISION> <INITIAL VALUE> 
149.  <PROC GP> ::= <PROC> ; 
150.  / <PROC GP> <PROC> ; 
151.  <PROC SECTION> ::= 

52
152. / PROCEDURES <PROC GP>
153. <ROE> ::= <PERIOD>
154. <B1> ::= <PERIOD>
155. <B2> ::= <PERIOD>
156. <BANK> ::= <NU>
157. / <NU> . <PI>
158. <NU> ::= *NUMBER*
159. <PI> ::= *NUMBER*
160. <QUALIFICATION> ::= 
161. / IF <EXPRESSION>
162. <EPISODE TIMING> ::= 
163. / : <ROE>
164. / : <ROE>, <B1>
165. / : <ROE>, <B1>, <B2>
166. / : <ROE>, <B1>, <B2>, <BANK>
167. <WHEN DO> ::= <QUALIFICATION> WHEN <NAME> <EPISODE TIMING> DO <TASK LIST>
168. <SIMPLE DO> ::= <QUALIFICATION> DO <TASK LIST> <RANK>
169. <EVERY> ::= <QUALIFICATION> EVERY <ROE> DO <TASK LIST>
170. <AT TIME> ::= <QUALIFICATION> AT <TIME> DO <TASK LIST>
171. <TASK LIST> ::= <NAME>
172. / <TASK LIST> THEN <NAME>
173. <CONTINGENCY DEF> ::= <WHEN DO>
174. / <SIMPLE DO>
175. / <EVERY>
176. / <AT TIME>
177. <LIST BODY> ::= <CONTINGENCY DEF> ;

53
178. / <LIST BODY> <CONTINGENCY D3F> ;
179. <CONTINGENCY LIST> ::= 
180. / CONTINGENCY LIST <LIST BODY>
181. <DESIGN CRITERIA> ::= 
182. / DESIGN CRITERIA METRIC <METRIC> ;
183. <METRIC> ::= FIRST
184. / COST
185. / POWER
186. <ENVIRONMENT SECTION> ::= 
187. / ENVIRONMENT <DEC GP>
188. <ID SECTION> ::= 
189. / IDENTIFICATION DESIGNER : *STRING*
190. <CONTROL SYSTEM DESIGN> ::= <ID SECTION>
191. <DESIGN CRITERIA> <ENVIRONMENT SECTION> <PROC SECTION>
192. <CONTINGENCY LIST>
APPENDIX C

PRIMITIVE LIST

10. generated for: SYSTEM

2s. MAIN ()

3d. FIRST :8,8,0,0,0,0,0.

4s. inputport (CONSA,TTL:8)

5s. inputport (CONST,TTL:8)

6s. inputport (FLGA,TTL:1)

7s. inputport (PINA,TTL:8)

Bs. inputport (FLGB,TTL:1)

9s. inputport (PINB,TTL:8)

10s. outputport (VA,TTL:8)

11s. outputport (VB,TTL:8)

12s. var (KCA:8,0)

13s. var (KCB:8,0)

14s. var (CNTA:8,0)

15s. var (ITIA:8,0)

16s. var (ITIB:8,0)

17s. var (AINT:8,0)

18s. var (ITDA:8,0)

19s. var (ITDB:8,0)

20s. var (BINT:8,0)

21s. var (VSA:8,0)

22s. var (VSB:8,0)

23s. var (BDIFF:8,0)

24s. var (PSA:8,0)

25s. var (PSB:8,0)

26s. var (COMPT:8,0)

26s. var (EA:8,0)

28s. var (EB:8,0)

29s. var (KPIA:8,0)

30s. var (EA1:8,0)

31s. var (EA2:8,0)

32s. var (EB1:8,0)

33s. var (EB2:8,0)

34s. var (KPIB:8,0)

35s. generated for: DATAA

36s. proc (DATAA:)

37s. sensecond (FLGA:1)

38s. eq (@T01,FLGA,EC01:8,1,8)

39s. jmpf (@T01,EC01:8)

40s. assign (DATAA,EC01:1,8)

41s. loc (@O1:)

42s. exitproc (DATAA:)

43s. generated for: DATABASE

44s. proc (DATABASE:)

45s. sensecond (FLGB:1)

46s. eq (@T01,FLGB,EC01:8,1,8)

47s. jmpf (@T01,EC02:8)

48s. assign (DATABASE,EC01:1,8)

49s. loc (@O2:)

50s. exitproc (DATABASE:)

51s. generated for: BCNT

52s. proc (BCNT:)

53s. ge (@T01,CNTB,EC02:8,8,8)
P 107s.assign (CMTB,e05:0,8)
P 108s.mult (e01,ITIB,BINT:8,8,8)
P 109s.add (e01,EB,e01:0,8,8)
P 110s.mult (e02,TOB,BDIFF:8,8,8)
P 111s.add (e01,e01,e02:8,8,8)
P 112s.mult (e01,KCB,e01:0,8,8)
P 113s.add (e01,VS,e01:0,8,8)
P 114s.assign (VB,e01:0,8,8)
P 115s.issueevent (VB:8)
P 116s.exitproc (BFX:)
P 117s.generated for:CONFLG

***************
P 118s.proc (CONFLG:)
P 119s.sensecond (COUNT:8)
P 120s.gt (e01,CONSIN,e05:0,8,8)
P 121s.jmpf (e01,e04:8)
P 122s.assign (CONFLG,e05:1,8)
P 123s.loc (e04:)
P 124s.exitproc (CONFLG:)
P 125s.generated for:CHGCN

***************
P 126s.proc (CHGCN:)
P 127s.sensecond (COUNT:8)
P 128s.jmpf (e01,COMPT,e01:0,8,8)
P 129s.assign (KCA,COUNT:8,8)
P 130s.loc (e05:)
P 131s.eq (e01,COMPT,e07:0,8,8)
P 132s.jmpf (e01,e06:8)
P 133s.divide (e01,CO1,CONST:8,8,8)
P 134s.assign (ITIA,e01:0,8,8)
P 135s.loc (e06:)
P 136s.eq (e01,COMPT,e03:0,8,8)
P 137s.jmpf (e01,e07:8)
P 138s.assign (TD,e01:0,8,8)
P 139s.loc (e07:)
P 140s.eq (e01,COMPT,e02:0,8,8)
P 141s.jmpf (e01,e08:8)
P 142s.assign (VSA,COUNT:8,8)
P 143s.loc (e08:)
P 144s.eq (e01,COMPT,e04:0,8,8)
P 145s.jmpf (e01,e09:8)
P 146s.assign (PSA,COUNT:8,8)
P 147s.loc (e09:)
P 148s.eq (e01,COMPT,e08:0,8,8)
P 149s.jmpf (e01,e10:8)
P 150s.assign (AINT,COUNT:8,8)
P 151s.loc (e10:)
P 152s.eq (e01,COMPT,e09:0,8,8)
P 153s.jmpf (e01,e11:8)
P 154s.assign (KCB,COUNT:8,8)
P 155s.loc (e11:)
P 156s.eq (e01,COMPT,e10:0,8,8)
P 157s.jmpf (e01,e12:8)
P 158s.divide (e01,CO1,CONST:8,8,8)
PROGRAM CSDE (DAT, INPUT, OUTPUT, PRIMFILE, TRANSLATE, SYMFILE);
(*This program uses the output from the CSDE syntax-directed*)
(*editor. The output is in standard CSDE form and is *)
(*translated into a primitive list, contained in PRIMFILE, *)
(*and the results of the parse in the OUTPUT file. *)

LABEL 99;

CONST
(* Constants generated by the translator as a result of the *)
(* CSDE syntax being fed through the automatic parser generator.*)

FRESIZE = 397;
FRESIZE = 205;
LSETSIZE = 83;

LSSIZE = 573;
PRODSPACE = 205;
FTRANSIZE = 397;
TRANSIZE = 1161;
ENTSIZE = 396;
LHSIZE = 190;
ENL = 190;
FSTATE = 7;

VOCSIZE = 182;
FIRSTRESMD = 24;
LASTRESMD = 81;
NUMTERMINALS = 83;

MAXENT = 182;
MAXFRES = 206;
MAXFTRANS = 1162;
MAXTRANS = 396;
MAXNSET = 62;
MAXPROD = 190;
MAXLS = 81;
MAXLSET = 574;
MAXLEN = 11;
MAXLHS = 182;

(*PROGRAM CONSTANTS*)

ENDOK = 42;
IDOK = 5;
NUMOK = 6;
ARROWOK = 21;
EQUIVALENCER = 20;
FSETOK = 4;
BECOMESTOK = 15;
NOTEQOK = 13;
LESSEQOK = 18;
GTSEQOK = 23;
STRINGOK = 7;

MAXSTK = 40;
TABSIZE = 84;
LINELENGTH = 120;
PAGESIZE = 53;
LINERRARRAYSIZE = 10;
(* SIZE OF STACK USED IN PARSER *)
(* SIZE OF SYMBOL TABLE *)
(* MAX LENGTH OF INPUT LINES *)
(* NUMBER OF LINES PRINTED PER PAGE *)
(* MAX NO OF ERRORS FLAGGED PER INPUT LINE +2 *)
MAXSTRINGS = 100; (* MAX LENGTH OF STRINGHEAD ARRAY *)
MAXSTORE = 1000; (* MAX LENGTH OF STRINGSTORE ARRAY *)
TEMPLISTMAX = 25; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)
MAXEMPS = 20; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)
TPoolSize = 10; (* CONSTANT USED IN SEMANTIC *)
MAXEVALSTACK = 20; (* DEPTH OF ARITH EVAL STACK *)
MAXOPS = 10; (* MAX NUMBER OF OPS HELD IN STORAGE FOR PRINTING *)
MAXS = 10; (* MAX NUMBER OF SELS HELD IN STORAGE FOR PRINTING *)
CSSMAX = 20; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)

TYPE

(*SYMBOL TABLE TYPES*)

ALFA = VARYING[MAXSTORE] OF CHAR;
KINDS = (UNDEFINED, RESWD, BINARY, ARITHMETIC, WORD, CHAREP,
TRANSDEC, TASK, FNCTN);
EXPTYPES = (INT, REAL, BOOL, STNG, ERRORS);
SYMTR = SYMENTRY;
SYMENTRY = RECORD
SYNAME : ALFA;
LINK : SYMTR;
CASE KIND : KINDS OF
(*1*) RESWD : (KEY1 : INTEGER);
(*2*) BINARY : (PRECISION2, IVAL2 : INTEGER);
(*3*) ARITHMETIC : (PRECISION3, IVAL3 : INTEGER);
(*4*) WORD : (CODID4 : SYMTR, STRINGP4 : INTEGER);
(*5*) CHAREP : (CODIDS : SYMTR, IVAL5 : INTEGER);
(*6*) TRANSDEC : (TYPE6, TECHNOLOGY6 : SYMTR;
PRECISION6 : INTEGER);
(*7*) TASK : (PARAMLIST7 : SYMTR);
(*8*) FNCTN : (PARAMLIST8, TYPE8 : SYMTR;
PRECISION8, IVAL8 : INTEGER);
END;

(*SEMANTIC TYPES*)

SWITCHES = (TRACEPARSE, TRACETOK, PRINTTABLE);
DESCRIPTOR = RECORD (* DESCRIBES THE CURRENT TOKEN *)
SYNAME, TYPNAME : ALFA;
LINEPOS,
INTVAL : INTEGER; (* INTEGER VAL OF CURRENT NUMBER *)
REALVAL : REAL; (* REAL VAL OF CURRENT NUMBER *)
CHARVAL : CHAR; (* VAL OF CURRENT CHAR LITERAL *)
SYMLOC : SYMTR; (* SYMBOL LOC IN SYMBOL TABLE *)
END;
TAGTYPE = (QNUM, QNAME);
VAR
PRIMFILE : TEXT;
DAT : TEXT;
TRANSLATE : TEXT;
SYMFILE : TEXT;

FRED : PACKED ARRAY[1..FREDSIZE] OF 1..MAXFRED;
HSET : PACKED ARRAY[1..HSETSIZE] OF 1..MAXHSET;
LSET : PACKED ARRAY[1..LSETSIZE] OF 1..MAXLSET;
LS : PACKED ARRAY[1..LSSIZE] OF 1..MAXLS;
PROD : PACKED ARRAY[1..PRODSIZE] OF 1..MAXPROD;

FTRN : PACKED ARRAY[1..FTRNSIZE] OF 1..MAXFTRN;
TRAN : PACKED ARRAY[1..TRANSIZE] OF 1..MAXTRAN;
ENT : PACKED ARRAY[1..ENTSIZE] OF 1..MAXENT;

LEN : PACKED ARRAY[1..LENSIZE] OF 0..MAXLEN;
LHS : PACKED ARRAY[1..LHSSIZE] OF 1..MAXLHS;

(* PROCEDURE PARSE *)
NEXTSYM, (* NEXT SYMBOL IN INPUT STREAM *)
NOWSTA, (* CURRENT STATE *)
REDUCTION, (* POSSIBLE REDUCTIONS *)
TRANSITION : INTEGER; (* POSSIBLE TRANSITIONS *)

(* STACK VARIABLES *)
STACK : ARRAY[1..MAXSTK] OF RECORD
STATE : INTEGER; (* STATE STACK *)
TOK : INTEGER; (* TOKEN STACK *)
DES : DESCRIPTOR; (* TOKEN DESCRIPTOR *)
EXPTYPE : EXPTYPES;
END;
SIPTR : INTEGER; (* TOP OF STACK POINTER *)

(* ERROR HANDLING VARIABLES *)
ERRLIST : SET OF 1..58; (* COMPILATION ERROR LIST *)
LINERRORS : ARRAY[1..LINERRARRAYSIZE] OF RECORD
ERRPOSITION, (* ERROR POSITION *)
ERRNUM, (* WHICH ERROR *)
STATE : INTEGER; (* PARSER STATE WHERE ERROR OCCURRED *)
END;
LINERRPTR : (* POINTER INTO LINERRORS *)
MAXLINES : INTEGER;
PROGRAMERRFLAG,
OVERFLOWLOGGED : BOOLEAN;

(* PROCEDURE NEXTSYM *)
SPS : ARRAY[CHAR] OF INTEGER;
CC, (* POINTER TO CURRENT CHAR ON LINE *)
LL, (* LENGTH OF CURRENT INPUT SYMBOL *)
SOURCELINECOUNT, (* INPUT LINE COUNTER *)
PAGENUMBER,
PAGELINECOUNT : INTEGER;
CH : CHAR; (* NEXT CHAR IN LINE *)
LINE : ARRAY[1..LINELENGTH] OF CHAR; (* INPUT LINE BUFFER *)
ZDATE, ZTIME : PACKED ARRAY[1..11] OF CHAR;
LASTTOK : BOOLEAN;
BUFFER : ALFA;

