/A PROGRAM DESIGN LANGUAGE FOR COBOL/

BY

Robert Shih-pei Chou

B.A., Tunghai University, 1973

A MASTER'S REPORT

submitted in partial fulfillment of the

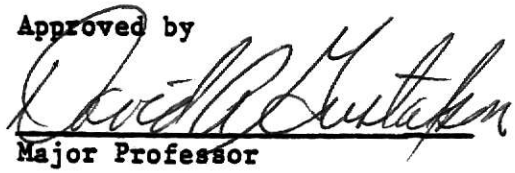
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1985

Approved by



Major Professor

LD
2668
.R4
1985
C46
c.2

A11202 996143

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Dr. David A. Gustafson, for his constant advise and assistance during the writing of this study. I also wish to express my gratitude to the members of my graduate committee, Dr. Virgil E. Wallentine and Dr. William J. Hankley, for their helps and guidances in the completion of my work.

Especial thanks go to my parents, Mr. and Mrs. Hon-Wan Chou, for their encouragement and support during my graduate study.

Finally, I wish to thank my family, especially my wife, Yu-Jung, for her understanding, encouraging love and dedicated assistance during the past several years. I thank my lovely son, Danny, for his understanding and patience during the past years.

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

TABLE OF CONTENTS

Page

Chapter 1: INTRODUCTION AND MOTIVATION 1

1.1. Structured programming 1

1.2. COBOL in structured programming 4

1.3. The Program Design Language(PDL) 5

Chapter 2: DESIGN PHASE 9

2.1. Definitions of the Program Design Language 9

2.2. The equivalent COBOL codes of each PDL statement 11

2.2.1. The control flow of the PDL control structures 12

2.2.2. The standard COBOL modules for
the PDL control structures 13

2.2.3. The equivalence between PDL and COBOL 16

2.3. Translation design 18

2.3.1. The translation semantics and the translation
instructions of each PDL statement 19

2.3.2. The execution flag of the translator 21

2.3.3. The software techniques used
to design this translator 21

Chapter 3: IMPLEMENTATION 34

3.1. The translation outline 35

3.1.1. The data transfer between
the translator and its peripheral 36

3.1.2. The data flow within the translator 36

TABLE OF CONTENTS (CONTINUED)

Page

3.1.3. The "stack" data structure 38

3.2. The translation tables 39

3.2.1. The translation instructions 39

3.2.2. The construction of the translation tables ... 41

3.3. The translator and its performances 43

3.3.1. The flags 43

3.3.2. Data flow 44

3.3.3. The results of test runs 46

Chapter 4: CONCLUSION 48

REFERENCES 51

APPENDIX A: USER MANUAL 52

1. Keywords, Delimiters, and Symbols of PDL 52

1.01. The PDL keywords 53

1.02. The PDL delimiters 53

1.03. The valid characters used in PDL statements ... 53

2. The translation mode of PDL 54

2.01. Translation mode of the Kall statement 55

2.02. Translation mode of the SET-ON statement 55

2.03. Translation mode of the SET-OFF statement 56

2.04. Translation mode of the ENABLE statement 56

2.05. Translation mode of the DISABLE statement 57

TABLE OF CONTENTS (CONTINUED)

Page

2.06. Translation mode of th IFF structure 58

2.07. Translation mode of the CASE structure 59

2.08. Translation mode of the CASE88 structure 61

2.09. Translation mode of the WHILE structure 63

2.10. Translation mode of the WHILENOT structure 64

2.11. Translation mode of the REPEAT structure 65

3. The limitations and other coding restrictions 66

APPENDIX B: A Comparison listing of PDL statements and their COBOL equivalences with the translation semantics and the translation instructions 67

APPENDIX C: Module Specification and storage requirement of the translator 86

1. Module specification 86

2. The storage requirement 88

APPENDIX D: 90

1. A listing of all the translation instructions 91

2. The text of the TABLE BUILDER program 95

3. The contents of the command table 97

4. The contents of the address table 99

5. The contents of the string table 100

TABLE OF CONTENTS (CONTINUED)

Page

6. A listing of dumping the variables of the TABLE BUILDER program during converting the translation instructions into the three tables ... 101

APPENDIX E: 106

1. The text of the translator program - PREP 107
2. A sample PDL text 115
3. The output code of the sample PDL text 116
4. A listing of variable contents of the PREP program during translating the sample PDL text 117

LIST OF FIGURES

Figure		Page
1-1	Sequence Control Structure	7
1-2	Selection Control Structure	7
1-3-1	Iteration Control Structure "REPEAT-UNTIL"	7
1-3-2	Iteration Control Structure "WHILE-ENDWHILE"	7
1-4	Three Approaches to Structured Programming	8
2-1	The Syntax of PDL Statements	25
2-2	The PDL Control Structures with Their Flow Chart and the Designed COBOL Equivalences	26
2-2-1	The REPEAT-UNTIL Structure	26
2-2-2	The WHILE-ENDWHILE Structure	27
2-2-3	The WHILENOT-ENDWHILENOT Structure	27
2-2-4	The IFF-ELSE-ENDIFF structure	28
2-2-5	The CASE-selections-ENDCASE Structure	29
2-2-6	The CASE88-selections-ENDCASE88 Structure	30
2-3	The Selection Structure with a Null Internal Structure	31
2-4	Control Flow Shown on a COBOL Procedure Division	32
2-5	The Outline of the Translation	33
2-6	How the "Stack" Data Structure Manage the Translation of the Nesting Control Structures ...	33
3-1	Data Flow within the Translator - the PREP Program ..	47
3-2	Data Flow within the TRANS Program.....	47

A PROGRAM DESIGN LANGUAGE FOR COBOL

Chapter 1:

INTRODUCTION AND MOTIVATION

The practice of structured programming increases the productivity of programmers and the reliablility of computer programs. However, the lack of adequate control structures in COBOL makes it hard for the user of COBOL to apply the structured programming technique. To make the structured programming task easier in COBOL, adequate control structures are required. The purpose of the present study is to develop a generalized Program Design Language (PDL) by adding these control structures to COBOL. This PDL will greatly enhance the structured programming capability in COBOL.

1.1. Structured Programming

Structured programming improves the design and maintenance of computer programs. Structured programming provides a method of organizing and coding programs that makes the programs easily understood and modified [Donaldson-73]. It converts arbitrarily large and complex flowcharts into standard forms so that they can be represented by iteration and nesting of a small number of basic and standard control logic structures [Mills-72].

From a theoretical view, the basic control structures, introduced by Bohm and Jacopini, are the sequence, the selection, and the iteration structures (see figure 1-1 through 1-3) [Bohm-66]. Normally, selection statements include IF-THEN-ELSE and CASE statements. The iteration structures include the 'WHILE-ENDWHILE' and the 'REPEAT-UNTIL' structures and 'FOR' statements. (For statements are not included in the COBOL PDL). Each control structure is shown in the figures as an one-entrance-one-exit "box". The control processing of these boxes is defined by following the arrows. All these arrows either obey the sequential processing rule or are decided by checking a given condition.

Based on Wirth's concept [Wirth-71] of "stepwise refinement", structured programming can be understood as the application of a basic problem decomposition method to establish a manageable hierarchical problem structure. The highest conceptual level represents the general description of the problem, with each lower level providing greater magnification or additional problem detail. The process of refinement is carried out in a series of steps, with each step bringing greater problem detail into focus or refining the knowledge available in the preceding step [Jensen-81].

In terms of the control structures, the concept of "stepwise refinement" can be described as a processing which defines any rectangular box within a control structure by another control structure. The solution to a given problem can be any one of the structures, and it can be further described by

refining the internal boxes. This refinement will be terminated when the internal boxes can be easily represented by the instructions of a computer language.

There have been a lot of discussions on the meaning of "structured programming", but we are interested in its function and its advantages. Other topics of debate about its meaning include: "If structured programming is NON-GOTO", "If sturctured programming is a panacea for all problems", and "If more controls are needed by the control structures - more entrance and exit points of a control structures", etc. These debates have subsided with some degree of consensus reached.

The advantage of structured programming is obvious in improving the development and maintenance of computer programs. Most of the computer languages are so different from the natural languages that it is not easy for a computer programmer to make himself clear. The structured programming technique, by providing organized procedures, helps the programmers to have a better methodology and the users a better approach to understand an existing program. These advantages can reduce the possibilities of logic errors and of misunderstanding a computer program. Thus, the cost of software development can be reduced.

1.2. COBOL in Structured Programming

COBOL is too wordy in program development and too limited in its control structures structured programming. COBOL is a format-oriented language, its coding format creates inconvenience to the programmers. There are no easy coding options available for writing a precise and concise program. To focus on the wordy and coding problems, the "Shorthand Programming Language in a COBOL Environment" is a COBOL pre-compiler which reduces the initial coding of all COBOL programs by at least 50 percent [Crawford-74].

There are various methods to "reform" the unstructured programming language for the practice of structured programming. In summary, they are "imitation", "design", and "preprocessing". [presser-75] (See figure 1-4). The "imitation" approach is to appropriately place the existing language statements to form program texts which resemble the three control structures. The second method is to "design" a new language to replace the existing language. The "preprocessing" technique is to add the desired control statements to the existing language, and a preprocessor is used to convert the new statements into the existing language.

The preprocessing approach is the simplest and most flexible solution to make structured programming available for COBOL. The preprocessor statements can be used either to substitute the

wordy COBOL statements or to build up control structures on the existing COBOL statements. An ad-hoc translator has to be designed to convert the preprocessor statements into equivalent COBOl statements.

By keeping the original linguistic syntax, the existing COBOL can be embedded as the body of the control structures. The systems analysts and programmers only have to know the syntax of the added preprocessor statements. In translating the preprocessor statements, the embedded COBOL text will be simply copied as the output code.

The number of added preprocessor statements and their syntax can be easily altered to meet the user's demand. Besides, the type of the equivalent COBOL translated from the preprocessor statements can also be altered. The preprocessing technique is to create a friendly and flexible interface between COBOL and its user.

1.3. The Program Design Language

This paper is to demonstrate a type of Program Design Language (PDL) for COBOL and the translation process. The PDL statements include five PDL preprocessor statements to substitute the wordy COBOL and fourteen control statements to build up PDL control structures. Original COBOL statements are totally accepted as part of PDL without changing their syntax or

limitations.

The control structures of PDL will not include the "sequence" control structure of the three basic control structures. The control structures are the various structures of the "selection" and "iteration" structures. The "sequence" structure is eliminated from this PDL, since the sequential processing manner obeys the machine nature. Any "box" in figure 1-1 through figure 1-3-2, if it contains one or more structures, is basically a "sequence box". To augment the various types of the "selection" and "iteration" structures, the PDL will provide more convenience in expressing the program logic.

The translator of this PDL is required to be flexible, to be free-formated, and to be coded in PASCAL. To make a flexible interface between the user and the COBOL compilers, the translator should be easily modifiable to fit different environments. The PDL, unlike COBOL, has no restriction or special rule on coding foramt. All the statements can be started at any position of a coding line, normally an 80-character data structure, and can be continued into the next coding line.

PASCAL is a structured programming language, it provides enough control structures for the logics of the translator. The PASCAL compiler is available in most micro-computers Therefore, the translator can be executed on many machines.

Figure 1-1: Sequence Control Structure

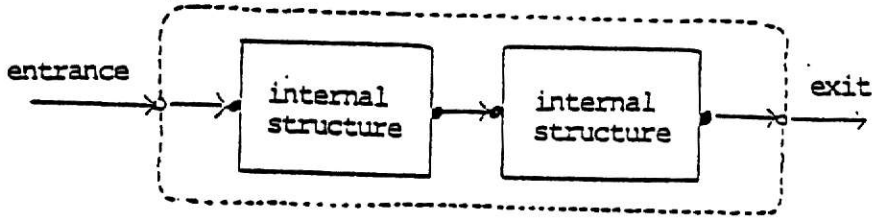


Figure 1-2: Selection Control Structure

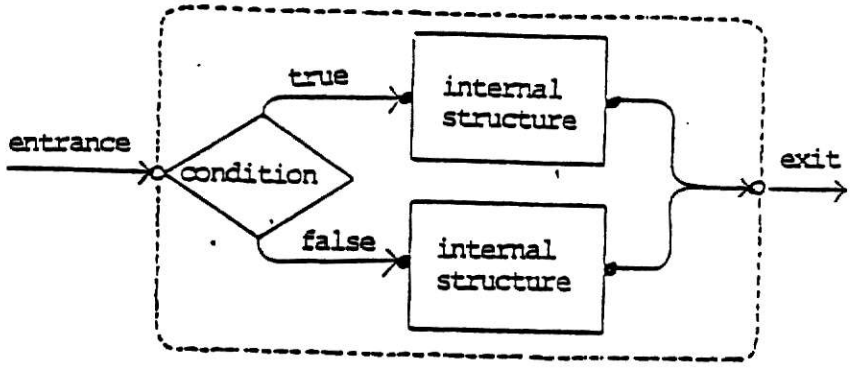


Figure 1-3: Iteration Control Structure "REPEAT - UNTIL"

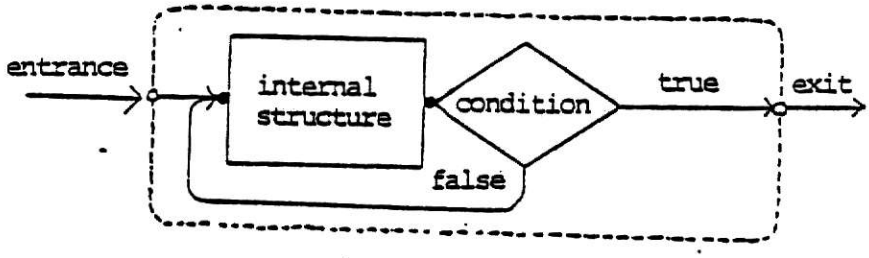
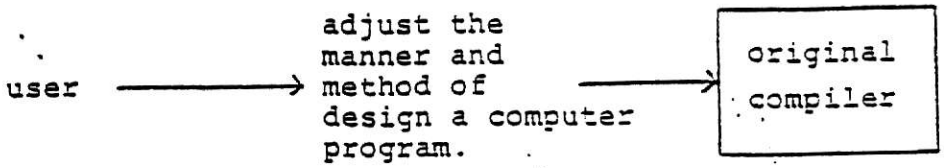
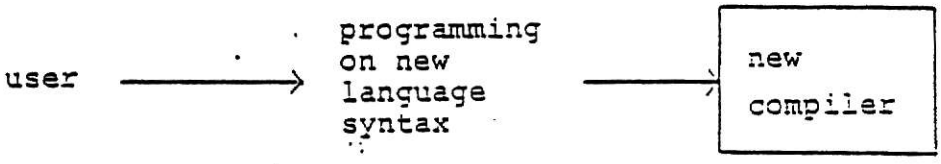


Figure 1-4: Three Approaches to Structured Programming

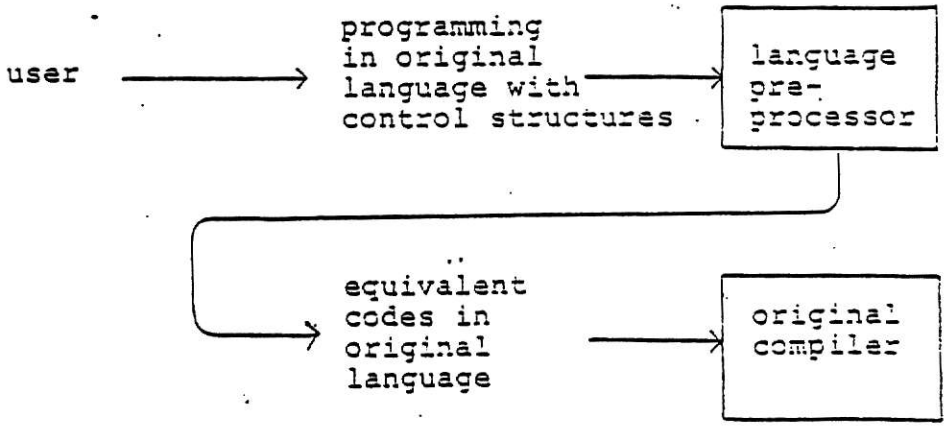
(1) Imitation approach



(2) Design approach



(3) Preprocessing approach



Chapter 2:

DESIGN PHASE

The process of creating a proper Program Design Language(PDL) and its translator consists of the definition of the PDL, the translation of the COBOL equivalence of each PDL statement, and the design for this translator. In the following discussion, the first section defines the syntax and the function of each PDL statement. In the second, the generated COBOL codes are designed to meet the meaning of each PDL statement. In the third, the translator design, based on language definitions, describes the outline of the translation procedures and the necessary modules.

2.1. Definitions of the Program Design Language

The current Program Design Language includes COBOL sentences and non-COBOL statements. The COBOL sentences are embedded into and serve as the content of a control structure. The non-COBOL statements consist of five preprocessor statements and fourteen control statements. The preprocessor statements substitute the for wordy COBOL, and the control statements construct six PDL control structures. The five preprocessor statements provide a convenient and concise format for COBOL. The six control

structures are three various types of control for both the selection and the iteration structures (See figure 2-1).

The non-COBOL statements are led and identified by the predefined PDL keywords (See Appendix A: The PDL User Manual). Examples of the non-COBOL statements and their meanings are described in the user manual (See Appendix A). The syntax of the non-COBOL statements are defined by graphics (See figure 2-1). In the figures, the undefined portions, which are specified by the lower case characters, use COBOL syntax. The non-COBOL statements use the same special characters as the delimiters or the terminators. A control statement can be just a single word; it does not need any termination. According to COBOL syntax, any COBOL sentence is led by a COBOL keyword and terminated by a period.

Non-COBOL statements include the "CALL" statement and the "IF" structure which are already defined in COBOL. The keyword of a non-COBOL statement must be different from that of any COBOL. The PDL "CALL" statement is identified by the keyword "KALL". The keywords "IFF" and 'END-IFF" are used to lead and to terminate the PDL "IF" control structure.

A PDL control structure is built by two or more PDL control statements. Two essential statements are the structure-open and the structure-close statements. The selection control structures include a series of selections chosen by checking

their labelling conditions. These labelling conditions are, therefore, respectively used as a marker for each individual program text.

Because there are more control structures than the three basic types, the flow chart of the PDL control structures is more flexible (See figure 2-2). The entrance point and the exit point of each control structure are the structure-open and the structure-close statements respectively. The selection structures contain more then one internal structures handled by the labelling condition. The intemediate control statements, like "ELSE" of "IFF" structure or ":selection:" of "CASE" and "CASE88"structures, express the meaning of the structure-close statement and the structure-open statement of two neighboring substructures. To eliminate the ambiguity of the nested "IFF" structures, this structure must always be equiped with the two substructures. The null substructure contains nothing; it is a dummy structure (See figure 2-3). This dummy structure can also be used in any PDL structure.

2.2. The Equivalent COBOL codes of each PDL Statement

With the imitation approach mentioned in the last chapter in mind, we define the standard modules of a COBOL equivalent code. All the PDL statements will be translated on this basis. The embedded COBOL, which is used as the content of a PDL control structure, will be directly copied into the output buffer (a data

structure to hold the generated COBOL) without any translation.

Since the PDL preprocessor statements are used to substitute for the existing COBOL sentences, they can be converted to COBOL by a specific method. The COBOL codes of a preprocessor statement will be generated in the output buffer when a preprocessor statement is accepted by the translator. The procedure of this translation is only to copy words from the input statement or the translation instructions to the output buffer.

The translation of the control statements is more complicated than that of other PDL statements. The control statements are used to build a control structure, they decide the processing sequence of the other two types of statements - the embedded COBOL and the preprocessor statements. The COBOL codes for the control statements are designed to paraphrase the control flow of PDL control structures.

2.2.1. The control flow of the PDL control structures

In the figures of the control structures (See figure 2-2), the arrows indicate the control flow of the instructions. The entrance point and the exit point become the starting point and the ending point of this flow. The direction of this flow is always from the entrance point to the exit point, but it can be altered by a given condition. The entrance and exit point, the control flow, and the condition are important in representing the

meaning of a control structure.

Since a control structure can be used as part of the body of another structure, the nesting construct of the control structures seem to produce a more complicated control flow. But The complication only occurs in the relationship and the ownership of the control structures. The control flow will not be further complicated, since any control structure can be treated as a single unit of its own external control structure. To an internal structure, the control flow always enters at the entrance point and leaves at the exit point.

2.2.2. The standard COBOL modules for the PDL control structures

The "procedure division" is part of COBOL program which contains the executing instructions. It consists of some "paragraphs". Each "paragraph" performs some functional operations, it is recognized by its label name. The label names are usually referred by a COBOL "GO TO" sentence, a "PERFORM" sentence, or the sequential processing (See figure 2-4).

In terms of translating PDL to COBOL, the important elements are the label names, the COBOL "GO TO" sentence, and the COBOL "IF" sentence. The entrance and the exit point of a control structure can be represented by the label names. The arrows in the figures become the "GO TO" sentences in COBOL. The condition to decide the direction of the arrows, or to pick up an arrow, is

expressed by the "IF" sentence. For instance, the condition of a PDL "REPEAT-UNTIL" structure is located at the bottom of the structure. The control flow is sent to the beginning label, if the condition is false.

By comparing the previous design with many COBOL programs, the ending label name seems redundant. But the ending label is essential to a control structure, it represents the tasks between two neighboring control structures. If there is nothing between the ending label and the beginning label of the next structure, the control flow will simply pass the dummy "paragraph" named by the ending label and enter the next "paragraph".

Each selection control (or CASE type) structure has more than one internal structure. Only one of the internal structures will be selected on each execution. The control flow from the exit point of each internal structure must be linked to the exit point of the selection structure. The selection condition is put at the starting position. A control flow, or a branch, is switched to the next selection, if the condition is false.

The nesting case of the control structure is to insert a "packed" COBOL paragraph into another one. The "GO TO" sentence of the internal level will never cross the borders, the beginning label and the ending label. In the selection structure, the "GO TO" sentences are still active within their beginning and ending labels of the selection structure. The intermediate labels,

which identify the selecting paragraphs, will be referred to by the "GO TO" sentence of the neighboring paragraphs.

It is important that a label must be unique; a label name can only be used once. In order to make the label names easily identified and produced, the label names of the generated COBOL are built by a specific format. The label names of a control structure have the same numerical prefix, it is a three digits number. For example, the beginning label of a 'REPEAT-UNTIL' structure which uses the identification nuber '005' is '005-REPEAT,' and the ending label is '005-END-REPEAT.' In spite of the ownership of a control structure, every structure has its own identification number. The identification number of a structure has to be different from that of an internal one.

Each selection structure has one or more intermediate label names to identify its internal structures. Since these "internal structures" are at the same control level within a select structure, their label name shares the same identification number with the selection structure. The label of an "internal structure" must be unique too. Therefore, a numerical suffix of three digits is used to indicate the "end" of these internal structures. For example, a CASE-ENDCASE structure which contains four internal structures and has the identification number "007" will have the label names of "007-CASE", "007-NOT-001", "007-NOT-002", "007-NOT-003", and "007-END-CASE". There are four spaces partitioned by the five labels, the first and the

last label enclose the entire selection structure.

The designed COBAL equivalent module of each control structure and each preprocessor statement is listed in the user manual (See Appendix A). A comparison listing of the PDL statement and its COBOL codes is used to explain the relationship between them (See Figure 2-2). This listing will be referred to later to explain the equivalence between PDL and its COBOL codes and the translation semantics of the PDL statements.

2.2.3. The equivalence between PDL and COBOL

The equivalence of the control statements and their COBOL is a notable problem. The embedded COBOL is simply copied into the output buffer area. The preprocessor statements are converted to standard COBOL codes, they are only an easier way to express the same meaning and there is no equivalence problem. The control statements, which are used to build the control processes on COBOL, may cause an equivalence problem. This problem does not occur in each individual control statement. A control structure is a basic unit which requires no further discussion, since a complete structure includes the complete meaning of a control flow. A more complicated problem exists in the relationship among the nested structures.

For each individual control structure, the generated COBOL

codes are designed to obey the meaning of the control flow (See the comparison in figure 2-2). Those COBOL codes are defined as the standard model for each PDL control structure. The spaces enclosed by the generated codes are reserved for the incoming program texts. The label preceding a space becomes the entrance of each space. The "GO TO" sentences perform the control flow, since they change the natural direction of the flow. The meaning of "control" is decided by the "IF" sentences. The equivalence between PDL control structures and their COBOL in the individual structure level can be directly compared to the control flow in the generated COBOL codes.

Since the conditional expression can be very complicated, the location of a generated conditional sentence is important. Unlike other languages, COBOL does not allow the adding of "NOT" to a "packed" condition to make it a negative. A COBOL clause "NEXT SENTENCE" is used to supplement this defect in COBOL.

In COBOL, the "NEXT SENTENCE" clause can be used to fill a dummy part of the "IF" sentence. A COBOL sentence, "IF condition NEXT SENTENCE ELSE GO TO label-name.", states that if the condition is true the next sentence after this "IF" sentence will be executed, otherwise the execution sequence will be branched to the specified label name. By using this model, all the control flow shown in the figures can be expressed in terms of COBOL (See Appendix A and figure 2-2).

The equivalence of the individual structure can be verified and the equivalence of the nested structures can also be verified on this basis. In the previous descriptions, we know that the control flow of a structure will not affect the control flow of the surrounding structures. The nested structures in COBOL will be equivalent, since all the individual structures in COBOL are identical with their PDL counterparts and the relationship among these structures in COBOL is the same as that among their corresponding PDL structures.

2.3. Translation Design

By comparing the construct of a PDL statement with that of its COBOL equivalence, we found that the functions of the translator consist of moving, concatenating, and comparing character strings. These translation operations must be appropriately managed by recognizing the end of a PDL statement, the end of a PDL control structure, and the end of an entire PDL source text. The translator basically consists of sub-routines to perform the functions necessary for each PDL structure.

The translation is performed on a statement by statement, and structure by structure basis. The translator contains some modules to perform the functions of character string manipulations. The main part of the translator uses these modules to achieve the desired translation.

The outline of the entire translation is to load the PDL statements one by one into the translator. The translator, by recognizing the attribute of the input PDL text, uses the proper translation instructions to generate the desired COBOL codes in the output buffer area. The input text and the designed outputs will be held in data storages. They are only used during the translation of a current input statement. It is important that some data be held longer, when for example these data are still potentially useful within a control structure (See figure 2-5).

2.3.1. The translation semantics and the translation instructions of each PDL statement

The comparison listing of a PDL statement and COBOL (See Appendix B) provides a model for the translation of each PDL statemnt. The translation semantics is designed by decomposing all the necessary operations which perform the function of the translation procedures. Basically, these operations are character string manipulations. A character string can be a word or a single character; it is separated from other character strings by a blank or a delimiter. A delimiter usually is a special character, such as parenthesis; it is recognized as a character string during translation.

The translation semantics is the blue print of the translation. The semantics is manually defined without a formal format, it can not be directly referred to by a computer program.

A type of translation instruction is defined to express the translation semantics by a defined format. An instruction performs a specific function and it is also a command to execute a submodule of the translator with parameters. An instruction set consists of some instructions and a termination. The translation of two PDL can share a set of translation instructions, if their translation procedures are similar.

The format of the translation instruction includes a command word and four parameters. The command word is a verb which indicates an action. The first and second parameters define the input and the output storages in the translator. These two parameters also define the meaning of the data flow. The third parameter takes thirty spaces to specify a predefined character string to be referred to. The last parameter defines the decision of branches.

There is no label field in the format of the translation instruction. The translation instruction will be identified by its index number. Therefore, all kinds of branches are defined by computing the value of the instruction pointer with a giving value. In normal cases, the instruction pointer is always increased by one in order to get into the next instruction. A condition branch occurs when two character strings are compared and found that they do not match.

2.3.2. The execution flag of the translator

The execution flag of the translator provides communiation between the inside and outside of the translator. It reflects the status of the peripheral interface and the management of the submodules. The result of performing the I/O routine will be reported to the translation monitor by this flag. The execution result of each translator submodules is obtained by the translation monitor to decide the next step of operation.

All the control units of the translator use this flag. In the beginning of a input statement, the instruction pointer will be reset. Storages are cleared or held for the current control structure. At the end of an input, if a PDL statement is incomplete, the I/O routine is called to fetch the next input line. The instruction pointer will not be changed, it will simply continue the unfinished part of the translation.

2.3.3. The software techniques used to design this translator

Based on the requirements of this systems, some software techniques are essential to build a flexible and efficient translator. The flexibility of this translator is to deal with the possibility of changes in the translation semantics. The efficiency of this translator is to provide a better user interface. The following decription will express how these techniques work.

The input and the output requirements of PASCAL are different from system to system. The data transportation among the translator and its peripheral modules is isolated from the regular operation of the translator. Two subprograms of the translator are used to do these transportation tasks. One is to load in PDL input text, the other is to write the formatted COBOL codes into output lines.

By isolating the data transportation, the data structures of the input text and the output code can be defined in a fixed format. A card format of eighty character is used, since it is the most popular data format. An output buffer,a working storage of the translator which will hold the unformatted temporary output, is defined as a string of two hundred and fifty six characters. Therefore, the data flow in the translator begins with the input line of the card format and ends with the output buffer.

Other than the I/O submodules, some basic submodules perform the simplest functions of character string manipulation tasks. One is to fetch a word from input line and to place it into a temporary store. The others are to move a word or to attach a word from a store to another store. All these submodules are called basic, because they are used by other submodules.

The rest of the submodules in the translator correspond to the defined commands of the translation instruction. The

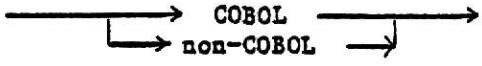
translation instructions are defined from the translation semantics, they perform the translation operations within the translator. The main submodule of the translator refers to the proper instruction by the instruction index, and each instruction advises the main submodule to pick up a submodule to do a step in the translation process. The parameters of the instruction will be used by the called submodule.

The table-driven technique provides a changeable environment for the translator by converting the translation semantics into a translation table. This isolates the problem of different translation semantics from the translator. The table-builder module stands on the same level as the translator; it always converts the translation methods into tables for the translator. The translation instructions read by the translator are not in their original formats; they have become the translation tables. Furthermore, these tables will be copied into the translator as its internal tables.

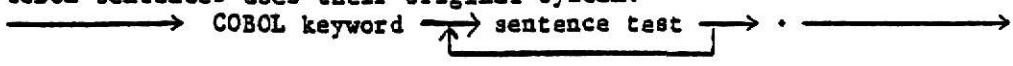
The problem of managing the nesting of the PDL control structure during the entire translation can be resolved by using a stack to hold the translation data on each control level. A stack is a first-in-and-last-out data structure. The data saved the earliest will be the last to go; the lastest saved data will be erased first. The nature of stack agrees with that of the control structure. The translation data of the current control structure is kept on the "top" of the stack, this data will be

discarded when it exits from the current control structure (See figure 2-6).

