A MODIFIED BASIC COMPILER IN PL/I
FOR THE M6800 MICROPROCESSOR

by

Fred Esenwein

Submitted in Partial Fulfillment of the Requirements
for the Degree of
Master of Science in Engineering
in the
Electrical Engineering
Program

3-25-80
Adviser Date

Leon Rand 3-24-80
Dean of the Graduate School Date

YOUNGSTOWN STATE UNIVERSITY

March, 1980

ABSTRACT

A MODIFIED BASIC COMPILER IN PL/I FOR THE M6800 MICROPROCESSOR

Fred Esenwein
Master of Science in Engineering
Youngstown State University, 1980

This paper presents a compiler which accepts a modified version of BASIC programming language as input, and produces hexadecimal assembly code for the Motorola M6800 microprocessor as output.

Included are discussions of microprocessor design considerations, compiler architecture, and the compiler program itself.

Finally, as a demonstration, the control of a real system is implemented by means of a microprocessor controller for which the operating program is written in BASIC.

ACKNOWLEDGEMENTS

I am obliged to my faculty adviser, Dr. Robert H. Foulkes, for his part in bringing this thesis to completion.

Thanks, also, to General Motors Corporation for financial support during the course of my study.

Most of all, I am grateful to my wife, Mary, whose forbearance throughout the two year span of this project was surpassed only by her expert technical assistance.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

LIST OF TABLES

CHAPTER I

INTRODUCTION

Purpose and Preliminary Concepts

It is difficult to overstate the impact of recent developments in microprocessor technology upon industrialized society. These compact, relatively inexpensive computer systems are finding applications in such diverse areas as business, education, transportation, and entertainment. The advent of the microcomputer has particularly benefitted the field of industrial control.

Dedicated control functions which would have previously been accomplished using hard-wired logic are now being implemented with microprocessor controllers. This results in a substantial savings in design time, provided that the necessary software can be developed expeditiously. A technique is presented here which simplifies the development of software for the Motorola M6800 microprocessor.

Microprocessor machine code is a collection of hexadecimal words that, in program form, is nearly useless to all but the most experienced programmer. A mnemonic form is available, but if it must be assembled by hand using pencil and paper, the process is slow and tedious. It is desirable to write the source program in a higher level language like BASIC or FORTRAN, and then invoke a machine translation of that source program into hexadecimal codes required by the microprocessor. Such packages are available from major manufacturers of microprocessor products as software support for their development systems. Unfortunately, these development systems are expensive enough to be unaffordable for many small businesses and all but a few individuals. This paper presents a compiler in PL/I which translates BASIC source code into M6800 machine language. Anyone with access to a time-sharing host computer is thereby able to develop microprocessor based control system software in BASIC language.

Approach

Industrial control requirements vary widely. In many instances intelligent control is needed, but the computing power of either a mainframe computer or a minicomputer would constitute an overkill situation. It is to these applications that the microcomputer is ideally suited. As a simple dedicated controller, the microcomputer is not required to perform complex mathematical operations. It is often not interfaced with a line printer. These and other simplifying factors lead to modification of the BASIC programming language for purposes of dedicated control. A later section will deal with the development of this modified BASIC language.

Although any of several microprocessors could be programmed in BASIC by means of a compiler similar to the one presented here, the Motorola M6800 was chosen in this particular case. The M6800 is an industry standard, and is supported by a wide variety of peripheral devices. In addition, a microcomputer built around the M6800 CPU was available to the author at the time this project began. Thorough testing of algorithms throughout the development of the compiler was therefore possible.

The compiler itself is divided into three separate parts: lexical analysis, syntax analysis, and code generation. The lexical analysis phase breaks each line of source code down into its component parts and checks for elementary errors. Syntax analysis discerns from the order of words and phrases in the source code what action must be performed by the program. These actions are translated into microprocessor machine code by the code generator.

The chapters which follow discuss the details involved in each segment of the compiler and how they accomplish the translation of BASIC into M6800 code. Chapter I has explained the purpose and basic approach for this paper. Chapter II deals with the peculiarities of the M6800 microprocessor and BASIC programming language. In Chapter III, the compiler proper is discussed, while Chapter IV presents results. A user's manual, PL/I listings for the compiler, sample programs, a demonstration program, and a glossary of compiler terms constitute the five appendices.

CHAPTER II

DESIGN CONSIDERATIONS

M6800 Hex Code as Compiler Output

Prerequisite to the design of a compiler that produces proper hex code for the M6800 microprocessor is an understanding of the features, limitations, and general operation of the machine itself. A cursory examination of M6800 fundamentals is therefore in order.

A program for the M6800 consists of a series of instructions, each of which is followed by one or more operands. There are over one hundred hexadecimal instructions (opcodes) for the 6800 MPU [1]. It is the task of the compiler to express a source program, which is written in BASIC, solely in terms of this instruction set.

Many instructions cause the MPU to either store data in memory or to retrieve data from memory. When this is the case, the programmer often has the choice of addressing the desired memory location in one of several ways. The four possible choices are the immediate, direct, indexed and extended addressing modes. Efficient use of available memory and speed of execution can be maximized by the proper choice of addressing mode.

When an instruction is executed in the immediate addressing mode, the byte immediately following the opcode is used as data. This is useful when the data does not change during the execution of the program.

Direct addressing interprets the byte following an opcode as a pointer to the memory location from which the data for the operation is to be retrieved. Since the 6800 MPU uses a 16 bit address bus, a specific memory location can not be completely defined by one eight bit byte. Direct addressing assumes the higher order byte to be 00_{16}. Direct addressing is sometimes called zero page addressing.

The use of extended addressing allows any memory location to be accessed via the 16 bit address bus. An opcode is followed by two bytes in the program, the first of which is the higher order byte of the address where the data resides, and the second of which is the lower order byte. This mode of addressing requires more execution time, but it is more flexible than immediate or zero page modes.

Indexed addressing will not be discussed in this paper.

The need for an understanding of various addressing modes will become apparent later when memory assignment is discussed.

Although the 6800 MPU is capable of addressing over 64,000 memory locations, very few systems actually incorporate that much memory. Indeed, memory size is a limiting factor in almost all microprocessor system designs. The machine used by the author is marketed by Heath Company. It has only 512 bytes of random access memory (RAM). Of this, addresses 00C5 through 00FF are reserved for use by the operating system [2]. This lack of memory places severe restrictions on the configuration of a BASIC compiler to be used with this machine. The size of the BASIC source program acceptable for compilation is likewise restricted.

Input/output (I/O) devices are assigned addresses by the system designer. Transfers of data between I/O devices and the MPU are handled exactly like memory transfers. Generally speaking, I/O devices are assigned addresses above the highest address assigned to a RAM element.

This completes the overview of microprocessor fundamentals relevant to the development of the BASIC compiler. A detailed treatment of this subject is outside the scope of this paper. Readers who are not familiar with the operation of the Motorola M6800 should seek a good tutorial such as the Heathkit individual learning program [1].

As stated previously, the output of the BASIC compiler is a program in hexadecimal codes that are part of the M6800 instruction set or data. This can be enhanced for ease of human interpretation by the addition of mnemonic instructions and comments in English. These concepts are incorporated into the finished BASIC compiler.

Choice of BASIC as Source Language

Although no single high level computer language is best for all applications, BASIC seems to be a good choice in this instance. It is easily learned and universally used.

In addition, with minor modifications, it is ideally suited for industrial control functions.

Many versions of BASIC are currently in use. Some are so expanded with respect to the original Dartmouth BASIC that they approach FORTRAN or PL/I in complexity. Table 1 shows commonly used elements of BASIC language as derived from Digital Equipment Corporation's BASIC-PLUS [3].

TABLE 1
FUNDAMENTAL ELEMENTS OF BASIC

Operators	Functions	Statements
-	ABS	LET
+	ATN	PRINT
/	COS	GOTO
*	DEF	INPUT
↑	EXP	STOP
=	INT	END
(	LOG	IF
)	RND	FOR
>	SGN	NEXT
<	SIN	GOSUB
,	SQR	RETURN
>=	TAN	DIM
<=		REM
		DATA
<>		READ
		RESTORE
		ON
		matrix operations
		alphanumeric operations
		file handling operations

There are many items in the above listing that serve no useful purpose as far as dedicated industrial control is concerned. On the other hand, there are some instructions that would be very nice to have which do not appear above. The task at hand becomes one of deleting those statements, functions, or operators which are either meaningless or worthless to the intended application, and of adding special statements, functions, or operators to facilitate working with the microprocessor.

First of all, the PRINT statement is meaningless without a printer. While data manipulation on a mainframe computer almost invariably results in a report from the system printer, printers are often left out of microprocessor systems altogether. When they are incorporated, they may be treated just like any other I/O device. The PRINT statement is therefore of little use and will be deleted.

Similarly, the alphanumeric and quoted text features serve a limited usefulness and are also deleted.

The INPUT statement is a system command. It is used in conjunction with a terminal for interactive processing. Any time the INPUT statement is encountered in the execution of a BASIC program, the execution halts and the user is prompted to enter a value at the terminal. Interactive processing is not a concern here, so the INPUT statement is deleted.