(* SYMBOL TABLE VARIABLES *)
SYMTABLE : ARRAY[1..TABSIZE] OF SYMPTR;
AMULT : REAL;
STRINGHEAD : ARRAY[1..MAXSTRINGS] OF INTEGER;
STRINGSTORE : PACKED ARRAY[1..MAXSTORE] OF CHAR;

(* SEMANTIC VARIABLES *)
SWITCH : ARRAY[SWITCHES] OF BOOLEAN;
TEMPLIST : ARRAY[1..TEMPLISTMAX] OF SYMPTR;
TLI : INTEGER;
LINECOUNT,
ALINECOUNT : INTEGER;
FIRSTPARAM : SYMPTR;
LABELCOUNT : INTEGER;

TEMPNAME : ARRAY[1..MAXNAME] OF RECORD

NAME : ALFA;
PRECISION : INTEGER;
INUSE,USED : BOOLEAN;
END;

EVALSTACK : ARRAY[1..MAXSTACK] OF RECORD

NAME : ALFA;
PRECISION : INTEGER;
END;

ESI : INTEGER;

CONSTANTSTORE : ARRAY[1..CSSMAX] OF RECORD

VAL : INTEGER;
0213 PRECISION: INTEGER;
0214 NAME: ALFA;
0215 END;
0216
0217 CSI: INTEGER;
0218
0219 OPSTORE: ARRAY[1..MAXOPS] OF ALFA;
0220 OPI: INTEGER;
0221
0222 SELSTORE: ARRAY[1..MAXSELS] OF INTEGER;
0223 SELI: INTEGER;
0224 NIL: INTEGER;
0225
0226 DESCRIPTION: DESCRIPTOR;
0227
0228 (* INITIALIZED VALUES GENERATED BY THE AUTOMATIC PARSER *)
0229
0230 VALUE
0231
0232 (*LANGUAGE TERMINALS*)
0233 STRINGSTORE := ('(' ', ')', '. ', ', ', 'I', 'D', 'O', 'N', 'U', 'M',
0235 '>', '<', '<', 'a', 'c', 'd', 'b', 'i', 'n', 'A',
0239 'P', 'U', 'X', 'I', 'D', 'E', 'C', 'L', 'E', 'N', 'D',
0243 'I', 'F', 'H', 'P', 'U', 'T', 'E',
0244 'L', 'I', 'M', 'I', 'T', 'I', 'C', 'M', 'O', 'N', 'I', 'T', 'O',
0251
0252 666 OF ' ');
0253
0254 (*IN STRINGSTORE THE POSITION OF THE BEGINNING OF EACH TERMINAL*)
0255 STRINGHEAD := (1, 2, 3, 4, 6, 10, 18, 26, 27, 28, 29, 30, 31,
0256 33, 34, 36, 37, 38, 40, 41, 43, 45, 46, 48, 51, 61, 67, 73, 75, 78,
0257 84, 88, 99, 103, 111, 115, 121, 129, 131, 137, 144, 147, 150, 161, 166, 171, 174,
0258 178, 186, 187, 201, 203, 218, 219, 222, 223, 229, 237, 239, 242, 244, 246,
0259 252, 257, 267, 274, 275, 280, 284, 288, 292, 294, 297, 302, 304, 313, 320, 324, 328)
| 0425 | 361,176,362,363,176,273,364,229,275,365,366,200,367,368,369,370,281, |
| 0426 | 371,281,372,175,176,177,176,179,180,181,182,373,184,185,186,187,188, |
| 0427 | 189,190,191,192,139,374,375,254,376,377,378,379,380,331,381,382,331, |
| 0428 | 383,176,273,384,254,385,219,386,175,176,177,175,179,180,181,182,387, |
| 0429 | 184,185,186,187,188,180,191,192,220,388,389,331,390,254,391,359, |
| 0430 | 392,229,393,394,395,330,332,396); |
| 0431 | |
| 0432 | |
| 0433 | |
| 0434 | |
| 0435 | NSET := (59,58,61,57,56,51,51,51,51,51,51,51, |
| 0436 | 61,51,51,51,51,51,51,51,51,51,51,51,51, |
| 0437 | 34,53,35,35,43,51,35,42,51,34,55,51,51,51,51,51, |
| 0439 | 4,10,10,5,5,8,5,4,4,4,5,1,51,51,51,51,51, |
| 0440 | 35,51,51,51,51,29,33,19,62,60,31,2,51,51,51,51, |
| 0441 | 51,51,51,51,51,51,51,51,51,51,51,51,51, |
| 0442 | 29,8,8,2,2,2,2,2,1,1,54,40,37,41,60,59, |
| 0443 | 28,51,51,51,51,51,36,36,36,36,36,36,2, |
| 0444 | 51,32,51,10,3,7,6,5,10,6,5,4,40,45,50,51,51, |
| 0445 | 35,47,51,51,9,29,51,51,51,51,51,51, |
| 0446 | 51,51,51,51,51,51,51,51,51,51,51,51, |
| 0447 | 60,60,41,60); |
| 0448 | |
| 0449 | |
| 0450 | |
| 0451 | |
| 0452 | |
| 0453 | |
| 0454 | PROD := (188,181,1,186,151,83,82,84,85,187,91,69,90,88, |
| 0455 | 179,184,183,185,92,139,152,138,190,106,106,97,93,136,136,141,149, |
| 0456 | 141,160,103,100,110,94,98,130,145,142,150,175,176,160,180,174,173, |
| 0457 | 108,96,104,95,101,118,86,87,131,143,44,133,41,71,78,72,73, |
| 0458 | 69,67,79,70,74,76,75,68,133,12,13,2,3,161,32,30,28, |
| 0459 | 18,11,16,20,24,26,10,17,107,11,12,12,12,12,12,12,12,12,12,12,12,12, |
| 0460 | 119,115,116,114,99,130,144,38,62,12,66,63,49,14,47,80, |
| 0461 | 34,22,21,11,6,7,8,9,10,4,5,178,59,171,153,162,189, |
| 0462 | 105,105,102,117,132,39,133,44,77,52,53,57,54,56,58,48, |
| 0463 | 146,81,140,15,33,31,29,27,17,23,25,19,60,156,156,160,113, |
| 0464 | 122,137,112,112,134,130,51,50,64,65,46,40,170,172,169,163,182, |
| 0465 | 124,121,123,148,147,36,42,37,61,39,159,157,167,45,135,164,154, |
| 0466 | 43,165,155,166); |
| 0467 | |
| 0468 | |
| 0469 | (* LOOK AHEAD SETS *) |
| 0470 | LS := (1,5,6,7,1,5,6,7,8,10,61,2,9,14,16,20,28,38,44,72, |
| 0471 | 73,80,2,3,8,9,10,12,13,14,16,17,18,19,20,21,22,23,24,28,38,44, |
| 0472 | 63,72,73,80,2,9,14,16,20,21,24,28,38,44,63,72,73,80,2,9,14,16, |
| 0473 | 20,21,28,38,44,63,72,73,80,2,9,14,16,20,21,28,38,44,72,73,80,2, |
| 0474 | 8,9,10,13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,73,80,2, |
| 0475 | 3,4,5,6,9,10,12,13,14,15,16,17,18,19,20,21,22,23,24,28,38,44, |
| 0476 | 63,72,73,80,2,3,4,8,9,10,12,13,14,16,17,18,19,20,21,22,23,24, |
| 0477 | 28,38,44,63,72,73,80,2,9,14,16,20,28,38,44,72,80,2,3,8,9,10, |
0533 114, 114, 128, 128, 106);
0534 
0535 (* PARSE INITIALIZATION *)
0536
0537 FUNCTION HASH (SYM : ALFA) : INTEGER;
0538 (* USED IN ENTER TO CREATE THE INDEX IN THE SYMBOL *)
0539 (* TABLE FOR EACH SYMBOL *)
0540 
0541 VAR KEY : INTEGER;
0542 BEGIN
0543 KEY := 17*ORD(SYM[1]) + 15*ORD(SYM[2]) + 13*ORD(SYM[3]) +
0544 3*ORD(SYM[5]) + ORD(SYM[7]);
0545 HASH := ROUND(KEY*AMULT)MOD TABSIZE + 1
0546 END; (*HASH*)
0547
0548
0549 FUNCTION ENTER (SYM : ALFA) : SYMPTR;
0550 (* ENTERS VALUES IN THE SYMBOL TABLE AND *)
0551 (* RETURNS THE POINTER TO THE SYMBOL *)
0552 
0553 VAR PTR : SYMPTR;
0554 HASHINDEX : INTEGER;
0555 BEGIN
0556 NEW(PTR);
0557 PTR. SYMNAME := SYM;
0558 PTR. KIND := UNDEFINED;
0559 HASHINDEX := HASH(SYM);
0560 PTR. LINK := SYMTABLE[HASHINDEX];
0561 SYMTABLE[HASHINDEX] := PTR;
0562 ENTER := PTR
0563 END; (*ENTER*)
0564
0565
0566 FUNCTION LOOKUP (SYM : ALFA) : SYMPTR;
0567 (* LOOKS UP SYMBOLS IN THE SYMBOL TABLE *)
0568 (* AND RETURNS THE POINTER TO THE SYMBOL DESIRED *)
0569 
0570 VAR PTR, SYMLOC : SYMPTR;
0571 HASHINDEX : INTEGER;
0572 BEGIN
0573 SYMLOC := NIL;
0574 HASHINDEX := HASH(SYM);
0575 PTR := SYMTABLE[HASHINDEX];
0576 WHILE PTR <> NIL DO
0577 IF SYM = PTR. SYMNAME THEN
0578 BEGIN
0579 SYMLOC := PTR;
0580 PTR := NIL
0581 END
PROCEDURE HEADER:
(* PUTS HEADER IN TRANSLATE FILE *)

BEGIN
PAGENUMBER := PAGENUMBER + 1;
WRITE (TRANSLATE, 'CSDL TRANSLATOR ', ':', '50', 'PAGE ', PAGENUMBER);
WRITE (TRANSLATE, 'NAVAL POSTGRADUATE SCHOOL', ':', '47');
WRITE (TRANSLATE, ZDATE : 10, ZTIME : 10);
WRITE (TRANSLATE);
PAGELINECOUNTH := 3
END: (**HEADER*)

FUNCTION CHARVAL (NUMBER : INTEGER) : ALFA;
(* USED TO CREATE LOCATION AND TEMPORARY PRIMITIVE VALUES *)

BEGIN
CASE NUMBER OF
0 : CHARVAL := '00';
1 : CHARVAL := '01';
2 : CHARVAL := '02';
3 : CHARVAL := '03';
4 : CHARVAL := '04';
5 : CHARVAL := '05';
6 : CHARVAL := '06';
7 : CHARVAL := '07';
8 : CHARVAL := '08';
9 : CHARVAL := '09';
10 : CHARVAL := '10';
11 : CHARVAL := '11';
12 : CHARVAL := '12';
13 : CHARVAL := '13';
14 : CHARVAL := '14';
15 : CHARVAL := '15';
16 : CHARVAL := '16';
17 : CHARVAL := '17';
18 : CHARVAL := '18';
19 : CHARVAL := '19';
20 : CHARVAL := '20';
END
END: (**CHARVAL*)

PROCEDURE INITIALIZE;
(* INITIALIZES ALL VARIABLES IN THE PROGRAM AND *)

(* CREATES THE STACKS FOR USE IN THE PARSER *)
0680  ERRLIST := [];  
0681  LINERPRTR := 0;  
0682  OVERFLOWLOGGED := FALSE;  
0683  MAXLINEERRORS := LINERRARRAYSIZE - 2;  
0684  NEXTSA := 1;  
0685  NEXSYM := ENDTOK;  
0686  stkptr := 1;  
0687  stack[i].STATE := 1;  
0688  stack[i].TOK := 0;  
0689  DESCRIPTION.SYNAME := ' ';  
0690  DESCRIPTION.INTVAL := 0;  
0691  DESCRIPTION.CHARVAL := '0';  
0692  DESCRIPTION.REALVAL := 0.0;  
0693  DESCRIPTION.SYMLOC := NIL;  
0694  BUFFER := ' ';  
0695  FOR K := 1 TO MAXTEMPS DO  
0696  SWITCH[K] := FALSE;  
0697  TLI := 0;  
0698  (* START SEMANTIC INITIALIZATION *)  
0699  FOR I := 1 TO MAXTEMPS DO  
0700  BEGIN  
0701  TEMPNAME[I].NAME := 'ot';  
0702  TEMPNAME[I].NAME := TEMPNAME[I].NAME + CHARVAL(I);  
0703  IF I IN [1..TPOOLBSIZE] THEN  
0704  TEMPNAME[I].PRECISION := 8  
0705  ELSE  
0706  TEMPNAME[I].PRECISION := 16;  
0707  TEMPNAME[I].INUSE := FALSE;  
0708  TEMPNAME[I].USED := FALSE  
0709  END;  
0710  ESL := 0;  
0711  FOR I := 1 TO CSSMAX DO  
0712  BEGIN  
0713  CONSTANTSTORE[I].NAME := 'sc';  
0714  CONSTANTSTORE[I].NAME := CONSTANTSTORE[I].NAME +  
0715  CONSTANTSTORE[I].NAME + CHARVAL(I);  
0716  CONSTANTSTORE[I].VAL := 0;  
0717  CONSTANTSTORE[I].PRECISION := 8;  
0718  END;  
0719  CSI := 0;  
0720  NLI := 0;  
0721  END;  
0722  (*PARSING ROUTINES*)  
0723  PROCEDURE PUTT (TITLE : ALFA);  
0724  (*PLACES TITLE IN PRIMITIVE FILE.*)  
0725  BEGIN  
0726  LINECOUNT := LINECOUNT + 1;  
0727  WRITE(PRIMFILE, ' ', LINECOUNT : 4, ' t.generated for:', ' ', TITLE : 10,  
0728  ' ', ' ', 10, ' ***************');  
0729  END;  
0730  (*PUTT*)
procedure puts (primname : alfa; opi, seli : integer);
(*loads primitive file*)

var i, j : integer;
begin
linecount := linecount + 1;
write (primfile, 'p', linecount : 4, ' ', primname : 10, ' ');
if opi > 0 then
begin
for i := 1 to opi - 1 do
begin
for j := 1 to 10 do
if opstore[i][j] <> ' ' then
write (primfile, opstore[i][j] : 1);
write (primfile, ' ');
end;
for j := 1 to 10 do
if opstore[opi][j] <> ' ' then
write (primfile, opstore[opi][j] : 1);
write (primfile, ' ');
end;
if seli > 0 then
begin
for i := 1 to seli - 1 do
write (primfile, selstore[i] : 1, ' ');
write (primfile, selstore[seli] : 1, ' ');
end;
else
write (primfile, ' ');
writeln (primfile);
end; (* puts *)

procedure putsym (primname : alfa; opi, seli : integer);
(* loads the symbol table *)

var i, j : integer;
begin
write (symfile, 's', primname : 10, ' ');
if opi > 0 then
begin
for i := 1 to opi - 1 do
begin
for j := 1 to 10 do
if opstore[i][j] <> ' ' then
write (symfile, opstore[i][j] : 1);
write (symfile, ' ');
end;
for j := 1 to 10 do
if opstore[opi][j] <> ' ' then
write (symfile, opstore[opi][j] : 1);