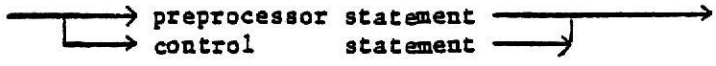
Figure 2-1: The Syntax of PDL Statements



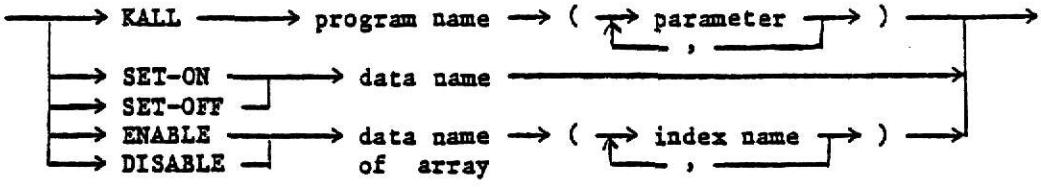
The COBOL sentences uses their original syntax:



The non-COBOL statement:



The preprocessor statements:



The control statements:

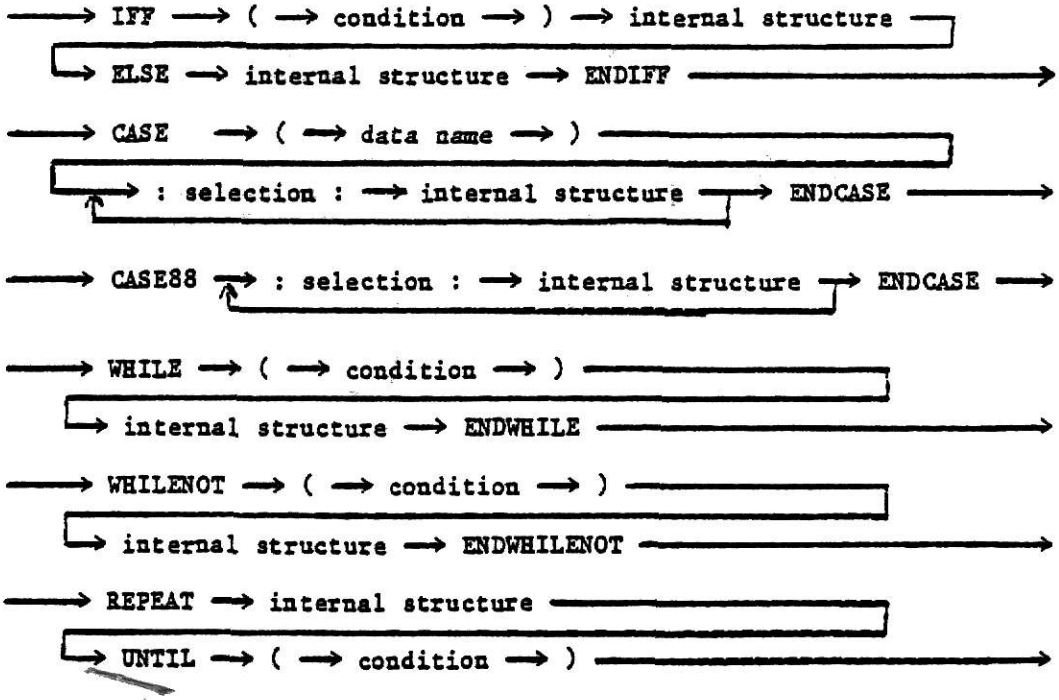


Figure 2-2: The PDL Control Structures with Their Flow Chart and the Designed COBOL Equivalences

Descriptions of the symbols in the figures:

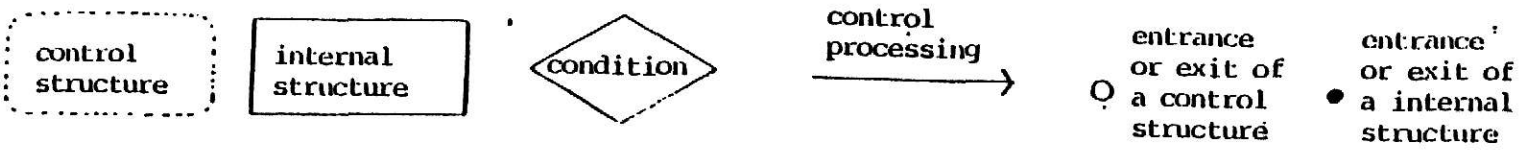
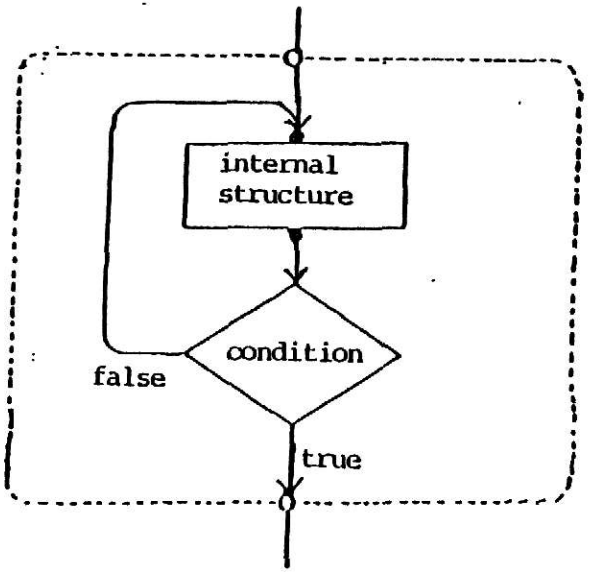
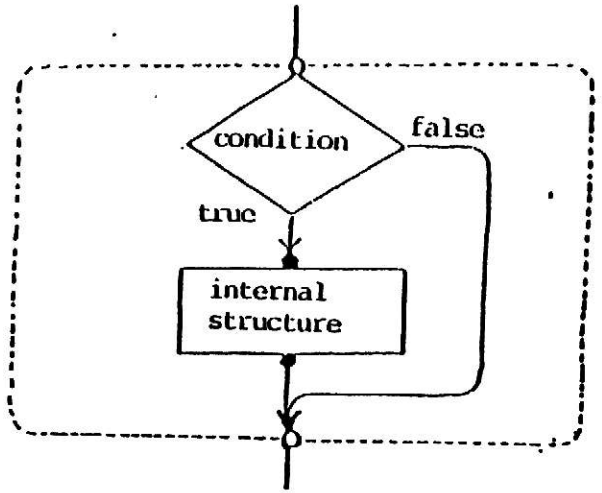


Figure 2-2-1: The REPEAT-UNTIL Structure



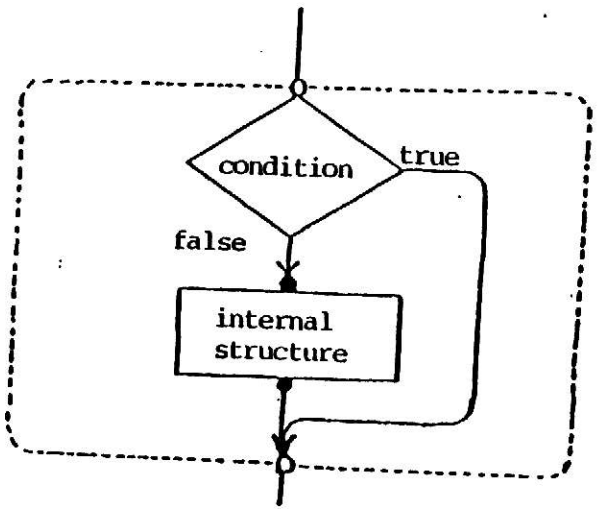
PDL	COBOL
REPEAT	num-REPEAT.
internal structure	internal structure
UNTIL (condition)	IF condition GO TO num-END-REPEAT ELSE GO TO num-REPEAT. num-END-REPEAT.

Figure 2-2-2: The WHILE-ENDWHILE Structure



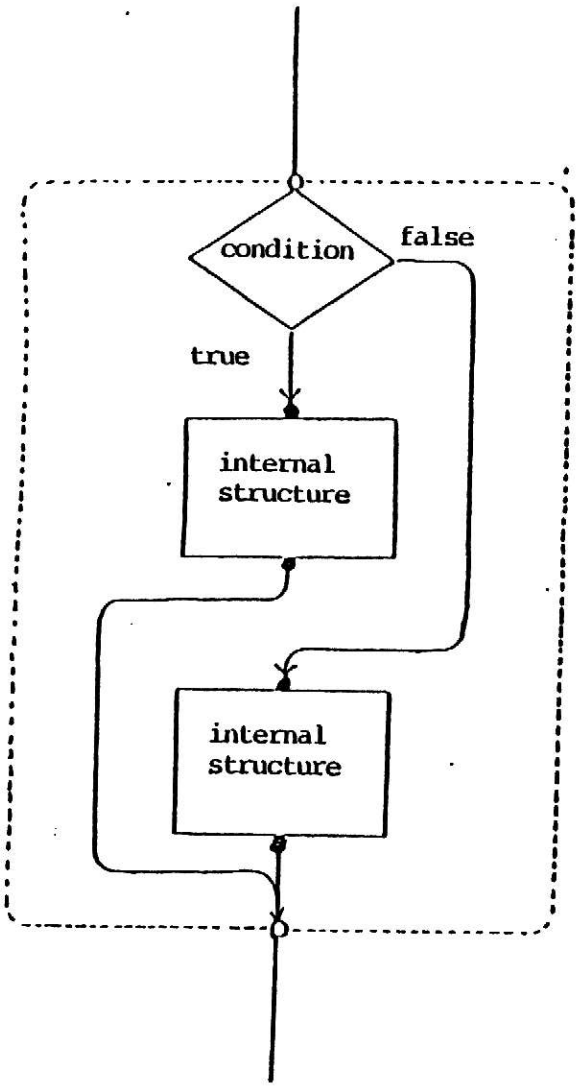
PDL	COBOL
WHILE (condition)	num-WHILE. IF condition NEXT SENTENCE ELSE GO TO num-END-WHILE.
internal structure	internal structure
ENDWHILE	GO TO num-WHILE. num-END-WHILE.

Figure 2-2-3: The WHILENOT-ENDWHILENOT Structure



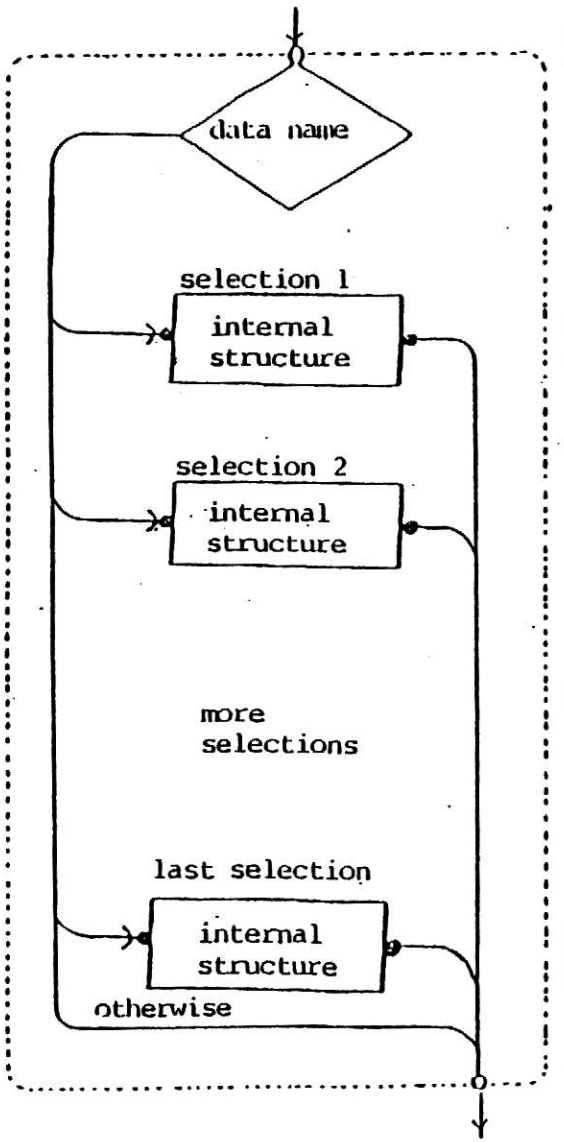
PDL	COBOL
WHILENOT (condition)	num-WHILENOT. IF condition GO TO num-END-WHILENOT.
internal structure	internal structure
ENDWHILENOT	GO TO num-WHILENOT. num-END-WHILENOT.

Figure 2-2-4: The IFF-ELSE-ENDIFF Structure



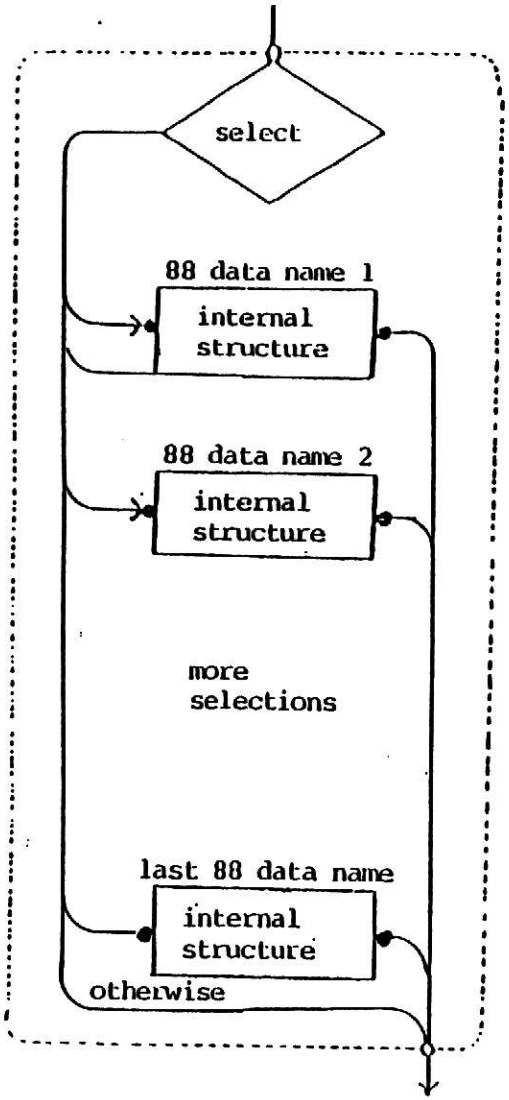
PDL	COBOL
IFF (condition)	num-IFF. IF condition NEXT SENTENCE ELSE GO TO num-IFF-NOT.
internal structure	internal structure
ELSE	GO TO num-END-IFF. num-IFF-NOT.
internal structure	internal structure
ENDIFF	GO TO num-END-IFF. num-END-IFF.

Figure 2-2-5: The CASE-selections-ENDCASE Structure



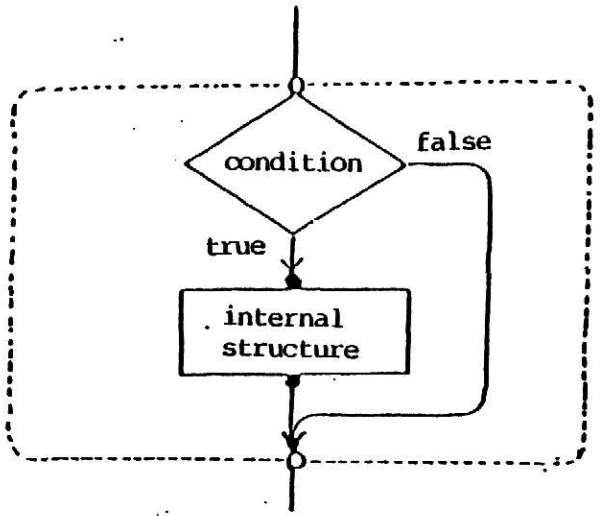
PDL	COBOL
CASE (data name)	num-CASE.
: Selection 1 :	IF data name EQUAL TO selection-1 NEXT SENTENCE ELSE GO TO num-NOT-001.
internal structure	internal structure
: Selection 2 :	GO TO num-END-CASE. num-NOT-001. IF data name EQUAL TO selection-2 NEXT SENTENCE ELSE GO TO num-NOT-002.
internal structure	internal structure
more selections	more selections
: last selection :	GO TO num-END-CASE. num-NOT-nnn. IF data name EQUAL TO selection-n NEXT SENTENCE ELSE GO TO num-NOT-nnn.
internal structure	internal structure
ENDCASE	GO TO num-END-CASE. num-NOT-nnn. num-END-CASE.

Figure 2-2-6: The CASE88-selections-ENDCASE88 Structure

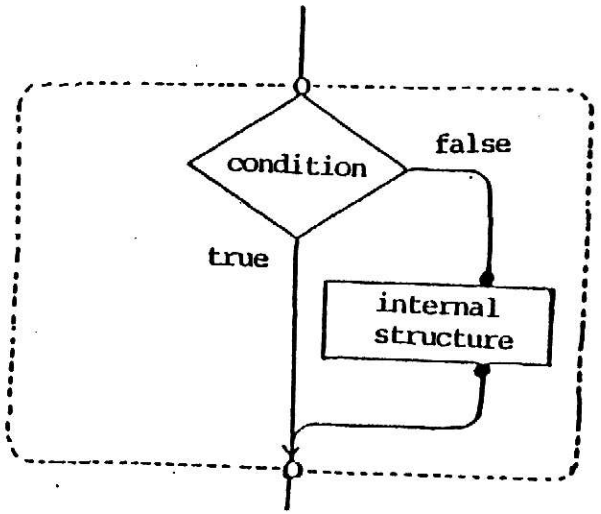


PDL	COBOL
CASE88	num-CASE88.
: 88-data-name-1 :	IF 88-data-name-1 : NEXT SENTENCE ELSE GO TO num-NOT-001.
internal structure	internal structure
: 88-data-name-2 :	GO TO num-END-CASE88 num-NOT-001. IF 88-data-name-2 NEXT SENTENCE ELSE GO TO num-NOT-002.
internal structure	internal structure
. more selections	more selections
: last-88-data-name :	GO TO num-END-CASE88. num-NOT-nnn. IF last-88-data-name NEXT SENTENCE ELSE GO TO num-NOT-nnn.
internal structure	internal structure
ENDCASE88	GO TO num-END-CASE88. num-NOT-nnn. num-END-CASE88.

Figure 2-3: The Select Structure with a Null Internal Structure



PDL	COBOL
IFF (condition)	num-IFF. IF condition NEXT SENTENCE ELSE GO TO num-IFF-NOT.
internal structure	internal structure
ELSE	GO TO num-END-IFF. num-IFF-NOT.
nothing	
ENDIFF	GO TO num-END-IFF. num-END-IFF.



PDL	COBOL
IFF (condition)	num-IFF. IF condition NEXT SENTENCE ELSE GO TO num-IFF-NOT.
nothing	
ELSE	GO TO num-END-IFF. num-IFF-NOT.
internal structure	internal structure
ENDIFF	GO TO num-END-IFF. num-END-IFF.

Figure 2-4: Control Flow Shown on a COBOL Procedure Division

COBOL Sentences	Control Flow
PROCEDURE DIVISION.	sequence - next
OPEN INPUT FILE-IN, OUTPUT FILE-OUT.	sequence - next
WRITE LINE-OUT FROM HEADING-LINE AFTER ADVANCING TO-NEW-PAGE.	sequence - next
MOVE SPACES TO LINE-OUT.	sequence - next
WRITE LINE-OUT AFTER ADVANCING 1.	sequence - next
MAIN-ROUTINE.	sequence - next
READ FILE-IN AT END GO TO STOP-RUN.	conditional branch
MOVE LINE-IN TO LINE-OUT.	sequence - next
WRITE LINE-OUT AFTER ADVANCING 1.	sequence - next
GO TO MAIN-ROUTINE.	unconditional branch
STOP-RUN.	sequence - next
CLOSE FILE-IN, FILE-OUT.	sequence - next
STOP RUN.	

Figure 2-5: The Outline of the Translation

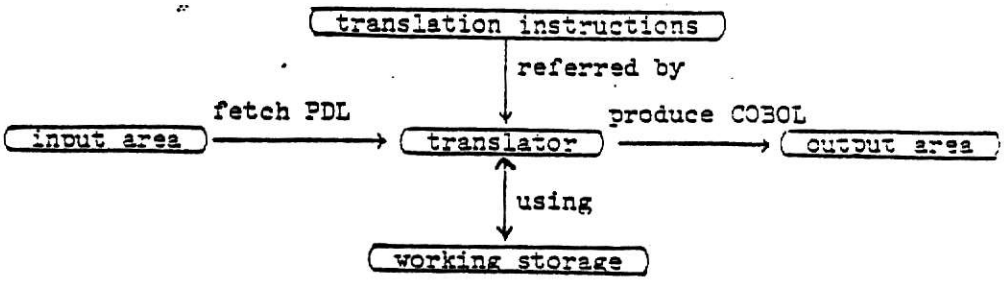


Figure 2-6: How the "Stack" data structure manage the translation of the nesting control structures

The nested PDL control structure	nested level	the stack data storage	the stack pointer
IFF (condition)	1		
statement	1		
WHILE (condition)	2		
statement	2		
IFF (condition)	3		
statement	3		
ELSE	3		
statement	3		
REPEAT	4		
statement	4		
UNTIL (condition)	4		
statement	3		
ENDIFF	3		
statement	2		
ENDWHILE	2		
statement	1		
ENDIFF	1		

Chapter 3:

IMPLEMENTATION

In the design phase, the authors defined a type of Program Design Language (PDL), the COBOL equivalence of the PDL statements, and the process of the translation between PDL and COBOL. There are some translation principles and software techniques suggested in the design phase to achieve the expected translation results. These methods and principles are to isolate input/output problems from the translator, to use the table-driven technique to make the translation flexible under different environments, and to use a "stack" data structure to manipulate the data during the translation of the nested PDL control structures.

There are several steps to accomplish these goals. First: the data transportation of the translation are defined to simplify the input and output interfaces of the translator and its peripherals and to standardize the data transmission within the translator. The data flow within the translator is isolated from the peripheral. Second: the translation tables are made to increase the modifiabilty of the translator. Third: a "stack" data structure is defined to handle the nested PDL control structures.

In this chapter, there are three sections to describe the

entire process of the translation and the performance of this task. The input/output isolation and nested control structure manipulation, which deal with the subjects of data flow and communication among the translation modules, are discussed in the first section, the translation outline. The creation and construction of the translation tables are described in the second. The construction of the translator is introduced in the last section and the execution and the performances of the translator will be discussed.

3.1. The Translation Outline

The figure of the translation outline (See figure 2-6) defines five modules and four relationships, and illustrates the basic translation operations. The relationship among these modules determines how the "words" of an PDL statement flow among the system modules and how these "words" are translated.

The purpose of using the techniques introduced in the design phase is to simplify the function and the construction of the translator. Therefore, the mode of translation can be easily modified by loading new translation semantics into the translator. The change of the translation environment - the execution of the translator under different computer systems or the employment of the translator as subsystem of a system - will not have any impact on the normal operation of this translator, because the simplification of the translator tends to

isolate the volatile elements from the translator.

3.1.1. The data transfer between the translator and its peripheral

The five "boxes" in the figure of the translation outline can be divided into two levels. The "stack" and the translation tables are actually parts of the translator. The input area and output area are usually system data stores: they are different form system to system. It is important that the medium of the data transportation between these areas and the translator must be fixed.

First of all, the data structure used to load PDL statements into the translator and to unload the COBOL code from the translator is only a "card" format structure, an array structure of eighty characters. The "loader" utility of the host computer system will take care the data transportation between the translator and the input/output area by using the "card" format structure. There are two stores defined in the translator, the "input-line" and the "output-line", to temporarily hold the currently accepted and produced data. These data stores can also be treated as the beginning point and the ending point of the data flow within the translator.

3.1.2. The data flow within the translator

Since the length of the output COBOL codes always exceed

that of the PDL input statements, a "buffer" storage is prepared to keep the entire output codes of an PDL statement before they are loaded on the "output-line". This "buffer" area is defined as an array of two hundred and fifty six characters. An "editor" subprogram of the translator converts the "buffer" contents into COBOL-oriented format and loads COBOL codes on the "output-line" one by one.

From the "input-line" to the "buffer" or the "output-line", a "word" type of data structure is defined as the basic translation unit. The "word" type data structure is an array data structure of thirty characters, since all data names can be up to thirty characters in COBOL. The unused portion of this data store is filled with blanks. A "word" defined in the translator can be a COBOL data name, a word from the PDL statement, an operator in COBOL, or a delimiter.

During the translation to an PDL statement, a "word" is fetched by a subprogram of the translator and put into a data store, the "temp-word". The content of the "temp-word" will be processed by other subprograms of the translator to produce the desired outputs. The content of a "stack" entry hold the semi-product of the translation.

The "input-line" is not the only area which provides the translation source. The translation tables also contribute some data for the translator. The data flow within the

translator starts from the "input-line" and the translation tables, passes the "temp-word", the "stack" stores, the "buffer", and ends at the "output-line" (See figure 3-1 and 3-2).

3.1.3. The "stack" data structure

The function of the "stack" data structure has already been introduced in the last chapter. The "stack" contains ten entries (See Appendix C-2). Each entry has ten pairs of records, and each record has an individual store to hold a "word" and a counter to tell the length of this "word". Seven of the ten stores are designed to be used by the PDL control statements, the other three stores are used to convert the PDL preprocessor statements and to copy the embedded COBOL.

The "stack" is defined as an array data structure. The entries are be referred to by their indices. A "stack pointer" holds the index number of the currently used entry. At the beginning, the "stack pointer" points to the first entry, all the stores in this entry are cleared to blank, and the counters are set to zero. When a "word" is loaded into one of these stores, the counter will record the length of this "word".

When appending a "word" to one of the stores, the subprogram will add the length of the incoming "word" to the counter. If the total length exceeds thirty characters, the maximum length of the store capacity, an error message of 'storage overflow' will be

encountered and the translation will be terminated due to error.

Since an open statement of an PDL control structure will increase the "stack pointer" by one and the close statement will decrease it, this pointer can produce two types of errors. When the pointer is on the first entry and an PDL close statement is read, this may cause the error of 'stack underflow.' When the pointer is on the top entry, the last available entry, and an PDL close statement is read, this may cause 'stack overflow.'

3.2. The Translation Tables

The translation tables are the bases of the translator they supervise the process of the translation. These tables include the command table (command table), the command address table (address table), and the instant character table (string table). To the translator, the translation tables are the coded version of the translation semantics (See Appendix B). The translation semantics will be coded by using the format of translation instructions (See Appendix B). A TABLE-BUILDER program produces these tables by using the translation instructions.

3.2.1. The translation instructions

The translation instructions (See Appendix D-1) consist of five fields to express a single step of the translation process. The fields are the "command", the "source storage name", the

"target storage name", the "branch index", and the "instant character string". The first three fields specify the name of a subprogram or a store in the translator. The fourth field, the "branch index", provides a constant value to change the translation status. Each of the four fields takes ten characters. The last field, the "instant character string", which may contain a COBOL data name, is thirty character long.

The "command" field defines the subprogram in the translator to perform an operation by using the parameters specified in the rest of fields (See Appendix C-1). The "commands" can be classified into the LOADERs, the CONTROLLERs, and the INSPECTORs. The LOADERs use data from the "source" field and produce data to the "target" field. The "controllers" conduct the operation of the translator. The "inspectors" change the operation of the translator by comparing the contents of two data stores.

The "source storage" and the "target storage" define the data stores of the "temp-word", the "buffer", or the ten data stores of each "stack" entry (See Appendix C-2). All the processing data are kept in these stores. These two fields may be omitted when a command does not refer to any data store, or when a data store defaults. Most "controllers" are defined to maintain the counter or the flag variables.

The "branch index" contains a numeric value within the range

from -99 to 99, and it changes the translation sequence by adding this value to the instruction pointer when the process of the current instruction fails. The translation will be completely terminated, if this field is 99. A positive value will skip some instructions, a negative value will set the instruction pointer back to a previous instruction. The instruction pointer is always increased by one after the process of an instrcution, a backward branch to the fifth step prior to the current one must be specified by a negative number six in this field.

The "instant character string" specified in the last field of a translation instruction is the "read only" data of the translator. The possible names defined in the two data storage fields are the twelve data stores which can be randomly used, they are coded by the number from one to twelve. There are eighty-seven other stores defined in the translator to hold these strings, they are coded by the numbers from thirteen to ninety-nine.

3.2.2. The construct of the translation tables

The table-builder program (See Appendix D-2) uses the translation instructions to produce three translation tables in separate files. The translator reads these files into its internal data structures. The translation mode can be modified by loading new translation instructions into the "table-builder"

program to obtain new translation tables. Thus, the problem of different translation is isolated from the translator.

The "command table" contains four numeric values in each entry. The "command" field has the code of the translation command of a translation instruction. Next to the "command field," the "source field" is the code of the "source" store name, or the index of the "character string" of a translation instruction. The third field is the code of the "target" store name. The branch index in the translation instruction is copied into the last field. Anything undefined in the translation instruction will be complemented by zero (See Appendix D-3). The "command table" is defined as an array of three hundred entries in this format.

The "address" table is also generated from the translation instruction (See Appendix D-4). The "table-builder" program will write a PDL keyword with the current index derived from the listing of instructions, when the instruction has "KEYWORD" in the "command" field. The PDL keyword is specified in the last field of this instruction. The counter does not increase the index for this instruction, since the data of this instruction is not used in the "command" table.

The "string" table has eighty-seven character strings found from all the translation instructions (See Appendix D-5). The unused entries of this table are blanks. Any character string can

be reached by its index in the array structure. To save storage and to reduce the size of this table, the same character string found from different translation instructions share the same index.

3.3. The Translator and Its Performances

The translator is a computer program coded in PASCAL (See Appendix E-1). The outline of the translation and the major data stores of the translator have already been introduced in the early part of this chapter. The construct and the operations of the translator will be described by using the data flow within the subprograms of the translator.

3.3.1. The flags

First of all, there is a flag to reflect the status of the input area. An "unfinished" status occurs, when the end of "input-line" is encountered and the translation of the current PDL statement is not finished. The translator will retain the current translation result and read another input to continue. If the translation of a PDL statement is done and there still is other PDL statements on the same "input-line", The translator processes this statement as usual. This flag does not actually exist in the translator; it is just the counter of the "input-line". However, the status of the "input-line" has no impact on

the status of the translation.

The translation flag is a numeric variable, it is zero when the translator starts to translating a PDL statement. An error is encountered when the flag contains a value greater than zero. If so, the translation stops and an error message is ejected. A translation is normal when the flag value ends up with zero. (This flag has the initial value -1.) At the end of the "input-line", the value of this flag turns -2. It is reset to -1 to continue the translation. At the end of all PDL text, this flag contains the value of -3.