File handling statements are nice for organizing data in applications like payroll management or license plate registrations. They are not necessary in this case.

Matrix operations have been dropped for two reasons. First of all, matrices require tremendous amounts of storage.

Secondly, matrix operations are of very little use in a real time control system. In most cases the real time event would be long past by the time the matrix operations were completed.

All of the BASIC functions are deleted with the exception of SQR. SQR is retained as an academic exercise in the development of functional algorithms.

All of the operators are retained.

The statement list is now down to a much more manageable size. At this point, however, there exists a total lack of I/O statements for peripheral devices. This, of course, is a direct result of the elimination of the PRINT and INPUT commands. They must now be replaced by more general I/O statements.

The problem with the PRINT and INPUT statements is that they address only two devices, whose addresses are known to the mainframe computer by design. The microprocessor must be able to accept data from or transmit data to a multitude of devices at arbitrary addresses. This is accomplished by defining two new BASIC statements, RDIN and WRTOUT. These statements, in conjunction with the addresses of the appropriate I/O devices, allow the exchange of data between the 6800 MPU and its peripherals. Use and syntax of these new statements are demonstrated in Appendix A.

One more change is necessary. The character "@" must be substituted for the up-arrow operator. No up-arrow exists on the IBM terminal through which this program was developed.

This concludes the modification of the BASIC language. Table 2 summarizes the results.

TABLE 2

ELEMENTS OF MODIFIED BASIC

Operators	Functions	Statements
- + / * @ = ( ) > < , GEQ (>=) LEQ (<=) NEQ (<>)	SQR	LET GOTO STOP END IF FOR NEXT GOSUB RETURN REM DATA READ RESTORE RDIN WRTOUT

No attempt to instruct the reader in the use of this modified version of BASIC is made here. A complete user's manual appears in Appendix A.

CHAPTER III

SYNTHESIS OF A COMPILER IN PL/I FOR THE GENERATION OF M6800 HEX CODE FROM BASIC SOURCE PROGRAMS

Compiler Architecture and Operation

A compiler translates a higher level programming language into machine language acceptable to the computer being programmed. This is accomplished in three major steps: (1) lexical analysis, (2) syntax analysis and interpretation, and (3) code generation.

The purpose of lexical analysis is to break the source program down into a series of basic elements (tokens), and to discover elementary errors. Each line of source text is scanned sequentially. Identifiers (variable names), literals (constants), and terminal symbols (operators and keywords) are individually broken out of the source line, checked for validity, and entered into the parse table. The source program is then converted into a series of "uniform symbols". Each entry in the uniform symbol table is three characters long, and is accompanied by a pointer showing where in the tables the actual token may be found. For instance, a uniform symbol table entry like "TRM 4" refers to the fourth token in the terminal table.

The use of uniform symbols makes comparison of long character strings unnecessary. It also makes later phases of the compiler less complicated.

The outputs from the lexical analysis phase are identifier (IDN), literal (LIT), and uniform symbol tables. Lexical analysis also provides input to the error table if mistakes in the source program are detected. The terminal (TRM) and keyword (KEY) tables are permanent tables and are used by lexical analysis as input. All of the above tables, with the exception of the error table, are forwarded as inputs to the syntax analysis phase.

Once through the lexical analysis phase, the compiler has ascertained that the source program is made up of a series of allowable tokens. However, if the program is to have any meaning, the tokens must be arranged in the proper order according to the rules of syntax for the source language.

During syntax analysis the uniform symbol table is scanned sequentially while the computer looks for a meaningful phrase. When a meaningful phrase is recognized, the compiler breaks it down into elementary operations and enters the information into a matrix. Each entry of the matrix consists of a simple operator and one or two operands. In this way complex expressions are reduced to several simpler operations.

As before, if any syntax errors are discovered during the course of the syntax analysis, they are recorded in the error table.

The matrix is an intermediate form of the program that is derived solely from the source program and is in no way related to the machine for which the compiler is used. In other words, up to this point, the compiler is "machine independent". In the next phase, which is code generation, the output must be tailored for a specific piece of hardware.

The code generator translates each line of the matrix into machine code. The compiler uses a code generation routine dictated by the matrix operator along with the matrix line operands to output a machine language equivalent of the matrix line. Each matrix line may actually result in several lines of machine code.

Compiler theory is the subject of many books. The material in the above discussion was drawn largely from a work by John J. Donovan [4]. The chapter on compilers should be of interest to the reader desiring a deeper understanding of the ideas involved.

The modified BASIC to M6800 hex code compiler, which is the topic of this paper, was developed according to the generalized model of the compiler just presented. The higher level language input is the modified BASIC of Appendix A, and the machine language output is Motorola M6800 hexadecimal code.

The compiler itself is a computer program, written in PL/I which accomplishes the tasks of lexical analysis, syntax analysis, and code generation on an IBM 370 computer. Because of its structure and ability to perform string operations with relative ease, PL/I was chosen over the other higher level languages available for the development of this compiler.

As previously stated, there are certain areas within the microprocessor's memory of which the compiler may not avail itself when making memory assignments. Figure 1 shows the memory map for the author's system.

SYSTEM ASSIGNMENTS	HEXADECIMAL ADDRESSES	USER ASSIGNMENTS
MONITOR ROM (OPERATING SYSTEM)	FFFF FC00	NOT AVAILABLE
NOT ASSIGNED	FBFF C200	I/O DEVICES
RESERVED FOR KEYBOARD, DISPLAYS, AND MONITOR	C1FF C000	NOT AVAILABLE
NOT ASSIGNED	BFFF 0200	I/O DEVICES
USER RAM	01FF 0100	PROGRAM
RESERVED FOR MONITOR	00FF 00C5	NOT AVAILABLE
USER RAM	00C4 0000	DATA, CONSTANTS, VARIABLES, TEMPORARY STORAGE

Fig. 1. Memory Map.

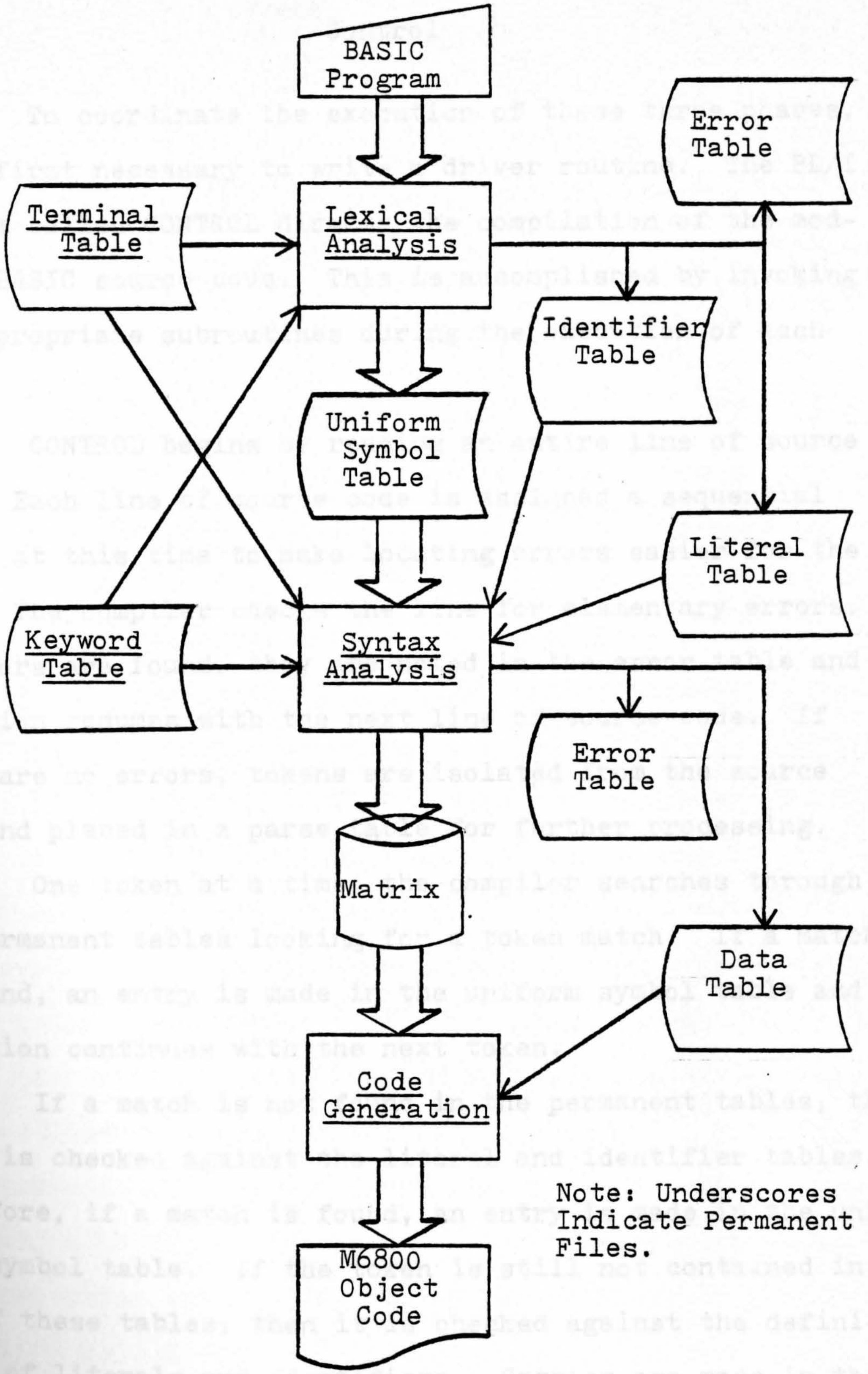
Addresses 0000 through 01FF are occupied by RAM devices. The system, however, has reserved a section of memory for the microprocessor's operating system in the middle of user RAM. The compiler has therefore been designed to begin address assignments for the actual program at address 0100. If the program were started at a lower address, provision would have to be made for execution to branch around the reserved locations. The addresses 0000 through 00C4 are used by the compiler for the storage of data, constants, variables, and temporary values. These items are frequently transferred to and from the MPU during program execution, so placing them in zero page allows the advantage of using the direct addressing mode.