0706       WRITE(SYMFILE, '':');
0707       END;
0708       IF SELI > 0 THEN
0709         BEGIN
0800       FOR I := 1 TO SELI - 1 DO
0801         WRITE(SYMFILE, SELSTORE[I] : 1, ':');
0802         WRITE(SYMFILE, SELSTORE[SELI] : 1, ':');
0803       END
0804       ELSE
0805         WRITE(SYMFILE, ':');
0806         WRITELN(SYMFILE)
0807       END; (* PUTSYM *)
0808
0809       PROCEDURE GETSYM (VAR NEXTSYM : INTEGER; VAR DES : DESCRIPTOR); FORWARD;
0810       PROCEDURE ERROIR (ERRORNUM, INTENSITY, PTOFFSET : INTEGER); FORWARD;
0811
0812       PROCEDURE PUTA (CONTINAME, TASKNAME : ALFA; SELI : INTEGER);
0813       (* UNITS (MS),RHO,BETA1,BETA2,ORDER,PI,GAMMA1,GAMMA2,BCKGRD : *)
0814       (* LOADS THE TIMING CONSTRAINTS *)
0815       VAR I : INTEGER;
0816       BEGIN
0817       ALINECOUNT := ALINECOUNT + 1;
0818       WRITE(PRIMFILE, 'A', ALINECOUNT : 4, ':', CONTINAME : 10, ':',
0819       TASKNAME : 10, ':', MS: ');
0820       FOR I := 1 TO SELI - 1 DO
0821       WRITE(PRIMFILE, SELSTORE[I] : 4, ':');
0822       WRITE(PRIMFILE, SELSTORE[SELI] : 4)
0823       END; (* PUTA *)
0824
0825       PROCEDURE PUTD (METRIC : ALFA; NUMVOLUMES, NUMMONITORS : INTEGER);
0826       (* LOADS THE DESIGN CRITERIA *)
0827       VAR I : INTEGER;
0828       BEGIN
0829       LINECOUNT := LINECOUNT + 1;
0830       WRITE(PRIMFILE, 'P', LINECOUNT : 4, ':', METRIC : 10, ':');
0831       FOR I := 1 TO NUMVOLUMES - 1 DO
0832       WRITE(PRIMFILE, SELSTORE[I] : 1, ':');
0833       WRITE(PRIMFILE, SELSTORE[NUMVOLUMES], ':');
0834       FOR I := NUMVOLUMES + 1 TO NUMMONITORS DO
0835       WRITE(PRIMFILE, SELSTORE[I], ':');
0836       WRITELN(PRIMFILE, SELSTORE[NUMVOLUMES + 1 + NUMMONITORS], ':')
0837       END; (* PUTD *)
0838
0839       FUNCTION CHECKTYPE (EXP1TYPE, EXP2TYPE : EXPTYPES) : EXPTYPES;
0840       (* CHECKS THE TYPES OF INPUT VALUES *)
0841       BEGIN
0842       IF EXP1TYPE = EXP2TYPE THEN
0843       CHECKTYPE := EXP1TYPE
0844       ELSE
0845       CHECKTYPE := ERRORS
END; (* CHECKTYPE *)

FUNCTION COMPUTPRE (PRE1, PRE2 : INTEGER) : INTEGER;
(* COMPUTS THE PRECISION REQUIRED FOR THE *)
(* RESULT OF AN ARITHMETIC OPERATION *)
BEGIN
IF PRE1 > PRE2 THEN
  COMPUTPRE := PRE1
ELSE
  COMPUTPRE := PRE2
END; (* COMPUTPRE *)

PROCEDURE NEWTEMP (VAR ZPRECISION : INTEGER; VAR TEMP : ALFA);
(* CHECKS TO SEE IF A VARIABLE NAME HAS BEEN PREVIOUSLY *)
(* DEFINED. IF IT HAS, NEWTEMP RETURNS TRUE. IF NOT, IT *)
(* RETURNS FALSE. *)
CONST LASTB = 10;
VAR I, STOPSEARCH : INTEGER;
BEGIN
CASE ZPRECISION OF
  1,2,3,4,5,6,7,8 :
    BEGIN
      I := 1;
      STOPSEARCH := LASTB
      END;
    9,10,11,12,13,14,15,16 :
      BEGIN
        I := LASTB + 1;
        STOPSEARCH := MAXTEMPS
      END;
OTHERWISE ERROR(35,1,-1);
END;
WHILE (I <= STOPSEARCH) AND (TEMPNAME[I].INUSE) DO
  I := I + 1;
IF I > STOPSEARCH THEN
  ERROR(22,1,-1)
ELSE
BEGIN
  TEMP := TEMPNAME[I].NAME;
  ZPRECISION := TEMPNAME[I].PRECISION;
  TEMPERATURE[I].INUSE := TRUE;
  IF NOT TEMPERATURE[I].USED THEN
    TEMPERATURE[I].USED := TRUE
  END
END; (* NEWTEMP *)

PROCEDURE RETURNTEMP (TEMP : ALFA);
(* RETURNS FALSE IF A TEMP IS NEW *)
VAR I : INTEGER;
BEGIN

I := 1;
WHILE (TEMPNAME[I].NAME <> TEMP) AND (I <= MAXTEMPS) DO
  I := I + 1;
IF I > MAXTEMPS THEN
  WRITELN(TRANSLATE, 'FIX THE ERROR IN RETURNTEMP')
ELSE
  TEMPNAME[I].INUSE := FALSE
END; (* RETURNTEMP *)

PROCEDURE PRINTEMPS;
(* LOADS THE VARIABLES *)
VAR I : INTEGER;
BEGIN
  FOR I := 1 TO MAXTEMPS DO
    IF TEMPNAME[I].USED THEN
      BEGIN
        OPSTORE[I] := TEMPNAME[I].NAME;
        SELSTORE[I] := TEMPNAME[I].PRECISION;
        PUTS('var ', I, I)
      END
    END; (* PRINTEMPS *)

PROCEDURE PUSHEVALSTACK (ZNAME : ALFA; ZPRECISION : INTEGER);
(* PUSHES A VARIABLE ON TO THE STACK *)
BEGIN
  ESI := ESI + 1;
  EVALSTACK[ESI].NAME := ZNAME;
  EVALSTACK[ESI].PRECISION := ZPRECISION
END; (*PUSHEVALSTACK *)

PROCEDURE POPEVALSTACK (VAR NAME : ALFA; VAR ZPRECISION : INTEGER);
(* POPS A VARIABLE FROM THE STACK *)
BEGIN
  NAME := EVALSTACK[ESI].NAME;
  ZPRECISION := EVALSTACK[ESI].PRECISION;
  IF (NAME[1] = 'x') AND (NAME[2] = 't') THEN
    RETURNTEMP(NAME);
  ESI := ESI - 1
END; (* POPEVALSTACK *)

PROCEDURE NEWCONS (ZVALUE : INTEGER; VAR ZNAME : ALFA;
                    ZPRECISION : INTEGER);
(* CREATES A NEW CONSTANT IF NEEDED. OTHERWISE *)
BEGIN
  VAR I : INTEGER;
  BEGIN

0955   I := 1;
0956   WHILE (I <= CSI) AND (CONSTANTSTORE[I].VAL <> ZVALUE) DO
0957     I := I + 1;
0958   IF I > CSI THEN
0959     BEGIN
0960       CSI := CSI + 1;
0961       CONSTANTSTORE[CSI].VAL := ZVALUE;
0962       IF (ZVALUE >= -127) AND (ZVALUE <= 128) THEN
0963         ZPRECISION := 8
0964       ELSE
0965         IF (ZVALUE >= -32768) AND (ZVALUE <= 32768) THEN
0966           ZPRECISION := 16
0967         ELSE
0968           BEGIN
0969             ERROR(31,1,-1);
0970             ZPRECISION := 16
0971           END;
0972           CONSTANTSTORE[CSI].PRECISION := ZPRECISION;
0973           ZNAME := CONSTANTSTORE[I].NAME
0974         END
0975       ELSE
0976       BEGIN
0977         ZNAME := CONSTANTSTORE[I].NAME;
0978         ZVALUE := CONSTANTSTORE[I].VAL;
0979         ZPRECISION := CONSTANTSTORE[I].PRECISION
0980       END
0981     END;
0982   END; (* NEWCONS *)
0983   NEWLABEL (VAR LABELNAME : ALFA);
0984   VAR PLACE, LCCOPY : INTEGER;
0985     BEGIN
0986       LABELNAME := 'O';
0987       LABELCOUNT := LABELCOUNT + 1;
0988       LCCOPY := LABELCOUNT;
0989       IF LCCOPY > 0 THEN
0990         LABELNAME := LABELNAME + CHARVAL(LCCOPY);
0991     END; (* NEWLABEL *)
0992
0993   FUNCTION FINDTRANSITION (CSTATE,COKEN : INTEGER) : INTEGER;
0994     (* CHECKS TO SEE IF ANY TRANSITIONS EXIST *)
0995   VAR I : INTEGER;
1000   BEGIN
1001     FINDTRANSITION := 0;
1002     FOR I := FTRN[CSTATE] TO FTRN[CSTATE + 1] - 1 DO
1003       IF COKEN = ENT[TRAN[I]] THEN
1004         FINDTRANSITION := TRAN[I]
1005     END; (*FINDTRANSITION*)
1006
1007
FUNCTION FINDREDUCTION (CSTATE,COKEN : INTEGER) : INTEGER;
(* CHECKS TO SEE IF ANY REDUCTIONS EXIST *)
VAR I, J : INTEGER;
BEGIN
FINDREDUCTION := 0;
FOR I := PRED[CSTATE] TO PRED[CSTATE + 1] - 1 DO
  FOR J := LSET[NSET[I]] TO LSET[NSET[I] + 1] - 1 DO
    IF COKEN = LS[J] THEN
      FINDREDUCTION := PROD[I]
END; (*FINDREDUCTION*)

PROCEDURE DOTRANITION (NEWSTA : INTEGER);
(* GIVEN THE TRANSITION NUMBER, THIS MODULE *)
(* EXECUTES THE TRANSITION AND RETURNS TO *)
(* PARSE TO CONTINUE THE LOOP *)
BEGIN
  IF SWITCH[TRACEPARSE] THEN
    WRITELN('TRANSLATE; ' TO STATE ', NOWSTA :2,
    TO STATE ', NEWSTA :2);
    STKPTR := STKPTR + 1;
    IF STKPTR <= MAXSTK THEN
      BEGIN
        STACK[STKPTR].TOK := NEXTSYM;
        STACK[STKPTR].DES := DESCRIPTION;
        STACK[STKPTR].STATE := NEWSTA;
        NOWSTA := NEWSTA;
        GETSYM(NEXTSYM,DESCRIPTION);
        IF SWITCH[TRACTION] THEN
          WRITELN('NEXTSYM = ', NEXTSYM :2);
      END
    ELSE
      BEGIN
        WRITELN('GOING INTO ERROR IN DOTRAN');
        ERROR(0,0)
      END
END; (*DOTRANITION*)

PROCEDURE SEMANTIC (PRODUCTION : INTEGER); FORWARD;
PROCEDURE SEMANTICI(PRODUCTION : INTEGER); FORWARD;

PROCEDURE DOREDUCTION (PROD : INTEGER);
(* GIVEN THE REDUCTION, THIS MODULE DOES THE REDUCTION *)
(* BY CALLING SEMANTIC WITH THE PARTICULAR CONSTRUCT *)
(* IT HAS RECOGNIZED. DOREDUCTION THEN REMOVES THE *)
(* APPROPRIATE NUMBER OF ITEMS FROM THE STACK AND *)
(* PLACES THE REDUCED VERSION ON THE STACK. *)
BEGIN
  SEMANTIC(0);
STKPTR := STKPTR - LEN[PROD];
IF STKPTR <= MAXSTK THEN
BEGIN
NOWSTA := FINDTRANSITION(STACK[STKPTR].STATE, LHS[PROD]);
IF SWITCH[TRACEPARSE] THEN
WRITELN(TRANSLATE,'PRODUCTION ', PROD ; 2,
' AND TRANSITION FROM STATE ', STACK[STKPTR].STATE ; 2,
' TO STATE ', NOWSTA ; 2);
STKPTR := STKPTR + 1;
STACK[STKPTR].TON := LHS[PROD];
STACK[STKPTR].STATE := NOWSTA
END
ELSE
BEGIN
WRITELN(TRANSLATE,'GOING INTO ERROR IN DOREDUCTION');
ERROR(9,5,0)
END
END; (* DOREDUCTION *)

PROCEDURE PARSE;

(* REPEATEDLY EXECUTES UNTIL IT TRANSITIONS TO A FINAL *)
(* STATE. CALLS FINDREDOCUTION TO SEE IF ANY REDUCTIONS *)
(* EXIST. IF ONE DOES, IT DOES IT BY CALLING DOREDUCTION *)
(* AND GOES BACK TO REPEAT THE LOOP. IF NO REDUCTIONS *)
(* EXIST, IT CALLS FINDTRANSITION TO SEE IF ANY TRANSITIONS *)
(* EXIST. IF ONE DOES, IT CALLS DOTRANSITION AND THEN *)
(* REPEATS THIS LOOP. IF NEITHER EXISTS, THEN EITHER AN *)
(* ERROR EXISTS OR WE HAVE TRANSITIONED TO A FINAL STATE *)

BEGIN
REPEAT
REDUCTION := FINDREDUCTION(NOWSTA,NEXTSYM);
IF REDUCTION > 0 THEN
DOREDUCTION(REDUCTION)
ELSE
BEGIN
TRANSITION := FINDTRANSITION(NOWSTA,NEXTSYM);
IF TRANSITION <> FSTATE THEN
BEGIN
IF TRANSITION > 0 Then
DOTRANSITION(TRANSITION)
ELSE
BEGIN
WRITELN(TRANSLATE,'GOING INTO ERROR IN PARSE');
ERROR(5,4,-1)
END
END
END
END
END
NOWSTA := TRANSITION

UNTIL NOWSTA = FSTATE
END; (* PARSE *)

(* ERROR HANDLING ROUTINES *)