3.3.2. Data flow

Having read the contents of the translation tables and set the initial value to all the working storages and flags, the translator starts to request the input PDL text. The system loader puts the PDL text in the "input-line" and a "get-word" subprogram copies the first "word" from the "input-line" to the "temp-word". The "get-word" subprogram takes the operater or the delimiter as a "word". For example, "IFF (COUNTER EQUAL 99)" is the same as "IFF(COUNTER EQUAL 99)". This text contains six "words".

Since the first word of a PDL statement is the keyword, a "recognizer" subprogram identifies the type of this "word". If this "word" is not found in the "address table", the current PDL

statement is COBOL sentence. The text will be copied until a period is read to indicate the end of this COBOL sentence. Otherwise, the address obtained from the "address table" will be referred to as the starting index of the translation instruction which is defined in the "command table".

From the starting index of the "command table", the translator uses the translation instructions to translate the PDL statement until an "ENDUP" instruction is processed. At this moment, an "p-endup" subprogram edits the contents of "buffer" to the "output-line" and puts '0' into the translation flag.

The outputs held in "buffer" are basically all upper case character or numeric characters. The lower case characters 'a,' 'b,' 'c,' and 'd' are used as the editting control characters. The 'd' is blank, since there is no blank written to "buffer". The first blank found in the buffer tells that it is the end of buffer contents.

The "command table" has ten kinds of translation commands, there are ten subprograms defined in the translator to perform the function of each corresponding translation command in the "command table". The data "loader" subprograms do not directly process the related data; there are some subroutines defined in the translator to serve the "loader" subprograms. Almost all the subprograms will change the value of the translation flag to

terminate the translation or eject an error.

3.3.3. The result of test run

The two major modules, the table builder program and the translator, were modified for the purpose of testing. Each program will list all the results during processing. From these listings, we can verify their correctness.

The listing produced from the table builder program includes everything in the three translation tables. Each line of the listing includes all the original fields of a translation instruction and the code of these fields. (See Appendix D-6).

An example of PDL input text is listed in User Manual (See Appendix E-2). The translator uses this text to generate COBOL codes in appendix. (See Appendix E-3). The translator also provides a dump listing of the variables during execution (See Appendix E-4). Whenever the translation is changed, the translation tables are updated.

Figure 3-1: Data Flow within the Translator
- the PREP program

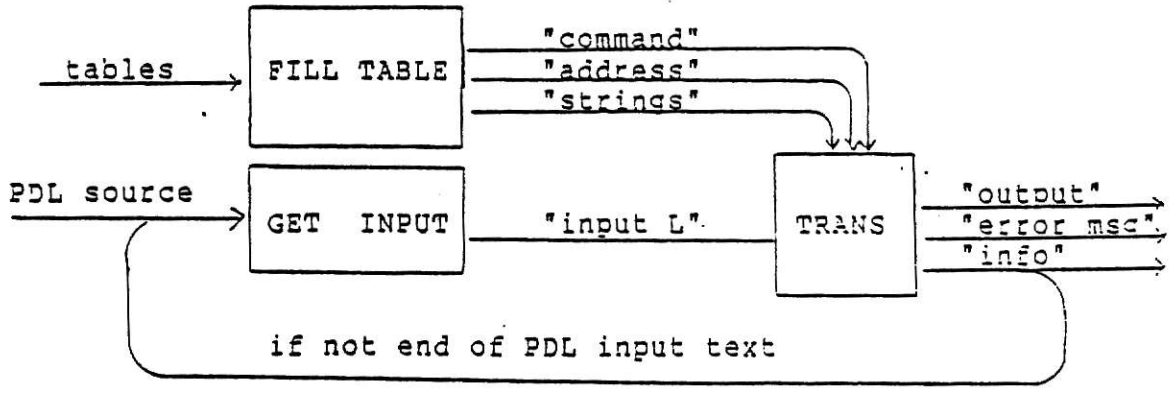
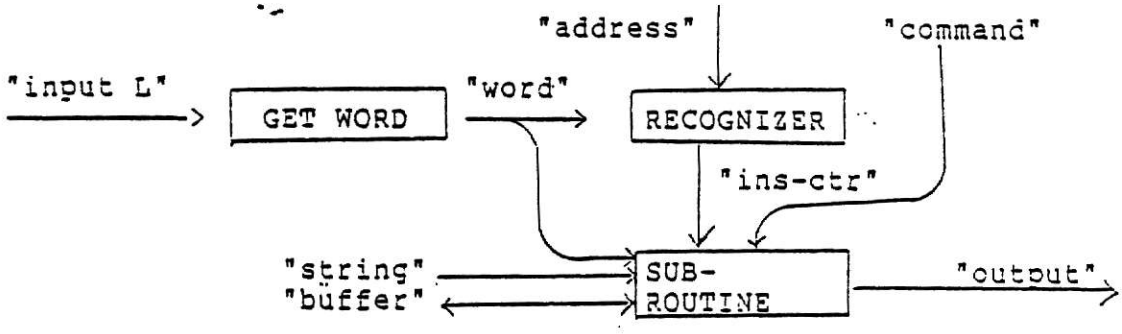


Figure 3-2: Data Flow within the TRANS program



Chapter 4:
CONCLUSION

Using the Program Design Language (PDL), the structured programming technique can be applied to COBOL without changing the compiler. The programmers can be free from using a insufficient language feature for complicated logic. The computer program will be more easily understood and modified.

The hierarchically arranged control structures allow the program designers and the users to approach the solution of a giving problem with "stepwise refinement". The complicated logic design can be simplified by hiding the details in the deeper level of control. The various types of control structures make the logic clearer.

The "CASE" structure extends the selection options to more then two. The COBOL has only the "IF" statement to do a binary selection, the "CASE" structure replaces a bunch of COBOL "IF" statements with more selections. The special structure, the "CASECOC" structure, is an ad hoc design of COBOL. The COBOL "88" level data name can be directly tested like a boolean variable.

The translation design employs some techniques to isolate the volatile conditions from the translator. A simply built

computer program tends to be less prone to error and more efficient. The translator itself, besides doing the job of the execution management and storage maintenance, is a simple machine to perform the functions of character string manipulations. This design makes the translation more flexible and easier to avoid errors.

The translator is restricted under some conditions, such as the depth of the nesting level of the PDL control structures and the maximum length of the outputs. These restrictions can be changed by increasing the constants in the translator program. When the host system can provide enough memory space, this problem can be solved.

In the subprogram which is to fetch a PDL input text, there are two control characters defined to delete the previous character or the entire input line. This design is specially done for users who uses this translator as an interactive system. This subprogram can be modified to fix the change of input specification.

The translation return code can be used as a communication tool with its surrounding modules, when the translator is employed as a module of a system. In this case, the subprogram of fetching PDL texts can be reduced. The module which used the translator can based on this code to do a decision.

The current PDL does not include a "DO" loop, but this feature can be added by coding the translation instructions for this structure. There are more special types of control structures possible. For example, the "EXIT" statement of "WHILE" loops or a "WHILE" control with a "DO" loop is a good feature. To put these structures into the current PDL may cause more work on the translator and the construction of the translator will be further complicated.

The translator does not process the logic decomposition of the conditional expressions. Therefore, we have to do some unnatural translation to make the antithesis of a conditional expression. To solve this problem will involve more logic problems, since the complication caused by having different expressions is totally unpredictable. The method we have developed to solve this problem is not a difficult one to work with.

REFERENCES

[Donaldson-73]
J.R. Donaldson,
Structured Programming
Datamation, Vol. 19, No. 12, Dec 1973, pp. 52-54

[Mills-72]
H.D. Mills
Chief Programmer Teams: Principles and Precedures
IBM Technical Report FSC 71-5108,
International Business Machines Corp.,
Gaithersburg, Md., June 1972

[Bohm-66]
C. Bohm and G. Jacopini,
Folw Diagrams, Turing Machines and Languages with
only Two Formation Rules
Comm. ACM. Vol. 9, No. 5. May 1966, pp. 366-371

[Wirth-71]
Program Development by stepwise Refinement
Comm. ACM, Vol 14, No. 4. Apr. 1971, pp 221-227

[Jensen-81]
Randull W. Jensen
Structured Programming
Hughes Aircraft Company
Computer, Vol. 14, No. 4 pp 31-48, March 1981

[Crawford-74]
Marshal A. Crawford
Splice- A COBOL Pre-Computer
ARDS MONITOR-Apr. 1974

[Presser-75]
Leon Presser
Structured Languages
Proc. AFIPS 1975, NCC, pp 291-292

APPENDIX A: USER MANUAL

1. Keywords, Delimiters, and Symbols of PDL

1.01. The PDL Keywords

1.02. The PDL Delimiters

1.03. The valid characters used in PDL statements

2. The Translation Mode of PDL

2.01. Translation mode of the Kall statement

2.02. Translation mode of the SET-ON statement

2.03. Translation mode of the SET-OFF statement

2.04. Translation mode of the ENABLE statement

2.05. Translation mode of the DISABLE statement

2.06. Translation mode of th IFF structure

2.07. Translation mode of the CASE structure

2.08. Translation mode of the CASE88 structure

2.09. Translation mode of the WHILE structure

2.10. Translation mode of the WHILENOT structure

2.11. Translation mode of the REPEAT structure

3. The Limitations and Other Coding Restrictions

1. Keywords, Delimiters, and Symbols of PDL

The PDL statements consist of COBOL and nonCOBOL statements. The COBOL statements use their original syntax, they are not described in this manual. The PDL (the nonCOBOL portion) statements are recognized by their leading word (the keyword), and using special characters as the delimiters.

1.01. The PDL keywords include:

```
KALL
ENABLE, DISABLE, SET-ON, SET-OFF,
CSE, :, END-CASE,
CASE88, END-CASE88,
IFF, ELSE, END-IFF,
REPEAT, UNTIL,
WHILE, END-WHILE,
WHILENOT, END-WHILENOT
```

1.02. The PDL delimitërs include

blands, commas, colons, parentheses, and periods.

1.03. The valid characters used in PDL statements can be:

1. The alphabetical characters:
   'A' ... 'Z'
2. The numberical characters:
   '0' ... '9'
3. The delimeters:
   ' ' ':' ',' '(' ')' '.'
4. The COBOL operators:
   '+' '-' '*' '/' '='

2. The Translation Mode of PDL

The PDL statements (nonCOBOL portion) can be classified into the control statements and the preprocessor statements. There are five PDL preprocessor statements to substitude the wordy COBOL, and the control statements form the six PDL control structures.

The five preprocessor statements is:

'KALL', 'SET-ON', 'SET-OFF', 'ENABLE', AND 'DISABLE'

statements.

The six control structures are formed by the fourteen PDL control statements.

- The 'IFF' sturcture includes the 'IFF,' 'ELSE,' and 'END-IF' statements.
- The 'CASE' sturcture includes the 'CASE,' ':<VALUE>:,' and 'END-CASE' statements.
- The 'CASE88' structure includes the 'CASE 88,' ':<VALUE":,' and 'END-CASE88' statements.
- The 'REPEAT' structure includes the 'REPEAT' and 'UNTIL' statements.
- The 'WHILE' structure includes the 'WHILE'and 'END-WHILE' statements.
- The 'WHILENOT' structure includes the 'WHILENOT' and 'END-WHILENOT' statements.

2.01. The KALL statement substitutes COBOL CALL sentence.

The code: KALL <pgm-name> (<parm-1>[, parm-n])

will be translated as below:

```
CALL "<pgm-name>" USING
    <parm-1>
    [parm-n].
```

Example: In COBOL data-division defines:

```
01  PASSING-DATA.
    05  FORMAT-CODE        PIC X.
    05  PAGE-LENGTH        PIC 99.
    05  LINE-WIDTH         PIC 999.
    05  DIAMENTION         PIC 9.
```

And a sub-program is named as 'BUILD-FORM'
When input statement is:

```
CALL BUILD-FORM (FORMAT-CODE,
                 PAGE-LENGTH,
                 LINE-WIDTH)
```

Its output code will be:

```
CALL "BUILD-FORM" USING
    FORMAT-CODE
    PGE-LENGTH
    LINE-WIDTH).
```

2.02. The SET-ON statement assigns a pre-defined value in the SET-TO-ON data name to other data names.

The code: SET-ON <DATA-NAME>

will be translated as: MOVE SET-TO-ON TO <data-name>.

Example: In COBOL data-division defines:

```
01  SWITCHES.
    05  SET-TO-ON          PIC 9 VALUE 1.
    05  SET-TO-OFF         PIC 9 VALUE 0.
01  VALID-SW               PIC 9 VALUE 0.
    88  VALID-SW-OFF               VALUE 0.
    88  VALID-SW-ON                VALUE 1.
```

When input statement is coded as: SET-ON VALID-SW

Its output code will be: MOVE SET-TO-ON TO VALID-SW.

2.03. The SET-OFF statement assign a pre-defined value in the SET-TO-OFF data name to other data names.

The code: SET-OFF <data-name>

will be translated as: MOVE SET-TO-OFF TO <data-name>.

Example: In COBOL data-division defines:

```
01  SWITCHES.
    05  SET-TO-ON          PIC 9 VALUE 1.
    05  SET-TO-OFF         PIC 9 VALUE 0.
01  VALID-SW               PIC 9 VALUE 0.
    88  VALID-SW-OFF             VALUE 0.
    88  VALID-SW-ON              VALUE 1.
```

When input statemetn is coded as: SET-ON VALID-SW

Its output code will be: MOVE SET-TOOFF TO VALID-SW.

2.04. The ENABLE statement assigns a pre-defined value to the data items of an array structure.

The code:

ENABLE <array-name>(<index-1>[, <index-n>])

will be translated as below:

```
MOVE ENABLE-<array-id> TO
    <array-id> (<index-1>)
    [<array-id> (<index-n>)].
```

Example: In COBOL data-division defines:

```
01  MISC-VARS.
    05  FIRST              PIC 9  VALUE 0.
    05  SECOND             PIC 9  VALUE 1.
    05  THIRD              PIC 9  VALUE 2.
    05  FOURTH             PIC 9  VALUE 3.
    05  ENABLE-EXEC        PIC 99 VALUE 99.
    05  DISABLE-EXEC       PIC 99 VALUE 0.
01  EXEC                   OCCURS 4 TIMES PIC 99.
```

When input statement is coded as:

```
ENABLE EXEC (FIRST, SECOND, THIRD, FOURTH)
```

Its output code will be:

```
MOVE ENABLE-EXEC TO
    ENABLE (FIRST)
    ENABLE (SECOND)
    ENABLE (THIRD)
    ENABLE (FOURTH).
```

2.05. The DISABLE statement assigns pre-defined value to the data items of an array structure.

The code:

```
DISABLE <array-name> (<index-1>[, <index-n>])
```

will be translated as below:

```
MOVE DISABLE-<array-id> TO
    <array-id> (<index-1>)
    [<array-id> (<index-n>)].
```

Example: In COBOL data-division defines:

```
01  MISC-VARS.
    05  FIRST              PIC 9  VALUE 0.
    05  SECOND             PIC 9  VALUE 1.
    05  THIRD              PIC 9  VALUE 2.
    05  FOURTH             PIC 9  VALUE 3.
    05  ENABLE-EXEC        PIC 99 VALUE 99.
    05  DISABLE-EXEC       PIC 99 VALUE 0.
01  EXEC                   OCCURS 4 TIMES PIC 99.
```

When input statement is coded as:

```
DISABLE EXEC (FIRST, SECOND, THIRD, FOURTH)
```

Its output code will be:

```
MOVE DISABLE-EXEC TO
    DISABLE (FIRST)
    DISABLE (SECOND)
    DISABLE (THIRD)
    DISABLE (FOURTH).
```

2.06. The IFF structure is formed by 'IFF,' 'ELSE,' and 'END-IFF' statements.

The code:

```
IFF (<condition>)
    <text-of-true>
ELSE
    <text-of-false>
END-IFF
```

will be translated as below:

```
<num>-IFF.
    IF <condition>
        NEXT SENTENCE
    ELSE
        GO TO <num>-IFF-NOT.
    <text-of-true>
    GO TO <num>-END-IFF.
<num>-IFF-NOT.
    <text-of-false>
    GO TO <num>-END-IFF.
<num>-END-IFF.
```

Example: When input statement is coded as:

```
IF (RATE GREATER 100 AND HOUR GREATER 200)
    MOVE 15 TO CODE.
    CALL ERROR-MSG (NAME, ADDR, RATE, HOUR, CODE, )
ELSE
END-IF
```

and the identification number of this structure is '123'.

Its output code will be:

```
123-IFF.
    IF RATE GREATER 100 AND HOUR GREATER 200
        NEXT SENTENCE
    ELSE
        GO TO 123-IFF-NOT.
    MOVE 15 TO CODE.
    CALL "ERROR-MSG" USING
        NAME
        ADDR
        RATE
        HOUR
        CODE.
    GO TO 123-END-IFF.
123-IFF-NOT.
    GO TO 123-END-IFF.
123-END-IFF.
```

2.07. The CASE structure is formed by 'CASE,' ':<selection>:,' and 'END-CASE' statements.

The code:

```
CASE (<data-name>)
:<select-1>:  <text-of-select-1>
[:<select-i>:  <text-of-select-i>]
:<select-n>:  <text-of-select-n>
END-CASE
```

will be translated as below:

```
<num>-CASE.
    IF <data-name> EQUAL TO <select-1>
        NEXT SENTENCE
    ELSE
        GO TO <num>-NOT-001.
    <text-of-select-1>
    GO TO <num>-END-CASE.
<num>-NOT-001.
    IF <data-name> EQUAL <select-2>
        NEXT SENTENCE
    ELSE
        GO TO <num>-NOT-002.
    <text-of-select-2>
    GO TO <num>-END-CASE.

    [more texts]

    GO TO <num>-END-CASE.
<num>-NOT-mmm.
    IF <data-name> EQUAL TO <selext-nnn>
        NEXT SENTENCE
    ELSE
        GO TO <num>-NOT-nnn.
    <text-of-select-n>
    GO TO <num>-END-CASE.
<num>-NOT-nnn.
<num>-END-CASE.
```

*** mmm is nnn - 1.

Example: When input statement is coded as:

```
CASE (SCORE)
:'A': MOVE 'EXCELLENT' TO COMMENT.
      CALL PRINT-SHEET (NAME, SCORE, COMMENT)
:'B': MOVE 'GOOD' TO COMMENT.
      CALL PRINT-SHEET (NAME, SCORE, COMMENT)
:'C': MOVE 'NOT BAD' TO COMMENT.
      CALL PRINT-SHEET (NAME, SCORE, COMMENT)
:'D': MOVE 'POOR' TO COMMENT.
      CALL PRINT-SHEET (NAME, SCORE, COMMENT)
END-CASE
```

its output code will be:

```
456-CASE.
    IF SCORE EQUAL TO 'A'
        NEXT SENTENCE
    ELSE
        GO TO 456-NOT-001.
    MOVE 'EXCELLENT' TO COMMENT.
    CALL "PRINT-SHEET" USING
        NAME,
        SCORE,
        COMMENT.
    GO TO 456-END-CASE.
456-NOT-001.
    IF SCORE EQUAL TO B-GRADE
        NEXT SENTENCE
    ELSE
        GO TO 456-NOT-002.
    MOVE 'GOOD' TO COMMENT.
    CALL "PRINT-SHEET" USING
        NAME,
        SCORE,
        COMMENT.
    GO TO 456-END-CASE.
456-NOT-002.
    IF SCORE EQUAL TO C-GRADE
        NEXT SENTENCE
    ELSE
        GO TO 456-NOT-003.
    MOVE "NOT BAD" TO COMMENT.
    CALL "PRINT-SHEET" USING
        NAME,
        SCORE,
        COMMETN.
    GO TO 456-END-CASE.
456-NOT-003.
    IF SCORE EQUAL TO D-GRADE
        NEXT SENTENCE
    ELSE
        GO TO 456-NOT-004.
    MOVE 'POOR' TO COMMENT.
    CALL "PRINT-SHEET" USING
        NAME,
        SCORE,
        COMMETN.
    GO TO <num>-END-CASE.
456-NOT-004.
456-END-CASE.
```

2.08. The CASE88 structure is formed by 'CASE88,' ':<selection>:,' and 'END-CASE88' statements.

The code:

```
CASE88
 :<select-1>:  <text-of-select-1>
 :<select-2>"  <text-of-select-2>
 [:more: <more text>]
 :<select-n>:  <text-0f-select-n>
 END-CASE88
```

will be translated as below:

```
<num>-CASE88.
    IF <select-1>
        NEXT SENTENCE
    ELSE
        GO TO <num>-NOT-001.
    <text-of-select-1>
    GO TO <num>-END-CASE88.
<num>-NOT-001.
    IF <select-2>
        NEXT SENTENCE
    ELSE
        GO TO <num>-NOT-002.
    <text-of-select-2>
    GO TO <num>-END-CASE88.

    [move COBOL paragraphes]

    GO TO <num>-END-CASE88.
<num>-NOT-mmm.
    IF <select-n>
        NEXT SENTENCE
    ELSE
        GO TO <num>-NOT-nnn.
    <text-of select-n>
    GO TO <num>-END-CASE88.
<num>-NOT-nnn.
<num>-END-CASE88.
```

*** mmm is nnn - 1

Example: In COBOL data division defines:

```
01  SCORE.
    88  GRADE-A          PIC X(1) VALUE 'A'.
    88  GRADE-B          PIC X(1) VALUE 'B'.
    88  GRADE-C          PIC X(1) VALUE 'C'.
    88  GRADE-D          PIC X(1) VALUE 'D'.
```

When input statement is coded as:

```
CASE88
:GRADE-A: MOVE 'EXCELLENT' TO COMMENT.
:GRADE-B: MOVE 'GOOD' TO COMMENT.
:GRADE-C: MOVE 'NOT BAD' TO COMMENT.
:GRADE-D: MOVE 'POOR' TO COMMENT.
END-CASE
CALL PRING-SHEET (NAME, SCORE, COMMENT, )
```

and the indentification number of this structure is "123".
Its output code will be:

```
123-CASE88.
    IF GRADE-A
        NEXT SENTENCE
    ELSE
        GO TO 123-CASE88-001.
    MOVE 'EXCELLENT' TO COMMENT.
        GO TO 123-END-CASE88.
123-CASE88-001.
    IF GRADE-B
        NEXT SENTENCE
    ELSE
        GO TO <num>-CASE88-B-GRADE.
        MOVE 'GOOD' TO COMMENT.
        GO TO <num>-END-CASE88.
123-CASE88-002.
    IF GRADE-C
        NEXT SENTENCE
    ELSE
        GO TO <num>-CASE88-C-GRADE.
        MOVE 'NOT BAD' TO COMMENT.
        GO TO <num>-END-CASE88.
123-CASE88-003.
    IF GRADE-D
        NEXT SENTENCE
    ELSE
        GO TO <num>-CASE88-D-GRADE.
        MOVE'POOR' TO COMMENT.
        GO TO <num>-END-CASE88.
123-CASE88-004.
123-END-CASE88.
    CALL "PRINT-SHEET" UNSIGN
        NAME,
        SCORE,
        COMMENT.
```

2.09. The WHILE structure is formed by 'WHILE' and 'END-WHILE' statements.

The code:

```
WHILE (<condition>)
   <text>
END-WHILE
```

will be translated as below:

```
<num>-WHILE.
    IF <condtion>
        NEXT SENTENCE
    ELSE
        GO TO <num>-END-WHILE.
    <text>
    GO TO <num>-WHILE.
<num>-END-WHILE.
```

Example: When an input PDL structure is coded as:

```
MOVE 1 TO CTR.
WHILE (CTR LESS 101)
   CALL READ-DATA(NAME, SCORE)
   ADD 1 TO CTR.
END-WHILE
```

and the indentification number of this structure is "123".

Its output code will be:

```
    MOVE 1 TO CTR.
123-WHILE.
    IF CTR LESS 101
        NEXT SENTENCE
    ELSE
        GO TO 123-END-WHILE.
    CALL "READ-DATA" USING
        NAME
        SCORE.
    ADD 1 TO CTR.
    GO TO 123-WHILE.
123-END-WHILE.
```

2.10. The WHILENOT structure is formed by 'WHILENOT' and 'END-WHILENOT' statements.

The code:

```
WHILENOT (<condition>)
   <text>
END-WHILENOT
```

will be translated as below:

```
<num>-WHILENOT.
    IF <condtion>
        GO TO <num>-END-WHILENOT.
    <text>
    GO TO <num>-WHILENOT.
<num>-END-WHILENOT.
```

Example: When an input PDL structure is coded as:

```
MOVE 1 TO CTR.
WHILENOT (CTR GREATER 100)
   CALL READ-DATA(NAME, SCORE)
   ADD 1 TO CTR.
END-WHILENOT
```

and the indentification number of this structure is "456". Its output code will be:

```
    MOVE 1 TO CTR.
456-WHILENOT.
    IF CTR GREATER 100
        GO TO 456-END-WHILENOT.
    CALL "READ-DATA" USING
        NAME
        SCORE.
    ADD 1 TO CTR.
    GO TO 456-WHILENOT.
456-END-WHILENOT.
```

2.11. The REPEAT structure is formed by 'REPEAT' and 'UNTIL' statements.

The code:

```
REPEAT
   <text>
UNTIL (<condition>)
```

will be translated as below:

```
<num>-REPEAT.
    <text>
    IF <condtion>
        GO TO <num>-END-REPEAT.
    ELSE
        GO TO <num>-REPEAT.
<num>-END-REPEAT.
```

Example: When an input PDL structure is coded as:

```
MOVE 1 TO CTR.
REPEAT
   CALL READ-DATA(NAME, SCORE)
   ADD 1 TO CTR.
UNTIL (CTR EQUAL 101)
```

and the indentification number of this structure is "789". Its output code will be:

```
    MOVE 1 TO CTR.
789-WHILENOT.
    CALL "READ-DATA" USING
        NAME
        SCORE.
    ADD 1 TO CTR.
    IF CTR EQUAL 101
        GO TO 789-END-REPEAT
    ELSE
        GO TO 789-REPEAT.
789-END-REPEAT.
```

3. The Limitations and Other Coding Restrictions

Due to the limitation of software and hardware, the translator is designed under some conditions. The use of the currently develeped PDL statements has the following limitations.

1. The number of the nested control structures can not exceed ten levels.

2. Any PDL statement can be coded on any position of an "input-line", and it can continue to the next "input-line". It can not be seperated within a "word".

3. Any "word" in a PDL statement can not longer than thirty characters. Otherwise, this "word" will be treated as two or more "words".

4. The total outputs of any PDL statement can not exceed two handred and fifty-six charcters. Otherwise, it will cause to 'BUFFER OVERFLOW'.

APPENDIX B: A Comparison listing of PDL statements and their COBOL equivalences with the translation semantics and the translation instructions.

The following listing defines the meanings of the symbols which are used in the comparison listing and the translation semantics.

<condition> is a statement coded in COBOL which will be either true or false.

<word> is just a word or a delimiter like a comma, a period, or a parenthesis, etc.

<variable> is a pre-defined data name.

<select-i> can be any given constant value or a data name.

<proc-id> is the identifier of an external procedure - a program.

<parm-i> are data names used as the parameters.

<array-id> is an array data name.

<index-i> are elements of an array data name.

<num> is a three-digit number managed by the translator, it is also the unique index of the control structures.

<iii> is a three-digit number to identify the different occurrences of the CASE structures. It is always coded as '001', '002', ... 'mmm','nnn'.

'nnn' indicates the last possible occurrence.

'mmm' is a number prior to 'nnn'.