The remaining memory addresses that are not reserved for system use may be assigned to I/O devices.

The necessary theory and physical constraints have now been brought into focus. The next section deals with the PL/I program that produces M6800 code from modified BASIC input.

Discussion of PL/I Routines

A flow chart of the compiler appears in figure 2. The reader may also wish to refer to the PL/I listings of compiler routines which appear in Appendix B. Each PL/I routine is discussed individually below. The routines are presented in three groups corresponding to the lexical, syntax, and code generation phases of the compiler.



Note: Underscores Indicate Permanent Files.

Fig. 2. Modified BASIC to M6800 Hex Code Compiler Flow Chart.

Control

To coordinate the execution of these three phases, it is first necessary to write a driver routine. The PL/I program called CONTROL directs the compilation of the modified BASIC source code. This is accomplished by invoking the appropriate subroutines during the execution of each phase.

CONTROL begins by reading an entire line of source code. Each line of source code is assigned a sequential number at this time to make locating errors easier for the user. The compiler checks the line for elementary errors. If errors are found, they are noted in the error table and execution resumes with the next line of source code. If there are no errors, tokens are isolated from the source line and placed in a parse table for further processing.

One token at a time, the compiler searches through the permanent tables looking for a token match. If a match is found, an entry is made in the uniform symbol table and execution continues with the next token.

If a match is not found in the permanent tables, the token is checked against the literal and identifier tables. As before, if a match is found, an entry is made in the uniform symbol table. If the token is still not contained in any of these tables, then it is checked against the definitions of literals and identifiers. Entries are made in the uniform symbol table and either the literal or identifier table if the token is found to be valid. Otherwise, the only conclusion is that the token is an illegal character string. This information is sent to the error table.

Execution continues in like manner, one token after another, until all the tokens in the source program have been examined. The result is a complete uniform symbol table, and filled literal and identifier tables with no duplicate entries.

Once the uniform symbol table has been established, the compiler proceeds with syntax analysis, working with one line of source code at a time, as taken from the uniform symbol table.

The first thing which must appear at the beginning of a new line is a valid line number. An invalid line number in the first position causes the generation of an error message. Execution then skips around the remainder of the tokens in that line and begins again at the start of the next source line in the uniform symbol table. In fact, the discovery of an error anywhere within a program line during syntax analysis will result in this same action.

According to the rules of syntax for BASIC, a keyword must always appear in the second position. For every BASIC keyword, there is an associated action routine in the compiler which processes the remainder of the source line. When the syntax analyzer reads the second token in a source line from the uniform symbol table, it checks this token against valid keywords until a match is found. If no match is found, an error is indicated.

A valid keyword in the second position causes execution to transfer to an action routine where the remainder of the line is checked for syntax, interpreted, and finally converted into matrix entries for use by the code generation phase.

The last statement in any BASIC program must be END. The compiler checks for this during syntax analysis and issues an error message if the condition is not met.

Although the entire source program is scanned for lexical and syntax errors, an error of any kind causes compilation to halt at the end of syntax analysis. Code generation is thus suppressed and the programmer is obliged to correct his source program before attempting to compile again.

If the execution continues through the lexical and syntax phases without incident, the final task of code generation is undertaken.

The purpose of the code generation phase is to convert the source program, now in matrix form, into microprocessor hex code and to make memory assignments within user RAM.

The compiler begins at address 0000 and assigns data, literals, identifiers, and temporary storage values to zero page locations. The compiler then begins reading matrix lines, one at a time, and generating equivalent microprocessor code. This microprocessor code is entered in consecutive memory locations, beginning at address 0100.

Code generation works in much the same way as syntax analysis. The compiler reads a matrix line. Each matrix line consists of an operator and one or two operands. Corresponding to every matrix operator is a code generation subroutine. When a matrix operator is recognized, execution is transferred to the associated code generation subroutine where microprocessor code is generated according to the operands in the matrix line.

This completes compilation of the source program. CONTROL sends the source listing, the error table, the terminal table, the literal table, the identifier table, the uniform symbol table, and the data table, along with the finished microprocessor hex code version of the source program to the system printer. The matrix is available in a file called MATRIX DATA.

The discussions which follow deal with the various subroutines used by CONTROL during the course of compilation.

Lexical Phase

Parsing the Source Code

Each line of BASIC source code is resolved into its component parts (tokens), or parsed, by a PL/I routine called PARSE2. PARSE2 scans the source line looking for blanks, terminal symbols, or operators which delineate tokens. The characters between delimiters are assumed to be tokens and are entered into the parse table in the order in which they are encountered. PARSE2 places a "$" symbol in the parse table at the end of each source line to serve as a line's end flag for the syntax phase.

Discovery of Elementary Errors

ERRCHK examines an entire source line and looks for three error conditions: (1) no numeric in column one, (2) a line number of more than five digits, and (3) characters beyond column 72.

The REMARK Statement

A PL/I routine called REMARK is called by CONTROL whenever a REMARK statement is encountered in the source code. The appearance of "REM" in the parse table when the compiler is building the uniform symbol table causes the compiler to skip to the next program line. No entry is made in the uniform symbol table for any token appearing in the REMARK statement. Anything contained in a REMARK statement is therefore ignored.

Recognition Routines

All tokens must be classified as terminal symbols, keywords, identifiers, or literals. The definitions of these different types of tokens are contained in the recognition routines.

TRMREC, IDNREC, LITREC, KEYREC, and HEXREC are called by CONTROL to classify a token found in the parse table as a terminal symbol, identifier, literal, keyword, or hexadecimal literal, respectively. Once recognized as a specific type of token, an entry is made for the token in the uniform symbol table.

Syntax Phase

Checking for a Valid Line Number

The routine called TGTCHK checks to see if the first token in a new source line from the uniform symbol table is a valid line (target, for branching) number. To qualify, it must be a literal, and it must be all numeric.

Action Routines

The remainder of the syntax routines are action routines which are called by CONTROL when keywords are encountered in the uniform symbol table. Every keyword in BASIC language indicates a specific set of operations to be performed on the tokens which follow it. The action routine for a given keyword discerns the meaning of the phrase following that keyword according to the rules of syntax for BASIC and outputs a series of matrix lines which convey that meaning to the code generation phase.

All keywords have associated action routines. Included are LETAR, RDINAR, DATAAR, RETRNAR, STOPAR, WTOUTAR, GOTOAR, RESTRAR, READAR, ENDAR, FORAR, NEXTAR, and GOSUBAR. These PL/I routines generate matrix entries for LET, RDIN, DATA, RETURN, STOP, WRTOUT, GOTO, RESTORE, READ, END, FOR, NEXT, and GOSUB modified BASIC statements, respectively.

The action routines labeled IFAR and CDX are used together to process the IF statement. The IF statement consists of two arithmetic expressions on either side of a relational operator. CDX is used to process the condition on the left hand side of the relational operator and the assignment CXL = CDX is made. Similarly, CDX is used to process the condition on the right hand side of the relational operator and the assignment CXR = CDX is made. Finally, the IF action routine generates matrix entries that compare CXL to CXR prior to making a branch decision.

As before, an error anywhere within the program line results in output to the error table and sends execution to the beginning of the next source line.

Code Generation Phase

Code Generation Routines

The code generation subroutines are used by CONTROL to translate the program as it appears in the matrix into M6800 microprocessor hex code. Each matrix line begins with an operator which, when encountered, sends execution to the code generation subroutine associated with that particular operator. This usually results in several lines of microprocessor code for each matrix line.

The code generation subroutines are easily identified. Each is given a name which begins with the matrix operator it services, and ends in "CG" or "G". For example, RTRNCG is the code generation subroutine for the RETURN operator.

Table 3 summarizes the PL/I code generation subroutines and the matrix operators with which they are associated.