PROCEDURE PRINTERRORS;
VAR I : INTEGER;
BEGIN
IF PAGELINECOUNT + 10 > PAGESIZE THEN
WRITE(TRANSLATE, 'I');
I := 1;
WRITE(TRANSLATE);
WRITE(TRANSLATE);
WRITE(TRANSLATE, ':25, 'PROGRAM ERRORS');
WRITE(TRANSLATE, ':25, *---------------*
WRITE(TRANSLATE);
WRITE(TRANSLATE);
WRITE(TRANSLATE);
WHILE ERRLIST <> [] DO
BEGIN
WHILE NOT (I IN ERRLIST) DO
I := I + 1;
WRITE(TRANSLATE, I : 5);
CASE I OF
1: WRITE(TRANSLATE, ' : DIGIT EXPECTED');
2: WRITE(TRANSLATE, ' : ERROR IN IDENTIFIER');
3: WRITE(TRANSLATE, ' : ERROR IN NUMBER');
4: WRITE(TRANSLATE, ' : ERROR IN EXPONENT');
5: WRITE(TRANSLATE, ' : IMPROPER CONSTRUCTION, EXPECTED',
'SYMBOL LIST FOLLOWS');
6: WRITE(TRANSLATE, ' : BASES 2 THROUGH 16');
7: WRITE(TRANSLATE, ' : CHARACTER NOT DEFINED FOR THIS BASE');
8: WRITE(TRANSLATE, ' : STACK OVERFLOW IN DTRANSITION.');
9: WRITE(TRANSLATE, ' : TO CORRECT, INCREASE MAXSTK.');
10: WRITE(TRANSLATE, ' : TOO MANY ERRORS ON THIS LINE');
11: WRITE(TRANSLATE, ' : CHARACTERS ARE ITALICIZED');
12: WRITE(TRANSLATE, ' : CHARACTER ALREADY SPECIFIED');
13: WRITE(TRANSLATE, ' : CHARACTER REPRESENTATION MUST BE AN',
'INTEGER VALUE');
14: WRITE(TRANSLATE, ' : UNKNOWN OR MISSING COMPILER OPTION',
'INSTRUCTION');
15: WRITE(TRANSLATE, ' : REALS NOT IMPLEMENTED');
16: WRITE(TRANSLATE, ' : OPERATION NOT DEFINED FOR STRING',
'OPERANDS');
20:  writeln(translate, ' : TYPE MISMATCH');
21: writeln(translate, ' : OPERATION NOT DEFINED FOR BOOLEAN',
22: ' OPERANDS');
23: writeln(translate, ' : TOO MANY REQUESTS FOR TEMP NAMES--'.
24: ' INCREASE CONSTANT MAXTEMPS');
25: writeln(translate, ' : OPERATION NOT DEFINED FOR NUMERICAL',
26: ' OPERANDS');
27: writeln(translate, ' : PROCEDURE NAME EXPECTED');
28: writeln(translate, ' : MISMATCHING BEGIN-ENDING PAIR');
29: writeln(translate, ' : EXPRESSION TYPE MUST BE BOOLEAN');
30: writeln(translate, ' : ERROR IN FACTOR');
31: writeln(translate, ' : IDENTIFIER NOT DECLARED');
32: writeln(translate, ' : INDEX MUST BE DECLARED AS ARITHMETIC');
33: writeln(translate, ' : THE FOLLOWING ARGUMENT MUST ';
34: ' BE BINARY');
35: writeln(translate, ' : CONSTANT TOO LARGE--NOT IN'.
36: ' '); 37: writeln(translate, ' : IMPROPER ASSIGNMENT');
38: writeln(translate, ' : PARAMETERS NOT IMPLEMENTED');
39: writeln(translate, ' : TASK NAME EXPECTED');
40: writeln(translate, ' : REQUEST FOR A TEMPORARY WITH A',
41: ' PRECISION > 16');
42: writeln(translate, ' : STRUCTURED VARIABLES ARE ',
43: ' NOT IMPLEMENTED');
44: writeln(translate, ' : BIT FIELDS ARE NOT IMPLEMENTED');
45: writeln(translate, ' : EXPRESSION TYPES MUST BE INTEGER');
46: ');
47: end;
48: end;
49: end;
50: end; (*printerrors*)
51: end;
52: end;
53: end;
54: procedure recover;
55: (* this attempts to recover the parse from an *)
56: (* error such as an undefined variable *)
57: const numsolids = 21;
58: var solid : array[1..numsolids] of integer;
59: function soliddtoken (token : integer) : boolean;
60: var lowerbound, upperbound, index : integer;
61: found : boolean;
62: begin
63: lowerbound := 1;
64: upperbound := numsolids;
65: found := false;
66: while (lowerbound < upperbound) and (not found) do
67: begin
68: index := (lowerbound + upperbound) div 2;
69: if token < solid[index] then
70: upperbound := index - 1
71: else
72: if token > solid[index] then

LOWERBOUND := INDEX + 1
ELSE
  FOUND := TRUE
END; (*WHILE*)
SOLIDTOKEN := FOUND
END; (* SOLIDTOKEN *)
BEGIN
BEGIN
WHILE (NOT SOLIDTOKEN(STACK[STKPTR].TOK)) AND
(STKPTR > 2) DO
  STKPTR := STKPTR - 1;
  WHILE (FINDTRANSITION(STACK[STKPTR].STATE,NEXTSYM) = 0) AND
  (FINDREDUCTION(STACK[STKPTR].STATE,NEXTSYM) = 0) AND
  (NOT LASTTOK) DO
    GETSYM(NEXTSYM,DESCRIPTION);
    IF LASTTOK THEN
      ERROR(10.5,-1);
    NOWSTA := STACK[STKPTR].STATE
    END; (*RECOVER *)
PROCEDURE ERROR (*ERRORNUM, INTENSITY, PTROFFSET : INTEGER *);
VAR POSITION : INTEGER;
BEGIN
WRITELN(TRANSLATE,'IN ERROR SLC = ',SOURCENAMECOUNT : 3);
IF NOT PROGRAMERRFLAG THEN
  PROGRAMERRFLAG := TRUE;
IF LINERRP RT < MAXLINERRS THEN
  BEGIN
    LINERRPRT := LINERRPRT + 1;
    LINERRS[LINERRPRT].ERRORPOSITION := CC + PTROFFSET;
    LINERRS[LINERRPRT].ERRORNUM := ERRORNUM;
    LINERRS[LINERRPRT].STATE := NOWSTA;
    ERRLIST := ERRLIST + [ERRORNUM];
  END;
ELSE
  IF NOT OVERFLOWLOGGED THEN
    BEGIN
      OVERFLOWLOGGED := TRUE;
      LINERRPRT := LINERRPRT + 1;
      LINERRS[LINERRPRT].ERRORPOSITION := CC + PTROFFSET;
      LINERRS[LINERRPRT].ERRORNUM := 12;
      ERRLIST := ERRLIST + [12]
    END;
END;

IF INTENSITY = 5 THEN
GOTO 99;
IF INTENSITY = 4 THEN
RECOVER
END; (* ERROR *)

PROCEDURE PRINTLINES;
VAR LINESIZE, MARKER : INTEGER;
PROCEDURE PRINTERROR (X : INTEGER);
VAR I, J, K, SYM : INTEGER;
BEGIN
WITH LINESIZE DO
BEGIN
IF ERRPOSITION = LINESIZE + 1 THEN
BEGIN
WRITE(TRANSLATE,'.',ERRNUM : 2);
LINEPOSITION := LINEPOSITION + 3
END
ELSE
BEGIN
WRITE(TRANSLATE,'.',(ERRPOSITION - LINEPOSITION) - 1,'.'),
ERRNUM : 2);
LINEPOSITION := ERRPOSITION + 2;
END;
END;
IF ERRNUM = 5 THEN
BEGIN
WRITE(TRANSLATE,'');
LINEPOSITION := LINESIZE + 1;
FOR I := FRML[STATE] TO FRML[STATE + 1] DO
BEGIN
SYM := EN(TRAN[I]);
IF SYM <= NUMBERALS THEN
BEGIN
FOR J := STRINGHEAD[SYM] TO STRINGHEAD[SYM + 1] - 1 DO
BEGIN
WRITE(TRANSLATE, STRINGSTORE[J] : 1);
LINEPOSITION := LINEPOSITION + 1
END;
WRITE(TRANSLATE,'.');
LINEPOSITION := LINEPOSITION + 1
END;
END;
END;
END;
END;
END;
FOR I := FRED[STATE] TO FRED[STATE + 1] DO
FOR J := LSET[NSET[I]] TO LSET[NSET[I] + 1] - 1 DO
BEGIN
SYM := LS[J];
FOR K := STRINGHEAD[SYM] TO STRINGHEAD[SYM + 1] DO
BEGIN
WRITE(TRANSLATE, STRINGSTORE[K] : 1);
END;
LINEPOSITION := LINEPOSITION + 1
END;
WRITE(TRANSLATE,'');
LINEPOSITION := LINEPOSITION + 1
END
END
BEGIN
MORERRORS := TRUE;
LINEPOSITION := 0;
MARKER := 1;
WRITE(TRANSLATE,'****');
WHILE MORERRORS DO
BEGIN
WHILE MARKER <= LINERRPTR DO
BEGIN
IF LINERRORS[MARKER].ERRPOSITION > LINEPOSITION THEN
BEGIN
PRINTERROR(MARKER);
LINERRORS[MARKER].ERRPOSITION := 0
END;
MARKER := MARKER + 1
END;
MARKER := 1;
WHILE (MARKER <= LINERRPTR) AND
(LINERRORS[MARKER].ERRPOSITION = 0) DO
MARKER := MARKER + 1;
IF MARKER <= LINERRPTR THEN
BEGIN
WRITE(LINTRANSLATE);
LINEPOSITION := 0;
WRITE(TRANSLATE,'');
PAGELINECOUNT := PAGELINECOUNT + 1
END
ELSE
BEGIN
WRITE(LINTRANSLATE);
MORERRORS := FALSE
END
END
(* PRINTLINERRORS *)

PROCEDURE GPRINTABLE;
(* THIS IS USED BY A TOGGLE IN THE INPUT IF *)
(* THE DETAILS OF THE PARSE ARE DESIRED IN *)
(* AN OUTPUT FILE CALLED TRANSLATE *)
VAR PTR : SYMPTR;
I, J : INTEGER;
BEGIN
FOR I := 1 TO TABSIZE DO
BEGIN
PTR := SYMTABLE[I];
WRITELN(TRANSPOSE, I : 5);
WHILE PTR <> NIL DO
BEGIN
WRITE(TRANSPOSE, ' '); 
WRITE(TRANSPOSE, PTR, SYMPNAME : 10, ' ');
END;
WITH PTR DO
CASE KIND OF
UNDEFINED : WRITE(TRANSPOSE, 'UNDEFINED');
RESWD : WRITE(TRANSPOSE, 'RESWD ', KEY1 : 5);
BINARY : WRITE(TRANSPOSE, 'BINARY ', PRECISION2 : 5,
IVAL2 : 5);
ARITHMETIC : WRITE(TRANSPOSE, 'ARITHMETIC', PRECISION3 : 5,
IVAL3 : 5);
WORD : WRITE(TRANSPOSE, 'TEXT ', CODIS4, SYMPNAME : 10,
STRINGPTR4 : 5);
CHARREP : WRITE(TRANSPOSE, 'CHARREP ', CODIS5, SYMPNAME : 10,
IVAL5 : 5);
TRANSDEC : WRITE(TRANSPOSE, 'TRANSDEC ', TYPE6, SYMPNAME : 10,
TECHNOLOGY6, SYMPNAME : 10, PRECISION6 : 5);
TASK : WRITE(TRANSPOSE, 'TASK '); 
FUNCST : WRITE(TRANSPOSE, 'FUNCTION ', TYPE8, SYMPNAME : 10,
PRECISION8 : 5)
END;
WRITELN(TRANSPOSE);
PTR := PTR, LINK
END;
WRITELN(TRANSPOSE)
END; (* GPORINTABLE *)

(*--------------------- GET INPUT SYMBOLS -----------------*)

PROCEDURE GETSYM (*VAR MESTSYM : INTEGER; VAR DESC : DESCRIPTOR *);
(* GETS THE NEXT INPUT SYMBOL AND DETERMINES *)
(* THE TYPE AND DESCRIPTION OF THE TOKEN *)
VAR LEN, (* LENGTH OF INPUT SYMBOL *)
I, (* INDEX *)
BASE, (* BASE FOR BASED INTEGER *)
NUMDIGITS, (* NUMBER OF DIGITS IN CURRENT INTEGER *)
SCALE, (* EXPONENT ADJUSTMENT *)
EXPONENT : INTEGER, (* INTEGER VAL OF EXPONENT *)
SIGN, (* FLAG INDICATING NEGATIVE EXPONENT *)
DECIMALFLAG, (* FLAG INDICATING RADIX PT *)
BASEDNUMBER, (* FLAG INDICATING BASE OTHER THAN 10 *)
ENDSTRING : BOOLEAN, (* FLAG INDICATING END OF STRING *)
FAC, R : REAL, (* FOR EXPONENT ADJUSTMENT *)
FUNCTION FLIP (FLIPEE : BOOLEAN) : BOOLEAN;
BEGIN
IF FLIP = TRUE THEN
FLIP := FALSE
ELSE
FLIP := TRUE
END; (*FLIP*)

FUNCTION CHARVAL (CH : CHAR) : INTEGER;
BEGIN
IF CH IN ['A'..'F','O'..'9'] THEN
CASE CH OF
'0' : CHARVAL := 0;
'1' : CHARVAL := 1;
'2' : CHARVAL := 2;
'3' : CHARVAL := 3;
'4' : CHARVAL := 4;
'5' : CHARVAL := 5;
'6' : CHARVAL := 6;
'7' : CHARVAL := 7;
'8' : CHARVAL := 8;
'9' : CHARVAL := 9;
'A' : CHARVAL := 10;
'B' : CHARVAL := 11;
'C' : CHARVAL := 12;
'D' : CHARVAL := 13;
'E' : CHARVAL := 14;
'F' : CHARVAL := 15
END
ELSE
CHARVAL := 16
END; (*CHARVAL*)