1. The KALL statement

```
PROGRAM DESIGN LANGUAGE          COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
KALL <proc-id>                   CALL "<proc-id>" USING
   ( <parm-1>,                         <parm-1>
     ......                             ......
     <parm-n> )                        <parm-n>.
```

```
TRANSLATION SEMANTICS FOR 'KALL' STATEMENT
-------------------------------------------------------------------
010  read    <proc-id>
020  match   (     if unmatched, stop.
030  write   CALL "<proc-id>" USING
040  read    <parm-i>
050  match   ,     if unmatched, go to step 070.
060                if matched, go to step 040.
070  match   )     if unmatched, stop.
080                if matched, next.
090  write   .
```

```
TRANSLATION INSTRUCTIONS FOR 'KALL' STATEMENT
-------------------------------------------------------------------
INPUT                INNER1     00
MATCH     STRING                99        (
APPEND    STRING     BUFFER     00        bCALLd"
APPEND    INNER1     BUFFER     00
APPEND    STRING     BUFFER     00        "dUSING
INPUT                INNER1     00
APPEND    STRING     BUFFER     00        c
APPEND    INNER1     BUFFER     00
MATCH     STRING                01        ,
BRANCH                         -05
MATCH     STRING                99        )
APPEND    STRING     BUFFER     00        .
ENDUP
```

2. The SET-ON statement

```
PROGRAM DESIGN LANGUAGE        COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
SET-NO  <variable>             MOVE SET-TO-ON  TO <variable>.

TRANSLATION SEMANTICS FOR 'SET-ON' STATEMENT
-------------------------------------------------------------------
010  read    <variable>
020  write   MOVE SET-TO-ON TO <variable>.

TRANSLATION INSTRUCTIONS FOR 'SET-ON' AND 'SET-OFF' STATEMENTS
-------------------------------------------------------------------
MOVE       TEMP       INNER2     00
INPUT                 INNER1     00
EQUAL      STRING     INNER2     02        SET-ON
APPEND     STRING     BUFFER     00        bMOVEdSET-TO-ONdTOd
BRANCH                           01
APPEND     STRING     BUFFER     00        bMOVEdSET-TO-OFFdTOd
APPEND     INNER1     BUFFER     00
APPEND     STRING     BUFFER     00        .
ENDUP
```

3. The SET-OFF statement

```
PROGRAM DESIGN LANGUAGE        COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
SET-OFF <variable>             MOVE SET-TO-OFF TO <variable>.

TRANSLATION SEMANTICS FOR 'SET-OFF' STATEMENT
-------------------------------------------------------------------
010  read    <variable> .
020  write   MOVE SET-TO-OFF TO <variable>.

TRANSLATION INSTRUCTIONS FOR 'SET-ON' AND 'SET-OFF' STATEMENTS
-------------------------------------------------------------------
MOVE       TEMP       INNER2     00
INPUT                 INNER1     00
EQUAL      STRING     INNER2     02        SET-ON
APPEND     STRING     BUFFER     00        bMOVEdSET-TO-ONdTOd
BRANCH                           01
APPEND     STRING     BUFFER     00        bMOVEdSET-TO-OFFdTOd
APPEND     INNER1     BUFFER     00
APPEND     STRING     BUFFER     00        .
ENDUP
```

4. The ENABLE statement

```
PROGRAM DESIGN LANGUAGE          COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
ENABLE  <array-id>               MOVE ENABLE-<array-id>  TO
      ( <index-1>,                    <array-id> ( <index-1> )
        ......                              ......
        <array-id> )                    <array-id> ( <index-n> ).
```

```
TRANSLATION SEMANTICS FOR 'ENABLE' STATEMENT
-------------------------------------------------------------------
010  read    <array-id>
020  match   (     if unmatched, stop.
030                if matched, next.
040  write   MOVE ENABLE-<array-id> TO
050  read    <index-i>
060  write   <array-id> ( <index-i> )
070  match   ,     if unmatched, go to step 090.
080                if matched, go to step 050.
090  match   )     if unmatched, stop.
100                if matched, next.
110  write   .
```

```
TRANSLATION INSTRUCTIONS FOR 'ENABLE' AND 'DISABLE' STATEMENTS
-------------------------------------------------------------------
APPEND    TEMP      INNER2     00
INPUT               INNER1     00
MATCH     STRING               99        (
EQUAL     STRING    INNER2     02        ENABLE
APPEND    STRING    BUFFER     00        bMOVEdENABLE-
BRANCH                         01
APPEND    STRING    BUFFER     00        bMOVEdDISABLE-
APPEND    INNER1    BUFFER     00
APPEND    STRING    BUFFER     00        dTO
INPUT               INNER1     00
APPEND    STRING    BUFFER     00        c
APPEND    INNER2    BUFFER     00
APPEND    STRING    BUFFER     00        (
APPEND    INNER1    BUFFER     00
APPEND    STRING    BUFFER     00        )
MATCH     STRING               01        ,
BRANCH                        -08
MATCH     STRING               99        )
APPEND    STRING    BUFFER     00        .
ENDUP
```

5. The DISABLE statement

```
  PROGRAM DESIGN LANGUAGE          COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
  DISABLE <array-id>               MOVE DISABLE-<array-id> TO
        ( <index-1>,                    <array-id> ( <index-1> )
          ......                              ......
          <array-id> )                    <array-id> ( <index-n> ).

  TRANSLATION SEMANTICS FOR 'DISABLE' STATEMENT
-------------------------------------------------------------------
  010   read     <array-id>
  020   match    (     if unmatched, stop.
  030                  if matched, next.
  040   write    MOVE DISABLE-<array-id> TO
  050   read     <index-i>
  060   write    <array-id> ( <index-i> )
  070   match    ,     if unmatched, go to step 090.
  080                  if matched, go to step 050.
  090   match    )     if unmatched, stop.
  100                  if matched, next.
  110   write    .

  TRANSLATION INSTRUCTIONS FOR 'ENABLE' AND 'DISABLE' STATEMENTS
-------------------------------------------------------------------
  APPEND     TEMP       INNER2      00
  INPUT                 INNER1      00
  MATCH      STRING                 99        (
  EQUAL      STRING     INNER2      02        ENABLE
  APPEND     STRING     BUFFER      00        bMOVEdENABLE-
  BRANCH                            01
  APPEND     STRING     BUFFER      00        bMOVEdDISABLE-
  APPEND     INNER1     BUFFER      00
  APPEND     STRING     BUFFER      00        dTO
  INPUT                 INNER1      00
  APPEND     STRING     BUFFER      00        c
  APPEND     INNER2     BUFFER      00
  APPEND     STRING     BUFFER      00        (
  APPEND     INNER1     BUFFER      00
  APPEND     STRING     BUFFER      00        )
  MATCH      STRING                 01        ,
  BRANCH                           -08
  MATCH      STRING                 99        )
  APPEND     STRING     BUFFER      00        .
  ENDUP
```

6. The IFF structure

```
PROGRAM DESIGN LANGUAGE          COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
IFF (<condition>)              <num>-IFF.
     .                             IF <condition> NEXT SENTENCE
     .                             ELSE GO TO <num>-IF-NOT.
{Program text of}                  {Program text of}
{condition  true}                  {condition  true}
ELSE                               GO TO <num>-END-IFF.
                               <num>-IF-NOT.
{Program text of}                  {Program text of}
{condition false}                  {condition false}
END-IFF                            GO TO <num>-END-IFF.
                               <num>-END-IFF.
```

TRANSLATION SEMANTICS FOR 'IFF' STATEMENT

```
-------------------------------------------------------------------
010  match   (     if unmatched, stop.
020  setup   <num>
030  write   <num>-IFF.
040  write   IF
050  read    <word>
060  write   <word>
070  match   )     if unmatched, go to step 050.
080                if matched, next.
090  write   NEXT SENTENCE ELSE GO TO <num>-IFF-NOT.
-------------------------------------------------------------------
```

TRANSLATION INSTRUCTIONS FOR 'IFF' STATEMENT

```
-------------------------------------------------------------------
INCPTR                              00
MATCH     STRING                    99        (
RESET               COUNT1          00
MOVE      STRING    STORE1          00        END-IFF
MOVE      COUNT1    LABEL1          00
MOVE      COUNT1    LABEL2          00
MOVE      COUNT1    LABEL3          00
APPEND    STRING    LABEL1          00        -IFF
APPEND    STRING    LABEL2          00        -NOT
APPEND    STRING    LABEL3          00        -END-IFF
APPEND    STRING    BUFFER          00        a
APPEND    LABEL1    BUFFER          00
APPEND    STRING    BUFFER          00        .bIFd
INPUT               STORE2          00
APPEND    STORE2    BUFFER          00
APPEND    STRING    BUFFER          00        d
MATCH     STRING                    01        )
BRANCH                              01
BRANCH                             -06
APPEND    STRING    BUFFER          00        cNEXTdSENTENCEbELSEcGOdTOd
APPEND    LABEL2    BUFFER          00
APPEND    STRING    BUFFER          00        .
ENDUP
```

```
  TRANSLATION SEMANTICS FOR 'ELSE' STATEMENT
-------------------------------------------------------------------
  010  write   GO TO <num>-END-IFF.
  030  write   <num>-NOT.
-------------------------------------------------------------------

  TRANSLATION INSTRUCTIONS FOR 'ELSE' STATEMENT
-------------------------------------------------------------------
  APPEND     STRING     BUFFER     00        bGOdTOd
  APPEND     LABEL3     BUFFER     00
  APPEND     STRING     BUFFER     00        .a
  APPEND     LABEL2     BUFFER     00
  APPEND     STRING     BUFFER     00        .
  ENDUP

  TRANSLATION SEMANTICS FOR 'END-IFF' STATEMENT
-------------------------------------------------------------------
  010  write   GO TO <num>-END-IFF.
  030  write   <num>-END-IFF.

  TRANSLATION INSTRUCTIONS FOR 'END-IFF' STATEMENT
-------------------------------------------------------------------
  EQUAL      TEMP       STORE1     99
  APPEND     STRING     BUFFER     00        a
  APPEND     LABEL3     BUFFER     00
  APPEND     STRING     BUFFER     00        .
  DECPTR                           00
  ENDUP
```

7. The CASE structure

```
  PROGRAM DESIGN LANGUAGE          COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
  CASE ( <variable> )          <num>-CASE.
    :<select-1>:                   IF <variable> EQUAL TO <select-1>
                                       NEXT SENTENCE
                                   ELSE
                                       GO TO <num>-NOT-001.
       {Program text}              {Program text}
       {of  select-1}              {of  select-1}
    :<select-2>:                   GO TO <num>-END-CASE.
                                <num>-NOT-001.
                                   IF <variable> EQUAL TO <select-2>
                                       NEXT SENTENCE
                                   ELSE
                                       GO TO <num>-NOT-002.
       {Program text}              {Program text}
       {of  select-2}              {of  select-2}
                                   .......
    :<select-n>:                   GO TO <num>-END-CASE.
                                <num>-NOT-mmm.
                                   IF <variable> EQUAL TO <select-n>
                                       NEXT SENTENCE
                                   ELSE
                                       GO TO <num>-NOT-nnn.
       {Program text}              {Program text}
       {of  select-n}              {of  select-n}
  END-CASE                         GO TO <num>-END-CASE.
                                <num>-NOT-nnn.
                                <num>-END-CASE.
```

TRANSLATION SEMANTICS FOR 'CASE' STATEMENT

```
-------------------------------------------------------------------
  010  setup   <num>
  020  setup   <iii>
  030  match   (     if unmatched, stop.
  040                if matched, next.
  050  read    <variable>
  060  match   )     if unmatched, stop.
  070  write   <num>-CASE.
-------------------------------------------------------------------
```

TRANSLATION INSTRUCTIONS FOR 'CASE' AND 'CASE88' STATEMENTS

```
INCPTR                              00
RESET                 COUNT1        00
RESET                 COUNT2        00
MOVE       STRING     STORE1        00        END-
APPEND     TEMP       STORE1        00
EQUAL      STRING     STORE1        03        END-CASE
MATCH      STRING                   99        (
INPUT                 STORE2        00
MATCH      STRING                   99        )
MOVE       COUNT1     LABEL1        00
MOVE       COUNT2     LABEL2        00
MOVE       COUNT1     LABEL3        00
EQUAL      STRING     STORE1        03        END-CASE
APPEND     STRING     LABEL1        00        -CASE
APPEND     STRING     LABEL3        00        -END-CASE
BRANCH                              02
APPEND     STRING     LABEL1        00        -CASE88
APPEND     STRING     LABEL3        00        -END-CASE88
APPEND     STRING     BUFFER        00        a
APPEND     LABEL1     BUFFER        00
APPEND     STRING     BUFFER        00        .
ENDUP
```

TRANSLATION SEMANTICS FOR SELECTION STATEMENT OF CASE STRUCTURE

```
010  if <iii> equal to 001 go to step 060.
020                 else next
030  write    GO TO <num>-END-CASE.
040  write    <num>-NOT-<iii>.
050  setup    <iii>
060  read     <select-i>
070  match    :     if unmatched, stop.
080                 if matched, next.
090  write    IF <variable> EQUAL <select-i>
                   NEXT SENTENCE
              ELSE GO TO <num>-NOT-<iii>.
```

TRANSLATION INSTRUCTIONS FOR THE SELECTION STATEMENT

```
EQUAL     STRING    COUNT2    01    001
BRANCH                        08
APPEND    STRING    BUFFER    00    bGOdTOd
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00    .a
APPEND    COUNT1    BUFFER    00
APPEND    STRING    BUFFER    00    -NOT-
APPEND    LABEL2    BUFFER    00
APPEND    STRING    BUFFER    00    .
MOVE      COUNT2    LABEL2    00
RESET               COUNT2    00
INPUT               STORE3    00
MATCH     STRING              99    :
EQUAL     STRING    STORE1    04    END-CASE
APPEND    STRING    BUFFER    00    bIFd
APPEND    STORE2    BUFFER    00
APPEND    STRING    BUFFER    00    dNOTdEQUALd
BRANCH                        01
APPEND    STRING    BUFFER    00    bIFdNOTd
APPEND    STORE3    BUFFER    00
APPEND    STRING    BUFFER    00    cGOdTOd
APPEND    COUNT1    BUFFER    00
APPEND    STRING    BUFFER    00    -NOT-
APPEND    LABEL2    BUFFER    00
APPEND    STRING    BUFFER    00    .
ENDUP
```

TRANSLATION SEMANTICS FOR 'END-CASE' STATEMENT

```
010  write   GO TO <num>-NOT-<iii>.
020  write   <num>-NOT-<iii>.
030  write   <num>-END-CASE.
```

TRANSLATION INSTRUCTIONS
FOR 'END-CASE' AND 'END-CASE88' STATEMENTS

```
EQUAL     TEMP      STORE1    99
APPEND    STRING    BUFFER    00    bGOdTOd
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00    .a
APPEND    COUNT1    BUFFER    00
APPEND    STRING    BUFFER    00    -NOT-
APPEND    LABEL2    BUFFER    00
APPEND    STRING    BUFFER    00    .a
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00    .
DECPTR                        00
ENDUP
```

8. The CASE88 structure

PROGRAM DESIGN LANGUAGE	COBOL EQUIVALENT TEXT
CASE88	<num>-CASE88.
:<select-1>:	IF <select-1>
	NEXT SENTENCE
	ELSE
	GO TO <num>-NOT-001.
{Program text}	{Program text}
{of select-1}	{of select-1}
:<select-2>:	GO TO <num>-END-CASE88.
	<num>-NOT-001.
	IF <select-2>
	NEXT SENTENCE
	ELSE
	GO TO <num>-NOT-002.
{Program text}	{Program text}
{of select-2}	{of select-2}
	.....
:<select-n>:	GO TO <num>-END-CASE88.
	<num>-NOT-mmm.
	IF <select-n>
	NEXT SENTENCE
	ELSE
	GO TO <num>-NOT-nnn.
{Program text}	{Program text}
{of select-n}	{of select-n}
END-CASE88	GO TO <num>-END-CASE88.
	<num>-NOT-nnn.
	<num>-END-CASE88.

TRANSLATION SEMANTICS FOR 'CASE88' STATEMENT

010	setup	<num>
020	setup	<iii>
030	write	<num>-CASE88.

TRANSLATION INSTRUCTIONS FOR 'CASE' AND 'CASE88' STATEMENTS

```
INCPTR                              00
RESET                 COUNT1        00
RESET                 COUNT2        00
MOVE        STRING    STORE1        00        END-
APPEND      TEMP      STORE1        00
EQUAL       STRING    STORE1        03        END-CASE
MATCH       STRING                  99        (
INPUT                 STORE2        00
MATCH       STRING                  99        )
MOVE        COUNT1    LABEL1        00
MOVE        COUNT2    LABEL2        00
MOVE        COUNT1    LABEL3        00
EQUAL       STRING    STORE1        03        END-CASE
APPEND      STRING    LABEL1        00        -CASE
APPEND      STRING    LABEL3        00        -END-CASE
BRANCH                              02
APPEND      STRING    LABEL1        00        -CASE88
APPEND      STRING    LABEL3        00        -END-CASE88
APPEND      STRING    BUFFER        00        a
APPEND      LABEL1    BUFFER        00
APPEND      STRING    BUFFER        00        .
ENDUP
```

TRANSLATION SEMANTICS
FOR SELECTION STATEMENT OF CASE88 STRUCTURE

```
010  if <iii> equal to 001 go to step 060.
020                 else next.
030  write    GO TO <num>-END-CASE88.
040  write    <num>-NOT-<iii>.
050  setup    <iii>
060  read     <select-i> .
070  match    :     if unmatched, stop.
080                 if matched, next.
090  write    IF <select-i>
                  NEXT SENTENCE
              ELSE GO TO <num>-NOT-<iii>.
```

TRANSLATION INSTRUCTIONS FOR THE SELECTION STATEMENT

```
EQUAL     STRING    COUNT2    01    001
BRANCH                        08
APPEND    STRING    BUFFER    00    bGOdTOd
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00    .a
APPEND    COUNT1    BUFFER    00
APPEND    STRING    BUFFER    00    -NOT-
APPEND    LABEL2    BUFFER    00
APPEND    STRING    BUFFER    00    .
MOVE      COUNT2    LABEL2    00
RESET               COUNT2    00
INPUT               STORE3    00
MATCH     STRING              99    :
EQUAL     STRING    STORE1    04    END-CASE
APPEND    STRING    BUFFER    00    bIFd
APPEND    STORE2    BUFFER    00
APPEND    STRING    BUFFER    00    dNOTdEQUALd
BRANCH                        01
APPEND    STRING    BUFFER    00    bIFdNOTd
APPEND    STORE3    BUFFER    00
APPEND    STRING    BUFFER    00    cGOdTOd
APPEND    COUNT1    BUFFER    00
APPEND    STRING    BUFFER    00    -NOT-
APPEND    LABEL2    BUFFER    00
APPEND    STRING    BUFFER    00    .
ENDUP
```

TRANSLATION SEMANTICS FOR 'END-CASE88' STATEMENT

```
010  write   GO TO <num>-NOT-<iii>.
020  write   <num>-NOT-<iii>.
030  write   <num>-END-CASE88.
```

TRANSLATION INSTRUCTIONS
FOR 'END-CASE' AND 'END-CASE88' STATEMENTS

```
EQUAL     TEMP      STORE1    99
APPEND    STRING    BUFFER    00    bGOdTOd
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00    .a
APPEND    COUNT1    BUFFER    00
APPEND    STRING    BUFFER    00    -NOT-
APPEND    LABEL2    BUFFER    00
APPEND    STRING    BUFFER    00    .a
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00    .
DECPTR                        00
ENDUP
```

9. The WHILE structure

```
  PROGRAM DESIGN LANGUAGE       COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
  WHILE (<condition>)           <num>-WHILE.
                                    IF <condition>
                                        NEXT SENTENCE
                                    ELSE
                                        GO TO <num>-END-WHILE.
  (Program text of)                 (Program text of)
  (condition  true)                 (condition  true)
                                    GO TO <num>-WHILE.
  END-WHILE                     <num>-END-WHILE.
```

```
  TRANSLATION SEMANTICS FOR 'WHILE' STATEMENT
-------------------------------------------------------------------
  010   setup   <num>
  020   match   (     if unmatched, stop.
  030                 if matched, next.
  040   write   <num>-WHILE.
  050   write   IF
  060   read    <word>
  070   write   <word>
  080   match   )     if unmatched, go to step 060.
  090                 if matched, next.
  100   write   NEXT SENTENCE
                ELSE  GO TO <num>-END-WHILE.
-------------------------------------------------------------------
```

```
  TRANSLATION INSTRUCTIONS FOR 'WHILE' AND 'WHILENOT' STATEMENTS
-------------------------------------------------------------------
  INCPTR                              00
  RESET                 COUNT1        00
  MOVE      STRING      STORE1        00        END-
  APPEND    TEMP        STORE1        00
  MATCH     STRING                    99        (
  MOVE      COUNT1      LABEL1        00
  MOVE      COUNT1      LABEL3        00
  EQUAL     STRING      STORE1        03        END-WHILE
  APPEND    STRING      LABEL1        00        -WHILE
  APPEND    STRING      LABEL3        00        -END-WHILE
  BRANCH                              02
  APPEND    STRING      LABEL1        00        -WHILENOT
  APPEND    STRING      LABEL3        00        -END-WHILENOT
  APPEND    STRING      BUFFER        00        a
  APPEND    LABEL1      BUFFER        00
  APPEND    STRING      BUFFER        00        .bIFd
```

```
INPUT                  STORE2      00
APPEND     STORE2      BUFFER      00
APPEND     STRING      BUFFER      00        d
MATCH      STRING                  01        )
BRANCH                             01
BRANCH                            -06
EQUAL      STRING      STORE1      02        END-WHILE
APPEND     STRING      BUFFER      00        cNEXTdSENTENCEbELSEcGOdTOd
BRANCH                             01
APPEND     STRING      BUFFER      00        cGOdTOd
APPEND     LABEL3      BUFFER      00
APPEND     STRING      BUFFER      00        .
ENDUP
```

TRANSLATION SEMANTICS FOR 'END-WHILE' STATEMENT

```
010  write   GO TO <num>-WHILE.
020  write   <num>-END-WHILE.
```

TRANSLATION INSTRUCTIONS
FOR 'END-WHILE' AND 'END-WHILENOT' STATEMENTS

```
EQUAL      TEMP        STORE1      99
APPEND     STRING      BUFFER      00        bGOdTOd
APPEND     LABEL1      BUFFER      00
APPEND     STRING      BUFFER      00        .a
APPEND     LABEL3      BUFFER      00
APPEND     STRING      BUFFER      00        .
DECPTR                             00
ENDUP
```

10. The WHILENOT structure

```
PROGRAM DESIGN LANGUAGE        COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
WHILENOT (<condition>)     <num>-WHILENOT.
                                IF <condition>
                                    GO TO <num>-END-WHILENOT.
{Program text of}               {Program text of}
{condition false}               {condition false}
                                GO TO <num>-WHILENOT.
END-WHILENOT               <num>-END-WHILENOT.
```

TRANSLATION SEMANTICS FOR 'WHILENOT' STATEMENT

```
-------------------------------------------------------------------
010  setup   <num>
020  match   (     if unmatched, stop.
030                if matched, next.
040  write   <num>-WHILENOT.
050  write   IF
060  read    <word>
070  write   <word>
080  match   )     if unmatched, go to step 060.
090                if matched, next.
100  write   GO TO <num>-END-WHILENOT.
-------------------------------------------------------------------
```

TRANSLATION INSTRUCTIONS FOR 'WHILE' AND 'WHILENOT' STATEMENTS

```
-------------------------------------------------------------------
INCPTR                              00
RESET                 COUNT1        00
MOVE       STRING     STORE1        00        END-
APPEND     TEMP       STORE1        00
MATCH      STRING                   99        (
MOVE       COUNT1     LABEL1        00
MOVE       COUNT1     LABEL3        00
EQUAL      STRING     STORE1        03        END-WHILE
APPEND     STRING     LABEL1        00        -WHILE
APPEND     STRING     LABEL3        00        -END-WHILE
BRANCH                              02
APPEND     STRING     LABEL1        00        -WHILENOT
APPEND     STRING     LABEL3        00        -END-WHILENOT
APPEND     STRING     BUFFER        00        a
APPEND     LABEL1     BUFFER        00
APPEND     STRING     BUFFER        00        .bIFd
```

```
INPUT                STORE2     00
APPEND    STORE2     BUFFER     00
APPEND    STRING     BUFFER     00        d
MATCH     STRING                01        )
BRANCH                          01
BRANCH                         -06
EQUAL     STRING     STORE1     02        END-WHILE
APPEND    STRING     BUFFER     00        cNEXTdSENTENCEbELSEcGOdTOd
BRANCH                          01
APPEND    STRING     BUFFER     00        cGOdTOd
APPEND    LABEL3     BUFFER     00
APPEND    STRING     BUFFER     00        .
ENDUP
```

TRANSLATION SEMANTICS FOR 'END-WHILENOT' STATEMENT

```
010  write   GO TO <num>-WHILENOT.
020  write   <num>-END-WHILENOT.
```

TRANSLATION INSTRUCTIONS
FOR 'END-WHILE' AND 'END-WHILENOT' STATEMENTS

```
EQUAL     TEMP       STORE1     99
APPEND    STRING     BUFFER     00        bGOdTOd
APPEND    LABEL1     BUFFER     00
APPEND    STRING     BUFFER     00        .a
APPEND    LABEL3     BUFFER     00
APPEND    STRING     BUFFER     00        .
DECPTR                          00
ENDUP
```

11. The REPEAT structure

```
PROGRAM DESIGN LANGUAGE          COBOL EQUIVALENT TEXT
-------------------------------------------------------------------
REPEAT                          <num>-REPEAT.
{Program text of}                   {Program text of}
{condition false}                   {condition false}
UNTIL (<condition>)                 IF <condition>
                                        GO TO <num>-END-REPEAT
                                    ELSE
                                        GO TO <num>-REPEAT.
                                <num>-END-REPEAT.
```

TRANSLATION SEMANTICS FOR 'REPEAT' STATEMENT

```
010  setup   <num>
020  write   <num>-REPEAT.
```

TRANSLATION INSTRUCTIONS FOR 'REPEAT' STATEMENT

```
INCPTR                              00
RESET                 COUNT1        00
MOVE       STRING     STORE1        00        UNTIL
MOVE       COUNT1     LABEL1        00
MOVE       COUNT1     LABEL3        00
APPEND     STRING     LABEL1        00        -REPEAT
APPEND     STRING     LABEL3        00        -END-REPEAT
APPEND     STRING     BUFFER        00        a
APPEND     LABEL1     BUFFER        00
APPEND     STRING     BUFFER        00        .
ENDUP
```

TRANSLATION SEMANTICS FOR 'UNTIL' STATEMENT

```
010  match   (     if unmatched, stop.
020                if matched, next.
030  write   IF
040  read    <word>
050  write   <word>
060  match   )     if unmatched, go to step 040.
070                if matched, next.
080  write   GO TO <num>-END-REPEAT.
             ELSE  GO TO <num>-REPEAT.
090  write   <num>-END-REPEAT.
```

TRANSLATION INSTRUCTIONS FOR 'UNTIL' STATEMENT

EQUAL	TEMP	STORE1	99	
MATCH	STRING		99	(
APPEND	STRING	BUFFER	00	bIFd
INPUT		STORE2	00	
APPEND	STORE2	BUFFER	00	
APPEND	STRING	BUFFER	00	d
MATCH	STRING		01	)
BRANCH			01	
BRANCH			-06	
APPEND	STRING	BUFFER	00	bGOdTOd
APPEND	LABEL3	BUFFER	00	
APPEND	STRING	BUFFER	00	bELSEcGOdTOd
APPEND	LABEL1	BUFFER	00	
APPEND	STRING	BUFFER	00	.a
APPEND	LABEL3	BUFFER	00	
APPEND	STRING	BUFFER	00	.
DECPTR			00	
ENDUP				

APPENDIX C: Module Specification and storage requirement of the translator.

1. Module Specification

There are ten subprograms in the translator to perform the translation. A translation instruction with its parameters is to employ a corresponding subprogram of the translator to fulfil an expected operation by using these parameters.

There are eleven types of translation commands defined in the translation instruction, they are coded in the first field of a translation instruction. The translation command "KEYWORD" is used by the TABLE BUILDER program to create an "address table" by collecting the PDL keyword which is specified in the last field of the "KEYWORD" instruction.

Besides "KEYWORD" command which is not shown in the "command table", there are ten types of command to defined all the translation instructions. These commands can be classified into three groups: the LOADERs, the CONTROLLERs, and the INSPECTORs. The following descriptions are the module specification of the subprograms of the translator.

The LOADER commands include RESET, INPUT, MOVE, and APPEND commands, they alter the content of the related data storages by moving data into from the source area to the target area. The source and the target stores of the loading operation are defined in the second and the third field of the "command table" (See Appendix D-3).

Command: RESET
Corresponding subprogram in translator: P_RESET
Function: To increase the COUNT1 AND COUNT2 data stores of the "stack" storage by one.
Input specification: None
Output specification: 1 to 999.

Command: INPUT
Corresponding subprogram in translator: P_INPUT
Function: To fetch a "word" from "input-line", and store this "word" into a specific store.
Input specification: The "word" is a character string of thirty characters. Data is only obtained from the "input-line" - the temporary store of holding the input PDL text. If a character string in "input-line" is longer than thirty characters, the rest part of this string will be fetched as the other "word".
Output specification: The target store can only be the data fields on an entry of the "stack" storage. If the "word" is shorter than thirty charcters, the rest

spaces of the target store are filled with blanks.

Command: MOVE
Corresponding subprogram in translator: P_MOVE
Function: To move the contents of a data store into the other store.
Input specification: The data stores of an entry of the "stack" storage.
Output specification: A data store on the same entry of the "stack" storage as that of the input.

Command: APPEND
Corresponding subprogram in translator: P_APPEND
Function: To append the non-blank contents of source store to the target store.
To truncate the execeeding portion of the target store, when append to a data store of the "stack".
To eject an error, when this process cause to an overflow occurred on the "buffer" area.
Input specification: The source area can any store of the stack storage or the "temp word" store.
Output specification: The target store can be any data store on the "stack" storage or the "buffer" area.

The CONTROLLER commands include BRANCH, INCPTR, DECPTR, and ENDUP commands, they alter the process sequence of translation instructions (the sequence of using the entries of the command table) by modifying the content of the instruction counter (the numeric variable P_INSTR in the translator) using a giving number in the fourth field of the command table.

Command: BRANCH
Corresponding subprogram in translator: P_BRANCH
Function: To do an unconditional branch on the execution of the translation instructions by adding the giving value in the fourth field of the "command table" to the instruction counter of the translator.
Input specification: Any numeric value within -99 to 99
Output specification: Any numeric value within 1 to 250

Command: INCPTR
Corresponding subprogram in translator: P_INCPTR
Function: To increase the stack pointer (a variable P_STACK of the translator) by one.
Input specification: Numeric value 1
Output specification: Any numberic value within 1 to 10

Command: DECPTR
Corresponding subprogram in translator: P_DECPTR
Function: To decrease the stack pointer (a variable P_STACK of the translator) by one.
Input specification: Numeric value 1
Output specification: Any numberic value within 1 to 10

Command: ENDUP
Corresponding subprogram in translator: P_ENDUP
Function: To indicate the translation of a PDL statement is comlicated, and to edit the contents of "buffer" on the "output line".
Input specification: None
Output specification: None

The INSPECTOR commands include MATCH and EQUAL commands, they check the situation by comparing the content of two data stores. If the comaprison is fail, the translator uses the fourth field of the "command table" to make a decision on current transaltion.

Command: MATCH
Corresponding subprogram in translator: P_MATCH
Function: To match the next "word" on the "input-line" (the "word" on the "input-line" will be fetched next) with the content of a specific store
Input specification: The data stores on "stack" entries or a "word" in the "string" table
Output specification: None

Command: EQUAL
Corresponding subprogram in translator: P_EQUAL
Function: To compare the content of two data stores
Input specification: Any data stores of "word" type
Output specification: None

2. The Storage requirement

The translator uses ninety-nine units of storages to hold PDL fragments and the semi-products of the translator. All the units are coded from one through ninety-nine. The "buffer" area which is coded as 1 takes 255 characters, the other stores are defined as a "word" of 30 characters. They are defined as below:

code	name	specifications of using this storage area
1	BUFFER	To hold temporary output of the translator
2	COUNT1	A data store on "stack" entries to hold identification number of the current PDL control structure.
3	COUNT2	A data store on "stack" entries to hold identification number for an internal control strcuture of current selection control structure.
4	INNER1	A data store on "stack" entries to used as a working storage area for the translation of the noncontrol statements in PDL.
5	INNER2	Another data store on "stack" entries to used as another working storage area for the translation of the noncontrol statements in PDL.

6	LABEL1	A data store on "stack" entries to hold the starting label of the current PDL control structure.
7	LABEL2	A data store on "stack" entries to hold the label for an internal control structure of current selection control structure.
8	LABEL3	A data store on "stack" entries to hold the ending label of the current PDL control structure.
9	STORE1	A data store on "stack" entries to used as a working storage area for the translation of the PDL control stratements.
10	STORE2	Another data store on "stack" entries to used as another working storage area for the translation of PDL control statements.
11	STORE3	The other data store on "stack" entries to used as the other working storage area for the translation of PDL control statements.
12	TEMP	A temporary storage to hold a "word" fetched from the "input-line".

The other units of the working storage area in the translator which is coded from 13 to 99 are the character strings defined in the last field of a translation instruction. Rearranged by the TABLE BUILDER program, this character strings are collected into the "string" table. And the translator reads these character string into an array data structure where the array elements are indexed by the number from 13 to 99.