TABLE 3

CODE GENERATION SUBROUTINES

Code Generation Subroutine	Matrix Operator
READCG	READ
STOPCG	STOP
ENDCG	END
MTPLYCG	* (multiplication)
RESTRCG	RESTORE
EQUALCG	= (equality)
PLUSCG	+ (addition)
MINUSCG	- (subtraction)
DIVIDCG	/ (division)
EXPCG	@ (exponentiation)
SQRCG	SQR (square root)
RDINCG	RDIN
WTOUTCG	WRTOUT
GOSUBCG	GOSUB
GOTOCG	GOTO
RTRNCG	RETURN
BRANCHG	assigns destinations to branch statements
NEQCG	not equal
CONEQCG	conditionally equal
LTCG	less than
GTCG	greater than
GEQCG	greater than or equal
LEQCG	less than or equal

Utility Routines

The remaining PL/I routines are used by the code generation subroutines to accomplish routine, repetitive tasks.

HXDCCON converts a hexadecimal number to a decimal number. DCHXCO and DCHXCO2, on the other hand, convert decimal numbers to hexadecimal equivalents.

HEADING and RITEOUT are used to format the microprocessor code output that is sent to the system printer.

CHAPTER IV

RESULTS

Compiler Output

The report which the user receives from the IBM 370's printer at the completion of compilation consists of two parts: (1) the M6800 microprocessor code version of the source program, which is of primary interest, and (2) supplemental information to aid in the location of a problem, should one arise during compilation. The reader may wish to refer to one of the sample programs in Appendix C for the discussion that follows.

The microprocessor code version of the program is printed out at fifty lines to a page, with each page headed so as to make interpretation of the results easier for the user. In the first column are hexadecimal addresses for memory locations within user RAM. The second column lists the hexadecimal values that are to be programmed into the microprocessor. The first two columns alone are sufficient information for the user to enter and run his program on the M6800 microcomputer.

The last two columns are informatory in nature. Column number three is headed "MNEMONICS/DECIMAL CONTENTS". It contains mnemonics associated with various opcodes, which should be familiar to all experienced microcomputer users.

When helpful for clarity, certain decimal equivalents of hexadecimal constants appear in this column as well.

The last column describes the program in English phrases. This information, along with that in the previous column greatly enhances the program's readability.

When branch statements are encountered during compilation, it often occurs that the address within the microprocessor's memory to which execution is to be sent is at the time unknown. For this reason, all branch statements are saved until the compilation of the source program is complete. At that time, all memory assignments have been made, and the branch statements are listed in one block at the end of the printed output. Meanwhile, on the first pass, the memory locations to be filled with branch statements are simply filled with "XX".

Any time the hexadecimal contents are listed as "XX", it should be understood that the actual contents of that memory location do not matter. This convention is carried over into the locations reserved for computed values such as variable names. Since these locations will be written into later, the actual contents before execution of the microcomputer program can be anything.

Notice that memory addresses are assigned sequentially, beginning with 0000, for data, constants, variables, and temporary storage locations. Memory assignment then skips to address 0100 and continues to the end of the program.

At the end of the program listing, the user is reminded to start execution of his program from address 0100. The total amount of microprocessor memory required for the program is also given, in bytes. This information is very useful to the designer, because he can write a program to perform his control function, compile it, and know exactly how much RAM he must include in his system before actually purchasing any hardware.

Following the M6800 code program is a listing of the modified BASIC source program exactly as it was written by the user. The program lines are numbered consecutively on the right hand side. The error table on the next page of the printed output lists errors, if any, along with source line numbers corresponding to the ones found to the right of the BASIC code.

The last pages of the printed output contain the major tables used or created during compilation. They could easily be deleted from the report, but are included here for completeness. They might also be helpful in locating unusual problems encountered during compilation.

Application to HO Railroad Control

The final test of the results claimed in this paper is to build a working, physical system and control it with a microcomputer, using a control program written in BASIC and translated to microcomputer codes by means of the modified BASIC to M6800 code compiler presented herein.

The system chosen for purposes of demonstration is a model railroad. Parts for this set up are inexpensive and readily available. The track layout is shown in figure 3.

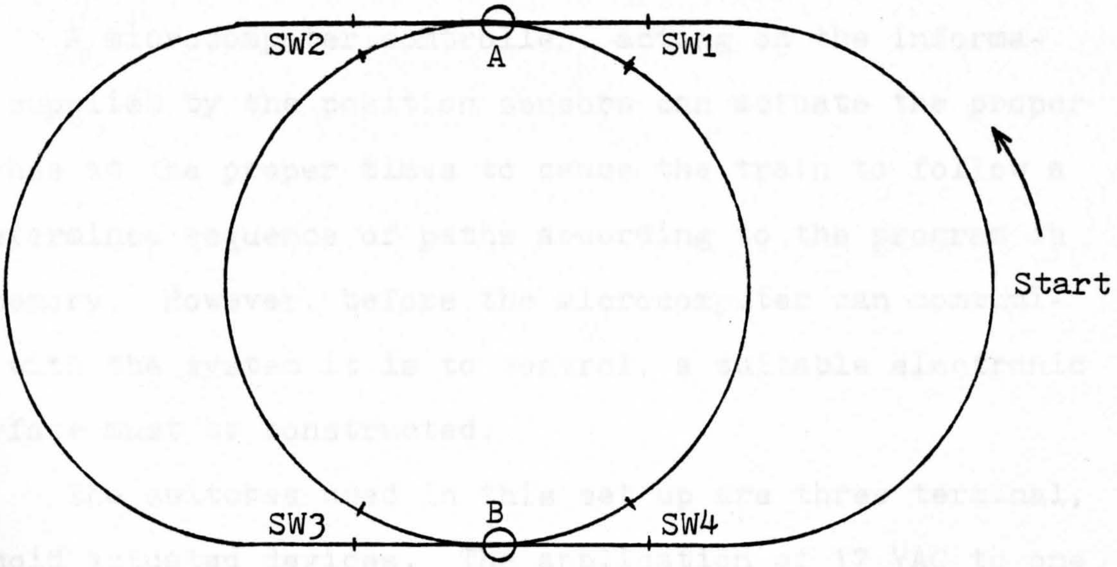


Fig. 3. Configuration of Train Track.

As can be seen in figure 3, a train starting from the location indicated may take any one of four circular paths defined by: (1) the large outside ellipse, (2) the inside circle, (3) the right hand ellipse consisting of the right side of the outside ellipse and the left side of the inside circle, or (4) the left hand ellipse consisting of the left side of the outside ellipse and the right side of the inside circle.

Position sensors are located at points A and B, which are the only two points common to all four possible paths. A cadmium sulfide photoresistor is positioned beneath the track bed at each location, with miniature incandescent lights above. These position sensors act as input devices.

The output devices are, of course, the four remote control switches labeled SW1, SW2, SW3, and SW4.

A microcomputer controller, acting on the information supplied by the position sensors can actuate the proper switches at the proper times to cause the train to follow a predetermined sequence of paths according to the program in its memory. However, before the microcomputer can communicate with the system it is to control, a suitable electronic interface must be constructed.

The switches used in this set up are three terminal, solenoid actuated devices. The application of 17 VAC to one pair of terminals causes the switch to set for straight through. Voltage applied to the other pair of terminals (one terminal is common in both cases) causes the switch to set for turn out. Three binary combinations are therefore sufficient for driving each switch. We choose them to be 00 for no action, 01 for straight through, and 10 for turn out.

Motorola manufactures an integrated circuit device called the Peripheral Interface Adapter (PIA). It is intended to simplify the chore of interfacing the M6800 microprocessor to physical systems. Coincidentally, the PIA is configured with two I/O ports of eight lines each. Either port can be initialized by the computer as an input port or an output port. This device lends itself perfectly to the application at hand. If we consider all four railroad switches to be one output device, then one eight bit PIA port can be designated as an output port to drive the four switches, which require two bits each.