PROCEDURE INCHAR (VAR CH : CHAR);
BEGIN
IF CC = LL THEN
BEGIN
IF EOF(DAT) THEN
BEGIN
WRITELN(TRANSLATE,'GOING INTO ERROR IN INCHAR');
ERROR(10,5,0)
END;
END;
IF LINERRPTR <> 0 THEN
BEGIN
PRINTLINERRORS;
LINERRPTR := 0;
OVERFLOWLOGGED := FALSE
END;
LL := 0;
CC := 0;
IF PAGELINECOUNT MOD PAGESIZE = 0 THEN (* NEW PAGE *)
BEGIN
WRITELN(TRANSLATE,'1');
HEADER
END;
1465 SOURCELINENOINT := SOURCELINENOINT \+ 1;
1466 WRITE(TRANSLATE, SOURCELINENOINT := 5, ' ');
1467 WHILE NOT EOF(DAT) DO
1468 BEGIN
1469 LL := LL \+ 1;
1470 READ(DAT, CH);
1471 WRITE(TRANSLATE, CH);
1472 LINE[LL] := CH;
1473 END;
1474 LL := LL \+ 1;
1475 LINE[LL] := ' ';
1476 WRITELN(TRANSLATE);
1477 READLN(DAT);
1478 PAGELINENOINT := PAGELINENOINT \+ 1
1479 END;
1500 CC := CC \+ 1;
1501 CH := LINE[CC];
1502 IF EOF(DAT) THEN
1503 IF CC = LL THEN
1504 LASTTOK := TRUE
1505 END; (*INCHAR*)
1506 PROEDURE GETNUMBER (VAR CH : CHAR; VAR BASE : INTEGER;
1507 DECIMALPART : BOOLEAN; VAR DIGITS, IVALUE : INTEGER;
1508 VAR RVALUE : REAL);
1509 BEGIN
1510 DIGITS := 1;
1511 IF CHARVAL(CH) >= BASE THEN
1512 ERROR(7,1,0);
1513 IF DECIMALPART THEN
1514 RVALUE := RVALUE \* BASE \+ CHARVAL(CH)
1515 ELSE
1516 IVALUE := CHARVAL(CH);
1517 INCHAR(CH);
1518 WHILE ((BASE = 10) AND (CH IN ['0'..'9',',' ])) OR
1519 ((BASE <> 10) AND (CH IN ['0'..'9','A'..'Z',','])) DO
1520 BEGIN
1521 IF CH = ' ' THEN
1522 INCHAR(CH);
1523 IF DECIMALPART THEN
1524 RVALUE := RVALUE \* BASE \+ CHARVAL(CH)
1525 ELSE
1526 IVALUE := IVALUE \* BASE \+ CHARVAL(CH);
1527 DIGITS := DIGITS \+ 1;
1528 IF CHARVAL(CH) >= BASE THEN
1529 ERROR(7,1,0);
1530 INCHAR(CH)
1531 END;
1532 END; (* GETNUMBER *)
1533 PROCEDURE GETEXPONENT (VAR SIGN : BOOLEAN; VAR EXP : INTEGER);
VAR NUMDIGITS, INTVAL : INTEGER;
REALVAL : REAL;
BEGIN
SIGN := FALSE;
INCHAR(CH);
IF CH = '+' THEN
INCHAR(CH)
ELSE
IF CH = '-' THEN
BEGIN
SIGN := TRUE;
INCHAR(CH)
END;
IF CH IN ['0'..'9'] THEN
BEGIN
GETNUMBER(CH,10,FALSE,NUMDIGITS,INTVAL,REALVAL);
EXP := INTVAL
END
ELSE
BEGIN
ERROR(4,1,0);
EXP := 0
END;
END;
BEGIN (* GETSYM *)
DESSYNAME := ' 
DES.INTVAL := 0;
DES.REALVAL := 0.0;
DES.TMPNAME := ' 
DES.CHARVAL := '0';
DES.SYMLOC := NIL;
DES.LINEPOS := LINELENGTH;
BUFFER := ' 
IF EOF(DAT) AND LASTTOK THEN
NEXTSYM := ENDTOK
ELSE
WHILE CH = ' ' DO
INCHAR(CH);
CASE CH OF
'A'..'B'..'C'..'D'..'F'..'G'..'H'..'I'..'J'..'L'..'M'..'N'..'O'..'P'..'Q'..'R'..'S'..'T'..'U'..'V'..'W'..'X'..'Y'..'Z':
BEGIN
I := 1;
BUFFER[I] := CH;
INCHAR(CH);
WHILE CH IN ['A'..'Z'..'0'..'9'] DO
BEGIN
IF I < 10 THEN
BEGIN
I := I + 1;
BUFFER[I] := CH
END
END
END

1580   END;  IF CH = '' THEN
1581   BEGIN
1582       INCHAR(CH);
1583       IF I < 10 THEN
1584         BEGIN
1585             I := I + 1;
1586             BUFFER[I] := CH
1587         END;
1588         IF NOT (CH IN ['A'..'Z','0'..'9']) THEN
1589             ERROR(2,1,0)
1590         END;
1591       INCHAR(CH)
1592     END;
1593     WHILE I < 10 DO
1594         BEGIN
1595             I := I + 1;
1596             BUFFER[I] := 'r'
1597         END;
1598         (* IDENTIFIER OR RESERVED WORD *)
1599         D.E.S.SYMLOC := LOOKUP(BUFFER);
1600         IF D.E.S.SYMLOC <> NIL THEN
1601             BEGIN
1602                 IF D.E.S.SYMLOC.KIND = RESWD THEN
1603                     BEGIN
1604                         NEXTSYM := D.E.S.SYMLOC.KEY1;
1605                         D.E.S.LINEPOS := CC - 1
1606                     END
1607                 ELSE
1608                     NEXTSYM := IDTOK
1609                 END
1610             ELSE
1611                 BEGIN
1612                     NEXTSYM := IDTOK;
1613                     D.E.S.SYMLOC := ENTER(BUFFER)
1614                 END;
1615                 D.E.S.SYNAME := D.E.S.SYMLOC.SYNAME
1616                 END;  (* CASE OF 'A'..'Z' *)
1617                 '0'..'1'..'2'..'3'..'4'..'5'..'6'..'7'..'8'..'9' :
1618         BEGIN
1619             NEXTSYM := NUMTOK;
1620             BASE := 10;
1621             SCALE := 0;
1622             DECIMALFLAG := FALSE;
1623             BASEDNUMBER := FALSE;
1624             GETNUMBER(CH,BASE,DECIMALFLAG,HUMDIGITS,D.E.S.INTVAL,D.E.S.REALVAL);
1625             IF CH = '' THEN  (*BASED NUMBER*)
1626         BEGIN
1627             BASEDNUMBER := TRUE;
1628             BASE := D.E.S.INTVAL;
1629             IF BASE > 16 THEN
1643    ERROR(6,1,-1);
1644    BASE := 16;
1645    END;
1646    INCHAR(CH);
1647    GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
1648    IF CH = ' ' THEN
1649      BEGIN
1650      DECIMALFLAG := TRUE;
1651      DES.REALVAL := DES.INTVAL;
1652      DES.INTVAL := 0;
1653      DES.CHARVAL := 'R';
1654      INCHAR(CH);
1655      GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
1656      SCALE := -NUMDIGITS
1657      END;
1658      IF CH <> ' ' THEN
1659      ERROR(3,1,0);
1660      INCHAR(CH)
1661      END;
1662      IF CH = '.' THEN (*DECIMAL PART *)
1663      BEGIN
1664      INCHAR(CH);
1665      IF BASEONNUMBER THEN
1666      ERROR(3,1,-1);
1667      DES.REALVAL := DES.INTVAL;
1668      DES.INTVAL := 0;
1669      DECIMALFLAG := TRUE;
1670      DES.CHARVAL := 'R';
1671      GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
1672      SCALE := -NUMDIGITS
1673      END;
1674      IF CH = 'E' THEN (* EXPONENT PART *)
1675      BEGIN
1676      DECIMALFLAG := FALSE;
1677      DES.CHARVAL := 'R';
1678      IF SCALE = 0 THEN
1679      DES.REALVAL := DES.INTVAL;
1680      GETEXPONENT(SIGNS,EXponent);
1681      IF SIGN THEN
1682      SCALE := SCALE - EXponent
1683      ELSE
1684      SCALE := SCALE + EXponent
1685      END;
1686      IF SCALE <> 0 THEN (*ADJUST SCALE *)
1687      BEGIN
1688      R := 1;
1689      SIGN := SCALE < 0;
1690      SCALE := ABS(SCALE);
1691      FAC := BASE;
1692      REPEAT
1693      IF ODD(SCALE) THEN
1694      R := R *FAC;
1695      ELSE
1696      R := R /FAC;
1697      UNTIL SCALE = 0;
FAC := SQR(FAC);
SCALE := SCALE DIV 2
UNTIL SCALE = 0;
IF SIGN THEN
  DES.REALVAL := DES.REALVAL / R
ELSE
  DES.REALVAL := DES.REALVAL * R
END;
(* CASE OF '0'..'9' *)
BEGIN
  '+' , '*' , ' ' , ' ' , '  ', ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' :
BEGIN
  NEXTSYM := SPS(CH);
  DES.LINEPOS := CC;
  INCHAR(CH);
END;
( * STRINGS * )
BEGIN
  INCHAR(CH);
ENDSTRING := FALSE;
WHILE NOT ENDSTRING DO
BEGIN
  WHILE CH <> ' ' DO
    INCHAR(CH);
  INCHAR(CH);
  IF CH = '=' THEN
    INCHAR(CH)
  ELSE
    ENDSTRING := TRUE
  END;
END;
NEXTSYM := STRINGTOK
END; ( * STRINGS * )
' ' :
BEGIN
  INCHAR(CH);
  IF CH = '=', ' ' THEN
BEGIN
  INCHAR(CH);
  IF CH = ' ' THEN
BEGIN
    INCHAR(CH);
    GETSYM(NEXTSYM,DES);
  IF NEXTSYM <> IDTOK THEN
    ERROR(17,1,-1)
ELSE
  IF DES.SYNAME = 'TRACEPARSE' THEN
    SWITCH[TRACEPARSE] := FLIP(SWITCH[TRACEPARSE])
ELSE IF DES.SYNAME = 'TRACETOK' THEN
    SWITCH[TRACETOK] := FLIP(SWITCH[TRACETOK])
ELSE IF DES.SYNAME = 'PRINTTABLE' THEN
    SWITCH[PRINTTABLE] := FLIP(SWITCH[PRINTTABLE])
ELSE
    ERROR(17,1,-1)
END;
WHILE CC < LL DO
    INCHAR(CH);
    GETSYM(NEWSYM,DES)
END
ELSE
    NEXTSYM := SPS['-']
END;
"=" :
BEGIN
    INCHAR(CH);
    IF CH = '"' THEN
        BEGIN
            NEXTSYM := ARROWTOK;
    END
    INCHAR(CH)
END
ELSE
    IF CH = '"' THEN
        BEGIN
            NEXTSYM := EQUIVALENCE;
            INCHAR(CH)
        END
    ELSE
        NEXTSYM := SPS['=' ]
    END; (* CASE OF '"' ')

"" :
BEGIN
    INCHAR(CH);
    IF CH = '"' THEN
        BEGIN
            NEXTSYM := PWRTOK;
        END
    INCHAR(CH)
END
ELSE
    NEXTSYM := SPS['"'
END;(* CASE OF '"' ')

' : :
BEGIN
    INCHAR(CH);
    IF CH = '"' THEN
        BEGIN
            NEXTSYM := BECOMESTOK;
        END
    INCHAR(CH)
END
ELSE
    NEXTSYM := SPS['']
END; (* CASE OF '"' ')

" : :
BEGIN
    INCHAR(CH);
    IF CH = '"' THEN
        BEGIN
            NEXTSYM := BECOMESTOK;
        END
    INCHAR(CH)
END
ELSE
    NEXTSYM := SPS['']
END; (* CASE OF '"' ')

'/' :  
BEGIN  
INCHAR(CH);  
IF CH = '/' THEN  
BEGIN  
NEXTSYM := NOTEQTK;  
INCHAR(CH)  
END  
ELSE  
NEXTSYM := SPS['/']  
END; (* CASE OF ' / ' *)  

'<':  
BEGIN  
INCHAR(CH);  
IF CH = '<' THEN  
BEGIN  
NEXTSYM := LESSEQTK;  
INCHAR(CH)  
END  
ELSE  
NEXTSYM := SPS['<']  
END; (* CASE OF '<' *)  

'>':  
BEGIN  
INCHAR(CH);  
IF CH = '>' THEN  
BEGIN  
NEXTSYM := GTREQTOK;  
INCHAR(CH)  
END  
ELSE  
NEXTSYM := SPS['>']  
END; (* CASE OF '>' *)  

'....', '$', '%', '*', '?', ';', 'o', '...', ':  
BEGIN  
ERROR(11,1,0);  
INCHAR(CH);  
GETSYM(NEXTSYM,DES)  
END  
END; (* CASE *)  

END; (* GETSYM *)  

PROCEDURE SEMANTIC (*PRODUCTION : INTEGER*);  
(* SEMANTIC AND SEMANTIC1 PERFORM THE STACK *)  
(* MANIPULATION BASED ON THE CURRENT PRO-*  
(* DUCTION NUMBER AND WILL EMIT THE PROPER *)  
(* VALUES TO THE PRIMITIVE LIST, SYMBOL *)  
(* TABLE AND TRANSLATE FILES *)
VAR PTR : SYMPTR;
TYPVAR : EXPTYPES;
TEMPNAME : ALFA;
PRCSN, I : INTEGER;
BEGIN
CASE PRODUCTION OF
  2, 3, 4, 5, 6, 7, 8, 9, 10, 11:
  (* <AAP> ::= +/- *)
  12:
  (* <AP> ::= <NUMBER> *)
  IF STACK[STKPTR].DES.CHARVAL = '0' THEN (*INTEGER*)
  BEGIN
    NEWCONS(STACK[STKPTR].DES.INTVAL, TEMPPNAME, PRCSN);
    PUSHEVALSTACK(TEMPPNAME, PRCSN);
    STACK[STKPTR].EXPTYPE := INT
  END
  ELSE
    ERROR(18, -3);
    PUSHEVALSTACK('JUNK ', 8);
    STACK[STKPTR].EXPTYPE := REAL
  END;
  13:
  (* <PRIMARY> ::= *STRING* *)
  BEGIN
    STACK[STKPTR].EXPTYPE := STRING;
    PUSHEVALSTACK('STRING ', 8);
  END;
  14:
  (* <PRIMARY> ::= <NAME> *)
  BEGIN
    WITH STACK[STKPTR].DES.SYMQOC DO
      CASE KIND OF TRANSDEC, ARITHMETIC, UNDEFINED:
      BEGIN
        IF KIND = TRANSDEC THEN
          PUSHEVALSTACK(SYMNAME, PRECISION8)
        ELSE
          IF KIND = ARITHMETIC THEN
            PUSHEVALSTACK(SYMNAME, PRECISION3)
          ELSE
            BEGIN
              PUSHEVALSTACK(SYMNAME, 8); ERROR(28, -3)
            END;
            STACK[STKPTR].EXPTYPE := INT
            END;
            FNCH:
            BEGIN
              IF TYPEB.SYMNAME = 'ARITHMETIC' THEN
              ERROR(27, -3);
PUSHEVALSTACK(SYMMNAME, PRECISION);  
STACK[STKPTR].EXPTYPE := INT  
END;  
OTHERWISE  
BEGIN  
ERROR(27,1,-3);  
PUSHEVALSTACK( 'JUNK ', 0);  
STACK[STKPTR].EXPTYPE := INT  
END  
END;  
END;  