APPENDIX - D

1. A listing of all the translation instructions
2. The text of the TABLE BUILDER program
3. The contents of the command table
4. The contents of the address table
5. The contents of the string table
6. A listing of dumping the variables of the TABLE BUILDER program during converting the translation instructions into the three tables

1. A listing of all the translation instructions

```
KEYWORD                                          COBOL
APPEND    STRING    BUFFER    00                 b
APPEND    TEMP      BUFFER    00
MATCH     STRING              02                 .
APPEND    STRING    BUFFER    00                 .
BRANCH                        04
INPUT               INNER1    00
APPEND    STRING    BUFFER    00                 d
APPEND    INNER1    BUFFER    00
BRANCH                        -07
ENDUP
KEYWORD                                          KALL
INPUT               INNER1    00
MATCH     STRING              99                 (
APPEND    STRING    BUFFER    00                 bCALLd"
APPEND    INNER1    BUFFER    00
APPEND    STRING    BUFFER    00                 "dUSING
INPUT               INNER1    00
APPEND    STRING    BUFFER    00                 c
APPEND    INNER1    BUFFER    00
MATCH     STRING              01                 ,
BRANCH                        -05
MATCH     STRING              99                 )
APPEND    STRING    BUFFER    00                 .
ENDUP
KEYWORD                                          SET-ON
KEYWORD                                          SET-OFF
MOVE      TEMP      INNER2    00
INPUT               INNER1    00
EQUAL     STRING    INNER2    02                 SET-ON
APPEND    STRING    BUFFER    00                 bMOVEdSET-TO-ONdTOd
BRANCH                        01
APPEND    STRING    BUFFER    00                 bMOVEdSET-TO-OFFdTOd
APPEND    INNER1    BUFFER    00
APPEND    STRING    BUFFER    00                 .
ENDUP
KEYWORD                                          ENABLE
KEYWORD                                          DISABLE
APPEND    TEMP      INNER2    00
INPUT               INNER1    00
MATCH     STRING              99                 (
EQUAL     STRING    INNER2    02                 ENABLE
APPEND    STRING    BUFFER    00                 bMOVEdENABLE-
BRANCH                        01
APPEND    STRING    BUFFER    00                 bMOVEdDISABLE-
APPEND    INNER1    BUFFER    00
APPEND    STRING    BUFFER    00                 dTO
INPUT               INNER1    00
APPEND    STRING    BUFFER    00                 c
APPEND    INNER2    BUFFER    00
APPEND    STRING    BUFFER    00                 (
APPEND    INNER1    BUFFER    00
APPEND    STRING    BUFFER    00                 )
MATCH     STRING              01                 ,
BRANCH                        -08
MATCH     STRING              99                 )
APPEND    STRING    BUFFER    00                 .
ENDUP
KEYWORD                                          CASE
KEYWORD                                          CASE88
INCPTR                        00
RESET               COUNT1    00
RESET               COUNT2    00
MOVE      STRING    STORE1    00                 END-
APPEND    TEMP      STORE1    00
EQUAL     STRING    STORE1    03                 END-CASE
```

```
MATCH     STRING                99        (
INPUT               STORE2      00
MATCH     STRING                99        )
MOVE      COUNT1    LABEL1      00
MOVE      COUNT2    LABEL2      00
MOVE      COUNT1    LABEL3      00
EQUAL     STRING    STORE1      03        END-CASE
APPEND    STRING    LABEL1      00        -CASE
APPEND    STRING    LABEL3      00        -END-CASE
BRANCH                          02
APPEND    STRING    LABEL1      00        -CASE88
APPEND    STRING    LABEL3      00        -END-CASE88
APPEND    STRING    BUFFER      00        a
APPEND    LABEL1    BUFFER      00
APPEND    STRING    BUFFER      00        .
ENDUP
KEYWORD                                   :
EQUAL     STRING    COUNT2      01        001
BRANCH                          08
APPEND    STRING    BUFFER      00        bGOdTOd
APPEND    LABEL3    BUFFER      00
APPEND    STRING    BUFFER      00        .a
APPEND    COUNT1    BUFFER      00
APPEND    STRING    BUFFER      00        -NOT-
APPEND    LABEL2    BUFFER      00
APPEND    STRING    BUFFER      00        .
MOVE      COUNT2    LABEL2      00
RESET               COUNT2      00
INPUT               STORE3      00
MATCH     STRING                99        ;
EQUAL     STRING    STORE1      04        END-CASE
APPEND    STRING    BUFFER      00        bIFd
APPEND    STORE2    BUFFER      00
APPEND    STRING    BUFFER      00        dNOTdEQUALd
BRANCH                          01
APPEND    STRING    BUFFER      00        bIFdNOTd
APPEND    STORE3    BUFFER      00
APPEND    STRING    BUFFER      00        cGOdTOd
APPEND    COUNT1    BUFFER      00
APPEND    STRING    BUFFER      00        -NOT-
APPEND    LABEL2    BUFFER      00
APPEND    STRING    BUFFER      00        .
ENDUP
KEYWORD                                   END-CASE
KEYWORD                                   END-CASE88
EQUAL     TEMP      STORE1      99
APPEND    STRING    BUFFER      00        bGOdTOd
APPEND    LABEL3    BUFFER      00
APPEND    STRING    BUFFER      00        .a
APPEND    COUNT1    BUFFER      00
APPEND    STRING    BUFFER      00        -NOT-
APPEND    LABEL2    BUFFER      00
APPEND    STRING    BUFFER      00        .a
APPEND    LABEL3    BUFFER      00
APPEND    STRING    BUFFER      00        .
DECPTR                          00
ENDUP
KEYWORD                                   IFF
INCPTR                          00
MATCH     STRING                99        (
RESET               COUNT1      00
MOVE      STRING    STORE1      00        END-IFF
MOVE      COUNT1    LABEL1      00
MOVE      COUNT1    LABEL2      00
MOVE      COUNT1    LABEL3      00
APPEND    STRING    LABEL1      00        -IFF
```

```
APPEND    STRING    LABEL2     00      -NOT
APPEND    STRING    LABEL3     00      -END-IFF
APPEND    STRING    BUFFER     00      a
APPEND    LABEL1    BUFFER     00
APPEND    STRING    BUFFER     00      .bIFd
INPUT               STORE2     00
APPEND    STORE2    BUFFER     00
APPEND    STRING    BUFFER     00      d
MATCH     STRING               01      )
BRANCH                         01
BRANCH                        -06
APPEND    STRING    BUFFER     00      cNEXTdSENTENCEbELSEcGOdTOd
APPEND    LABEL2    BUFFER     00
APPEND    STRING    BUFFER     00      .
ENDUP
KEYWORD                                 ELSE
APPEND    STRING    BUFFER     00      bGOdTOd
APPEND    LABEL3    BUFFER     00
APPEND    STRING    BUFFER     00      .a
APPEND    LABEL2    BUFFER     00
APPEND    STRING    BUFFER     00      .
ENDUP
KEYWORD                                 END-IFF
EQUAL     TEMP      STORE1     99
APPEND    STRING    BUFFER     00      a
APPEND    LABEL3    BUFFER     00
APPEND    STRING    BUFFER     00      .
DECPTR                         00
ENDUP
KEYWORD                                 REPEAT
INCPTR                         00
RESET               COUNT1     00
MOVE      STRING    STORE1     00      UNTIL
MOVE      COUNT1    LABEL1     00
MOVE      COUNT1    LABEL3     00
APPEND    STRING    LABEL1     00      -REPEAT
APPEND    STRING    LABEL3     00      -END-REPEAT
APPEND    STRING    BUFFER     00      .a
APPEND    LABEL1    BUFFER     00
APPEND    STRING    BUFFER     00      .
ENDUP
KEYWORD                                 UNTIL
EQUAL     TEMP      STORE1     99
MATCH     STRING               99      (
APPEND    STRING    BUFFER     00      bIFd
INPUT               STORE2     00
APPEND    STORE2    BUFFER     00
APPEND    STRING    BUFFER     00      d
MATCH     STRING               01      )
BRANCH                         01
BRANCH                        -06
APPEND    STRING    BUFFER     00      bGOdTOd
APPEND    LABEL3    BUFFER     00
APPEND    STRING    BUFFER     00      bELSEcGOdTOd
APPEND    LABEL1    BUFFER     00
APPEND    STRING    BUFFER     00      .a
APPEND    LABEL3    BUFFER     00
APPEND    STRING    BUFFER     00      .
DECPTR                         00
ENDUP
KEYWORD                                 WHILE
KEYWORD                                 WHILENOT
INCPTR                         00
RESET               COUNT1     00
MOVE      STRING    STORE1     00      END-
APPEND    TEMP      STORE1     00
MATCH     STRING               99      (
```

```
MOVE      COUNT1    LABEL1    00
MOVE      COUNT1    LABEL3    00
EQUAL     STRING    STORE1    03        END-WHILE
APPEND    STRING    LABEL1    00        -WHILE
APPEND    STRING    LABEL3    00        -END-WHILE
BRANCH                        02
APPEND    STRING    LABEL1    00        -WHILENOT
APPEND    STRING    LABEL3    00        -END-WHILENOT
APPEND    STRING    BUFFER    00        a
APPEND    LABEL1    BUFFER    00
APPEND    STRING    BUFFER    00        .bIFd
INPUT               STORE2    00
APPEND    STORE2    BUFFER    00
APPEND    STRING    BUFFER    00        d
MATCH     STRING              01        )
BRANCH                        01
BRANCH                       -06
EQUAL     STRING    STORE1    02        END-WHILE
APPEND    STRING    BUFFER    00        cNEXTdSENTENCEbELSEcGOdTOd
BRANCH                        01
APPEND    STRING    BUFFER    00        cGOdTOd
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00        .
ENDUP
KEYWORD                                 END-WHILE
KEYWORD                                 END-WHILENOT
EQUAL     TEMP      STORE1    99
APPEND    STRING    BUFFER    00        bGOdTOd
APPEND    LABEL1    BUFFER    00
APPEND    STRING    BUFFER    00        .a
APPEND    LABEL3    BUFFER    00
APPEND    STRING    BUFFER    00        .
DECPTR                        00
ENDUP
```

2. The text of the TABLE BUILDER program

```
PROGRAM TABLE_BUILDER (FINPUT, FTABLE, FKEYWD, FCHSTR, OUTPUT);

(*      This program converts the translation semantics into
    three file in numeric codes as the translation references   *)

CONST
   $NAME_LEN = 10;    (* length of name in translation semantics *)
   $WORD_LEN = 30;    (* length of word in translation semantics *)
   $TOTAL_INSTR = 10;          (* number of registered commands *)
   $TOTAL_STORE = 12;          (* number of registered storages *)

TYPE
   T_NAME = PACKED ARRAY [1..$NAME_LEN] OF CHAR;
   T_WORD = PACKED ARRAY [1..$WORD_LEN] OF CHAR;
   T_DATA = RECORD
       COMMAND, INSTORE, OUTAREA, BRANCH: T_NAME;
       STRINGS: T_WORD;
       UNUSED_SPACE: T_NAME                .          END;

VAR
   FINPUT: FILE OF T_DATA;      (* file of translation semantics *)
   FTABLE,                      (* file of the translation codes *)
   FKEYWD,                      (* file of the PDL keywords      *)
   FCHSTR: TEXT;                (* file of the character strings *)
   VA_INSTR:                    (* the registed trans. commands  *)
       ARRAY[1..$TOTAL_INSTR] OF T_NAME;
   VA_STORE:                    (* the registed runtime storages *)
       ARRAY[1..$TOTAL_STORE] OF T_NAME;
   VA_STRIN:                    (* the temperary store of strings*)
       ARRAY[1..99] OF T_WORD;
   DATA_IN: T_DATA;             (* the input record              *)
   CMD, INS, OUT, BRA,
   CTR_STR,                     (* counter of character strings  *)
   CTR_IND: INTEGER;            (* counter of trans. semantics   *)


(* This procedure defines all the referencing texts and
   initial value of all counter variables                       *)

PROCEDURE INIT_BUILDER;
BEGIN
  CTR_STR:= 1;     CTR_IND:= 1;
  VA_INSTR [1]:= 'APPEND    ';   VA_INSTR [2]:= 'BRANCH    ';
  VA_INSTR [3]:= 'DECPTR    ';   VA_INSTR [4]:= 'ENDUP     ';
  VA_INSTR [5]:= 'EQUAL     ';   VA_INSTR [6]:= 'INCPTR    ';
  VA_INSTR [7]:= 'INPUT     ';   VA_INSTR [8]:= 'MATCH     ';
  VA_INSTR [9]:= 'MOVE      ';   VA_INSTR[10]:= 'RESET     ';
  VA_STORE [1]:= 'BUFFER    ';   VA_STORE [2]:= 'COUNT1    ';
  VA_STORE [3]:= 'COUNT2    ';   VA_STORE [4]:= 'INNER1    ';
  VA_STORE [5]:= 'INNER2    ';   VA_STORE [6]:= 'LABEL1    ';
  VA_STORE [7]:= 'LABEL2    ';   VA_STORE [8]:= 'LABEL3    ';
  VA_STORE [9]:= 'STORE1    ';   VA_STORE[10]:= 'STORE2    ';
  VA_STORE[11]:= 'STORE3    ';   VA_STORE[12]:= 'TEMP      '
END;


(* This procedure builds up the code system for the PDL
   keywords, and stores them into an array structure.           *)

PROCEDURE P_SAVE_KEYWORD;
BEGIN   WRITELN(FKEYWD, DATA_IN.STRINGS, CTR_IND: 4);
  WRITE(CTR_IND:3, '|', DATA_IN.STRINGS);   WRITELN
END;
```

```
(* This procedure returns equivalent code for each name
   by referring to two (instruction or storage) lists.          *)

FUNCTION DECODE(NAME: T_NAME; WHAT: CHAR): INTEGER;
VAR ANS, I: INTEGER;
BEGIN   ANS:= 0;
  IF WHAT = 'A' THEN BEGIN (* This name is an instruction name *)
    FOR I:= 1 TO $TOTAL_INSTR DO
      IF VA_INSTR[I] = NAME THEN ANS:= I    END
  ELSE BEGIN                          (* This name is a storage name *)
    FOR I:= 1 TO $TOTAL_STORE DO
      IF VA_STORE[I] = NAME THEN ANS:= I  END;   DECODE:= ANS;
  IF (ANS = 0) AND  (NAME <> '          ')
  THEN BEGIN WRITE('BAD NAME= ', NAME);  WRITELN  END
END;


(* This procedure manages a listing of character string
   where stores the character strings pick up from the
   instructions, and provides code for them.                    *)

FUNCTION DESTRG(STR: T_WORD): INTEGER;
VAR ANS, I: INTEGER;
BEGIN  ANS:= 0;
  FOR I:= 1 TO CTR_STR DO
    IF STR = VA_STRIN[I] THEN ANS:= I + $TOTAL_STORE;
  IF ANS = 0 THEN BEGIN  ANS:= CTR_STR + $TOTAL_STORE;
     VA_STRIN[CTR_STR]:= STR;   CTR_STR:= CTR_STR + 1;
     WRITELN(FCHSTR, STR, ANS: 5)   END;
  DESTRG:= ANS
END;


(* This procedure reads a name from instruction text,
   and refers procedure 'P_FIND_CODE' to encode it.             *)

PROCEDURE P_CONVERT_CODES;
BEGIN
  WITH DATA_IN DO BEGIN
     CMD:= DECODE(COMMAND, 'A');
     IF INSTORE = 'STRING    '
     THEN INS:= DESTRG(STRINGS)
     ELSE INS:= DECODE(INSTORE, 'S');
      OUT:= DECODE(OUTAREA, 'S');
      IF BRANCH = '          ' THEN BRA:= 0
      ELSE BEGIN BRA:= (ORD(BRANCH[2]) - ORD('0')) * 10 +
         ORD(BRANCH[3]) - ORD('0');
         IF BRANCH[1] = '-' THEN BRA:= 0 - BRA  END;
      WRITE(CTR_IND: 3, '|', CMD: 3, INS: 3, OUT: 3, BRA: 3,
            '|',  STRINGS, '|');      WRITELN  END;
   WRITELN(FTABLE, CMD: 5, INS: 5, OUT: 5, BRA: 5);
   CTR_IND:= CTR_IND + 1
 END;

(* The main procedure reads instructions one by one from
   input file, and encodes instructions by recognizing
   their attributes.
BEGIN                                                           *)
  INIT_BUILDER;  RESET(FINPUT);  REWRITE(FTABLE);
  REWRITE(FKEYWD);  REWRITE(FCHSTR);
  REPEAT
    READ(FINPUT, DATA_IN);
    WRITE(' ', DATA_IN.COMMAND:7, DATA_IN.INSTORE:7,
              DATA_IN.OUTAREA:7, DATA_IN.BRANCH:4);
    IF DATA_IN.COMMAND = 'KEYWORD    ' THEN P_SAVE_KEYWORD
    ELSE P_CONVERT_CODES
  UNTIL EOF(FINPUT)

END.
```

3. The contents of the command table

index	cmd	st1	st2	bra
1	1	13	1	0
2	1	12	1	0
3	8	14	0	2
4	1	14	1	0
5	2	0	0	4
6	7	0	4	0
7	1	15	1	0
8	1	4	1	0
9	2	0	0	-7
10	4	0	0	0
11	7	0	4	0
12	8	16	0	99
13	1	17	1	0
14	1	4	1	0
15	1	18	1	0
16	7	0	4	0
17	1	19	1	0
18	1	4	1	0
19	8	20	0	1
20	2	0	0	-5
21	8	21	0	99
22	1	14	1	0
23	4	0	0	0
24	9	12	5	0
25	7	0	4	0
26	5	22	5	2
27	1	23	1	0
28	2	0	0	1
29	1	24	1	0
30	1	4	1	0
31	1	14	1	0
32	4	0	0	0
33	1	12	5	0
34	7	0	4	0
35	8	16	0	99
36	5	25	5	2
37	1	26	1	0
38	2	9	0	1
39	1	27	1	0
40	1	4	1	0
41	1	28	1	0
42	7	0	4	0
43	1	19	1	0
44	1	5	1	0
45	1	16	1	0
46	1	4	1	0
47	1	21	1	0
48	8	20	0	1
49	2	0	0	-8
50	8	21	0	99
51	1	14	1	0
52	4	0	0	0
53	6	0	0	0
54	10	0	2	0
55	10	0	3	0
56	9	29	9	0
57	1	12	9	0
58	5	30	9	3
59	8	16	0	99
60	7	0	10	0
61	8	21	0	99
62	9	2	6	0
63	9	3	7	0
64	9	2	8	0
65	5	30	9	3
66	1	31	6	0
67	1	32	8	0
68	2	0	0	2
69	1	33	6	0
70	1	34	8	0
71	1	35	1	0
72	1	6	1	0
73	1	14	1	0
74	4	0	0	0
75	5	36	3	1
76	2	0	0	8
77	1	37	1	0
78	1	8	1	0
79	1	38	1	0
80	1	2	1	0
81	1	39	1	0
82	1	7	1	0
83	1	14	1	0
84	9	3	7	0
85	10	0	3	0
86	7	0	11	0
87	8	40	0	99
88	5	30	9	4
89	1	41	1	0
90	1	10	1	0
91	1	42	1	0
92	2	0	0	1
93	1	43	1	0
94	1	11	1	0
95	1	44	1	0
96	1	2	1	0
97	1	39	1	0
98	1	7	1	0
99	1	14	1	0
100	4	0	0	0
101	5	12	9	99
102	1	37	1	0
103	1	8	1	0
104	1	38	1	0
105	1	2	1	0
106	1	39	1	0
107	1	7	1	0
108	1	38	1	0
109	1	8	1	0
110	1	14	1	0
111	3	0	0	0
112	4	0	0	0
113	6	0	0	0
114	8	16	0	99
115	10	0	2	0
116	9	45	9	0
117	9	2	6	0
118	9	2	7	0
119	9	2	8	0
120	1	46	6	0

index: the sequence in store

cmd: the code of translation command

st1: the code of the source store

st2: the code of the target store

bra: the giving value to alter the status of translation

index	cmd	st1	st2	bra
121	1	47	7	0
122	1	48	8	0
123	1	35	1	0
124	1	6	1	0
125	1	49	1	0
126	7	0	10	0
127	1	10	1	0
128	1	15	1	0
129	8	21	0	1
130	2	0	0	1
131	2	0	0	-6
132	1	50	1	0
133	1	7	1	0
134	1	14	1	0
135	4	0	0	0
136	1	37	1	0
137	1	8	1	0
138	1	38	1	0
139	1	7	1	0
140	1	14	1	0
141	4	0	0	0
142	5	12	9	99
143	1	35	1	0
144	1	8	1	0
145	1	14	1	0
146	3	0	0	0
147	4	0	0	0
148	6	0	0	0
149	10	0	2	0
150	9	51	9	0
151	9	2	6	0
152	9	2	8	0
153	1	52	6	0
154	1	53	8	0
155	1	35	1	0
156	1	6	1	0
157	1	14	1	0
158	4	0	0	0
159	5	12	9	99
160	8	16	0	99
161	1	41	1	0
162	7	0	10	0
163	1	10	1	0
164	1	15	1	0
165	8	21	0	1
166	2	0	0	1
167	2	0	0	-6
168	1	37	1	0
169	1	8	1	0
170	1	54	1	0
171	1	6	1	0
172	1	38	1	0
173	1	8	1	0
174	1	14	1	0
175	3	0	0	0
176	4	0	0	0
177	6	0	0	0
178	10	0	2	0
179	9	29	9	0
180	1	12	9	0

index: the sequence in store

cmd: the code of translation command

st1: the code of the source store

st2: the code of the target store

bra: the giving value to alter the status of translation

index	cmd	st1	st2	bra
181	8	16	0	99
182	9	2	6	0
183	9	2	8	0
184	5	55	9	3
185	1	56	6	0
186	1	57	8	0
187	2	0	0	2
188	1	58	6	0
189	1	59	8	0
190	1	35	1	0
191	1	6	1	0
192	1	49	1	0
193	7	0	10	0
194	1	10	1	0
195	1	15	1	0
196	8	21	0	1
197	2	0	0	1
198	2	0	0	-6
199	5	55	9	2
200	1	50	1	0
201	2	0	0	1
202	1	44	1	0
203	1	8	1	0
204	1	14	1	0
205	4	0	0	0
206	5	12	9	99
207	1	37	1	0
208	1	6	1	0
209	1	38	1	0
210	1	8	1	0
211	1	14	1	0
212	3	0	0	0
213	4	0	0	0

4. The contents of the address table

PDL statement lead by this keyword	index of the first translation instruction to translate this PDL
COBOL	1
KALL	11
SET-ON	24
SET-OFF	24
ENABLE	33
DISABLE	33
CASE	53
CASE88	53
:	75
END-CASE	101
END-CASE88	101
IFF	113
ELSE	136
END-IFF	142
REPEAT	148
UNTIL	159
WHILE	177
WHILENOT	177
END-WHILE	206
END-WHILENOT	206

The keyword "COBOL" indicates any embedded COBOL sentence.

5. The contents of the string table

the character strings defined in the translation instructions	the code of this string
b	13
.	14
d	15
(	16
bCALLd"	17
"dUSING	18
c	19
,	20
)	21
SET-ON	22
bMOVEdSET-TO-ONdTOd	23
bMOVEdSET-TO-OFFdTOd	24
ENABLE	25
bMOVEdENABLE-	26
bMOVEdDISABLE-	27
dTO	28
END-	29
END-CASE	30
-CASE	31
-END-CASE	32
-CASE88	33
-END-CASE88	34
a	35
001	36
bGOdTOd	37
.a	38
-NOT-	39
:	40
bIFd	41
dNOTdEQUALd	42
bIFdNOTd	43
cGOdTOd	44
END-IFF	45
-IFF	46
-NOT	47
-END-IFF	48
.bIFd	49
cNEXTdSENTENCEbELSEcGOdTOd	50
UNTIL	51
-REPEAT	52
-END-REPEAT	53
bELSEcGOdTOd	54
END-WHILE	55
-WHILE	56
-END-WHILE	57
-WHILENOT	58
-END-WHILENOT	59

6. A listing of dumping the variables of the TABLE BUILDER program during converting the translation instructions into the three tables

original text of the translation instruction				index	codes of the translation instruction					the "constant" character string of this instruction
KEYWORD					1	COBOL				
APPEND	STRING	BUFFER		00	1	1	13	1	0	b
APPEND	TEMP	BUFFER		00	2	1	12	1	0	
MATCH	STRING			02	3	8	14	0	2	.
APPEND	STRING	BUFFER		00	4	1	14	1	0	.
BRANCH				04	5	2	0	0	4	
INPUT		INNER1		00	6	7	0	4	0	
APPEND	STRING	BUFFER		00	7	1	15	1	0	d
APPEND	INNER1	BUFFER		00	8	1	4	1	0	
BRANCH				-07	9	2	0	0	-7	
ENDUP					10	4	0	0	0	
KEYWORD					11	KALL				
INPUT		INNER1		00	11	7	0	4	0	
MATCH	STRING			99	12	8	16	0	99	(
APPEND	STRING	BUFFER		00	13	1	17	1	0	bCALLd"
APPEND	INNER1	BUFFER		00	14	1	4	1	0	
APPEND	STRING	BUFFER		00	15	1	18	1	0	"dUSING
INPUT		INNER1		00	16	7	0	4	0	
APPEND	STRING	BUFFER		00	17	1	19	1	0	c
APPEND	INNER1	BUFFER		00	18	1	4	1	0	
MATCH	STRING			01	19	8	20	0	1	,
BRANCH				-05	20	2	0	0	-5	
MATCH	STRING			99	21	8	21	0	99	)
APPEND	STRING	BUFFER		00	22	1	14	1	0	.
ENDUP					23	4	0	0	0	
KEYWORD					24	SET-ON				
KEYWORD					24	SET-OFF				
MOVE	TEMP	INNER2		00	24	9	12	5	0	
INPUT		INNER1		00	25	7	0	4	0	
EQUAL	STRING	INNER2		02	26	5	22	5	2	SET-ON
APPEND	STRING	BUFFER		00	27	1	23	1	0	bMOVEdSET-TO-ONdTOd
BRANCH				01	28	2	0	0	1	
APPEND	STRING	BUFFER		00	29	1	24	1	0	bMOVEdSET-TO-OFFdTOd
APPEND	INNER1	BUFFER		00	30	1	4	1	0	
APPEND	STRING	BUFFER		00	31	1	14	1	0	.
ENDUP					32	4	0	0	0	
KEYWORD					33	ENABLE				
KEYWORD					33	DISABLE				
APPEND	TEMP	INNER2		00	33	1	12	5	0	
INPUT		INNER1		00	34	7	0	4	0	
MATCH	STRING			99	35	8	16	0	99	(
EQUAL	STRING	INNER2		02	36	5	25	5	2	ENABLE
APPEND	STRING	BUFFER		00	37	1	26	1	0	bMOVEdENABLE-
BRANCH				01	38	2	0	0	1	
APPEND	STRING	BUFFER		00	39	1	27	1	0	bMOVEdDISABLE-
APPEND	INNER1	BUFFER		00	40	1	4	1	0	
APPEND	STRING	BUFFER		00	41	1	28	1	0	dTO
INPUT		INNER1		00	42	7	0	4	0	
APPEND	STRING	BUFFER		00	43	1	19	1	0	c
APPEND	INNER2	BUFFER		00	44	1	5	1	0	
APPEND	STRING	BUFFER		00	45	1	16	1	0	(
APPEND	INNER1	BUFFER		00	46	1	4	1	0	
APPEND	STRING	BUFFER		00	47	1	21	1	0	)
MATCH	STRING			01	48	8	20	0	1	,
BRANCH				-08	49	2	0	0	-8	
MATCH	STRING			99	50	8	21	0	99	)

original text of the translation instruction			index	codes of the translation instruction					the "constant" character string of this instruction
APPEND	STRING	BUFFER	00	51	1	14	1	0	.
ENDUP				52	4	0	0	0	
KEYWORD				53	CASE				
KEYWORD				53	CASE88				
INCPTR			00	53	6	0	0	0	
RESET		COUNT1	00	54	10	0	2	0	
RESET		COUNT2	00	55	10	0	3	0	
MOVE	STRING	STORE1	00	56	9	29	9	0	END-
APPEND	TEMP	STORE1	00	57	1	12	9	0	
EQUAL	STRING	STORE1	03	58	5	30	9	3	END-CASE
MATCH	STRING		99	59	8	16	0	99	(
INPUT		STORE2	00	60	7	0	10	0	
MATCH	STRING		99	61	8	21	0	99	)
MOVE	COUNT1	LABEL1	00	62	9	2	6	0	
MOVE	COUNT2	LABEL2	00	63	9	3	7	0	
MOVE	COUNT1	LABEL3	00	64	9	2	8	0	
EQUAL	STRING	STORE1	03	65	5	30	9	3	END-CASE
APPEND	STRING	LABEL1	00	66	1	31	6	0	-CASE
APPEND	STRING	LABEL3	00	67	1	32	8	0	-END-CASE
BRANCH			02	68	2	0	0	2	
APPEND	STRING	LABEL1	00	69	1	33	6	0	-CASE88
APPEND	STRING	LABEL3	00	70	1	34	8	0	-END-CASE88
APPEND	STRING	BUFFER	00	71	1	35	1	0	a
APPEND	LABEL1	BUFFER	00	72	1	6	1	0	
APPEND	STRING	BUFFER	00	73	1	14	1	0	.
ENDUP				74	4	0	0	0	
KEYWORD				75	:				
EQUAL	STRING	COUNT2	01	75	5	36	3	1	001
BRANCH			08	76	2	0	0	8	
APPEND	STRING	BUFFER	00	77	1	37	1	0	bGOdTOd
APPEND	LABEL3	BUFFER	00	78	1	8	1	0	
APPEND	STRING	BUFFER	00	79	1	38	1	0	,a
APPEND	COUNT1	BUFFER	00	80	1	2	1	0	
APPEND	STRING	BUFFER	00	81	1	39	1	0	-NOT-
APPEND	LABEL2	BUFFER	00	82	1	7	1	0	
APPEND	STRING	BUFFER	00	83	1	14	1	0	.
MOVE	COUNT2	LABEL2	00	84	9	3	7	0	
RESET		COUNT2	00	85	10	0	3	0	
INPUT		STORE3	00	86	7	0	11	0	
MATCH	STRING		99	87	8	40	0	99	:
EQUAL	STRING	STORE1	04	88	5	30	9	4	END-CASE
APPEND	STRING	BUFFER	00	89	1	41	1	0	bIFd
APPEND	STORE2	BUFFER	00	90	1	10	1	0	
APPEND	STRING	BUFFER	00	91	1	42	1	0	dNOTdEQUALd
BRANCH			01	92	2	0	0	1	
APPEND	STRING	BUFFER	00	93	1	43	1	0	bIFdNOTd
APPEND	STORE3	BUFFER	00	94	1	11	1	0	
APPEND	STRING	BUFFER	00	95	1	44	1	0	cGOdTOd
APPEND	COUNT1	BUFFER	00	96	1	2	1	0	
APPEND	STRING	BUFFER	00	97	1	39	1	0	-NOT-
APPEND	LABEL2	BUFFER	00	98	1	7	1	0	
APPEND	STRING	BUFFER	00	99	1	14	1	0	.
ENDUP				100	4	0	0	0	