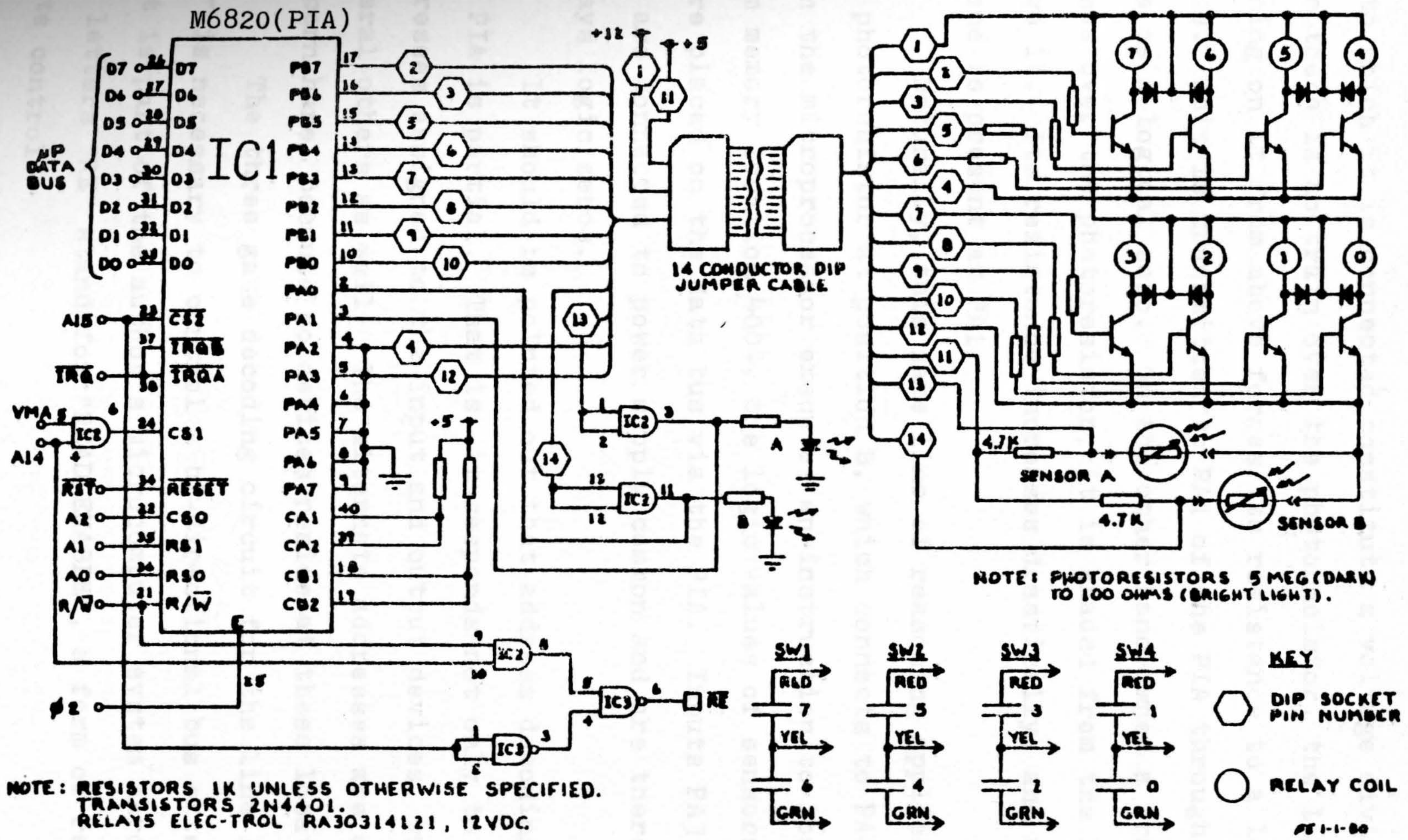
The remaining PIA I/O port is defined to be an input port. Two of its eight lines are used to convey position information to the microcomputer.

The schematic diagram of figure 4 shows the interface circuitry used to interconnect the microcomputer and the train layout. The M6820 PIA is labeled IC1. The only other integrated circuits are two common TTL gates.

In the author's system, all of the integrated circuits are located near the microprocessor itself. A ribbon cable connects the PIA to a set of relays and their transistor drivers, which are located on the track bed along with the position sensors.

The hexadecimal address 4006 is assigned to the four railroad switches, and I/O lines PB0 through PB7 are accordingly designated as output lines. The two position sensors are given the address 4004 and are connected to the PIA lines PA0 and PA1, which are defined as input lines. Lines D0 through D7 are connected to the microcomputer data bus.

When the microcomputer executes an instruction to store data in memory location 4006, the information on the data bus is transferred through the PIA to the eight lines which control the railroad switches. The lines having logical ones on them saturate the switching transistors to which they connect, energize the associated relay coils, and apply 17 VAC to the corresponding switch terminals.



NOTE: PHOTORESISTORS 5 MEG (DARK) TO 100 OHMS (BRIGHT LIGHT).

NOTE: RESISTORS 1K UNLESS OTHERWISE SPECIFIED.
TRANSISTORS 2N4401.
RELAYS ELEC-TROL RA30314121, 12VDC.

ƒB 1-1-80

Fig. 4. Interface Circuit Schematic.

The photoresistor at position A and the 4.7K resistor to which it is connected constitute a voltage divider. When there is no train over the photoresistor, the light shining on it from above forces its resistance to a low value. This is transmitted to PA1 of the PIA through the gate as a logical zero. On the other hand, when a train passes over the photoresistor, it is shaded from the light above it. Its resistance increases drastically, and a logical one is present at PA1.

Naturally, this same line of reasoning applies to the photoresistor at position B, which connects to PA0. When the microprocessor executes an instruction to load data from memory location 4004, the logic values of sensors A and B are placed on the data bus via the PIA. Inputs PA3 through PA7 are connected to power supply common and are therefore always logic zeros.

It should be pointed out that address decoding for the PIA is partial. That is, it responds not only to the addresses assigned to the input and output devices, but to several others as well. The alternate addresses are of no concern here, because no devices reside at these locations.

The three gate decoding circuit for the line labeled "$\overline{\text{RE}}$" is necessary to control a bi-directional bus extender that is part of the author's microcomputer system hardware. The letters "RE" stand for "READ ENABLE", a form of read/write control.

Now that the hardware interface has been taken care of, the control program must be approached. Let us start by developing a sequence of control events related to the system's I/O capabilities.

Let us say that we wish the train to take the following path: (1) once around the outside loop, (2) once around the inside circle, (3) once around the right side loop, (4) once around the left side loop, and (5) repeat the above steps until commanded to stop.

The train starts from the position indicated in figure 3. Table 4 outlines the sequence of events that must occur in order to accomplish the above objective.

TABLE 4

SEQUENCE OF EVENTS FOR TRAIN CONTROL

Train Position	SW1	SW2	SW3	SW4	PB Lines of PIA				Decimal Equivalent
					76	54	32	10	
A	X	X	X	X	00	00	00	00	0
B	X	S	X	X	00	10	00	00	32
A	X	X	S	S	00	00	10	10	10
B	S	X	X	X	10	00	00	00	128
A	X	X	X	T	00	00	00	01	1
B	T	X	X	X	01	00	00	00	64
A	X	X	X	S	00	00	00	10	2
B	S	T	X	X	10	01	00	00	144
A	X	X	T	X	00	00	01	00	4
B	X	X	X	X	00	00	00	00	0
A	X	X	X	T	00	00	00	01	1
B	T	X	X	X	01	00	00	00	64

Recall that the switch commands are 00 for no action (X), 01 for straight through (T), and 10 for turn out (S).

The command for each switch is given along with the binary values on each of the PIA output lines for each sequential position of the train. The decimal equivalents of the eight bit binary words are shown in the last column on the right.

The switches must be initialized so that they are all set straight through. Notice that at the end of the sequence they are again all straight through so that the sequence can be repeated indefinitely.

The BASIC program that achieves this control function is presented in Appendix D, along with the resultant M6800 hex code version used to program the microcomputer.

Conclusion

The compiler program presented in this paper provides any user who has access to a mainframe computer hosting PL/I the capability to produce useable M6800 assembly language programs from BASIC source code. The total cost of the demonstration control system, including the microcomputer, the interface components, and the train layout is less than two hundred dollars.

Admittedly, the control function performed by the train controller is not very complex. However, many industrial processes are equally simple. Automation of such processes might be very desirable if they could be done for a reasonable price. Reduction of engineering time through simplified software generation is the best way to reduce the cost of microprocessor control systems.

This paper has developed a compiler for simplifying microprocessor software generation in several logical steps. Chapter I describes the advantages of using microprocessor controllers instead of hard-wired logic and outlines the basic design philosophy behind the compiler itself.

Chapter II provides background information on the M6800 microprocessor, and establishes a special version of BASIC programming language, which is used as source code for the compiler program.

Chapter III is a treatment of compiler architecture in general, and the Modified BASIC to M6800 Hex Code Compiler in particular. All of the PL/I routines constituting the compiler are discussed in this chapter.

Finally, Chapter IV describes the compiler output and how it should be interpreted and used. As proof of the functionality of the compiler program, a model railroad layout is controlled by a microprocessor for which the control program is written in modified BASIC and translated by the compiler into microprocessor code.

Suggestions for expansion of this work include improved program storage capability and interrupt processing. The microprocessor hex code program resulting from compilation is presently stored permanently only on the printed output from the mainframe computer. Program storage on magnetic tape or disk with an appropriate interface to the microcomputer would greatly simplify the task of programming the controller. Also, the compiler presented here does not allow for the processing of interrupts. This is one of the microprocessor's more powerful features, and might be used to great advantage.

With the addition of sufficient memory to the microcomputer system, functions and special operations could be accomplished using the techniques developed here.

Whether or not the above improvements are undertaken in the future, the compiler as it stands is a useful and functional development tool for anyone operating on a budget.

APPENDIX A

Modified BASIC User's Manual

LANGUAGE SYNTAX

Statements

Keyword: DATA
Type: Non-executable
Format: line number DATA (value list)
Description: Supplies data to READ statement. DATA is never used without READ.

Keyword: END
Type: Executable
Format: line number END
Description: Terminates execution. The end statement is the last statement in a BASIC program.

Keyword: FOR
Type: Executable
Format: line number FOR (variable) = (positive integer or variable) TO (positive integer or variable) STEP (positive integer or variable)
Description: Causes execution to cycle through the designated loop in prescribed integer steps. The step value defaults to one.