15:  
(* <PRIMARY> ::= ( <EXPRESSION> ) *)  
(* RETAIN INFORMATION ON EXPRESSION *)  
BEGIN  
STACK[STKPTR - 2] := STACK[STKPTR - 1];  
END;  

16:  
(* <FACTOR> ::= <PRIMARY> *)  

17:  
(* <FACTOR> ::= <FACTOR><FACTOR> *)  

18:  
(* <TERM> ::= <FACTOR> *)  

19.  
(* <TERM> ::= <TERM> <MOP> <FACTOR> *)  

23.  
(* <SIMPLE EXP> ::= <SIMPLE EXP> <AOP> <TERM> *)  

28:(* <SIMPLE EXP> ::= <SIMPLE EXP> <RELATIONAL OP> <SIMPLE EXP> *)  
BEGIN  
TYPPAR := CHECKTYPE(STACK[STKPTR - 2].EXPTYPE,  
STACK[STKPTR].EXPTYPE);  
POPEVALSTACK(OPSSTORE[3].SELSSTORE[3]);  
POPEVALSTACK(OPSSTORE[2].SELSSTORE[2]);  
NEWTEMP(SELSSTORE[1], OPSSTORE[1]);  
PUSHEVALSTACK(OPSSTORE[1].SELSSTORE[1]);  
CASE TYPPAR OF  
INT: CASE STACK[STKPTR-1].DES.INTVAL OF  
2 : ('+ ') PUTS('add', ',3,3);  
3 : (' - ') PUTS('sub', ',3,3);  
4 : ('* ') PUTS('mul', ',3,3);  
5 : ('/ ') PUTS('div', ',3,3);  
6 : ('< ') PUTS('lt', ',3,3);  
7 : ('<= ') PUTS('le', ',3,3);  
8 : ('= ') PUTS('eq', ',3,3);  
9 : ('> ') PUTS('gt', ',3,3);  
10 : ('>= ') PUTS('ge', ',3,3);  
11 : ('/= ') PUTS('ne', ',3,3);  
END;  
(*END OF STACK*)  
STRING : ERROR(18,0,-1);  
BOOLEAN : ERROR(21,0,-1);  
ERRORS : ERROR(20,0,-1);  
END;  
(*END OF TYPPAR*)  
IF STACK[STKPTR-1].DES.INTVAL IN [6..13] THEN
STACK[STKPTR-2].EXPTYPE := BOOL;
END;
END;

20 :  (* <SIMPLE EXP> ::= <TERM> *)

21 :  (* <SIMPLE EXP> ::= <AQP><TERM> *)

22 :  (* <SIMPLE EXP> ::= NOT TERM *)

23 :  (* <RELATION> ::= <SIMPLE EXP> *)

26 :  (* <EXP4> ::= <RELATION> *)

27,  (* <EXP4> ::= <EXP4> AND <RELATION> *)

28,  (* <EXP3> ::= <EXP3> OR <EXP4> *)

31,  (* <EXP2> ::= <EXP2> => <EXP3> *)

33 :  (* <EXPRESSION> ::= <EXPRESSION> == <EXP2> *)

BEGIN
TYPVAR := CHECKTYPE(STACK[STKPTR - 2].EXPTYPE, 
STACK[STKPTR].EXPTYPE);
POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
SELSTORE[1] := COMPUTPRE(SELSTORE[2],SELSTORE[3]);
NEWTEMP(SELSTORE[1],OPSTORE[1]);
PUS(new,'a');
PUSHVALSTACK(OPSTORE[1],SELSTORE[1]);
END;

BEGIN
TYPVAR := CHECKTYPE(STACK[STKPTR - 2].EXPTYPE, 
STACK[STKPTR].EXPTYPE);
POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
SELSTORE[1] := COMPUTPRE(SELSTORE[2],SELSTORE[3]);
NEWTEMP(SELSTORE[1],OPSTORE[1]);
PUS(new,'a');
PUSHVALSTACK(OPSTORE[1],SELSTORE[1]);
END;

CASE TYPVAR OF
BOOL : CASE PRODUCTION OF
27 : PUS('and',3,3);
29 : PUS('or',3,3);
31 : PUS('implicat',3,3);
33 : PUS('equivallenc',3,3);
END;
INT : ERROR(23,1,-1);
STNG : ERROR(19,1,-1);
ERRORS : ERROR(20,1,-1)
END;

28 :  (* <EXP3> ::= <EXP4> *)
30 :  (* <EXP2> ::= <EXP3> *)
32 :  (* <EXPRESSION> ::= <EXP2> *)
34 :  (* <EXPR LIST> ::= *
      STACK[STKPTR + 1].DES.INTVAL := 0;
35 :  (* <EXPR LIST> ::= <EXPRESSION> *)
36 :  (* <EXPR LIST> ::= <EXPR LIST>,<EXPRESSION> *)
38 :  (* <IF HEAD> ::= IF<EXPRESSION> *)
39 :  (* <WHILE DO> ::= <WHILE> <HEAD> <DO> <STMT GP> END WHILE *)
40 :  (* <WHILE HEAD> ::= <WHILE> <EXPRESSION> <MAXLOOPCOUNT> *)

BEGIN
IF STACK[STKPTR].EXPTYPE <> BOOL THEN
BEGIN
ERROR(26,1,-1);
STACK[STKPTR].EXPTYPE := BOOL
END;
POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
NEWLABEL(OPSTORE[2]);
Puts('jmpf ',2,1);
END;

BEGIN
Puts('whend ',2,0)
END;

BEGIN
POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
NEWLABEL(OPSTORE[2]);
2067    STACK[stkptr - 3].DES.TMPNAME := OPSTORE[2];
2069    PUTS("whilecon\t'2,1,;
2070    END;
2071    2072    BEGIN
2073        (* <while> ::= while *)
2074        NEWLABEL(OPSTORE[1]);
2075        STACK[stkptr].DES.SYNAME := OPSTORE[1];
2076        PUTS("whileastart\t',1,0)
2077        END;
2078    2079    BEGIN
2080        (* <for loop> ::= <for head> <do stmt gp end for> *)
2081        WITH STACK[stkptr - 4] DO
2082            OPSTORE[1] := DES.SYLOC.SYNAME;
2083            SELSTORE[1] := DES.SYLOC.PRECISION3;
2084            OPSTORE[2] := DES.SYNAME;
2085            OPSTORE[3] := DES.TMPNAME;
2087            PUTS("forend\t',3,2)
2088            END;
2089    2090    BEGIN
2091        (* <forhead> ::= for *ID* from <expression> to *)
2092        (* <expression> ::= <maxloopcount> *)
2093        IF STACK[stkptr - 6].DES.SYLOC.KIND <> ARITHMETIC THEN
2094            ERROR(29,1,-1);
2095            OPSTORE[1] := STACK[stkptr - 6].DES.SYNAME;
2096            SELSTORE[1] := STACK[stkptr - 6].DES.SYLOC.PRECISION3;
2097            POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
2098            POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
2099            NEWLABEL(OPSTORE[4]);
2100            NEWLABEL(OPSTORE[5]);
2102            PUTS("formax\t',5,4);
2103            STACK[stkptr - 7].DES.SYNAME := OPSTORE[4];
2104            STACK[stkptr - 7].DES.TMPNAME := OPSTORE[6];
2105            STACK[stkptr - 7].DES.SYLOC := STACK[stkptr - 6].DES.SYLOC;
2106            STACK[stkptr - 7].DES.INTVAL := SELSTORE[4]
2107            END;
2108    2109    BEGIN
2110        (* <perform task> ::= *ID* *)
2111        IF STACK[stkptr].DES.SYLOC.KIND <> TASK) AND
2112            (STACK[stkptr].DES.SYLOC.KIND <> FNCTN) THEN
2113            ERROR(24,1,-1);
2115            PUTS("call\t',1,0)
2116            END;
2117    2118    BEGIN
2119        (* <perform task> ::= <id> (<expr list> : <id list> ) *)
2120        BEGIN
ERROR(36,1,-1);
IF (STACK[STKPTR - 6].DES.SYMLOC.KIND <> TASK) AND
(STACK[STKPTR - 5].DES.SYMLOC.KIND <> FNCTN) THEN
ERROR(24,1,STACK[STKPTR - 4].DES.LINEPOS - CC - 3);
Puts('call',1,0);
END;
46 : (* <MAXLOOPCOUNT> ::= <NUMBER> *)
IF STACK[STKPTR].DES.CHARVAL <> '"' (*)INTEGER*) THEN
ERROR(13,1,-1);
47 : (* <LEFT PART LIST> ::= <NAME> ::= *)
BEGIN
TLI := 1;
END;
48 : (* <LEFT PART LIST> ::= <LEFT PART LIST> <NAME> ::= *)
BEGIN
TLI := TLI + 1;
TEMPLIST[TLI] := STACK[STKPTR - 1].DES.SYMLOC;
END;
49 : (* <ASSIGNMENT STATEMENT> ::= <LEFT PART LIST><EXPRESSION>)
BEGIN
POPEVALSTACK(OPSTORE[2], SELSTORE[2]);
FOR I := 1 TO TLI DO
BEGIN
WITH TEMPLIST[1] DO
CASE KIND OF
  BINARY: SELSTORE[1] := PRECISION2;
  ARITHMETIC: SELSTORE[1] := PRECISION3;
  FNCTN: SELSTORE[1] := PRECISION8;
OTHERWISE BEGIN
  ERROR(32,1,-1);
  SELSTORE[1] := 0;
END;
END;
END;
END;
Puts('assign',2,2);
END;
50 : (* <DATA INPUT> ::= SENSE ( <NAME> ) *)
WITH STACK[STKPTR - 1].DES.SYMLOC DO
CASE KIND OF
  TRANSDEC: IF TYPE0.SYMNAME <> 'INPUT' THEN
    ERROR(20,1,2)
ELSE BEGIN
  OPSTORE[1] := SYMNAME;
  SELSTORE[1] := PRECISION6;
Puts('second

END;

UNDEFINED ERROR(20,1,-2);

OTHERWISE ERROR(20,1,-2);

END;

51:

(*<DATA OUTPUT> := ISSUE (<NAME>) *)

WITH STACK[STKPtr - 1].DES.SYMLOC DO

CASE KIND OF

TRANSDEC: IF TYPE6.SYMNAME <> 'OUTPUT' THEN

ERROR(20,1,-2)

ELSE BEGIN

OPSTORE[1] := SYMNAME;

SELSTORE[1] := PRECISION;

PUTS('issuevent',1,1)

END;

OTHERWISE ERROR(20,1,-2)

END;

52,53,54,55,56,57:

(*<TIME MEASURE> := H/M/S/MS/US/NS *)

STACK[STKPtr].DES.INTVAL := PRODUCTION;

58:

(*<PERIOD> := <NUMBER><TIME MEASURE> *)

BEGIN

IF STACK[STKPtr - 1].DES.CHARVAL = '0' THEN

STACK[STKPtr - 1].DES.REALVAL := STACK[STKPtr - 1].DES.INTVAL;

WITH STACK[STKPtr - 1] DO

CASE STACK[STKPtr].DES.INTVAL OF

(*CONVERT ALL TIMES TO MILLISECONDS *)

52: (*HOURS*) DES.REALVAL := DES.REALVAL * 3600000;

53: (*MINUTES*) DES.REALVAL := DES.REALVAL * 60000;

54: (*SECONDS*) DES.REALVAL := DES.REALVAL * 1000;

55: (*MILLISECONDS*)

56: (*MICROSECONDS*) DES.REALVAL := DES.REALVAL/10000;

57: (*NANOSECONDS*) DES.REALVAL := DES.REALVAL/1000000;

END

59:

(*<TIME> := <PERIOD> *)

60:

(*<TIME> := <TIME><PERIOD> *)

STACK[STKPtr-1].DES.REALVAL := STACK[STKPtr-1].DES.REALVAL +

STACK[STKPtr].DES.REALVAL;

61: (*<TIMED BLOCK> := <TIMEDBLOCKHEAD> ; <STMNT GP>END IM *)

PUTS('n

'0,0);

62:

(*<TIMEDBLOCKHEAD> := IN <PERIOD> *)

BEGIN

SELSTORE[1] := TRUNC(STACK[STKPtr].DES.REALVAL);

PUTS('in

'0,1);

END;
63: (* WAIT ::= WAIT <PERIOD> *)
2226 BEGIN
2227 SELSTORE[1] := TRUNC(STACK[stkptr].DES.REALVAL);
2228 PUTS('fixedwait',0,1);
2229 END;
2230
2231 64: (* WAIT ::= WAIT <EXPRESSION> <PERIOD> *)
2232 BEGIN
2233 POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
2234 SELSTORE[2] := TRUNC(STACK[stkptr].DES.REALVAL);
2235 PUTS('waitexpt',1,2);
2236 IF STACK[stkptr - 2].EXTYPE <> INT THEN
2237 ERROR(38,1,STACK[stkptr - 1].DES.LINEPOS - CC - 3)
2238 END;
2239
2240 65: (* WAIT UNTIL ::= <WAITHEAD><EXPRESSION> <PERIOD> *)
2241 BEGIN
2242 POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
2244 NEWLABEL(OPSTORE[3]);
2245 SELSTORE[2] := TRUNC(STACK[stkptr].DES.REALVAL);
2246 PUTS('boolwait',3,2);
2247 IF STACK[stkptr - 2].EXTYPE <> BOOL THEN
2248 ERROR(26,1,STACK[stkptr - 1].DES.LINEPOS - CC - 3)
2249 END;
2250
2251 66: (* WAITHEAD ::= WAIT UNTIL *)
2252 BEGIN
2253 NEWLABEL(OPSTORE[1]);
2254 PUTS('atboolwait',1,0);
2255 STACK[stkptr - 1].DES.SYNAME := OPSTORE[1]
2256 END
2257
2258 OTHERWISE SEMANTIC(productions);
2259 END (*CASE*)
2260 END; (*SEMANTIC*)
2261
2262 PROCEDURE SEMANTIC (*PRODUCTION : INTEGER *);
2263 VAR P : SYMPTR;
2264 TYPVAR : EXPTYPES;
2265 TEMPHAME : ALFA;
2266 I : INTEGER;
2267 BEGIN
2268 CASE PRODUCTION OF
2269 66: (*<INPUT SPEC> ::= INPUT : <TRANSMISSION BODY> END INPUT*)
2270 BEGIN
2271 FOR I := 1 TO TLI DO
2272 BEGIN
2273 TEMPLIST[I].TYPE := STACK[stkptr - 4].DES.SYMLOC;
OPSTORE[1] := TEMPLIST[I].SYNNAME;
SELSTORE[1] := TEMPLIST[I].PRECISION6;
PUTS('Inputport ',2,1);
PUTSYM('INPUTPORT',2,1)
END;
TLI := 0
END;