original text of the translation instruction			index	codes of the translation instruction				the "constant" character string of this instruction
KEYWORD			101	END-CASE				
KEYWORD			101	END-CASE38				
EQUAL	TEMP	STORE1	99 101	5	12	9	99	
APPEND	STRING	BUFFER	00 102	1	37	1	0	bGOdTOd
APPEND	LABEL3	BUFFER	00 103	1	8	1	0	
APPEND	STRING	BUFFER	00 104	1	38	1	0	.a
APPEND	COUNT1	BUFFER	00 105	1	2	1	0	
APPEND	STRING	BUFFER	00 106	1	39	1	0	-NOT-
APPEND	LABEL2	BUFFER	00 107	1	7	1	0	
APPEND	STRING	BUFFER	00 108	1	38	1	0	.a
APPEND	LABEL3	BUFFER	00 109	1	8	1	0	
APPEND	STRING	BUFFER	00 110	1	14	1	0	.
DECPTR			00 111	3	0	0	0	
ENDUP			112	4	0	0	0	
KEYWORD			113	IFF				
INCPTR			00 113	6	0	0	0	
MATCH	STRING		99 114	8	16	0	99	(
RESET		COUNT1	00 115	10	0	2	0	
MOVE	STRING	STORE1	00 116	9	45	9	0	END-IFF
MOVE	COUNT1	LABEL1	00 117	9	2	6	0	
MOVE	COUNT1	LABEL2	00 118	9	2	7	0	
MOVE	COUNT1	LABEL3	00 119	9	2	8	0	
APPEND	STRING	LABEL1	00 120	1	46	6	0	-IFF
APPEND	STRING	LABEL2	00 121	1	47	7	0	-NOT
APPEND	STRING	LABEL3	00 122	1	48	8	0	-END-IFF
APPEND	STRING	BUFFER	00 123	1	35	1	0	a
APPEND	LABEL1	BUFFER	00 124	1	6	1	0	
APPEND	STRING	BUFFER	00 125	1	49	1	0	.bIFd
INPUT		STORE2	00 126	7	0	10	0	
APPEND	STORE2	BUFFER	00 127	1	10	1	0	
APPEND	STRING	BUFFER	00 128	1	15	1	0	d
MATCH	STRING		01 129	8	21	0	1	)
BRANCH			01 130	2	0	0	1	
BRANCH			-06 131	2	0	0	-6	
APPEND	STRING	BUFFER	00 132	1	50	1	0	cNEXTdSENTENCEbELSEcGOdTOd
APPEND	LABEL2	BUFFER	00 133	1	7	1	0	
APPEND	STRING	BUFFER	00 134	1	14	1	0	.
ENDUP			135	4	0	0	0	
KEYWORD			136	ELSE				
APPEND	STRING	BUFFER	00 136	1	37	1	0	bGOdTOd
APPEND	LABEL3	BUFFER	00 137	1	8	1	0	
APPEND	STRING	BUFFER	00 138	1	38	1	0	.a
APPEND	LABEL2	BUFFER	00 139	1	7	1	0	
APPEND	STRING	BUFFER	00 140	1	14	1	0	.
ENDUP			141	4	0	0	0	
KEYWORD			142	END-IFF				
EQUAL	TEMP	STORE1	99 142	5	12	9	99	
APPEND	STRING	BUFFER	00 143	1	35	1	0	a
APPEND	LABEL3	BUFFER	00 144	1	8	1	0	
APPEND	STRING	BUFFER	00 145	1	14	1	0	.
DECPTR			00 146	3	0	0	0	
ENDUP			147	4	0	0	0	
KEYWORD			148	REPEAT				
INCPTR			00 148	6	0	0	0	
RESET		COUNT1	00 149	10	0	2	0	
MOVE	STRING	STORE1	00 150	9	51	9	0	UNTIL

original text of the translation instruction			index		codes of the translation instruction				the "constant" character string of this instruction
MOVE	COUNT1	LABEL1	00	151	9	2	6	0	
MOVE	COUNT1	LABEL3	00	152	9	2	8	0	
APPEND	STRING	LABEL1	00	153	1	52	6	0	-REPEAT
APPEND	STRING	LABEL3	00	154	1	53	8	0	-END-REPEAT
APPEND	STRING	BUFFER	00	155	1	35	1	0	a
APPEND	LABEL1	BUFFER	00	156	1	6	1	0	
APPEND	STRING	BUFFER	00	157	1	14	1	0	.
ENDUP				158	4	0	0	0	
KEYWORD				159	UNTIL				
EQUAL	TEMP	STORE1	99	159	5	12	9	99	
MATCH	STRING		99	160	8	16	0	99	(
APPEND	STRING	BUFFER	00	161	1	41	1	0	bIFd
INPUT		STORE2	00	162	7	0	10	0	
APPEND	STORE2	BUFFER	00	163	1	10	1	0	
APPEND	STRING	BUFFER	00	164	1	15	1	0	d
MATCH	STRING		01	165	8	21	0	1	)
BRANCH			01	166	2	0	0	1	
BRANCH			-06	167	2	0	0	-6	
APPEND	STRING	BUFFER	00	168	1	37	1	0	bGOdTOd
APPEND	LABEL3	BUFFER	00	169	1	8	1	0	
APPEND	STRING	BUFFER	00	170	1	54	1	0	bELSEcGOdTOd
APPEND	LABEL1	BUFFER	00	171	1	6	1	0	
APPEND	STRING	BUFFER	00	172	1	38	1	0	.a
APPEND	LABEL3	BUFFER	00	173	1	8	1	0	
APPEND	STRING	BUFFER	00	174	1	14	1	0	.
DECPTR			00	175	3	0	0	0	
ENDUP				176	4	0	0	0	
KEYWORD				177	WHILE				
KEYWORD				177	WHILENOT				
INCPTR			00	177	6	0	0	0	
RESET		COUNT1	00	178	10	0	2	0	
MOVE	STRING	STORE1	00	179	9	29	9	0	END-
APPEND	TEMP	STORE1	00	180	1	12	9	0	
MATCH	STRING		99	181	8	16	0	99	(
MOVE	COUNT1	LABEL1	00	182	9	2	6	0	
MOVE	COUNT1	LABEL3	00	183	9	2	8	0	
EQUAL	STRING	STORE1	03	184	5	55	9	3	END-WHILE
APPEND	STRING	LABEL1	00	185	1	56	6	0	-WHILE
APPEND	STRING	LABEL3	00	186	1	57	8	0	-END-WHILE
BRANCH			02	187	2	0	0	2	
APPEND	STRING	LABEL1	00	188	1	58	6	0	-WHILENOT
APPEND	STRING	LABEL3	00	189	1	59	8	0	-END-WHILENOT
APPEND	STRING	BUFFER	00	190	1	35	1	0	a
APPEND	LABEL1	BUFFER	00	191	1	6	1	0	
APPEND	STRING	BUFFER	00	192	1	49	1	0	.bIFd
INPUT		STORE2	00	193	7	0	10	0	
APPEND	STORE2	BUFFER	00	194	1	10	1	0	
APPEND	STRING	BUFFER	00	195	1	15	1	0	d
MATCH	STRING		01	196	8	21	0	1	)
BRANCH			01	197	2	0	0	1	
BRANCH			-06	198	2	0	0	-6	
EQUAL	STRING	STORE1	02	199	5	55	9	2	END-WHILE
APPEND	STRING	BUFFER	00	200	1	50	1	0	cNEXTdSENTENCEbELSEcGOdTOd

original text of the translation instruction	index	codes of the translation instruction	the "constant" character string of this instruction
BRANCH	01 201	2 0 0 1	
APPEND STRING BUFFER	00 202	1 44 1 0	cGOdTOd
APPEND LABEL3 BUFFER	00 203	1 8 1 0	
APPEND STRING BUFFER	00 204	1 14 1 0	.
ENDUP	205	4 0 0 0	
KEYWORD	206	END-WHILE	
KEYWORD	206	END-WHILENOT	
EQUAL TEMP STORE1	99 206	5 12 9 99	
APPEND STRING BUFFER	00 207	1 37 1 0	bGOdTOd
APPEND LABEL1 BUFFER	00 208	1 6 1 0	
APPEND STRING BUFFER	00 209	1 38 1 0	.a
APPEND LABEL3 BUFFER	00 210	1 8 1 0	
APPEND STRING BUFFER	00 211	1 14 1 0	.
DECPTR	00 212	3 0 0 0	
ENDUP	213	4 0 0 0	

APPENDIX - E

1. The text of the translator program - PREP
2. A sample PDL text
3. The output code of the sample PDL text
4. A listing of variable contents of the PREP program during translating the sample PDL text

1. The text of the translator program

```
PROGRAM PREP (FINSTR, FKEYWD, FCHSTR, FPDLIN, FCOBOL, OUTPUT);

(*      This program uses the translation references to
     translate PDS codes into designed COBOL codes.          *)

CONST
   $CMD = 25;        (* number of translation commands        *)
   $1ST = 13;        (* first code for all character strings  *)
   $LST = 99;        (* last code for all character strings   *)
   $LINE = 80;       (* line length                           *)
   $WORD = 30;       (* word length                           *)
   $INSTR = 300;     (* total number of translation semantics *)
   $STACK = 10;      (* total number of stack entries         *)
   $BUFFER = 255;    (* length of output buffer               *)
   $MARGIN = 72;     (* right margin of COBOL statement       *)

TYPE
   T_BUFFER = ARRAY [1 .. $BUFFER] OF CHAR;
   T_LINE   = ARRAY [1 .. $LINE] OF CHAR;
   T_WORD   = PACKED ARRAY [1 .. $WORD] OF CHAR;
   T_STACK  = RECORD
   (*
      Each stack entry is defined as a record of containing 10
      character strings and their corresponding length counter.
      STORE1/LST1 : The end-marker of a control structure;
      STORE2/LST2 : The primary storage of a control structure;
      STORE3/LST3 : The secondary storage of a control structure;
      COUNT1/LCT1 : The counter for the control structures;
      COUNT2/LCT2 : The subcounter of a control structure;
      LABEL1/LLL1 : The start label of a control structure;
      LABEL2/LLL2 : The internal label of a control structure;
      LABEL3/LLL3 : The end-label of a control structure;
      INNER1/LIN1 : The primary store of a non-control statements;
      INNER2/LIN2 : The secondary store of a non-control statement.
                                                              *)
      STORE1,  STORE2,  STORE3,  COUNT1,  COUNT2,
      LABEL1,  LABEL2,  LABEL3,  INNER1,  INNER2: T_WORD;
      LST1,    LST2,    LST3,    LCT1,    LCT2,
      LLL1,    LLL2,    LLL3,    LIN1,    LIN2:  INTEGER;
   END;

VAR
   FINSTR,           (* input file of the translation semantics  *)
   FKEYWD,           (* input file of the PDL keywords           *)
   FCHSTR,           (* input file of designed character strings *)
   FPDLIN,           (* input file of the PDL source text        *)
   FCOBOL:           (* output file of COBOL equivalent codes    *)
           TEXT;
   VA_CMD,           (* array to hold command code               *)
   VA_ST1,           (* array to hold the input storage names    *)
   VA_ST2,           (* array to hold the output storage names   *)
   VA_BRA:           (* array to hold a branch after comparison  *)
           ARRAY [1 .. $INSTR] OF INTEGER;
   A_KEYWORD:        (* array to hold the PDS keyword            *)
           ARRAY [1 .. $CMD] OF T_WORD;
   A_KEYADDR:        (* array to hold starting address of trans.
                     semantics in the VA_CMD array for a keyword *)
           ARRAY [1 .. $CMD] OF INTEGER;
   A_STACK:          (* The run-time storage of trans. status    *)
           ARRAY [1 .. $STACK] OF T_STACK;
   A_CHSTR:          (* The designed char. strings to be referred*)
           ARRAY [$1ST .. $LST] OF T_WORD;
```

The text of the translator program

```
  L_CURR,              (* A temp. space of the input PDL text.      *)
  L_NULL:              (* A data structure, an array, of 80 blanks *)
         T_LINE;
  BUFFER:              (* The output pool of the translator         *)
         T_BUFFER;
  W_CURR,              (* A word spacr of 30 char. of current use   *)
  W_NULL:              (* A word of blanks                          *)
         T_WORD;
  S_VALID_CH,          (* A valid character set for PDL statements *)
  S_DELIMETER:         (* Characters used as the delimiters in PDL *)
         SET OF CHAR;
  P_RESULT,            (* A numeric value to tell the trans. status *)
  P_STACK,             (* The stack pointer                         *)
  P_NUMBER,            (* Counter of managing the COBOL label names*)
  P_INSTR,             (* The index of currently used trans. codes *)
  P_BUFFER,            (* A pointer of next available buffer space *)
  $LEN_CUR_LINE,       (* Actual length of current input line       *)
  $LEN_CUR_WORD:       (* actual length of currently accessd word   *)
         INTEGER;
  P_CASES:             (* Counter of managing the CASEs label names*)
         ARRAY [1 .. $STACK] OF INTEGER;

(* A1 --- Clean fields of current stack entry for body statement*)

PROCEDURE CLEAN_INNER;
BEGIN
  WITH A_STACK[P_STACK] DO BEGIN
    INNER1:= W_NULL;  LIN1:= 0;  INNER2:= W_NULL;  LIN2:= 0  END
END;

(* A2 --- clean all fields of current stack entry                 *)

PROCEDURE CLEAN_ALL;
BEGIN
  WITH A_STACK[P_STACK] DO BEGIN P_CASES[P_STACK]:= 0;
    STORE1:= W_NULL;  LST1:= 0;  STORE2:= W_NULL;  LST2:= 0;
    STORE3:= W_NULL;  LST3:= 0;  COUNT1:= W_NULL;  LCT1:= 0;
    COUNT2:= W_NULL;  LCT2:= 0;  LABEL1:= W_NULL;  LLL1:= 0;
    LABEL2:= W_NULL;  LLL2:= 0;  LABEL3:= W_NULL;  LLL3:= 0;
  END;  CLEAN_INNER
END;

(* A3 --- give initial value to counters and character strings  *)

PROCEDURE INIT_PREPROCESSOR;
VAR CT: INTEGER;
BEGIN
  P_NUMBER:= 0;  P_STACK:= 1;  P_RESULT:= -1;  P_BUFFER:= 0;
  FOR CT:= 1 TO $WORD DO W_NULL[CT]:= ' ';
  FOR CT:= 1 TO $LINE DO L_NULL[CT]:= ' ';
  FOR CT:= 1 TO $BUFFER DO BUFFER[CT]:= ' ';
  S_DELIMETER:= ['(', ')', ',', ':', '.', ' '];
  S_VALID_CH:=  ['A' .. 'Z', '0' .. '9',
                 '+', '-', '*', '/', '='] + S_DELIMETER;
                                                    ';
  A_STACK[1].LABEL3:= '000-END
  CLEAN_ALL;   WRITELN(FCOBOL, '       000-BEGIN.')
END;
```

The text of the translator program

```
(* B --- read referencing data into defined data storages        *)
PROCEDURE FILL_TABLES;
VAR   I, J: INTEGER;  JUNK: CHAR;
BEGIN
  RESET(FINSTR);   I:= 1;
  REPEAT
    READ(FINSTR, VA_CMD[I], VA_ST1[I], VA_ST2[I], VA_BRA[I]);
    I:= I + 1;     READLN(FINSTR)
  UNTIL EOF(FINSTR);
  RESET(FKEYWD);   I:= 1;
  REPEAT
    A_KEYWORD[I]:= W_NULL;     J:= 1;
    REPEAT
      READ(FKEYWD, A_KEYWORD[I, J]);       J:= J + 1
    UNTIL J > $WORD;
    READ(FKEYWD, JUNK);  READ(FKEYWD, A_KEYADDR[I]);
    I:= I + 1;     READLN(FKEYWD)
  UNTIL EOF (FKEYWD);
  RESET (FCHSTR);   I:= $1ST;
  REPEAT
    A_CHSTR[I]:= W_NULL;     J:= 1;
    REPEAT
      READ(FCHSTR, A_CHSTR[I, J]);       J:= J + 1
    UNTIL J > $WORD;
    I:= I + 1;     READLN(FCHSTR)
  UNTIL EOF(FCHSTR);
END;


(* C --- translate PDL code into defined COBOL code              *)

PROCEDURE TRANSLATOR;
VAR
   P_PRINT,          (* A counter of printing translation result *)
   CUR_LINE_PTR,     (* The pointer of the current input line    *)
   PRI_LINE_PTR,     (* The pointer of holding the previous pos. *)
   STORE_1,          (* The copy of the operands of tran. codes  *)
   STORE_2,
   BRANCH: INTEGER;

(* C1 --- pick up a word or a character from input line          *)

PROCEDURE GET_WORD(VAR WD: T_WORD; VAR LN: INTEGER);
BEGIN
  WD:= W_NULL;   LN:= 0;   (* clean up the temporary space     *)
  WHILE (L_CURR[CUR_LINE_PTR] = ' ') AND
        (CUR_LINE_PTR < $LEN_CUR_LINE)  DO  (* no leading blank *)
     CUR_LINE_PTR:= CUR_LINE_PTR + 1;
  IF CUR_LINE_PTR > $LEN_CUR_LINE THEN P_RESULT:= -2
  ELSE IF L_CURR[CUR_LINE_PTR] IN S_DELIMETER THEN BEGIN
          WD[1]:= L_CURR[CUR_LINE_PTR];  LN:= 1;
          CUR_LINE_PTR:= CUR_LINE_PTR + 1  END
     ELSE BEGIN
        REPEAT  LN:= LN + 1; WD[LN]:= L_CURR[CUR_LINE_PTR];
          CUR_LINE_PTR:= CUR_LINE_PTR + 1
        UNTIL (LN > $WORD) OR (CUR_LINE_PTR > $LEN_CUR_LINE)
           OR (L_CURR[CUR_LINE_PTR] IN S_DELIMETER) END;
  WRITE(WD);  IF WD = W_NULL THEN P_RESULT:= -2;
  IF WD = 'END                                ' THEN P_RESULT:= -3
END;
```

The text of the translator program

```
(* C2 --- Find the required data storage                              *)

PROCEDURE REQUIRE
  (STORE_ID: INTEGER; VAR A_WORD: T_WORD; VAR LEN: INTEGER);
BEGIN
  WITH A_STACK[P_STACK] DO CASE STORE_ID OF
     2: BEGIN A_WORD:= COUNT1; LEN:= WORDLEN(A_WORD) END;
     3: BEGIN A_WORD:= COUNT2; LEN:= WORDLEN(A_WORD) END;
     4: BEGIN A_WORD:= INNER1; LEN:= WORDLEN(A_WORD) END;
     5: BEGIN A_WORD:= INNER2; LEN:= WORDLEN(A_WORD) END;
     6: BEGIN A_WORD:= LABEL1; LEN:= WORDLEN(A_WORD) END;
     7: BEGIN A_WORD:= LABEL2; LEN:= WORDLEN(A_WORD) END;
     8: BEGIN A_WORD:= LABEL3; LEN:= WORDLEN(A_WORD) END;
     9: BEGIN A_WORD:= STORE1; LEN:= WORDLEN(A_WORD) END;
    10: BEGIN A_WORD:= STORE2; LEN:= WORDLEN(A_WORD) END;
    11: BEGIN A_WORD:= STORE3; LEN:= WORDLEN(A_WORD) END;
    12: BEGIN A_WORD:= W_CURR; LEN:= $LEN_CUR_WORD END
  END
END;

(* C3 --- attach a word to another and truncated it if too long *)

PROCEDURE CON_WD(A:T_WORD; L:INTEGER; VAR B:T_WORD; VAR M:INTEGER);
VAR I: INTEGER;
BEGIN   I:= 0;
  WHILE (M <= $WORD) AND (I <= L) DO BEGIN  M:= M + 1;
    I:= I + 1;   B[M]:= A[I]   END  ; WRITE(B)
END;

(* C4 --- attach a word to buffer, error if buffer full              *)

PROCEDURE CON_BU(A_WORD: T_WORD; A_LEN: INTEGER);
VAR I: INTEGER;
BEGIN
  I:= 1;
  WHILE (P_BUFFER <= $BUFFER) AND (I <= A_LEN) DO BEGIN
    P_BUFFER:= P_BUFFER + 1;   BUFFER[P_BUFFER]:= A_WORD[I];
    I:= I + 1  END;
  IF P_BUFFER > $BUFFER THEN P_RESULT:= 5   (* buffer full          *)
END;

(* C5 --- recognize a keyword to decide translation or copying      *)

PROCEDURE RECOGNIZER;
VAR CT: INTEGER;
BEGIN
  P_INSTR:= 1;
  FOR CT:= 1 TO $CMD DO
    IF A_KEYWORD[CT] = W_CURR THEN P_INSTR:= A_KEYADDR[CT]
END;

(* C0 --- return length of a string in word type                     *)

FUNCTION WORDLEN (A_WORD: T_WORD): INTEGER;
VAR  I: INTEGER;
BEGIN   I:= $WORD;
  WHILE (I > 0) AND (A_WORD[I] = ' ') DO I:= I - 1;
  WORDLEN:= I
END;
```

The text of the translator program

```
(* 01 --- Append a char. string to one another, truncate extra  *)

PROCEDURE P_APPEND;
VAR A_WORD: T_WORD; A_LEN, I: INTEGER;
BEGIN
  IF STORE_1 = 1 THEN P_RESULT:= 7
  ELSE IF STORE_1 = STORE_2 THEN P_RESULT:= 8
  ELSE BEGIN
    IF STORE_1 = 12
    THEN BEGIN A_WORD:= W_CURR; A_LEN:= $LEN_CUR_WORD END
    ELSE IF STORE_1 < 12 THEN REQUIRE(STORE_1, A_WORD, A_LEN)
    ELSE BEGIN
      A_WORD:= A_CHSTR[STORE_1]; A_LEN:= WORDLEN(A_WORD) END;
    IF STORE_2 = 1 THEN CON_BU(A_WORD, A_LEN)
    ELSE IF NOT(STORE_2 IN [2..11]) THEN P_RESULT:= 9
    ELSE WITH A_STACK[P_STACK] DO CASE STORE_2 OF
      2: CON_WD(A_WORD, A_LEN, COUNT1, LCT1);
      3: CON_WD(A_WORD, A_LEN, COUNT2, LCT2);
      4: CON_WD(A_WORD, A_LEN, INNER1, LIN1);
      5: CON_WD(A_WORD, A_LEN, INNER2, LIN2);
      6: CON_WD(A_WORD, A_LEN, LABEL1, LLL1);
      7: CON_WD(A_WORD, A_LEN, LABEL2, LLL2);
      8: CON_WD(A_WORD, A_LEN, LABEL3, LLL3);
      9: CON_WD(A_WORD, A_LEN, STORE1, LST1);
     10: CON_WD(A_WORD, A_LEN, STORE2, LST2);
     11: CON_WD(A_WORD, A_LEN, STORE3, LST3)  END
  END
END;

(* 02 --- branch the execution route of the translation          *)

PROCEDURE P_BRANCH;
BEGIN
  IF BRANCH = -1 THEN P_RESULT:= 11
  ELSE P_INSTR:= P_INSTR + BRANCH;
END;

(* 03 --- Decrease the stack-pointer                             *)

PROCEDURE P_DECPTR;
BEGIN
  IF P_STACK = 1 THEN P_RESULT:= 2          (* stack underflow *)
  ELSE P_STACK:= P_STACK - 1;
END;

(* 04 --- edit buffer contents into COBOL format                 *)

PROCEDURE P_ENDUP;
VAR CT, START, I, J, K: INTEGER;
    L_TEMP: T_LINE;
BEGIN
  WRITELN;  FOR I:= 1 TO 65 DO WRITE('-');  I:= 1;  J:= 66;
  REPEAT  IF J = 66 THEN BEGIN WRITELN; J:= 1 END;
    WRITE(BUFFER[I]);  I:= I + 1;  J:= J + 1
  UNTIL I > P_BUFFER;
  WRITELN;   CT:= 1;
```

The text of the translator program

```
  REPEAT  J:= CT;
    IF BUFFER[CT] IN ['a' .. 'c']
    THEN START:= (ORD(BUFFER[CT]) - ORD('a') + 2) *4;
    REPEAT  J:= J + 1;  L_TEMP:= L_NULL;
      IF (BUFFER[J] IN ['a' .. 'd']) OR (J > P_BUFFER)
      THEN K:= J
    UNTIL (BUFFER[J] IN ['a' .. 'c'])
       OR (START + J - CT = $MARGIN) OR (J > P_BUFFER);
    FOR I:= CT + 1 TO K - 1 DO IF BUFFER[I] <> 'd'
       THEN L_TEMP[START+I-CT-1]:= BUFFER[I];
    FOR I:= 1 TO $LINE DO WRITE(FCOBOL, L_TEMP[I]);
    WRITELN(FCOBOL);  CT:= K
  UNTIL CT > P_BUFFER;
  FOR CT:= 1 TO $BUFFER DO BUFFER[CT]:= ' ';
  P_BUFFER:= 0;     P_RESULT:= -1;    CLEAN_INNER
END;

(* 05 --- Compare a storage content with one another's content  *)

PROCEDURE P_EQUAL;
VAR A_WORD, B_WORD: T_WORD; A_LEN, B_LEN: INTEGER;
BEGIN
  IF STORE_1 < 2 THEN P_RESULT:= 9
  ELSE IF STORE_1 > 12 THEN A_WORD:= A_CHSTR[STORE_1]
  ELSE REQUIRE(STORE_1, A_WORD, A_LEN);
  IF (STORE_2 < 2) OR (STORE_2 > 12) THEN P_RESULT:= 9
  ELSE REQUIRE(STORE_2, B_WORD, B_LEN);
  IF A_WORD <> B_WORD THEN BEGIN
     IF BRANCH = 99 THEN P_RESULT:= 6
     ELSE P_INSTR:= P_INSTR + BRANCH   END
END;

(* 06 --- Increase the stack pointer                              *)

PROCEDURE P_INCPTR;
BEGIN
  IF P_STACK = $STACK THEN P_RESULT:= 1      (* stack  overflow *)
  ELSE P_STACK:= P_STACK + 1;   CLEAN_ALL
END;

(* 07 --- Get a word from input text to a destinated storage   *)

PROCEDURE P_INPUT;
BEGIN
  IF CUR_LINE_PTR <= $LEN_CUR_LINE
  THEN IF STORE_2 IN [2..11]
    THEN WITH A_STACK[P_STACK] DO CASE STORE_2 OF
         2: GET_WORD(COUNT1, LCT1);    3: GET_WORD(COUNT2, LCT2);
         4: GET_WORD(INNER1, LIN1);    5: GET_WORD(INNER2, LIN2);
         6: GET_WORD(LABEL1, LLL1);    7: GET_WORD(LABEL2, LLL2);
         8: GET_WORD(LABEL3, LLL3);    9: GET_WORD(STORE1, LST1);
        10: GET_WORD(STORE2, LST2);   11: GET_WORD(STORE3, LST3)
       END
    ELSE P_RESULT:= 4     (* data can only move into into stack *)
  ELSE P_RESULT:= -2
END;
```

The text of the translator program

```
(* 08 --- match a character on input line with a giving char.  *)

PROCEDURE P_MATCH;
BEGIN
  IF CUR_LINE_PTR > $LEN_CUR_LINE
  THEN  P_RESULT:= -2  (* end of input line, try next line     *)
  ELSE BEGIN PRI_LINE_PTR:= CUR_LINE_PTR;
     GET_WORD(W_CURR, $LEN_CUR_WORD);
     IF W_CURR <> A_CHSTR[STORE_1]
     THEN IF BRANCH  = 99
        THEN P_RESULT:= 3 (* unmatch the expected word, error! *)
        ELSE BEGIN CUR_LINE_PTR:= PRI_LINE_PTR;
        P_INSTR:= P_INSTR + BRANCH     END
     ELSE     END
END;

(* 09 --- Move a storage content to one another data storage    *)

PROCEDURE P_MOVE;
VAR A_WORD: T_WORD; A_LEN, I: INTEGER;
BEGIN
  IF NOT(STORE_2 IN [2..11]) THEN P_RESULT:= 9 (* must on stack *)
  ELSE IF STORE_1 = STORE_2 THEN P_RESULT:= 8     (* in  =  out *)
  ELSE BEGIN
    IF STORE_1 IN [2..12] THEN REQUIRE(STORE_1, A_WORD, A_LEN)
    ELSE BEGIN A_WORD:= A_CHSTR[STORE_1];
      A_LEN:= WORDLEN(A_WORD) END;
    WITH A_STACK[P_STACK] DO CASE STORE_2 OF
       2: BEGIN  COUNT1:= A_WORD;  LCT1:= A_LEN END;
       3: BEGIN  COUNT2:= A_WORD;  LCT2:= A_LEN END;
       4: BEGIN  INNER1:= A_WORD;  LIN1:= A_LEN END;
       5: BEGIN  INNER2:= A_WORD;  LIN2:= A_LEN END;
       6: BEGIN  LABEL1:= A_WORD;  LLL1:= A_LEN END;
       7: BEGIN  LABEL2:= A_WORD;  LLL2:= A_LEN END;
       8: BEGIN  LABEL3:= A_WORD;  LLL3:= A_LEN END;
       9: BEGIN  STORE1:= A_WORD;  LST1:= A_LEN END;
      10: BEGIN  STORE2:= A_WORD;  LST2:= A_LEN END;
      11: BEGIN  STORE3:= A_WORD;  LST3:= A_LEN END;
    END
  END
END;

(* 10 --- Reset the stack counter for COBOL label names        *)

PROCEDURE P_RESET;
VAR I, J, K: INTEGER;  A_COUNT: T_WORD;
BEGIN
  IF NOT (STORE_2 IN [2,3])
  THEN P_RESULT:= 10     (* this command only works on counters *)
  ELSE WITH A_STACK[P_STACK] DO BEGIN
    IF STORE_2 = 2
    THEN BEGIN P_NUMBER:= P_NUMBER + 1; J:= P_NUMBER END
    ELSE BEGIN P_CASES[P_STACK]:= P_CASES[P_STACK] + 1;
      J:= P_CASES[P_STACK]  END;
    K:= 100;  A_COUNT:= W_NULL;
    FOR I:= 1 TO 3 DO BEGIN
      A_COUNT[I]:= CHR(J DIV K + ORD('0'));
      J:= J MOD K;  K:= K DIV 10  END;
    IF STORE_2 = 2
    THEN  BEGIN  COUNT1:= A_COUNT;  LCT1:= 3  END
    ELSE  BEGIN  COUNT2:= A_COUNT;  LCT2:= 3  END
  END
END;
```