Keyword: GOSUB
Type: Executable
Format: line number GOSUB (line number)
Description: Place subroutines at the end of the program after STOP and before DATA and END. When execution encounters GOSUB, control is transferred to the subroutine labeled with the indicated line number.

Keyword: GOTO
Type: Executable
Format: line number GOTO (line number)
Description: Unconditional branch. Causes an immediate jump to the specified line. The jump may be forward or backward.

Keyword: IF
Type: Executable
Format: line number IF (condition) $\left\{\begin{array}{l}\text{THEN (statement)}\\ \text{THEN (line number)}\\ \text{GOTO (line number)}\end{array}\right\}$
Description: Conditional statement. Directs the order of program execution depending upon the truth of some mathematical relation. The condition is tested. If false, execution continues with the next line number following the IF statement. If the condition is true, the statement following the THEN statement is executed or control is transferred to

the line number given after THEN or GOTO. The deciding condition is a simple relational expression in which two mathematical expressions are separated by a relational operator. The hierarchy of operations observed during the evaluation of the condition is the same as that for assignment statements.

Keyword: LET
Type: Arithmetic
Format: line number LET (variable) = (expression)
Description: Assigns a numeric value to a variable. The LET statement performs the calculations within the expression and assigns the numeric value to the indicated variable.

Keyword: NEXT
Type: Executable
Format: line number NEXT (variable)
Description: Together, the FOR and NEXT statements specify the boundaries of the program loop. The variable following NEXT is the same variable immediately following the associated FOR. When execution encounters the NEXT statement, the computer adds the STEP expression value to the variable and checks to see if the variable is still less than or equal to the terminal expression value. When the variable exceeds the terminal expression value, control falls through the loop to the statement following the NEXT statement.

Keyword: RDIN
Type: Executable
Format: line number RDIN (variable) from (hex address)
Description: Reads the information available at the input device residing at the specified hex address and assigns it to the variable.

Keyword: READ
Type: Executable
Format: line number READ (variable, variable, . . .)
Description: Read is used to assign to the listed variables those values which are obtained from the DATA statement. READ causes the variables listed to be assigned sequential values in the collection of DATA statements. Each time READ is encountered, the next value is assigned.

Keyword: REM
Type: Non-executable
Format: line number REM (comment)
Description: Provides a method of inserting notes and messages into the program source listing. The message can contain any characters on the keyboard. BASIC ignores anything on a line following the letters REM. In this version, the line number may not be used as a target for branching.

Keyword: RESTORE
Type: Executable
Format: line number RESTORE
Description: Causes the next READ statement to begin reading data from the first DATA statement in the program, regardless of where the last data value was found.

Keyword: RETURN
Type: Executable
Format: line number RETURN
Description: Used to exit a subroutine. Returns control to the line in the main program following the one containing the calling GOSUB.

Keyword: STOP
Type: Executable
Format: line number STOP
Description: Typically used to separate subroutines from the main program. The STOP statement is equivalent to GOTO END.

Keyword: WRTOUT
Type: Executable
Format: line number WRTOUT (variable) TO (hex address)
Description: Transfers the data in the variable to the output device residing at the microprocessor memory location given by the hex address.

Each BASIC program line is preceded by a line number. Only one statement per line is allowed, and each line has a maximum length of 72 characters.

General Rules of Syntax

Line Numbers

Each program line is preceded by a line number. Line numbers:

1. start in column 1;
2. are all numeric and range from 1 to 255;
3. serve as targets for branch statements.

This version of BASIC does not sort program lines into ascending order by line number prior to compilation.

Expressions

Expressions are combinations of numbers, variables, or functions in which the innermost parenthetical quantity is evaluated first, in the following order, left to right:

1. @ (exponentiation)
2. * or / (multiplication and division)
3. + or - (addition and subtraction)

Numbers

In this version of BASIC, only positive whole numbers through 255 may be used in calculations. Similarly, only positive whole numbers are returned as the results of calculations. This is attributable to the eight bit data bus of the microprocessor, which can handle numbers only up to and including FF_{16} at one time. It is most certainly possible to accommodate values beyond this range with the microprocessor, but such an undertaking is beyond the scope of this paper.

Variables

Variables are written as single letters, or as single letters followed by single digits. For example, A, Z3, and X7 are all legal variable names.

Hex Addresses

Hexadecimal numbers are used in this version of BASIC to specify microprocessor memory locations. Hexadecimal numbers are:

1. exactly four characters long;
2. any combination of "0123456789ABCDEF";
3. not permitted to contain radices.

Mathematical Operators, Relational Symbols, and Functions

The mathematical operators, relational operators, and functions available with this modified version of BASIC are as follows:

Operator	Example	Meaning
+	A+B	Add B to A.
-	A-B	Subtract B from A.
*	A*B	Multiply A by B.
/	A/B	Divide A by B.
@	A@B	Calculate A to the B power.
=	A=B	A is equal to B.
>	A>B	A is greater than B.
<	A<B	A is less than B.
NEQ	A NEQ B	A is not equal to B.
GEQ	A GEQ B	A is greater than or equal to B.
LEQ	A LEQ B	A is less than or equal to B.
SQR	SQR(A)	Calculate square root of A.

RUNNING THE COMPILER UNDER CMS

The compiler presented in this paper is designed to be run under IBM's Conversational Monitor System (CMS) [5]. Once the compiler has been loaded onto the user's disk (two cylinders required), it can be used to translate a BASIC program into M6800 code.

First the user must create a CMS file containing the BASIC source program. The compiler reads the source code from a file called SOURCE DATA. If several BASIC programs will be kept on a disk, it may be convenient to create the source program under another file name and file type, then copy it into SOURCE DATA before performing the compile.

Next, the user's virtual storage must be set to 512K, if it is not already. This is accomplished with the CP command DEFINE STORAGE AS 512K.

The program is now ready to run in the normal manner. The CMS EXEC procedure of figure 5 is helpful if the compiler is to be run many times.

USING THE PRINTED OUTPUT TO PROGRAM A MICROPROCESSOR

The first section of the printed report generated by the compiler is a sequential listing of the M6800 program steps that are equivalent to the program entered as BASIC source code.

The user simply loads the values listed in the "HEX CONTENTS" column into the microprocessor memory locations

```
FILE: RUNPRT    EXEC      A              YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

GLOBAL TXTLIB PLIOLIB
FILEDEF SOURCE DISK SOURCE DATA A (RECFM F LRECL 80 BLOCK 80
FILEDEF TRMINP DISK TRMINP DATA A (RECFM F LRECL 80 BLOCK 80
FILEDEF KEYINP DISK KEYINP DATA A (RECFM F LRECL 80 BLOCK 80
FILEDEF DATAOUT PRINTER
FILEDEF MATRIX DISK MATRIX DATA A (RECFM F LRECL 80 BLOCK 80
LOAD CONTROL (NODUP
```

Fig. 5. CMS EXEC Procedure RUNPRT EXEC.

listed in the "HEX ADDRESS" column. Execution is initiated from address 0100, as noted at the end of the program listing.

ERROR MESSAGES

Any errors encountered in the source program during compilation result in the suppression of hex code generation and a listing of errors along with the numbers of the BASIC program lines in which they were discovered. The meanings of the error messages are self-explanatory.

They are divided here into logical groups according to the BASIC statements which produce them.

General Errors

```
ILLEGAL CHARACTER GROUP
ILLEGAL KEYWORD FOLLOWS LINE NUMBER
LAST PROGRAM LINE MUST BE END STATEMENT
NUMERIC NOT FOUND IN COLUMN ONE
LINE NUMBER EXCEEDS FIVE DIGITS
CHARACTERS FOUND BEYOND COLUMN 72
LINE DOES NOT START WITH A VALID LINE NUMBER
```

LET

```
IDENTIFIER MUST FOLLOW LET
EQUAL SIGN IS NOT IN PROPER POSITION
PARENTHESES MUST ENCLOSE FUNCTION ARGUMENT
INVALID SYNTAX
```

RDIN

```
VARIABLE NAME MUST FOLLOW RDIN
FROM MUST FOLLOW VARIABLE NAME
HEX ADDRESS MUST FOLLOW FROM
MULTIPLE ENTRIES FOR HEX ADDRESS
```

IF

RELATIONAL OPERATOR MUST FOLLOW FIRST CONDITION
INVALID ACTION SPECIFIED
THEN OR GOTO MUST FOLLOW CONDITION
INVALID COMMAND FOLLOWS CONDITION
PARENTHESES MUST ENCLOSE FUNCTION ARGUMENT
INVALID SYNTAX
UNEQUAL NUMBER OF LEFT AND RIGHT PARENTHESES

DATA

DATA ENTRIES MUST BE NUMERIC
COMMAS REQUIRED BETWEEN DATA VALUES
DATA ENTRIES EXCEED CAPACITY OF 100

RETURN

CHARACTERS APPEAR AFTER RETURN STATEMENT

STOP

CHARACTERS APPEAR AFTER STOP STATEMENT

WRTOUT

VARIABLE NAME MUST FOLLOW WRTOUT
TO MUST FOLLOW VARIABLE NAME
HEX ADDRESS MUST FOLLOW TO
MULTIPLE ENTRIES FOR HEX ADDRESS