87 :(* <OUTPUT SPEC> ::= OUTPUT : *)
(* <TRANSMISSION BODY> END OUTPUT *)
BEGIN (* ASSIGN TYPE = 'OUTPUT' *)
FOR I := 1 TO TLI DO
BEGIN
TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
OPSTORE[1] := TEMPLIST[I].SYNNAME;
SELSTORE[1] := TEMPLIST[I].PRECISION6;
PUTS('outputport ',2,1);
PUTSYM('OUTPUTPORT',2,1)
END;
TLI := 0
END;

94 :(* <DUPLEX SPEC> ::= DUPLEX : *)
(* <TRANSMISSION BODY> END DUPLEX *)
BEGIN (* ASSIGN TYPE = 'DUPLEX' *)
FOR I := 1 TO TLI DO
BEGIN
TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
OPSTORE[1] := TEMPLIST[I].SYNNAME;
SELSTORE[1] := TEMPLIST[I].PRECISION6;
PUTS('in/output',2,1);
PUTSYM('IN/OUTPORT',2,1)
END;
TLI := 0
END;

95 :(* <BINARY SPEC> ::= BINARY : <BINARY BODY> END BINARY *);

96 : (* <ARITHMETIC SPEC> ::= ARITHMETIC : <ARITHMETIC BODY> END ARITHMETIC *);

99 : (* <TRANSMISSION DEC> ::= <ID> , <BINARY PRECISION> , <TECHNOLOGY> *)
BEGIN
PTR := STACK[STKPTR - 4].DES.SYMLOC;
IF PTR.KIND <> UNDEFINED THEN
ERROR(14,1,STACK[STKPTR - 3].DES.INTVAL - 1);
PTR.KIND := TRANDEC;
PTR.PRECISION6 := STACK[STKPTR - 2].DES.INTVAL;
PTR.TECHNOLOGY6 := STACK[STKPTR].DES.SYMLOC
END;

102 : (* <BINARY DEC> ::= <ID><STRUCTURE> , *)
    (* <BINARY PRECISION> <INITIAL VALUE> *)
BEGIN
    PTR := STACK(STKPTR - 4).DES.SYMLOC;
    IF PTR.KIND <> UNDEFINED THEN
        ERROR(14,1.0);
    PTR.KIND := BINARY;
    PTR.PRECISION2 := STACK(STKPTR - 1).DES.INTVAL;
    IF STACK(STKPTR).DES.CHARVAL = '0' THEN
        PTR.IVAL2 := 0
    ELSE
        PTR.IVAL2 := STACK(STKPTR).DES.INTVAL
    END;

105 : (* <ARITHMETIC DEC> ::= <ID><STRUCTURE> , *)
    (* <DECIMAL PRECISION> <INITIAL VALUE> *)
BEGIN
    PTR := STACK(STKPTR - 4).DES.SYMLOC;
    IF PTR.KIND <> UNDEFINED THEN
        ERROR(14,1.0);
    PTR.KIND := ARITHMETIC;
    PTR.PRECISION3 := STACK(STKPTR - 1).DES.INTVAL;
    IF STACK(STKPTR).DES.CHARVAL = '0' THEN
        PTR.IVAL3 := 0
    ELSE
        PTR.IVAL3 := STACK(STKPTR).DES.INTVAL;
    OPSTORE[1] := PTR.SYMNAME;
    SELSTORE[1] := PTR.PRECISION3;
    SELSTORE[2] := PTR.IVAL3;
    PUTS('var ',1,2);
    PUTSYM('VARIABLE ',1,2)
END;

106 : (* <STRUCTURE> ::= *)

107 : (* <STRUCTURE> ::= ( <NUMBER LIST> ) *)
    ERROR(36,1,-1);

108 : (* <NUMBER LIST> ::= *NUMBER* *)

109 : (* <NUMBER LIST> ::= <NUMBER LIST> , *NUMBER* *)
BEGIN
    IF STACK(STKPTR).DES.CHARVAL <> '0' THEN
        ERROR(18,1,-1);
    NL := NL + 1;
    SELSTORE[NL] := STACK(STKPTR).DES.INTVAL;
    IF PRODUCTION = 111 THEN
        STACK(STKPTR).DES.INTVAL := NL
    ELSE
        STACK(STKPTR - 2).DES.INTVAL := NL
END;
((<binary precision> ::= <number> *))

110:

(* <binary precision> ::= <number> *)

IF STACK[STKPTR].DES.CHARVAL <= '0' THEN

ERROR(13,1,-1);

111:

(* <decimal precision> ::= <number> *)

IF STACK[STKPTR].DES.CHARVAL <= '0' THEN

ERROR(10,1,-1);

112:

(* <initial value> *)

STACK[STKPTR + 1].DES.CHARVAL := '"';

STACK[STKPTR].DES.INTVAL := STACK[STKPTR].DES.INTVAL;

113:

(* <initial value> ::= <number> *)

IF STACK[STKPTR].DES.CHARVAL = '0' THEN

BEGIN

STACK[STKPTR - 1].DES.CHARVAL := '"';

STACK[STKPTR - 1].DES.INTVAL := STACK[STKPTR].DES.INTVAL;

END

114,115,116:

(* <technology> ::= TTL/ECL/IIL *)

117:

(* <code spec> ::= code ; *id* , <binary precision> ;
<character rep list> end code *)

118:

BEGIN

FOR I := 1 TO TLI DO

TEMPLIST[I].COIDS := STACK[STKPTR - 6].DES.SYMLOC;

TLI := 0;

END;

119:

BEGIN

TLI := TLI + 1;

TEMPLIST[TLI] := STACK[STKPTR - 1].DES.SYMLOC

END;

120:

BEGIN

TLI := TLI + 1;

TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC;

TLI := 1;

END;

121:

(* <id list> ::= *

(TLI := TLI + 1;

TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC)

122:

(* <idlist> ::= <id> *)

BEGIN

TEMPLIST[1] := STACK[STKPTR].DES.SYMLOC;

TLI := 1;

END;

123:

BEGIN

TLI := TLI + 1;

TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC

124:

BEGIN

TLI := TLI + 1;

TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC

125,126,127,128,129:

(* <code id> ::= <id>/ASCII6/

ASCII7/EBCDIC/BCD *)

130:

(* <id list> ::= *

(TLI := TLI + 1;

TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC)

131:

(* <idlist> ::= <id> *)

BEGIN

TEMPLIST[1] := STACK[STKPTR].DES.SYMLOC;

TLI := 1;

END;

132:

(* <id list> ::= <idlist> , <id> *)

BEGIN

TLI := TLI + 1;

TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC
133 :  (* <NAME> ::= <ID> *)
134 :  (* <NAME> ::= <ID> (<EXPRLIST>) *)
BEGIN
STACK[STKPTR - 3].DES.CHARVAL := 'A'; (*ARRAY*)
ERROR(36,1,-1)
END;
135 :
BEGIN
STACK[STKPTR - 5].DES.CHARVAL := 'B'; (*BIT FIELD*)
ERROR(37,1,-1)
END;
136 :  (* <FORMAL PARAM LIST> ::= *)
FIRSTPARAM := NIL;
137 :  (* <FORMAL PARAM LIST> ::= ( <ID LIST> ; <ID LIST> ) *)
ERROR(33,1,-1)
138 :  (* <PROC> ::= <TASK> *)
139 :  (* <PROC> ::= <FUNCTION> *)
140 :  (* <TASK> ::= <TASK HEAD> ; <ZOPT PROC DEC GP>
<STMT GP> END <ID> *)
146 :  (* <FUNCTION> ::= <FUNCTION HEAD> ; <ZOPT PROC DEC GP>
<STMT GP> END <ID> *)
BEGIN
IF STACK[STKPTR - 5].DES.SYMLOC <> STACK[STKPTR].DES.SYMLOC
THEN ERROR(26,1,-1);
PUTS('x1tproc ',1,0)
END;
147 :  (* <FUNCTION HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST>
BINARY , <BINARY PRECISION><INITIAL VALUE> *)
148 :  (* <FUNCTION HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST>
ARITHMETIC , <DECIMAL PRECISION><INITIAL VALUE> *)
BEGIN
PUTT(STACK[STKPTR - 6].DES.SYMNAME);
OPSTORE[1] := STACK[STKPTR - 6].DES.SYMNAME;
PUTS('proc ',1,0);
WITH STACK[STKPTR - 6].DES.SYMLOC DO
BEGIN
KIND := FNCTN;
PARAMLIST8 := FIRSTPARAM;
TYPE8 := STACK[STKPTR - 3].DES.SYMLOC;
PRECISION8 := STACK[STKPTR - 1].DES.INTVAL;
END;
IVALB := STACK[STKPTR].DES.INTVAL
END;
END;

145 : (* <TASK HEAD> := TASK *ID* <FORMAL PARAM LIST> *)
BEGIN
PUT(STACK[STKPTR - 1].DES.SYNAME);
PUTC("proc", 1, 0);
WITH STACK[STKPTR - 1].DES.SYMLOC DO
BEGIN
KIND := TASK;
PARAMLIST := FIRSTPARAM
END;
STACK[STKPTR - 2] := STACK[STKPTR - 1]
END;

156 : (* <RANK> ::= <NUM> *)
STACK[STKPTR].DES.LINEPOS := STACK[STKPTR].DES.INTVAL;

157 : (* <RANK> ::= <NUM>, <PI> *)
BEGIN
STACK[STKPTR - 2].DES.LINEPOS :=
STACK[STKPTR - 1].DES.INTVAL;
STACK[STKPTR - 2].DES.INTVAL := STACK[STKPTR].DES.INTVAL
END;

158 : (* <NUM> ::= <NUMBER> *)
159 : (* <PI> ::= <NUMBER> *)
IF STACK[STKPTR].DES.CHARVAL <> '0' THEN
ERROR(13,1,-1);

160 : (* <QUALIFICATION> ::= *)
STACK[STKPTR + 1].DES.CHARVAL := '0';

161 : (* <QUALIFICATION> ::= IF <EXPRESSION> *)
BEGIN
STACK[STKPTR - 1].DES.CHARVAL := 'Q';
IF STACK[STKPTR].EXPTYPE <> BOOL THEN
ERROR(26,1,-1);
POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
NEWLABEL(OPSTORE[2]);
PUTC('jmpf',2,1);
END;

162,163,164,165,166 : (* <EPISODE TIMING> ::= / <ROE> /
: <ROE>, <B1> /: <ROE>, <B1>, <B2> /
: <ROE>, <B1> , <B2> , <RANK> *)
BEGIN
FOR I := 1 TO 5 DO
SELSTORE[1] := 0;
CASE PRODUCTION OF
162 : 
164 : BEGIN
END;
END
END
END
END
167 : (*WHEN DO* := <QUALIFICATION> WHEN <NAME>
168 : <EPISODE TIMING> DO <TASK LIST>*)
169 : BEGIN
170 : WITH STACK[STKPTR - 3].DES.SYMLOC DO
171 : CASE KIND OF
172 : UNDEFINED : ERROR(28,1,STACK[STKPTR - 4].DES.LINEPOS-CC+2);
173 : BINARY :=
174 : FNCTN : IF TYPEB.SYMNAME <> 'BINARY ' THEN
175 : ERROR(30,1,STACK[STKPTR-4].DES.LINEPOS-CC+2);
176 : OTHERWISE ERROR(30,1,STACK[STKPTR-4].DES.LINEPOS-CC)
END;
177 : FOR I := 1 TO OPI DO
178 : PUTA(STACK[STKPTR-3].DES.SYMNAME,OPSTORE[I],5);
179 : IF STACK[STKPTR-5].DES.CHARVAL = 'Q' THEN
180 : BEGIN
181 : OPSTORE[I] := STACK[STKPTR-5].DES.SYMNAME;
182 : PUTFV('L0C',,1,0);
183 : PUTFV('LOC',,1,0)
184 : END
185 : FOR I := 1 TO OPI DO
186 : PUTA('OPSTORE[I],5);
187 : IF STACK[STKPTR-3].DES.CHARVAL = 'Q' THEN
188 : BEGIN
189 : SELSTORE[1] := 0;
190 : FOR I := 1 TO OPI DO
191 : PUTA('OPSTORE[I],5);
192 : IF STACK[STKPTR-3].DES.CHARVAL = 'Q' THEN
193 : BEGIN
194 :
PUTS(’loc’, 1, 0)
END
END;
169. (* <EVERY> ::= <QUALIFICATION> EVERY <ROE>
DO <TASK LIST> *)
170 : (* <AT TIME> ::= <QUALIFICATION> AT <TIME>
DO <TASK LIST> *)
BEGIN
FOR I := 1 TO OPI DO
PUTA(’.’, OPSTORE[1], 5);
IF STACK[STKPTR - 3].DES_CHARVAL = ’Q’ THEN
BEGIN
PUTS(’loc’, 1, 0)
END
END;
END;
171 : (* <TASK LIST> ::= <NAME> *)
BEGIN
WITH STACK[STKPTR].DES_SYMLOC DO
CASE KIND OF
UNDEFINED : ERROR(28, 1, -1);
TASK : ;
FNCTN : ;
ELSE ERROR(34, 1, -1)
END;
OPI := 1
END;
END;
END;
172 : (* <TASK LIST> ::= <TASK LIST> THEN <NAME> *)
BEGIN
WITH STACK[STKPTR].DES_SYMLOC DO
CASE KIND OF
UNDEFINED : ERROR(28, 1, -1);
TASK : ;
FNCTN : ;
ELSE ERROR(34, 1, -1)
END;
OPI := OPI + 1;
OPSTORE[OPI] := STACK[STKPTR].DES_SYMNAME
END;
END;
182 : (* <DESIGN CRITERIA> ::= DESIGN CRITERIA
METRIC <METRIC> ; VOLUMES <NUMBER LIST> ;

BEGIN

PUTD(STACK[STKPTR-7].DES.SYNAME, STACK[STKPTR-4].DES.INTVAL,
      STACK[STKPTR-1].DES.INTVAL-STACK[STKPTR-4].DES.INTVAL);
M1I := 0
END;

183, 184, 185 : (* <METIRC> ::= FIRST/COST/POWER *);

190 : (* PRINT ALL CONSTANTS USED DURING COMPILATION *)

BEGIN
  PUTT(' SYSTEM ');
  FOR I := 1 TO CSI DO
    BEGIN
      OPSTORE[1] := CONSTANTSTORE[I].NAME;
      SELSTORE[1] := CONSTANTSTORE[I].VAL;
      PUTS('cons ', '1,2);
      PUTF('CONS ', '1,2)
    END;
    PRINTEMPS
  END;
END; (*CASE PRODUCTION OF *)