The text of the translator program

```
(* 99 --- main procedure of the translator                          *)
BEGIN
  FOR P_PRINT:= 1 TO 65 DO WRITE('=');  WRITELN;
  FOR P_PRINT:= 1 TO $LINE DO WRITE(L_CURR[P_PRINT]);  WRITELN;
  CUR_LINE_PTR:= 1;
  WHILE (CUR_LINE_PTR < $LEN_CUR_LINE) AND (P_RESULT > -3)
    AND (P_RESULT < 1) DO BEGIN
    IF P_RESULT = - 1 THEN BEGIN  GET_WORD(W_CURR, $LEN_CUR_WORD);
       WRITELN;   RECOGNIZER  END;
    IF (P_RESULT = -2) OR (P_RESULT = -1) THEN P_RESULT:= 0;
    REPEAT BRANCH:= VA_BRA[P_INSTR];
      STORE_1:= VA_ST1[P_INSTR];   STORE_2:= VA_ST2[P_INSTR];
      WRITELN; WRITE(' R', P_RESULT:2, ' L=', $LEN_CUR_LINE:2,
      ' C=', CUR_LINE_PTR:2, ' S=', P_STACK:2, ' N=', P_NUMBER:2,
      ' I=', P_INSTR:3, ' B=', P_BUFFER:3, ' W=',$LEN_CUR_WORD:2,
      ' [',VA_CMD[P_INSTR]:3, STORE_1:3, STORE_2:3, BRANCH:3,']');
      CASE VA_CMD[P_INSTR] OF
         1: P_APPEND;   2: P_BRANCH;   3: P_DECPTR;   4: P_ENDUP;
         5: P_EQUAL;    6: P_INCPTR;   7: P_INPUT;    8: P_MATCH;
         9: P_MOVE;    10: P_RESET;
      END;   IF P_RESULT = 0 THEN P_INSTR:= P_INSTR + 1
    UNTIL P_RESULT <> 0   END
END;

(* D  --- read a input text line and hold it                        *)

PROCEDURE GET_INPUT;
VAR CH: CHAR;   I: INTEGER;
BEGIN
  L_CURR:= L_NULL;   I:= 1;
  REPEAT
    READ(FPDLIN, CH);
    IF CH IN S_VALID_CH THEN BEGIN
       L_CURR[I]:= CH;  I:= I + 1  END
    ELSE IF CH IN ['a','<']
       THEN IF I > 1
          THEN IF CH = 'a'
             THEN BEGIN L_CURR[I]:= ' ';  I:= I - 1  END
             ELSE I:= 1
          ELSE WRITELN('THIS IS A NULL LINE!')
       ELSE
   UNTIL (I > $LINE) OR EOLN;  I:= 80;
   WHILE (I > 0) AND (L_CURR[I] = ' ') DO I:= I - 1;
   $LEN_CUR_LINE:= I
END;

(* E  --- print out error messages and translation results          *)

PROCEDURE ERROR_MESSAGE;
BEGIN
  WRITELN;  WRITELN;  WRITELN;
  CASE P_RESULT OF
    1: WRITELN('Error! Stack overflow.');
    2: WRITELN('Error! Stack underflow.');
    3: WRITELN('Error! Unmatch the expected character.');
    4: WRITELN('Error! Incorrect translation code.');
    5: WRITELN('Error! Buffer full.');
    6: WRITELN('Error! Unmatch block-end label.');
    7: WRITELN('Trans. code error! Input store must be a word.');
    8: WRITELN('Trans. code error! Input equal to output.');
    9: WRITELN('Trans. code error! Stack storages required.');
   10: WRITELN('Trans. code error! Counter field required.');
   11: WRITELN('Trans. code error! bad branch code.')
  END
END;
```

```
(* Z  --- main procedure; the primary controller                    *)
BEGIN
  INIT_PREPROCESSOR;    FILL_TABLES;
  RESET(FPDLIN);       REWRITE(FCOBOL);
  REPEAT
    GET_INPUT;      TRANSLATOR;
  UNTIL (P_RESULT > 0) OR (P_RESULT = -3);
  IF P_RESULT > 0 THEN ERROR_MESSAGE
  ELSE WRITELN(FCOBOL, '       000-END.')
END.
```

2. A sample PDL text

```
KALL EMPLOYEE  (NAME,ADDR,SSN)
CASE (EMPL-CODE)
   :1: SET-ON EXEC-CODE
   :2: IFF (SSN EQUAL BAD-SSN)
          DISABLE EXEC-STAT (EXEC-NUM, EXEC-SEQ)
          MOVE 99 TO SET-DOWN.
       ELSE
          ADD 1 TO CTR.
       END-IFF
   :3: WHILE  (EXEC-STAT NOT EQUAL 0)
          KALL EXEC (ERROR, NAME, SSN, ADDR)
       END-WHILE
END-CASE
MOVE 1 TO CTR-EMPL.
REPEAT
   KALL PRINT-FORM (CTR-EMPL, NAME, SSN)
   ADD 1 TO CTR-EMPL.
UNTIL  (CTR-EMPL LARGE THAN 100)
SET-OFF EXEC
END
```

3. The output code of the sample PDL text

```
000-BEGIN.
    CALL "EMPLOYEE" USING
         NAME
         ADDR
         SSN.
001-CASE.
    IF EMPL-CODE NOT EQUAL 1
         GO TO 001-NOT-001.
    MOVE SET-TO-ON TO EXEC-CODE.
    GO TO 001-END-CASE.
001-NOT-001.
    IF EMPL-CODE NOT EQUAL 2
         GO TO 001-NOT-002.
002-IFF.
    IF SSN EQUAL BAD-SSN
         NEXT SENTENCE
    ELSE
         GO TO 002-NOT.
    MOVE DISABLE-EXEC-STAT TO
         DISABLE(EXEC-NUM)
         DISABLE(EXEC-SEQ).
    MOVE 99 TO SET-DOWN.
    GO TO 002-END-IFF.
002-NOT.
    ADD 1 TO CTR.
002-END-IFF.
    GO TO 001-END-CASE.
001-NOT-002.
    IF EMPL-CODE NOT EQUAL 3
         GO TO 001-NOT-003.
003-WHILE.
    IF EXEC-STAT NOT EQUAL 0
         NEXT SENTENCE
    ELSE
         GO TO 003-END-WHILE.
    CALL "EXEC" USING
         ERROR
         NAME
         SSN
         ADDR.
    GO TO 003-WHILE.
003-END-WHILE.
    GO TO 001-END-CASE.
001-NOT-003.
001-END-CASE.
    MOVE 1 TO CTR-EMPL.
004-REPEAT.
    CALL "PRINT-FORM" USING
         CTR-EMPL
         NAME
         SSN.
    ADD 1 TO CTR-EMPL.
    IF CTR-EMPL LARGE THAN 100
    GO TO 004-END-REPEAT
    ELSE
         GO TO 004-REPEAT.
004-END-REPEAT.
    MOVE SET-TO-OFF TO EXEC.
000-END.
```

4. A listing of variable contents of the PREP program during translating the sample PDL text

This listing is created during the execution of the translator. The following descriptions explain the meaning of each numeric values or symbols.

'==========' line indicates the beginning point of translating a line of PDL text.

The text right after '=======' line is the current PDL input text.

A "word" after the PDL input text is a keyword read by the "GET_WORD" subprogram of the translator.

The lines after the blank line are the value of the selected variable in executing a subprogram of the translator.

'R' is the return code of the translator;
When an error occurred, the value of 'R' is any number greater than zero.
When 'R' is zero, the translator is in normal translation status.
When 'R' is '-1', it is the end of an PDL statement.
When 'R' is '-2', it is the end of the input line.
When 'R' is '-3', it is the end of entire PDL text.
'L' is the actual length of current PDL input line.
'C' is the position on the "input line" where a "word" will be fetched.
'S' is the value of the stack pointer.
'N' is the number to identify a PDL control structure.
'I' is the value of the instruction counter which is the index of the translation instruction.
'B' is the number of characters in "buffer" area.
'W' is the length of the currently used "word" on the "input line".
The numbers within '[' and ']' are the currently used translation instructions in "command table".
The character string after ']' is currently read from the "input line".

'----------' line indicates the end of translating a PDL statement.