GOTO

LINE NUMBER MUST FOLLOW GOTO STATEMENT
MULTIPLE ARGUMENTS IN GOTO STATEMENT

RESTORE

CHARACTERS APPEAR AFTER RESTORE STATEMENT

READ

```
READ ARGUMENT MUST BE A VARIABLE NAME
COMMAS REQUIRED BETWEEN READ ARGUMENTS
```

END

```
CHARACTERS APPEAR AFTER END STATEMENT
```

FOR

```
IDENTIFIER MUST FOLLOW FOR
EQUAL SIGN IS NOT IN PROPER POSITION
POSITIVE INTEGER OR VARIABLE MUST PRECEDE TO
LITERAL PRECEDING TO IS NOT INTEGER
TO IS MISPLACED OR MISSING
MISSING NEXT STATEMENT
POSITIVE INTEGER OR VARIABLE MUST FOLLOW TO
LITERAL FOLLOWING TO IS NOT INTEGER
ONLY STEP MAY FOLLOW INTEGER AFTER TO
POSITIVE INTEGER MUST FOLLOW STEP
LITERAL FOLLOWING STEP IS NOT INTEGER
EXTRANEOUS CHARACTER AT END OF LINE
```

NEXT

```
NEXT MUST BE FOLLOWED BY A VARIABLE
IMPROPER FOR-NEXT PAIR
CHARACTERS APPEAR AFTER VARIABLE NAME
```

GOSUB

```
LINE NUMBER MUST FOLLOW GOSUB STATEMENT
MULTIPLE ARGUMENTS IN GOSUB STATEMENT
```

APPENDIX B

Compiler PL/I Listings

```
FILE: CONTROL  PLIOPT   A            YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

          /*YSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSU
          YSUYSU                                               YSUYSU
          YSUYSU   MASTER'S THESIS   FRED ESENWEIN  1980     YSUYSU
          YSUYSU                                               YSUYSU
          YSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSUYSU*/

 CONTROL : PROCEDURE OPTIONS (MAIN);

    /* TABLES */

      DCL   1    ERR_TBL                  (75)      EXT,
                 2    ERROR               CHAR      (40),
                 2    SRS_LIN             FIXED     (3);
      DCL   1    PRS_TBL                  (999)     EXT,
                 2    TOKEN               CHAR      (10),
                 2    CARD                FIXED     (4);
      DCL   1    SRS_TBL                  (500)     EXT,
                 2    COMMAND             CHAR      (72);
      DCL   1    UST                      (999)     EXT,
                 2    US_NTRY             CHAR      (3),
                 2    UST_PTR             FIXED     (4);
      DCL   1    KEY_TBL                  (30)      EXT,
                 2    KY_NTRY             CHAR      (10);
      DCL   1    TRM_TBL                  (30)      EXT,
                 2    TM_NTRY             CHAR      (3),
                 2    TRM_PRI             FIXED     (1);
      DCL   1    LIT_TBL                  (500)     EXT,
                 2    LT_NTRY             CHAR      (9);
      DCL   1    IDN_TBL                  (99)      EXT,
                 2    ID_NTRY             CHAR      (2);
      DCL   1    DTA_TBL                  (101)     EXT,
                 2    DT_PTR              FIXED     (3);
      DCL   1    LBL_AD                   (500)     EXT,
                 2    LBL_PTR             FIXED     (4),
                 2    LBL_ADR             CHAR      (4);
      DCL   1    TMP_AD                   (70)      EXT,
                 2    TMP_ADR             CHAR      (4);
      DCL   1    LIT_AD                   (500)     EXT,
                 2    LIT_ADR             CHAR      (4);
      DCL   1    IDN_AD                   (99)      EXT,
                 2    IDN_ADR             CHAR      (4);
      DCL 1 GOTBL                         (100)     EXT,
                 2    LAB_PTR             FIXED     (4),
                 2    GO_ADR              FIXED     (5),
                 2    BRANCH              CHAR      (5),
                 2    DESTN               FIXED     (4),
                 2    OFFSET              FIXED     (4);

    /* FILES */

      DCL   SOURCE     FILE       RECORD     INPUT,
            DATAOUT    STREAM     PRINT,
            TRMINP     FILE       RECORD     INPUT,
            MATRIX     FILE       RECORD;
```

```
FILE: CONTROL  PLIOPT   A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

         KEYINP     FILE        RECORD       INPUT:                          CON00560
                                                                             CON00570
      /* ITEMS */                                                            CON00580
                                                                             CON00590
      DCL   SUB                                FIXED     (3)   EXT,          CON00600
            SUB2                               FIXED     (3)   INIT(1),      CON00610
            SUB3                               FIXED     (3),                CON00620
            TRMSYM                             CHAR      (80),               CON00630
            KEYWORD                            CHAR      (80),               CON00640
            UST_IDX                            FIXED     (4)   EXT,          CON00650
            TKN_CNT                            FIXED     (3),                CON00660
            K                                  FIXED     (4),                CON00670
            TKN_END                            FIXED     (4),                CON00680
            CNT                                FIXED     (2)   EXT,          CON00690
            SRS_CNT                            FIXED     (3),                CON00700
            DT_IDX                             FIXED     (3)   EXT,          CON00710
            TGT_PTR                            FIXED     (4)   EXT,          CON00720
            CRD_CTR                            FIXED     (3)   EXT,          CON00730
            MAXTEMP                            FIXED     (3)   INIT(0),      CON00740
            TMP_STR                            FIXED     (3)   EXT,          CON00750
            LIN_PTR                            FIXED     (4)   EXT,          CON00760
            OPR                                CHAR      (3)   EXT,          CON00770
            OPR_PTR                            FIXED     (4)   EXT,          CON00780
            OP1                                CHAR      (3)   EXT,          CON00790
            OP1_PTR                            FIXED     (4)   EXT,          CON00800
            OP2                                CHAR      (3)   EXT,          CON00810
            OP2_PTR                            FIXED     (4)   EXT,          CON00820
            HEXADR                             CHAR      (4)   EXT,          CON00830
            HEXCON                             CHAR      (2)   EXT,          CON00840
            MDCON                              CHAR      (9)   EXT,          CON00850
            CMTS                               CHAR      (45)  EXT,          CON00860
            PGCNT                              FIXED     (2)   EXT,          CON00870
            GT_IDX                             FIXED     (4)   EXT,          CON00880
            LBL_IDX                            FIXED     (4)   EXT,          CON00890
            MEMORY_CNT                         FIXED     (5),                CON00900
            DECADR                             FIXED     (5)   EXT,          CON00910
            TMP_IDX                            FIXED     (2)   EXT,          CON00920
            LINE                               CHAR      (80)  EXT:          CON00930
                                                                             CON00940
      /* SUBROUTINES */                                                      CON00950
                                                                             CON00960
      DCL   PARSEZ      ENTRY:                                               CON00970
      DCL   ERRCHK      ENTRY:                                               CON00980
      DCL   KEYREC      ENTRY       (FIXED(4)):                              CON00990
      DCL   IDNREC      ENTRY       (FIXED(4)):                              CON01000
      DCL   TRMREC      ENTRY       (FIXED(4)):                              CON01010
      DCL   LITREC      ENTRY       (FIXED(4)):                              CON01020
      DCL   REMARK      ENTRY       (FIXED(4)):                              CON01030
      DCL   HEXREC      ENTRY       (FIXED(4)):                              CON01040
      DCL   TGTCHK      ENTRY:                                               CON01050
      DCL   LETAR       ENTRY:                                               CON01060
      DCL   GOTOAR      ENTRY:                                               CON01070