BEGIN (*MAIN*)

INITIALIZE;
PUTT(' SYSTEM ');
PUTS('MAIN ', '0,0);
PARSE;

(*THE BELOW STMTS CHECK TO SEE IF TOGGLES HAVE*)
(*BEEN TURNED ON TO TRACE THE PARSE EXECUTION *)
(*AND CALLS THE PROCEDURE TO PRINT THE DETAILS*)

99: IF LINERRPR < 0 THEN PRINTLINERRORS;
    IF PROGRAMERFLAG THEN PRINTERRORS;
    IF SWITCH[PRINTABLE] THEN GOPRINTABLE
END.
APPENDIX

CSDL TRANSLATOR
NAVAL POSTGRADUATE SCHOOL
18-APR-1981:17:03.1

1 -- PRINTABLE

2

3 IDENTIFICATION

4 DESIGNER : "ALAN ROSS"

5 DATE : "12-28-63"

6 PROJECT : "DUAL PROCESS CONTROL APPLICATION"

7

8 DESIGN CRITERIA

9 METRIC FIRST:

10 VOLUMES 0:

11 MONITORS 0:

12

13 ENVIRONMENT

14

15 INPUT : CONIN,0,TTL ; CONST,0,TTL ; FLGA,1,TTL ;

16 PINA,0,TTL ; FLSB,1,TTL ;

17 PINB,0,TTL ; END INPUT ;

18

19 OUTPUT : VA,0,TTL ; VB,0,TTL ; END OUTPUT ;

20

21 ARITHMETIC : KCA,0 ; KCB,0 ; CNTB,0 ; ITIA,0 ; ITIB,0 ; AINT,0 ; TDA,0 ;

22 TDA',0 ; DIN,0 ; VSA,0 ; VSB,0 ; BDIFF,0 ; PSA,0 ; PSB,0 ; COMPT,0 ;

23 EA 3 ; EB,0 ; KPIA,0 ; EA1,0 ; EA2,0 ; EB1,0 ; EB2,0 ; KP1B,0 ;

24 END ARITHMETIC ;

25

26 PROCEDURES

27

28 FUNCTION DATA:

29

30 FUNCTION (GABA:

31 IF FLGA = 1 THEN DATA := 1; END IF ;

32 END DATA ;

33

34 FUNCTION DAB:

35

36 FUNCTION (GLGB);

37 IF FLGB = 1 THEN DAB := 1; END IF ;

38 END DAB ;

39

40 FUNCTION BCNT:

41

42 IF CNTB >= 4 THEN BCNT := 1; END IF ;

43 END BCNT ;

44

45 TASK AFIX:

46 ARITHMETIC : ADIFF,0 ; END ARITHMETIC ;

47

48

49

50
CSDL TRANSLATOR
NAVAL POSTGRADUATE SCHOOL
18-APR-1981:17:03:1
51 SENSE (PINA);
52 EA := PINA*KCA - PSA;
53 ADIFF := (3*EA - 4*EA1 + EA2)*5;
54 AINT := AINT + EA/KCA;
55 VA := VSA + KCA*(EA + ITIA*AINI + TDA*ADIFF);
56 ISSUE (VA);
57 DATA2 := 0;
58 EA2 := EA1;
59 EA1 := EA;
60 END AFIx;
61
62 TASK BCALC;
63 SENSE (PINE);
64 EB := PINB*KCB - PSB;
65 BDIFF := (3*EB - 4*EB1 + EB2)*10;
66 BINT := BINT + EB/KCB;
67 CNTB := CNTB + 1;
68 DATABASE := 0;
69 END BCALC;
70
71 TASK BFIX;
72 CNTB := 0;
73 VB := VSB + KCB*(EB + ITIB*BINT + TDB*BDIFF);
74 ISSUE (VB);
75 END BFIX;
76
77 FUNCTION CONFLG:
78 BINARY,;
79 SENSE (CONSIN);
80 IF CONSIN > 0 THEN CONFLG := 0; END IF;
81 END CONFLG;
82
83 TASK CMGCON;
84 SENSE (CONST);
85 IF CNGPPT = 1 THEN KCA := CONST; END IF;
86 IF CNGPPT = 2 THEN ITIA := 1/CONST; END IF;
87 IF CNGPPT = 3 THEN TDA := CONST; END IF;
88 IF CNGPPT = 4 THEN VSA := CONST; END IF;
89 IF CNGPPT = 5 THEN PSA := CONST; END IF;
90 IF CNGPPT = 6 THEN AINT := CONST; END IF;
91 IF CNGPPT = 7 THEN KCB := CONST; END IF;
92 IF CNGPPT = 8 THEN ITIB := 1/CONST; END IF;
93 IF CNGPPT = 9 THEN TDB := CONST; END IF;
94 IF CNGPPT = 10 THEN VSB := CONST; END IF;
95 IF CNGPPT = 11 THEN PSB := CONST; END IF;
96 IF CNGPPT = 12 THEN BINT := CONST; END IF;
97 END CMGCON;
100 CONTINGENCY LIST

CSOL TRANSLATOR
NAVAL POSTGRADUATE SCHOOL
18-APR-1981 17:03:1

101 WHEN DATAA;100 MS DO AFIX1;
102 WHEN DATAB; 50 MS DO BCALC1;
103 WHEN BCNT; 100 MS DO BFIX1;
104 WHEN CONFG DO CHGCON1;
105
106 END

FUNCTION RESWD 48
2
DATAB FUNCTION BINARY 1
M RESWD 57
3
DESIGN RESWD 36
4
5
6
THEN RESWD 72
7
PROJECT RESWD 67
8
9
BCNT FUNCTION BINARY 1
TERM RESWD 71
10
DATAA FUNCTION BINARY 1
FOR RESWD 48
11
ARITHMETIC RESWD 25
12
AND RESWD 24
13
PRINTABLE UNDEFINED
14
EA ARITHMETIC 8 0
VSB ARITHMETIC 8 0
15
EBCDIC CODE RESWD 40
16
CHGCON TASK
17
VSA ARITHMETIC 8 0
VOLUMES RESWD 78
18
MS RESWD 80
19
UNTIL RESWD 75
KCB ARITHMETIC 8 0
TDL RESWD 74
<table>
<thead>
<tr>
<th>TASK</th>
<th>RESWD</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER</td>
<td>RESWD</td>
<td>65</td>
</tr>
<tr>
<td>ISSUE</td>
<td>RESWD</td>
<td>55</td>
</tr>
<tr>
<td>KCA</td>
<td>ARITHMETIC</td>
<td>8</td>
</tr>
<tr>
<td>AT</td>
<td>RESWD</td>
<td>28</td>
</tr>
<tr>
<td>BINT</td>
<td>ARITHMETIC</td>
<td>8</td>
</tr>
<tr>
<td>PINB</td>
<td>TRANSDEC INPUT</td>
<td>TTL</td>
</tr>
<tr>
<td>PINA</td>
<td>TRANSDEC INPUT</td>
<td>TTL</td>
</tr>
<tr>
<td>WHILE</td>
<td>RESWD</td>
<td>81</td>
</tr>
<tr>
<td>CONTINGENCY</td>
<td>RESWD</td>
<td>32</td>
</tr>
<tr>
<td>TDB</td>
<td>ARITHMETIC</td>
<td>8</td>
</tr>
<tr>
<td>WHEN</td>
<td>RESWD</td>
<td>80</td>
</tr>
<tr>
<td>EVERY</td>
<td>RESWD</td>
<td>44</td>
</tr>
<tr>
<td>BINARY</td>
<td>RESWD</td>
<td>30</td>
</tr>
<tr>
<td>TDA</td>
<td>ARITHMETIC</td>
<td>8</td>
</tr>
<tr>
<td>CDIFF</td>
<td>ARITHMETIC</td>
<td>8</td>
</tr>
<tr>
<td>IIL</td>
<td>RESWD</td>
<td>52</td>
</tr>
<tr>
<td>BCD</td>
<td>RESWD</td>
<td>29</td>
</tr>
<tr>
<td>CEB</td>
<td>ARITHMETIC</td>
<td>8</td>
</tr>
<tr>
<td>CISMINT</td>
<td>TRANSDEC INPUT</td>
<td>TTL</td>
</tr>
<tr>
<td>DATE</td>
<td>RESWD</td>
<td>35</td>
</tr>
<tr>
<td>AFIX</td>
<td>TASK</td>
<td>73</td>
</tr>
<tr>
<td>TO</td>
<td>RESWD</td>
<td></td>
</tr>
<tr>
<td>ECL</td>
<td>RESWD</td>
<td>41</td>
</tr>
<tr>
<td>VA</td>
<td>TRANSDEC OUTPUT</td>
<td>TTL</td>
</tr>
<tr>
<td>LIST</td>
<td>RESWD</td>
<td>56</td>
</tr>
<tr>
<td>NOT</td>
<td>RESWD</td>
<td>61</td>
</tr>
<tr>
<td>METRIC</td>
<td>RESWD</td>
<td>58</td>
</tr>
<tr>
<td>IN</td>
<td>RESWD</td>
<td>53</td>
</tr>
<tr>
<td>IDENTIFICA</td>
<td>RESWD</td>
<td>50</td>
</tr>
<tr>
<td>S</td>
<td>RESWD</td>
<td>68</td>
</tr>
<tr>
<td>DO</td>
<td>RESWD</td>
<td>36</td>
</tr>
<tr>
<td>CNTB</td>
<td>ARITHMETIC</td>
<td>8</td>
</tr>
<tr>
<td>MONITORS</td>
<td>RESWD</td>
<td>50</td>
</tr>
<tr>
<td>FIRST</td>
<td>RESWD</td>
<td>45</td>
</tr>
<tr>
<td>DUPEX</td>
<td>RESWD</td>
<td>39</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>----</td>
</tr>
<tr>
<td>PSB</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>ENVIRONMEN</td>
<td>RESWD 43</td>
<td></td>
</tr>
<tr>
<td>ASCII7</td>
<td>RESWD 27</td>
<td></td>
</tr>
<tr>
<td>ASCII8</td>
<td>RESWD 26</td>
<td></td>
</tr>
<tr>
<td>COMPTT</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>PSA</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>JTB</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>ITIA</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>CONST</td>
<td>TRANSDEC INPUT TTL 0</td>
<td></td>
</tr>
<tr>
<td>VB</td>
<td>TRANSDEC OUTPUT TTL 0</td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td>RESWD 64</td>
<td></td>
</tr>
<tr>
<td>PROCEDURES</td>
<td>RESWD 66</td>
<td></td>
</tr>
<tr>
<td>KPIB</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>KPIA</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>DESIGNER</td>
<td>RESWD 37</td>
<td></td>
</tr>
<tr>
<td>EA2</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>EA1</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>FLGB</td>
<td>TRANSDEC INPUT TTL 1</td>
<td></td>
</tr>
<tr>
<td>FLGA</td>
<td>TRANSDEC INPUT TTL 1</td>
<td></td>
</tr>
<tr>
<td>WAIT</td>
<td>RESWD 79</td>
<td></td>
</tr>
<tr>
<td>FROM</td>
<td>RESWD 47</td>
<td></td>
</tr>
<tr>
<td>COST</td>
<td>RESWD 33</td>
<td></td>
</tr>
<tr>
<td>BFIX</td>
<td>TASK</td>
<td></td>
</tr>
<tr>
<td>BCALC</td>
<td>TASK</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>RESWD 49</td>
<td></td>
</tr>
<tr>
<td>VARIABLES</td>
<td>RESWD 77</td>
<td></td>
</tr>
<tr>
<td>SENSE</td>
<td>RESWD 69</td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>RESWD 42</td>
<td></td>
</tr>
<tr>
<td>AINT</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>RESWD 63</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>RESWD 51</td>
<td></td>
</tr>
<tr>
<td>CONFLG</td>
<td>FUNCTION BINARY 1</td>
<td></td>
</tr>
<tr>
<td>EB2</td>
<td>ARITHMETIC 8 0</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX F
SYMBOL TABLE

S. INPUTPORT(CONSM, TTL:8)
S. INPUTPORT(CONSTM, TTL:8)
S. INPUTPORT(LFLA, TTL:1)
S. INPUTPORT(PINA, TTL:8)
S. INPUTPORT(PINB, TTL:8)
S. OUTPUTPORT(VA, TTL:8)
S. OUTPUTPORT(VB, TTL:8)
S. VARIABLE (KCA:8,0)
S. VARIABLE (KCB:8,0)
S. VARIABLE (CNTB:8,0)
S. VARIABLE (ITIB:8,0)
S. VARIABLE (ITT:8,0)
S. VARIABLE (ATT:8,0)
S. VARIABLE (TOA:8,0)
S. VARIABLE (TOB:8,0)
S. VARIABLE (BIN:8,0)
S. VARIABLE (VSA:8,0)
S. VARIABLE (VSB:8,0)
S. VARIABLE (BOFF:8,0)
S. VARIABLE (PSA:8,0)
S. VARIABLE (PSB:8,0)
S. VARIABLE (COMPT:8,0)
S. VARIABLE (EA:8,0)
S. VARIABLE (EB:8,0)
S. VARIABLE (KIPI:8,0)
S. VARIABLE (EA1:8,0)
S. VARIABLE (EA2:8,0)
S. VARIABLE (EB1:8,0)
S. VARIABLE (EB2:8,0)
S. LOC (e01:)
S. LOC (e02:)
S. LOC (e03:)
S. VARIABLE (ADIFF:8,0)
S. LOC (e04:)
S. LOC (e05:)
S. LOC (e06:)
S. LOC (e07:)
S. LOC (e08:)
S. LOC (e09:)
S. LOC (e10:)
S. LOC (e11:)
S. LOC (e12:)
S. LOC (e13:)
S. LOC (e14:)
S. LOC (e15:)
S. LOC (e16:)
S. CONS (eC01:1,8)
S. CONS (eC02:4,8)
S. CONS (eC03:3,8)
S. CONS (eC04:5,8)
S. CONS (eC05:0,8)
LIST OF REFERENCES


**INITIAL DISTRIBUTION LIST**

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Distribution List</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>2</td>
<td>Defense Technical Information Center</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cameron Station</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alexandria, Virginia 22314</td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>Dudley Knox Library, Code 0142</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>3.</td>
<td>1</td>
<td>Department Chairman, Code 52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Department of Computer Science</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>4.</td>
<td>1</td>
<td>Office of Research Administration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code 012A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>5.</td>
<td>1</td>
<td>Computer Technologies Curricular Office</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code 37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>6.</td>
<td>2</td>
<td>Thomas H. Carson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>458 Papaya Court</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jacksonville, Florida 32225</td>
</tr>
<tr>
<td>7.</td>
<td>3</td>
<td>LtCol. Alan A. Ross, USAF, Code 52Rs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Department of Computer Science</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
</tbody>
</table>

120