The line(s) after the '----------' line is the contents of "buffer" area.

```
================================================================
KALL EMPLOYEE  (NAME,ADDR,SSN)
KALL

 R 0 L=30 C= 5 S= 1 N= 0 I= 11 B=  0 W= 4 [  7  0  4  0]EMPLOYEE
 R 0 L=30 C=14 S= 1 N= 0 I= 12 B=  0 W= 4 [  8 16  0 99](
 R 0 L=30 C=17 S= 1 N= 0 I= 13 B=  0 W= 1 [  1 17  1  0]
 R 0 L=30 C=17 S= 1 N= 0 I= 14 B=  7 W= 1 [  1  4  1  0]
 R 0 L=30 C=17 S= 1 N= 0 I= 15 B= 15 W= 1 [  1 18  1  0]
 R 0 L=30 C=17 S= 1 N= 0 I= 16 B= 22 W= 1 [  7  0  4  0]NAME
 R 0 L=30 C=21 S= 1 N= 0 I= 17 B= 22 W= 1 [  1 19  1  0]
 R 0 L=30 C=21 S= 1 N= 0 I= 18 B= 23 W= 1 [  1  4  1  0]
 R 0 L=30 C=21 S= 1 N= 0 I= 19 B= 27 W= 1 [  8 20  0  1],
 R 0 L=30 C=22 S= 1 N= 0 I= 20 B= 27 W= 1 [  2  0  0 -5]
 R 0 L=30 C=22 S= 1 N= 0 I= 16 B= 27 W= 1 [  7  0  4  0]ADDR
 R 0 L=30 C=26 S= 1 N= 0 I= 17 B= 27 W= 1 [  1 19  1  0]
 R 0 L=30 C=26 S= 1 N= 0 I= 18 B= 28 W= 1 [  1  4  1  0]
 R 0 L=30 C=26 S= 1 N= 0 I= 19 B= 32 W= 1 [  8 20  0  1],
 R 0 L=30 C=27 S= 1 N= 0 I= 20 B= 32 W= 1 [  2  0  0 -5]
 R 0 L=30 C=27 S= 1 N= 0 I= 16 B= 32 W= 1 [  7  0  4  0]SSN
 R 0 L=30 C=30 S= 1 N= 0 I= 17 B= 32 W= 1 [  1 19  1  0]
 R 0 L=30 C=30 S= 1 N= 0 I= 18 B= 33 W= 1 [  1  4  1  0]
 R 0 L=30 C=30 S= 1 N= 0 I= 19 B= 36 W= 1 [  8 20  0  1])
 R 0 L=30 C=30 S= 1 N= 0 I= 21 B= 36 W= 1 [  8 21  0 99])
 R 0 L=30 C=31 S= 1 N= 0 I= 22 B= 36 W= 1 [  1 14  1  0]
 R 0 L=30 C=31 S= 1 N= 0 I= 23 B= 37 W= 1 [  4  0  0  0]
----------------------------------------------------------------
bCALLd"EMPLOYEE"dUSINGcNAMEcADDRcSSN.
================================================================
 CASE (EMPL-CODE)
CASE

 R 0 L=17 C= 6 S= 1 N= 0 I= 53 B=  0 W= 4 [  6  0  0  0]
 R 0 L=17 C= 6 S= 2 N= 0 I= 54 B=  0 W= 4 [ 10  0  2  0]
 R 0 L=17 C= 6 S= 2 N= 1 I= 55 B=  0 W= 4 [ 10  0  3  0]
 R 0 L=17 C= 6 S= 2 N= 1 I= 56 B=  0 W= 4 [  9 29  9  0]
 R 0 L=17 C= 6 S= 2 N= 1 I= 57 B=  0 W= 4 [  1 12  9  0]END-CASE
 R 0 L=17 C= 6 S= 2 N= 1 I= 58 B=  0 W= 4 [  5 30  9  3]
 R 0 L=17 C= 6 S= 2 N= 1 I= 59 B=  0 W= 4 [  8 16  0 99](
 R 0 L=17 C= 8 S= 2 N= 1 I= 60 B=  0 W= 1 [  7  0 10  0]EMPL-CODE
 R 0 L=17 C=17 S= 2 N= 1 I= 61 B=  0 W= 1 [  8 21  0 99])
 R 0 L=17 C=18 S= 2 N= 1 I= 62 B=  0 W= 1 [  9  2  6  0]
 R 0 L=17 C=18 S= 2 N= 1 I= 63 B=  0 W= 1 [  9  3  7  0]
 R 0 L=17 C=18 S= 2 N= 1 I= 64 B=  0 W= 1 [  9  2  8  0]
 R 0 L=17 C=18 S= 2 N= 1 I= 65 B=  0 W= 1 [  5 30  9  3]
 R 0 L=17 C=18 S= 2 N= 1 I= 66 B=  0 W= 1 [  1 31  6  0]001-CASE
 R 0 L=17 C=18 S= 2 N= 1 I= 67 B=  0 W= 1 [  1 32  8  0]001-END-CASE
 R 0 L=17 C=18 S= 2 N= 1 I= 68 B=  0 W= 1 [  2  0  0  2]
 R 0 L=17 C=18 S= 2 N= 1 I= 71 B=  0 W= 1 [  1 35  1  0]
 R 0 L=17 C=18 S= 2 N= 1 I= 72 B=  1 W= 1 [  1  6  1  0]
 R 0 L=17 C=18 S= 2 N= 1 I= 73 B=  9 W= 1 [  1 14  1  0]
 R 0 L=17 C=18 S= 2 N= 1 I= 74 B= 10 W= 1 [  4  0  0  0]
----------------------------------------------------------------
a001-CASE.
================================================================
     :1: SET-ON EXEC-CODE
:

 R 0 L=25 C= 7 S= 2 N= 1 I= 75 B=  0 W= 1 [  5 36  3  1]
 R 0 L=25 C= 7 S= 2 N= 1 I= 76 B=  0 W= 1 [  2  0  0  8]
 R 0 L=25 C= 7 S= 2 N= 1 I= 85 B=  0 W= 1 [ 10  0  3  0]
 R 0 L=25 C= 7 S= 2 N= 1 I= 86 B=  0 W= 1 [  7  0 11  0]:
 R 0 L=25 C= 8 S= 2 N= 1 I= 87 B=  0 W= 1 [  8 40  0 99]:
 R 0 L=25 C= 9 S= 2 N= 1 I= 88 B=  0 W= 1 [  5 30  9  4]
 R 0 L=25 C= 9 S= 2 N= 1 I= 89 B=  0 W= 1 [  1 41  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I= 90 B=  4 W= 1 [  1 10  1  0]
```

```
 R 0 L=25 C= 9 S= 2 N= 1 I= 91 B= 13 W= 1 [  1 42  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I= 92 B= 24 W= 1 [  2  0  0  1]
 R 0 L=25 C= 9 S= 2 N= 1 I= 94 B= 24 W= 1 [  1 11  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I= 95 B= 25 W= 1 [  1 44  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I= 96 B= 32 W= 1 [  1  2  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I= 97 B= 35 W= 1 [  1 39  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I= 98 B= 40 W= 1 [  1  7  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I= 99 B= 43 W= 1 [  1 14  1  0]
 R 0 L=25 C= 9 S= 2 N= 1 I=100 B= 44 W= 1 [  4  0  0  0]
----------------------------------------------------------------
bIFdEMPL-CODEdNOTdEQUALd1cGOdTOd001-NOT-001.
SET-ON

 R 0 L=25 C=16 S= 2 N= 1 I= 24 B=  0 W= 6 [  9 12  5  0]
 R 0 L=25 C=16 S= 2 N= 1 I= 25 B=  0 W= 6 [  7  0  4  0]EXEC-CODE
 R 0 L=25 C=26 S= 2 N= 1 I= 26 B=  0 W= 6 [  5 22  5  2]
 R 0 L=25 C=26 S= 2 N= 1 I= 27 B=  0 W= 6 [  1 23  1  0]
 R 0 L=25 C=26 S= 2 N= 1 I= 28 B= 19 W= 6 [  2  0  0  1]
 R 0 L=25 C=26 S= 2 N= 1 I= 30 B= 19 W= 6 [  1  4  1  0]
 R 0 L=25 C=26 S= 2 N= 1 I= 31 B= 28 W= 6 [  1 14  1  0]
 R 0 L=25 C=26 S= 2 N= 1 I= 32 B= 29 W= 6 [  4  0  0  0]
----------------------------------------------------------------
bMOVEdSET-TO-ONdTOdEXEC-CODE.
================================================================
      :2: IFF (SSN EQUAL BAD-SSN)
:

 R 0 L=33 C= 8 S= 2 N= 1 I= 75 B=  0 W= 1 [  5 36  3  1]
 R 0 L=33 C= 8 S= 2 N= 1 I= 77 B=  0 W= 1 [  1 37  1  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 78 B=  7 W= 1 [  1  8  1  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 79 B= 19 W= 1 [  1 38  1  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 80 B= 21 W= 1 [  1  2  1  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 81 B= 24 W= 1 [  1 39  1  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 82 B= 29 W= 1 [  1  7  1  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 83 B= 32 W= 1 [  1 14  1  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 84 B= 33 W= 1 [  9  3  7  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 85 B= 33 W= 1 [ 10  0  3  0]
 R 0 L=33 C= 8 S= 2 N= 1 I= 86 B= 33 W= 1 [  7  0 11  0]2
 R 0 L=33 C= 9 S= 2 N= 1 I= 87 B= 33 W= 1 [  8 40  0 99]:
 R 0 L=33 C=10 S= 2 N= 1 I= 88 B= 33 W= 1 [  5 30  9  4]
 R 0 L=33 C=10 S= 2 N= 1 I= 89 B= 33 W= 1 [  1 41  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 90 B= 37 W= 1 [  1 10  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 91 B= 46 W= 1 [  1 42  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 92 B= 57 W= 1 [  2  0  0  1]
 R 0 L=33 C=10 S= 2 N= 1 I= 94 B= 57 W= 1 [  1 11  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 95 B= 58 W= 1 [  1 44  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 96 B= 65 W= 1 [  1  2  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 97 B= 68 W= 1 [  1 39  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 98 B= 73 W= 1 [  1  7  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I= 99 B= 76 W= 1 [  1 14  1  0]
 R 0 L=33 C=10 S= 2 N= 1 I=100 B= 77 W= 1 [  4  0  0  0]
----------------------------------------------------------------
bGOdTOd001-END-CASE.a001-NOT-001.bIFdEMPL-CODEdNOTdEQUALd2cGOdTOd
001-NOT-002.
IFF

 R 0 L=33 C=14 S= 2 N= 1 I=113 B=  0 W= 3 [  6  0  0  0]
 R 0 L=33 C=14 S= 3 N= 1 I=114 B=  0 W= 3 [  8 16  0 99](
 R 0 L=33 C=16 S= 3 N= 1 I=115 B=  0 W= 1 [ 10  0  2  0]
 R 0 L=33 C=16 S= 3 N= 2 I=116 B=  0 W= 1 [  9 45  9  0]
 R 0 L=33 C=16 S= 3 N= 2 I=117 B=  0 W= 1 [  9  2  6  0]
 R 0 L=33 C=16 S= 3 N= 2 I=118 B=  0 W= 1 [  9  2  7  0]
 R 0 L=33 C=16 S= 3 N= 2 I=119 B=  0 W= 1 [  9  2  8  0]
 R 0 L=33 C=16 S= 3 N= 2 I=120 B=  0 W= 1 [  1 46  6  0]002-IFF
 R 0 L=33 C=16 S= 3 N= 2 I=121 B=  0 W= 1 [  1 47  7  0]002-NOT
 R 0 L=33 C=16 S= 3 N= 2 I=122 B=  0 W= 1 [  1 48  8  0]002-END-IFF
```

```
 R 0 L=33 C=16 S= 3 N= 2 I=123 B=  0 W= 1 [  1 35  1  0]
 R 0 L=33 C=16 S= 3 N= 2 I=124 B=  1 W= 1 [  1  6  1  0]
 R 0 L=33 C=16 S= 3 N= 2 I=125 B=  8 W= 1 [  1 49  1  0]
 R 0 L=33 C=16 S= 3 N= 2 I=126 B= 13 W= 1 [  7  0 10  0]SSN
 R 0 L=33 C=19 S= 3 N= 2 I=127 B= 13 W= 1 [  1 10  1  0]
 R 0 L=33 C=19 S= 3 N= 2 I=128 B= 16 W= 1 [  1 15  1  0]
 R 0 L=33 C=19 S= 3 N= 2 I=129 B= 17 W= 1 [  8 21  0  1]EQUAL
 R 0 L=33 C=19 S= 3 N= 2 I=131 B= 17 W= 5 [  2  0  0 -6]
 R 0 L=33 C=19 S= 3 N= 2 I=126 B= 17 W= 5 [  7  0 10  0]EQUAL
 R 0 L=33 C=25 S= 3 N= 2 I=127 B= 17 W= 5 [  1 10  1  0]
 R 0 L=33 C=25 S= 3 N= 2 I=128 B= 22 W= 5 [  1 15  1  0]
 R 0 L=33 C=25 S= 3 N= 2 I=129 B= 23 W= 5 [  8 21  0  1]BAD-SSN
 R 0 L=33 C=25 S= 3 N= 2 I=131 B= 23 W= 7 [  2  0  0 -6]
 R 0 L=33 C=25 S= 3 N= 2 I=126 B= 23 W= 7 [  7  0 10  0]BAD-SSN
 R 0 L=33 C=33 S= 3 N= 2 I=127 B= 23 W= 7 [  1 10  1  0]
 R 0 L=33 C=33 S= 3 N= 2 I=128 B= 30 W= 7 [  1 15  1  0]
 R 0 L=33 C=33 S= 3 N= 2 I=129 B= 31 W= 7 [  8 21  0  1])
 R 0 L=33 C=34 S= 3 N= 2 I=130 B= 31 W= 1 [  2  0  0  1]
 R 0 L=33 C=34 S= 3 N= 2 I=132 B= 31 W= 1 [  1 50  1  0]
 R 0 L=33 C=34 S= 3 N= 2 I=133 B= 57 W= 1 [  1  7  1  0]
 R 0 L=33 C=34 S= 3 N= 2 I=134 B= 64 W= 1 [  1 14  1  0]
 R 0 L=33 C=34 S= 3 N= 2 I=135 B= 65 W= 1 [  4  0  0  0]
-------------------------------------------------------------------------
a002-IFF.bIFdSSNdEQUALdBAD-SSNdcNEXTdSENTENCEbELSEcGOdTOd002-NOT.
=========================================================================
                DISABLE EXEC-STAT (EXEC-NUM, EXEC-SEQ)
DISABLE

 R 0 L=52 C=22 S= 3 N= 2 I= 33 B=  0 W= 7 [  1 12  5  0]DISABLE
 R 0 L=52 C=22 S= 3 N= 2 I= 34 B=  0 W= 7 [  7  0  4  0]EXEC-STAT
 R 0 L=52 C=32 S= 3 N= 2 I= 35 B=  0 W= 7 [  8 16  0 99](
 R 0 L=52 C=34 S= 3 N= 2 I= 36 B=  0 W= 1 [  5 25  5  2]
 R 0 L=52 C=34 S= 3 N= 2 I= 39 B=  0 W= 1 [  1 27  1  0]
 R 0 L=52 C=34 S= 3 N= 2 I= 40 B= 14 W= 1 [  1  4  1  0]  ----
 R 0 L=52 C=34 S= 3 N= 2 I= 41 B= 23 W= 1 [  1 28  1  0]
 R 0 L=52 C=34 S= 3 N= 2 I= 42 B= 26 W= 1 [  7  0  4  0]EXEC-NUM
 R 0 L=52 C=42 S= 3 N= 2 I= 43 B= 26 W= 1 [  1 19  1  0]
 R 0 L=52 C=42 S= 3 N= 2 I= 44 B= 27 W= 1 [  1  5  1  0]
 R 0 L=52 C=42 S= 3 N= 2 I= 45 B= 34 W= 1 [  1 16  1  0]
 R 0 L=52 C=42 S= 3 N= 2 I= 46 B= 35 W= 1 [  1  4  1  0]
 R 0 L=52 C=42 S= 3 N= 2 I= 47 B= 43 W= 1 [  1 21  1  0]
 R 0 L=52 C=42 S= 3 N= 2 I= 48 B= 44 W= 1 [  8 20  0  1],
 R 0 L=52 C=43 S= 3 N= 2 I= 49 B= 44 W= 1 [  2  0  0 -8]
 R 0 L=52 C=43 S= 3 N= 2 I= 42 B= 44 W= 1 [  7  0  4  0]EXEC-SEQ
 R 0 L=52 C=52 S= 3 N= 2 I= 43 B= 44 W= 1 [  1 19  1  0]
 R 0 L=52 C=52 S= 3 N= 2 I= 44 B= 45 W= 1 [  1  5  1  0]
 R 0 L=52 C=52 S= 3 N= 2 I= 45 B= 52 W= 1 [  1 16  1  0]
 R 0 L=52 C=52 S= 3 N= 2 I= 46 B= 53 W= 1 [  1  4  1  0]
 R 0 L=52 C=52 S= 3 N= 2 I= 47 B= 61 W= 1 [  1 21  1  0]
 R 0 L=52 C=52 S= 3 N= 2 I= 48 B= 62 W= 1 [  8 20  0  1])
 R 0 L=52 C=52 S= 3 N= 2 I= 50 B= 62 W= 1 [  8 21  0 99])
 R 0 L=52 C=53 S= 3 N= 2 I= 51 B= 62 W= 1 [  1 14  1  0]
 R 0 L=52 C=53 S= 3 N= 2 I= 52 B= 63 W= 1 [  4  0  0  0]
-------------------------------------------------------------------------
bMOVEdDISABLE-EXEC-STATdTOcDISABLE(EXEC-NUM)cDISABLE(EXEC-SEQ).
=========================================================================
                MOVE 99 TO SET-DOWN.
MOVE

 R 0 L=35 C=20 S= 3 N= 2 I=  1 B=  0 W= 4 [  1 13  1  0]
 R 0 L=35 C=20 S= 3 N= 2 I=  2 B=  1 W= 4 [  1 12  1  0]
 R 0 L=35 C=20 S= 3 N= 2 I=  3 B=  5 W= 4 [  8 14  0  2]99
 R 0 L=35 C=20 S= 3 N= 2 I=  6 B=  5 W= 2 [  7  0  4  0]99
 R 0 L=35 C=23 S= 3 N= 2 I=  7 B=  5 W= 2 [  1 15  1  0]
 R 0 L=35 C=23 S= 3 N= 2 I=  8 B=  6 W= 2 [  1  4  1  0]
 R 0 L=35 C=23 S= 3 N= 2 I=  9 B=  8 W= 2 [  2  0  0 -7]
```

```
 R 0 L=35 C=23 S= 3 N= 2 I=  3 B=  8 W= 2 [  8 14  0  2]TO
 R 0 L=35 C=23 S= 3 N= 2 I=  6 B=  8 W= 2 [  7  0  4  0]TO
 R 0 L=35 C=26 S= 3 N= 2 I=  7 B=  8 W= 2 [  1 15  1  0]
 R 0 L=35 C=26 S= 3 N= 2 I=  8 B=  9 W= 2 [  1  4  1  0]
 R 0 L=35 C=26 S= 3 N= 2 I=  9 B= 11 W= 2 [  2  0  0 -7]
 R 0 L=35 C=26 S= 3 N= 2 I=  3 B= 11 W= 2 [  8 14  0  2]SET-DOWN
 R 0 L=35 C=26 S= 3 N= 2 I=  6 B= 11 W= 8 [  7  0  4  0]SET-DOWN
 R 0 L=35 C=35 S= 3 N= 2 I=  7 B= 11 W= 8 [  1 15  1  0]
 R 0 L=35 C=35 S= 3 N= 2 I=  8 B= 12 W= 8 [  1  4  1  0]
 R 0 L=35 C=35 S= 3 N= 2 I=  9 B= 20 W= 8 [  2  0  0 -7]
 R 0 L=35 C=35 S= 3 N= 2 I=  3 B= 20 W= 8 [  8 14  0  2].
 R 0 L=35 C=36 S= 3 N= 2 I=  4 B= 20 W= 1 [  1 14  1  0]
 R 0 L=35 C=36 S= 3 N= 2 I=  5 B= 21 W= 1 [  2  0  0  4]
 R 0 L=35 C=36 S= 3 N= 2 I= 10 B= 21 W= 1 [  4  0  0  0]
-----------------------------------------------------------------------
bMOVEd99dTOdSET-DOWN.
=======================================================================
                ELSE
ELSE

 R 0 L=17 C=18 S= 3 N= 2 I=136 B=  0 W= 4 [  1 37  1  0]
 R 0 L=17 C=18 S= 3 N= 2 I=137 B=  7 W= 4 [  1  8  1  0]
 R 0 L=17 C=18 S= 3 N= 2 I=138 B= 18 W= 4 [  1 38  1  0]
 R 0 L=17 C=18 S= 3 N= 2 I=139 B= 20 W= 4 [  1  7  1  0]
 R 0 L=17 C=18 S= 3 N= 2 I=140 B= 27 W= 4 [  1 14  1  0]
 R 0 L=17 C=18 S= 3 N= 2 I=141 B= 28 W= 4 [  4  0  0  0]
-----------------------------------------------------------------------
bGOdTOd002-END-IFF.a002-NOT.
=======================================================================
                    ADD 1 TO CTR.
ADD

 R 0 L=30 C=21 S= 3 N= 2 I=  1 B=  0 W= 3 [  1 13  1  0]
 R 0 L=30 C=21 S= 3 N= 2 I=  2 B=  1 W= 3 [  1 12  1  0]
 R 0 L=30 C=21 S= 3 N= 2 I=  3 B=  4 W= 3 [  8 14  0  2]1
 R 0 L=30 C=21 S= 3 N= 2 I=  6 B=  4 W= 1 [  7  0  4  0]1
 R 0 L=30 C=23 S= 3 N= 2 I=  7 B=  4 W= 1 [  1 15  1  0]
 R 0 L=30 C=23 S= 3 N= 2 I=  8 B=  5 W= 1 [  1  4  1  0]
 R 0 L=30 C=23 S= 3 N= 2 I=  9 B=  6 W= 1 [  2  0  0 -7]
 R 0 L=30 C=23 S= 3 N= 2 I=  3 B=  6 W= 1 [  8 14  0  2]TO
 R 0 L=30 C=23 S= 3 N= 2 I=  6 B=  6 W= 2 [  7  0  4  0]TO
 R 0 L=30 C=26 S= 3 N= 2 I=  7 B=  6 W= 2 [  1 15  1  0]
 R 0 L=30 C=26 S= 3 N= 2 I=  8 B=  7 W= 2 [  1  4  1  0]
 R 0 L=30 C=26 S= 3 N= 2 I=  9 B=  9 W= 2 [  2  0  0 -7]
 R 0 L=30 C=26 S= 3 N= 2 I=  3 B=  9 W= 2 [  8 14  0  2]CTR
 R 0 L=30 C=26 S= 3 N= 2 I=  6 B=  9 W= 3 [  7  0  4  0]CTR
 R 0 L=30 C=30 S= 3 N= 2 I=  7 B=  9 W= 3 [  1 15  1  0]
 R 0 L=30 C=30 S= 3 N= 2 I=  8 B= 10 W= 3 [  1  4  1  0]
 R 0 L=30 C=30 S= 3 N= 2 I=  9 B= 13 W= 3 [  2  0  0 -7]
 R 0 L=30 C=30 S= 3 N= 2 I=  3 B= 13 W= 3 [  8 14  0  2].
 R 0 L=30 C=31 S= 3 N= 2 I=  4 B= 13 W= 1 [  1 14  1  0]
 R 0 L=30 C=31 S= 3 N= 2 I=  5 B= 14 W= 1 [  2  0  0  4]
 R 0 L=30 C=31 S= 3 N= 2 I= 10 B= 14 W= 1 [  4  0  0  0]
-----------------------------------------------------------------------
bADDd1dTOdCTR.
=======================================================================
                  END-IFF
END-IFF

 R 0 L=22 C=23 S= 3 N= 2 I=142 B=  0 W= 7 [  5 12  9 99]
 R 0 L=22 C=23 S= 3 N= 2 I=143 B=  0 W= 7 [  1 35  1  0]
 R 0 L=22 C=23 S= 3 N= 2 I=144 B=  1 W= 7 [  1  8  1  0]
 R 0 L=22 C=23 S= 3 N= 2 I=145 B= 12 W= 7 [  1 14  1  0]
 R 0 L=22 C=23 S= 3 N= 2 I=146 B= 13 W= 7 [  3  0  0  0]
 R 0 L=22 C=23 S= 2 N= 2 I=147 B= 13 W= 7 [  4  0  0  0]
-----------------------------------------------------------------------
a002-END-IFF.
=======================================================================
```

```
                :3: WHILE  (EXEC-STAT NOT EQUAL 0)
:

 R 0 L=46 C=14 S= 2 N= 2 I= 75 B=  0 W= 1 [  5 36  3  1]
 R 0 L=46 C=14 S= 2 N= 2 I= 77 B=  0 W= 1 [  1 37  1  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 78 B=  7 W= 1 [  1  8  1  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 79 B= 19 W= 1 [  1 38  1  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 80 B= 21 W= 1 [  1  2  1  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 81 B= 24 W= 1 [  1 39  1  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 82 B= 29 W= 1 [  1  7  1  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 83 B= 32 W= 1 [  1 14  1  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 84 B= 33 W= 1 [  9  3  7  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 85 B= 33 W= 1 [ 10  0  3  0]
 R 0 L=46 C=14 S= 2 N= 2 I= 86 B= 33 W= 1 [  7  0 11  0]3
 R 0 L=46 C=15 S= 2 N= 2 I= 87 B= 33 W= 1 [  8 40  0 99]:
 R 0 L=46 C=16 S= 2 N= 2 I= 88 B= 33 W= 1 [  5 30  9  4]
 R 0 L=46 C=16 S= 2 N= 2 I= 89 B= 33 W= 1 [  1 41  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 90 B= 37 W= 1 [  1 10  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 91 B= 46 W= 1 [  1 42  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 92 B= 57 W= 1 [  2  0  0  1]
 R 0 L=46 C=16 S= 2 N= 2 I= 94 B= 57 W= 1 [  1 11  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 95 B= 58 W= 1 [  1 44  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 96 B= 65 W= 1 [  1  2  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 97 B= 68 W= 1 [  1 39  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 98 B= 73 W= 1 [  1  7  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I= 99 B= 76 W= 1 [  1 14  1  0]
 R 0 L=46 C=16 S= 2 N= 2 I=100 B= 77 W= 1 [  4  0  0  0]
-------------------------------------------------------------------------
bG0dT0d001-END-CASE.a001-NOT-002.bIFdEMPL-CODEdNOTdEQUALd3cG0dT0d
001-NOT-003.
WHILE

 R 0 L=46 C=22 S= 2 N= 2 I=177 B=  0 W= 5 [  6  0  0  0]
 R 0 L=46 C=22 S= 3 N= 2 I=178 B=  0 W= 5 [ 10  0  2  0]
 R 0 L=46 C=22 S= 3 N= 3 I=179 B=  0 W= 5 [  9 29  9  0]
 R 0 L=46 C=22 S= 3 N= 3 I=180 B=  0 W= 5 [  1 12  9  0]END-WHILE
 R 0 L=46 C=22 S= 3 N= 3 I=181 B=  0 W= 5 [  8 16  0 99](
 R 0 L=46 C=25 S= 3 N= 3 I=182 B=  0 W= 1 [  9  2  6  0]
 R 0 L=46 C=25 S= 3 N= 3 I=183 B=  0 W= 1 [  9  2  8  0]
 R 0 L=46 C=25 S= 3 N= 3 I=184 B=  0 W= 1 [  5 55  9  3]
 R 0 L=46 C=25 S= 3 N= 3 I=185 B=  0 W= 1 [  1 56  6  0]003-WHILE
 R 0 L=46 C=25 S= 3 N= 3 I=186 B=  0 W= 1 [  1 57  8  0]003-END-WHILE
 R 0 L=46 C=25 S= 3 N= 3 I=187 B=  0 W= 1 [  2  0  0  2]
 R 0 L=46 C=25 S= 3 N= 3 I=190 B=  0 W= 1 [  1 35  1  0]
 R 0 L=46 C=25 S= 3 N= 3 I=191 B=  1 W= 1 [  1  6  1  0]
 R 0 L=46 C=25 S= 3 N= 3 I=192 B= 10 W= 1 [  1 49  1  0]
 R 0 L=46 C=25 S= 3 N= 3 I=193 B= 15 W= 1 [  7  0 10  0]EXEC-STAT
 R 0 L=46 C=34 S= 3 N= 3 I=194 B= 15 W= 1 [  1 10  1  0]
 R 0 L=46 C=34 S= 3 N= 3 I=195 B= 24 W= 1 [  1 15  1  0]
 R 0 L=46 C=34 S= 3 N= 3 I=196 B= 25 W= 1 [  8 21  0  1]NOT
 R 0 L=46 C=34 S= 3 N= 3 I=198 B= 25 W= 3 [  2  0  0 -6]
 R 0 L=46 C=34 S= 3 N= 3 I=193 B= 25 W= 3 [  7  0 10  0]NOT
 R 0 L=46 C=38 S= 3 N= 3 I=194 B= 25 W= 3 [  1 10  1  0]
 R 0 L=46 C=38 S= 3 N= 3 I=195 B= 28 W= 3 [  1 15  1  0]
 R 0 L=46 C=38 S= 3 N= 3 I=196 B= 29 W= 3 [  8 21  0  1]EQUAL
 R 0 L=46 C=38 S= 3 N= 3 I=198 B= 29 W= 5 [  2  0  0 -6]
 R 0 L=46 C=38 S= 3 N= 3 I=193 B= 29 W= 5 [  7  0 10  0]EQUAL
 R 0 L=46 C=44 S= 3 N= 3 I=194 B= 29 W= 5 [  1 10  1  0]
 R 0 L=46 C=44 S= 3 N= 3 I=195 B= 34 W= 5 [  1 15  1  0]
 R 0 L=46 C=44 S= 3 N= 3 I=196 B= 35 W= 5 [  8 21  0  1]0
 R 0 L=46 C=44 S= 3 N= 3 I=198 B= 35 W= 1 [  2  0  0 -6]
 R 0 L=46 C=44 S= 3 N= 3 I=193 B= 35 W= 1 [  7  0 10  0]0
 R 0 L=46 C=46 S= 3 N= 3 I=194 B= 35 W= 1 [  1 10  1  0]
 R 0 L=46 C=46 S= 3 N= 3 I=195 B= 36 W= 1 [  1 15  1  0]
 R 0 L=46 C=46 S= 3 N= 3 I=196 B= 37 W= 1 [  8 21  0  1])
 R 0 L=46 C=47 S= 3 N= 3 I=197 B= 37 W= 1 [  2  0  0  1]
```

```
 R 0 L=46 C=47 S= 3 N= 3 I=199 B= 37 W= 1 [  5 55  9  2]
 R 0 L=46 C=47 S= 3 N= 3 I=200 B= 37 W= 1 [  1 50  1  0]
 R 0 L=46 C=47 S= 3 N= 3 I=201 B= 63 W= 1 [  2  0  0  1]
 R 0 L=46 C=47 S= 3 N= 3 I=203 B= 63 W= 1 [  1  8  1  0]
 R 0 L=46 C=47 S= 3 N= 3 I=204 B= 76 W= 1 [  1 14  1  0]
 R 0 L=46 C=47 S= 3 N= 3 I=205 B= 77 W= 1 [  4  0  0  0]
------------------------------------------------------------
a003-WHILE.bIFdEXEC-STATdNOTdEQUALd0dcNEXTdSENTENCEbELSEcGOdTOd00
3-END-WHILE.
============================================================
                    KALL EXEC (ERROR, NAME, SSN, ADDR)
KALL

 R 0 L=54 C=25 S= 3 N= 3 I= 11 B=  0 W= 4 [  7  0  4  0]EXEC
 R 0 L=54 C=30 S= 3 N= 3 I= 12 B=  0 W= 4 [  8 16  0 99](
 R 0 L=54 C=32 S= 3 N= 3 I= 13 B=  0 W= 1 [  1 17  1  0]
 R 0 L=54 C=32 S= 3 N= 3 I= 14 B=  7 W= 1 [  1  4  1  0]
 R 0 L=54 C=32 S= 3 N= 3 I= 15 B= 11 W= 1 [  1 18  1  0]
 R 0 L=54 C=32 S= 3 N= 3 I= 16 B= 18 W= 1 [  7  0  4  0]ERROR
 R 0 L=54 C=37 S= 3 N= 3 I= 17 B= 18 W= 1 [  1 19  1  0]
 R 0 L=54 C=37 S= 3 N= 3 I= 18 B= 19 W= 1 [  1  4  1  0]
 R 0 L=54 C=37 S= 3 N= 3 I= 19 B= 24 W= 1 [  8 20  0  1],
 R 0 L=54 C=38 S= 3 N= 3 I= 20 B= 24 W= 1 [  2  0  0 -5]
 R 0 L=54 C=38 S= 3 N= 3 I= 16 B= 24 W= 1 [  7  0  4  0]NAME
 R 0 L=54 C=43 S= 3 N= 3 I= 17 B= 24 W= 1 [  1 19  1  0]
 R 0 L=54 C=43 S= 3 N= 3 I= 18 B= 25 W= 1 [  1  4  1  0]
 R 0 L=54 C=43 S= 3 N= 3 I= 19 B= 29 W= 1 [  8 20  0  1],
 R 0 L=54 C=44 S= 3 N= 3 I= 20 B= 29 W= 1 [  2  0  0 -5]
 R 0 L=54 C=44 S= 3 N= 3 I= 16 B= 29 W= 1 [  7  0  4  0]SSN
 R 0 L=54 C=48 S= 3 N= 3 I= 17 B= 29 W= 1 [  1 19  1  0]
 R 0 L=54 C=48 S= 3 N= 3 I= 18 B= 30 W= 1 [  1  4  1  0]
 R 0 L=54 C=48 S= 3 N= 3 I= 19 B= 33 W= 1 [  8 20  0  1],
 R 0 L=54 C=49 S= 3 N= 3 I= 20 B= 33 W= 1 [  2  0  0 -5]
 R 0 L=54 C=49 S= 3 N= 3 I= 16 B= 33 W= 1 [  7  0  4  0]ADDR
 R 0 L=54 C=54 S= 3 N= 3 I= 17 B= 33 W= 1 [  1 19  1  0]
 R 0 L=54 C=54 S= 3 N= 3 I= 18 B= 34 W= 1 [  1  4  1  0]
 R 0 L=54 C=54 S= 3 N= 3 I= 19 B= 38 W= 1 [  8 20  0  1])
 R 0 L=54 C=54 S= 3 N= 3 I= 21 B= 38 W= 1 [  8 21  0 99])
 R 0 L=54 C=55 S= 3 N= 3 I= 22 B= 38 W= 1 [  1 14  1  0]
 R 0 L=54 C=55 S= 3 N= 3 I= 23 B= 39 W= 1 [  4  0  0  0]
------------------------------------------------------------
bCALLd"EXEC"dUSINGcERRORcNAMEcSSNcADDR.
============================================================
                    END-WHILE
END-WHILE

 R 0 L=27 C=28 S= 3 N= 3 I=206 B=  0 W= 9 [  5 12  9 99]
 R 0 L=27 C=28 S= 3 N= 3 I=207 B=  0 W= 9 [  1 37  1  0]
 R 0 L=27 C=28 S= 3 N= 3 I=208 B=  7 W= 9 [  1  6  1  0]
 R 0 L=27 C=28 S= 3 N= 3 I=209 B= 16 W= 9 [  1 38  1  0]
 R 0 L=27 C=28 S= 3 N= 3 I=210 B= 18 W= 9 [  1  8  1  0]
 R 0 L=27 C=28 S= 3 N= 3 I=211 B= 31 W= 9 [  1 14  1  0]
 R 0 L=27 C=28 S= 3 N= 3 I=212 B= 32 W= 9 [  3  0  0  0]
 R 0 L=27 C=28 S= 2 N= 3 I=213 B= 32 W= 9 [  4  0  0  0]
------------------------------------------------------------
bGOdTOd003-WHILE.a003-END-WHILE.
============================================================
                END-CASE
END-CASE

 R 0 L=20 C=21 S= 2 N= 3 I=101 B=  0 W= 8 [  5 12  9 99]
 R 0 L=20 C=21 S= 2 N= 3 I=102 B=  0 W= 8 [  1 37  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=103 B=  7 W= 8 [  1  8  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=104 B= 19 W= 8 [  1 38  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=105 B= 21 W= 8 [  1  2  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=106 B= 24 W= 8 [  1 39  1  0]
```

```
 R 0 L=20 C=21 S= 2 N= 3 I=107 B= 29 W= 8 [  1  7  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=108 B= 32 W= 8 [  1 38  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=109 B= 34 W= 8 [  1  8  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=110 B= 46 W= 8 [  1 14  1  0]
 R 0 L=20 C=21 S= 2 N= 3 I=111 B= 47 W= 8 [  3  0  0  0]
 R 0 L=20 C=21 S= 1 N= 3 I=112 B= 47 W= 8 [  4  0  0  0]
-----------------------------------------------------------------
bGOdTOd001-END-CASE.a001-NOT-003.a001-END-CASE.
=================================================================
              MOVE 1 TO CTR-EMPL.
MOVE

 R 0 L=32 C=18 S= 1 N= 3 I=  1 B=  0 W= 4 [  1 13  1  0]
 R 0 L=32 C=18 S= 1 N= 3 I=  2 B=  1 W= 4 [  1 12  1  0]
 R 0 L=32 C=18 S= 1 N= 3 I=  3 B=  5 W= 4 [  8 14  0  2]1
 R 0 L=32 C=18 S= 1 N= 3 I=  6 B=  5 W= 1 [  7  0  4  0]1
 R 0 L=32 C=20 S= 1 N= 3 I=  7 B=  5 W= 1 [  1 15  1  0]
 R 0 L=32 C=20 S= 1 N= 3 I=  8 B=  6 W= 1 [  1  4  1  0]
 R 0 L=32 C=20 S= 1 N= 3 I=  9 B=  7 W= 1 [  2  0  0 -7]
 R 0 L=32 C=20 S= 1 N= 3 I=  3 B=  7 W= 1 [  8 14  0  2]TO
 R 0 L=32 C=20 S= 1 N= 3 I=  6 B=  7 W= 2 [  7  0  4  0]TO
 R 0 L=32 C=23 S= 1 N= 3 I=  7 B=  7 W= 2 [  1 15  1  0]
 R 0 L=32 C=23 S= 1 N= 3 I=  8 B=  8 W= 2 [  1  4  1  0]
 R 0 L=32 C=23 S= 1 N= 3 I=  9 B= 10 W= 2 [  2  0  0 -7]
 R 0 L=32 C=23 S= 1 N= 3 I=  3 B= 10 W= 2 [  8 14  0  2]CTR-EMPL
 R 0 L=32 C=23 S= 1 N= 3 I=  6 B= 10 W= 8 [  7  0  4  0]CTR-EMPL
 R 0 L=32 C=32 S= 1 N= 3 I=  7 B= 10 W= 8 [  1 15  1  0]
 R 0 L=32 C=32 S= 1 N= 3 I=  8 B= 11 W= 8 [  1  4  1  0]
 R 0 L=32 C=32 S= 1 N= 3 I=  9 B= 19 W= 8 [  2  0  0 -7]
 R 0 L=32 C=32 S= 1 N= 3 I=  3 B= 19 W= 8 [  8 14  0  2].
 R 0 L=32 C=33 S= 1 N= 3 I=  4 B= 19 W= 1 [  1 14  1  0]
 R 0 L=32 C=33 S= 1 N= 3 I=  5 B= 20 W= 1 [  2  0  0  4]
 R 0 L=32 C=33 S= 1 N= 3 I= 10 B= 20 W= 1 [  4  0  0  0]
-----------------------------------------------------------------
bMOVEd1dTOdCTR-EMPL.
=================================================================
               REPEAT
REPEAT

 R 0 L=20 C=21 S= 1 N= 3 I=148 B=  0 W= 6 [  6  0  0  0]
 R 0 L=20 C=21 S= 2 N= 3 I=149 B=  0 W= 6 [ 10  0  2  0]
 R 0 L=20 C=21 S= 2 N= 4 I=150 B=  0 W= 6 [  9 51  9  0]
 R 0 L=20 C=21 S= 2 N= 4 I=151 B=  0 W= 6 [  9  2  6  0]
 R 0 L=20 C=21 S= 2 N= 4 I=152 B=  0 W= 6 [  9  2  8  0]
 R 0 L=20 C=21 S= 2 N= 4 I=153 B=  0 W= 6 [  1 52  6  0]004-REPEAT
 R 0 L=20 C=21 S= 2 N= 4 I=154 B=  0 W= 6 [  1 53  8  0]004-END-REPEAT
 R 0 L=20 C=21 S= 2 N= 4 I=155 B=  0 W= 6 [  1 35  1  0]
 R 0 L=20 C=21 S= 2 N= 4 I=156 B=  1 W= 6 [  1  6  1  0]
 R 0 L=20 C=21 S= 2 N= 4 I=157 B= 11 W= 6 [  1 14  1  0]
 R 0 L=20 C=21 S= 2 N= 4 I=158 B= 12 W= 6 [  4  0  0  0]
-----------------------------------------------------------------
a004-REPEAT.
=================================================================
                    KALL PRINT-FORM (CTR-EMPL, NAME, SSN)
KALL

 R 0 L=55 C=23 S= 2 N= 4 I= 11 B=  0 W= 4 [  7  0  4  0]PRINT-FORM
 R 0 L=55 C=34 S= 2 N= 4 I= 12 B=  0 W= 4 [  8 16  0 99](
 R 0 L=55 C=36 S= 2 N= 4 I= 13 B=  0 W= 1 [  1 17  1  0]
 R 0 L=55 C=36 S= 2 N= 4 I= 14 B=  7 W= 1 [  1  4  1  0]
 R 0 L=55 C=36 S= 2 N= 4 I= 15 B= 17 W= 1 [  1 18  1  0]
 R 0 L=55 C=36 S= 2 N= 4 I= 16 B= 24 W= 1 [  7  0  4  0]CTR-EMPL
 R 0 L=55 C=44 S= 2 N= 4 I= 17 B= 24 W= 1 [  1 19  1  0]
 R 0 L=55 C=44 S= 2 N= 4 I= 18 B= 25 W= 1 [  1  4  1  0]
 R 0 L=55 C=44 S= 2 N= 4 I= 19 B= 33 W= 1 [  8 20  0  1],
 R 0 L=55 C=45 S= 2 N= 4 I= 20 B= 33 W= 1 [  2  0  0 -5]
```

```
R 0 L=55 C=45 S= 2 N= 4 I= 16 B= 33 W= 1 [  7  0  4  0]NAME
R 0 L=55 C=50 S= 2 N= 4 I= 17 B= 33 W= 1 [  1 19  1  0]
R 0 L=55 C=50 S= 2 N= 4 I= 18 B= 34 W= 1 [  1  4  1  0]
R 0 L=55 C=50 S= 2 N= 4 I= 19 B= 38 W= 1 [  8 20  0  1],
R 0 L=55 C=51 S= 2 N= 4 I= 20 B= 38 W= 1 [  2  0  0 -5]
R 0 L=55 C=51 S= 2 N= 4 I= 16 B= 38 W= 1 [  7  0  4  0]SSN
R 0 L=55 C=55 S= 2 N= 4 I= 17 B= 38 W= 1 [  1 19  1  0]
R 0 L=55 C=55 S= 2 N= 4 I= 18 B= 39 W= 1 [  1  4  1  0]
R 0 L=55 C=55 S= 2 N= 4 I= 19 B= 42 W= 1 [  8 20  0  1])
R 0 L=55 C=55 S= 2 N= 4 I= 21 B= 42 W= 1 [  8 21  0 99])
R 0 L=55 C=56 S= 2 N= 4 I= 22 B= 42 W= 1 [  1 14  1  0]
R 0 L=55 C=56 S= 2 N= 4 I= 23 B= 43 W= 1 [  4  0  0  0]
```

```
bCALLd"PRINT-FORM"dUSINGcCTR-EMPLcNAMEcSSN.
```

```
                    ADD 1 TO CTR-EMPL.
ADD
```

```
R 0 L=37 C=23 S= 2 N= 4 I=  1 B=  0 W= 3 [  1 13  1  0]
R 0 L=37 C=23 S= 2 N= 4 I=  2 B=  1 W= 3 [  1 12  1  0]
R 0 L=37 C=23 S= 2 N= 4 I=  3 B=  4 W= 3 [  8 14  0  2]1
R 0 L=37 C=23 S= 2 N= 4 I=  6 B=  4 W= 1 [  7  0  4  0]1
R 0 L=37 C=25 S= 2 N= 4 I=  7 B=  4 W= 1 [  1 15  1  0]
R 0 L=37 C=25 S= 2 N= 4 I=  8 B=  5 W= 1 [  1  4  1  0]
R 0 L=37 C=25 S= 2 N= 4 I=  9 B=  6 W= 1 [  2  0  0 -7]
R 0 L=37 C=25 S= 2 N= 4 I=  3 B=  6 W= 1 [  8 14  0  2]TO
R 0 L=37 C=25 S= 2 N= 4 I=  6 B=  6 W= 2 [  7  0  4  0]TO
R 0 L=37 C=28 S= 2 N= 4 I=  7 B=  6 W= 2 [  1 15  1  0]
R 0 L=37 C=28 S= 2 N= 4 I=  8 B=  7 W= 2 [  1  4  1  0]
R 0 L=37 C=28 S= 2 N= 4 I=  9 B=  9 W= 2 [  2  0  0 -7]
R 0 L=37 C=28 S= 2 N= 4 I=  3 B=  9 W= 2 [  8 14  0  2]CTR-EMPL
R 0 L=37 C=28 S= 2 N= 4 I=  6 B=  9 W= 8 [  7  0  4  0]CTR-EMPL
R 0 L=37 C=37 S= 2 N= 4 I=  7 B=  9 W= 8 [  1 15  1  0]
R 0 L=37 C=37 S= 2 N= 4 I=  8 B= 10 W= 8 [  1  4  1  0]
R 0 L=37 C=37 S= 2 N= 4 I=  9 B= 18 W= 8 [  2  0  0 -7]
R 0 L=37 C=37 S= 2 N= 4 I=  3 B= 18 W= 8 [  8 14  0  2].
R 0 L=37 C=38 S= 2 N= 4 I=  4 B= 18 W= 1 [  1 14  1  0]
R 0 L=37 C=38 S= 2 N= 4 I=  5 B= 19 W= 1 [  2  0  0  4]
R 0 L=37 C=38 S= 2 N= 4 I= 10 B= 19 W= 1 [  4  0  0  0]
```

```
bADDd1dTOdCTR-EMPL.
```

```
                  UNTIL  (CTR-EMPL LARGE THAN 100)
UNTIL
```

```
R 0 L=49 C=23 S= 2 N= 4 I=159 B=  0 W= 5 [  5 12  9 99]
R 0 L=49 C=23 S= 2 N= 4 I=160 B=  0 W= 5 [  8 16  0 99](
R 0 L=49 C=26 S= 2 N= 4 I=161 B=  0 W= 1 [  1 41  1  0]
R 0 L=49 C=26 S= 2 N= 4 I=162 B=  4 W= 1 [  7  0 10  0]CTR-EMPL
R 0 L=49 C=34 S= 2 N= 4 I=163 B=  4 W= 1 [  1 10  1  0]
R 0 L=49 C=34 S= 2 N= 4 I=164 B= 12 W= 1 [  1 15  1  0]
R 0 L=49 C=34 S= 2 N= 4 I=165 B= 13 W= 1 [  8 21  0  1]LARGE
R 0 L=49 C=34 S= 2 N= 4 I=167 B= 13 W= 5 [  2  0  0 -6]
R 0 L=49 C=34 S= 2 N= 4 I=162 B= 13 W= 5 [  7  0 10  0]LARGE
R 0 L=49 C=40 S= 2 N= 4 I=163 B= 13 W= 5 [  1 10  1  0]
R 0 L=49 C=40 S= 2 N= 4 I=164 B= 18 W= 5 [  1 15  1  0]
R 0 L=49 C=40 S= 2 N= 4 I=165 B= 19 W= 5 [  8 21  0  1]THAN
R 0 L=49 C=40 S= 2 N= 4 I=167 B= 19 W= 4 [  2  0  0 -6]
R 0 L=49 C=40 S= 2 N= 4 I=162 B= 19 W= 4 [  7  0 10  0]THAN
R 0 L=49 C=45 S= 2 N= 4 I=163 B= 19 W= 4 [  1 10  1  0]
R 0 L=49 C=45 S= 2 N= 4 I=164 B= 23 W= 4 [  1 15  1  0]
R 0 L=49 C=45 S= 2 N= 4 I=165 B= 24 W= 4 [  8 21  0  1]100
R 0 L=49 C=45 S= 2 N= 4 I=167 B= 24 W= 3 [  2  0  0 -6]
R 0 L=49 C=45 S= 2 N= 4 I=162 B= 24 W= 3 [  7  0 10  0]100
R 0 L=49 C=49 S= 2 N= 4 I=163 B= 24 W= 3 [  1 10  1  0]
R 0 L=49 C=49 S= 2 N= 4 I=164 B= 27 W= 3 [  1 15  1  0]
```

```
 R  0  L=49  C=49  S=  2  N=  4  I=165  B=  28  W=  3  [   8  21   0   1])
 R  0  L=49  C=50  S=  2  N=  4  I=166  B=  28  W=  1  [   2   0   0   1]
 R  0  L=49  C=50  S=  2  N=  4  I=168  B=  28  W=  1  [   1  37   1   0]
 R  0  L=49  C=50  S=  2  N=  4  I=169  B=  35  W=  1  [   1   8   1   0]
 R  0  L=49  C=50  S=  2  N=  4  I=170  B=  49  W=  1  [   1  54   1   0]
 R  0  L=49  C=50  S=  2  N=  4  I=171  B=  61  W=  1  [   1   6   1   0]
 R  0  L=49  C=50  S=  2  N=  4  I=172  B=  71  W=  1  [   1  38   1   0]
 R  0  L=49  C=50  S=  2  N=  4  I=173  B=  73  W=  1  [   1   8   1   0]
 R  0  L=49  C=50  S=  2  N=  4  I=174  B=  87  W=  1  [   1  14   1   0]
 R  0  L=49  C=50  S=  2  N=  4  I=175  B=  88  W=  1  [   3   0   0   0]
 R  0  L=49  C=50  S=  1  N=  4  I=176  B=  88  W=  1  [   4   0   0   0]
--------------------------------------------------------------------------
bIFdCTR-EMPLdLARGEdTHANd100dbGOdTOd004-END-REPEATbELSEcGOdTOd004-
REPEAT.a004-END-REPEAT.
==========================================================================
                      SET-OFF EXEC
SET-OFF

 R  0  L=30  C=26  S=  1  N=  4  I=  24  B=   0  W=  7  [   9  12   5   0]
 R  0  L=30  C=26  S=  1  N=  4  I=  25  B=   0  W=  7  [   7   0   4   0]EXEC
 R  0  L=30  C=31  S=  1  N=  4  I=  26  B=   0  W=  7  [   5  22   5   2]
 R  0  L=30  C=31  S=  1  N=  4  I=  29  B=   0  W=  7  [   1  24   1   0]
 R  0  L=30  C=31  S=  1  N=  4  I=  30  B=  20  W=  7  [   1   4   1   0]
 R  0  L=30  C=31  S=  1  N=  4  I=  31  B=  24  W=  7  [   1  14   1   0]
 R  0  L=30  C=31  S=  1  N=  4  I=  32  B=  25  W=  7  [   4   0   0   0]
--------------------------------------------------------------------------
bMOVEdSET-TO-OFFdTOdEXEC.
==========================================================================
                       END
END

 R-3 L=22 C=23 S=  1 N=  4 I=   1 B=   0 W=  3 [   1  13   1   0]
```

A Program Design Language for COBOL

BY

Robert Shihpei Chou

B. A., Tugnhai University, 1973

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

The structured programming method improves the readability and modifiability of computer programs. It reduces the cost of sortware development. This method is not sasily applied to COBOL, since COBOL is not equipt with enough control statements. The purpose of this study is to define a program design language(PDL) for COBOL and to create a tranalator for the transltion of this PDL.

The statements of the PDL include the control statements and the preprocessor statements. The control statements are used to build up sequence, selection, and iteration control structures in a COBOL program. The preprocessor statements substitute some original COBOL sentences are inserted into the control structures as the structure body.

A table-driven technique is employed to make a flexible transltor for this PDL. The table-builder program converts the translation sementics of the PDL statements into entries in the tables. The translator uses these tables to translate the PDL statements. When changes occur in PDL, new translation tables will be built for the new PDL.

This study provides the COBOL programmers with an easy approach to structured programming and allow the user a flexible flexible implementation for the PDL.

1430-60
CD-53