      DCL   IFAR        ENTRY:                                               CON01080
      DCL   FORAR       ENTRY:                                               CON01090
      DCL   NEXTAR      ENTRY:                                               CON01100
```

```
     DCL  GOSUBAR   ENTRY;
     DCL  RETRNAR   ENTRY;
     DCL  DATAAR    ENTRY;
     DCL  READAR    ENTRY;
     DCL  RESTRAR   ENTRY;
     DCL  STOPAR    ENTRY;
     DCL  ENDAR     ENTRY;
     DCL  WTOUTAR   ENTRY;
     DCL  RDINAR    ENTRY;
     DCL  RITEOUT   ENTRY;
     DCL  HEADING   ENTRY;
     DCL  DCHXCO    ENTRY;
     DCL  DCHXCO2   ENTRY;
     DCL  READCG    ENTRY;
     DCL  RESTRCG   ENTRY;
     DCL  RTRNCG    ENTRY;
     DCL  RDINCG    ENTRY;
     DCL  WTOUTCG   ENTRY;
     DCL  PLUSCG    ENTRY;
     DCL  MINUSCG   ENTRY;
     DCL  MTPLYCG   ENTRY;
     DCL  DIVIDCG   ENTRY;
     DCL  EQUALCG   ENTRY;
     DCL  GTCG      ENTRY;
     DCL  LTCG      ENTRY;
     DCL  LEQCG     ENTRY;
     DCL  GEQCG     ENTRY;
     DCL  NEQCG     ENTRY;
     DCL  EXPCG     ENTRY;
     DCL  STOPCG    ENTRY;
     DCL  ENDCG     ENTRY;
     DCL  GOTOCG    ENTRY;
     DCL  GOSUBCG   ENTRY;
     DCL  CONEOCG   ENTRY;
     DCL  SQRCG     ENTRY;
     DCL  BRANCHG   ENTRY;

      /* INITIALIZE */

          SUB=1;
          CRD_CTR = 0;
          CNT = 1;
          UST_IDX = 1;

     /* FILL ALL TABLES WITH INERT DATA */

          DO K = 1 TO 75 BY 1;
               ERROR(K) = ' ';
               SRS_LIN(K) = 0;
          END;

          DO K = 1 TO 999 BY 1;
               TOKEN(K) = ' ';
          END;
      DO K = 1 TO 999 BY 1;
             US_NTRY(K) = ' ';
             UST_PTR(K) = 0;
      END;

      DO K = 1 TO 30 BY 1;
             KY_NTRY(K) = ' ';
      END;

      DO K = 1 TO 30 BY 1;
             TM_NTRY(K) = ' ';
             TRM_PRI(K) = 9;
      END;

      LT_NTRY(1) = '1';
      LIT_ADR(1) = ' ';
      LT_NTRY(2) = '99';
      LIT_ADR(2) = ' ';
      DO K = 3 TO 500 BY 1;
             LT_NTRY(K) = ' ';
             LIT_ADR(K) = ' ';
      END;

      DO K = 1 TO 99 BY 1;
             ID_NTRY(K) = ' ';
             IDN_ADR(K) = ' ';
      END;

      DO K = 1 TO 101 BY 1;
             DT_PTR(K) = 0;
      END;

      DO K = 1 TO 500 BY 1;
             LBL_PTR(K) = 0;
             LBL_ADR(K) = ' ';
      END;

      DO K = 1 TO 70 BY 1;
             TMP_ADR(K) = ' ';
      END;

      DO K = 1 TO 100 BY 1;
             LAB_PTR(K) = 0;
             GO_ADR(K) = 0;
             BRANCH(K) = ' ';
             DESTN(K) = 0;
             OFFSET(K) =0;
       END;
/* OPEN FILES */

       ON ENDFILE(SOURCE) GO TO T002;
       ON ENDFILE(KEYINP) GO TO T001;
       ON ENDFILE(TRMINP) GO TO C005;
       OPEN FILE(TRMINP);
      OPEN FILE(KEYINP);
      OPEN FILE(SOURCE);
      OPEN FILE (MATRIX) OUTPUT;

   /* FILL TERMINAL TABLE FROM EXTERNAL FILE */

      K = 1;
   C004:READ FILE(TRMINP) INTO (TRMSYM);
      TM_NTRY(K) = TRMSYM;
      K = K + 1;
      GO TO C004;

   /* FILL KEYWORD TABLE FROM EXTERNAL FILE */

   C005:K = 1;
   C003:READ FILE(KEYINP) INTO (KEYWORD);
      KY_NTRY(K) = KEYWORD;
      K = K + 1;
      GO TO C003;

      /*******************************************
      ***********    LEXICAL ANALYSIS   ***********
      ******************************************/

   /* READ SOURCE LINE */

   T001:READ FILE (SOURCE) INTO (LINE);
      DISPLAY (CRD_CTR);
      CRD_CTR = CRD_CTR + 1;
      SRS_CNT = CRD_CTR;

   /* FOR ADDITION OF LINE NUMBER TO SOURCE LISTING */

      COMMAND(CRD_CTR) = SUBSTR(LINE, 1, 72);

   /* IGNORE ADDITIONAL ERRORS IF MORE THAN 25 ARE FOUND */

      IF_CNT > 25 THEN GO TO CABND;

   /* CHECK SOURCE LINE FOR ERRORS */

      CALL ERRCHK ;

   /* PARSE SOURCE LINE AND BUILD PARSE TABLE */

   C001:CALL PARSE2;
      GO TO T001;

   /* SEARCH TABLES FOR TOKEN MATCH */

   T002:K = 1;
      DO WHILE (TOKEN(K) ¬= ' ');
         TKN_END = UST_IDX;
         IF TOKEN(K) = 'REM       ' THEN CALL REMARK(K);
         CALL KEYREC(K);
```

```
        IF UST_IDX > TKN_END THEN GO TO C006;
        CALL TRMREC(K);
        IF UST_IDX > TKN_END THEN GO TO C006;
        CALL IDNREC(K);
        IF UST_IDX > TKN_END THEN GO TO C006;
        CALL LITREC(K);
        IF UST_IDX > TKN_END THEN GO TO C006;
        CALL HEXREC(K);
        IF UST_IDX > TKN_END THEN GO TO C006;
        ERROR(CNT) = 'ILLEGAL CHARACTER GROUP';
        SRS_LIN(CNT) = CARD(K);
        CNT = CNT + 1;
    C006:K = K + 1;
    END;

    /*********************************************
    ***********   SYNTAX ANALYSIS   ***********
    *********************************************/

  /* INITIALIZE */

    DT_IDX = 1;
    UST_IDX = 1;
    CRD_CTR = 1;
C007:DISPLAY (CRD_CTR);
    IF (US_NTRY(UST_IDX)='_') THEN GO TO END_OF_SYNTAX;
    SUB3 = CNT;

  /* IGNORE ADDITIONAL ERRORS IF MORE THAN 50 ARE FOUND */

    IF CNT > 50 THEN GO TO CABND;
    CALL TGTCHK;
    IF SUB3¬=CNT THEN GO TO C007;

  /* LINE NUMBER HAS BEEN VERIFIED */

    IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=23) THEN
        DO;
            CRD_CTR=CRD_CTR+1;
            UST_IDX=UST_IDX+1;
            GO TO C007;
        END;
    IF US_NTRY(UST_IDX)¬='KEY' THEN GO TO C008;
    IF UST_PTR(UST_IDX)=1 THEN
        DO;
            CALL LFTAR;
            IF TMP_STR > MAXTEMP THEN MAXTEMP = TMP_STR;
            GO TO C007;
        END;
    IF UST_PTR(UST_IDX)=2 THEN
        DO;
            CALL GOTOAR;
            GO TO C007;
        END;
    IF UST_PTR(UST_IDX)=3 THEN
```

```
         DO:
              CALL IFAR:
              IF TMP_STR > MAXTEMP THEN MAXTEMP = TMP_STR:
              IF MAXTEMP < 2 THEN MAXTEMP = 2:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=6 THEN
         DO:
              CALL FORAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=7 THEN
         DO:
              CALL NEXTAR:
              IF MAXTEMP = 0 THEN MAXTEMP = 1:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=8 THEN
         DO:
              CALL GOSUBAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=9 THEN
         DO:
              CALL RETRNAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=12 THEN
         DO:
              CALL DATAAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=13 THEN
         DO:
              CALL READAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=14 THEN
         DO:
              CALL RESTRAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=16 THEN
         DO:
              CALL STOPAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=17 THEN
         DO:
              CALL ENDAR:
              GO TO C007:
         END:
    IF UST_PTR(UST_IDX)=18 THEN
         DO:
              CALL WTOUTAR:
                GO TO C007;
            END:
        IF UST_PTR(UST_IDX)=19 THEN
            DO:
                CALL BDINAR:
                GO TO C007:
            END:
  C008:ERROR(CNT)='ILLEGAL KEYWORD FOLLOWS LINE NUMBER';
        SRS_LIN(CNT)=CRD_CTR:
        CNT=CNT+1;
        DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
            UST_IDX = UST_IDX + 1:
        END:
        UST_IDX = UST_IDX + 1:
        CRD_CTR = CRD_CTR + 1:
        GO TO C007:
  END_OF_SYNTAX:

  /* IS LAST SYMBOL AN END STATEMENT ? */
        IF (US_NTRY(UST_IDX-2) ¬= 'KEY')|(UST_PTR(UST_IDX-2) ¬= 17)
            THEN DO;
                ERROR(CNT) = 'LAST PROGRAM LINE MUST BE END STMT':
                SRS_LIN(CNT) = CRD_CTR:
            END:

  /* DIAGNOSTIC SUMMARY */

        IF ERROR(1)¬=' ' THEN
            DO:
                CNT = CNT + 2;
                ERROR(CNT)='ABEND - SEVERE ERRORS DETECTED.':
                CNT = CNT + 1:
                ERROR(CNT)='CODE GENERATION SUPPRESSED.':
                CNT = CNT + 1:
                GO TO CABND:
            END:
        ERROR(1)='NO DIAGNOSTICS GENERATED':
        ERROR(3)='MAXIMUM TEMPORARY STORAGE USED -':
        SRS_LIN(3)=MAXTEMP:
        CNT=4:

        /****************************************
        **********  CODE GENERATION  **********
        ****************************************/

        PGCNT=0:
        DECADR=0:

    /* HEAD FIRST OUTPUT PAGE */

        CALL HEADING:

    /* ENTER DATA TABLE INTO UP MEMORY */

        K = 1;
```

```
          DO WHILE (DT_PTR(K) ¬= 0);
               MDCON = LT_NTRY(DT_PTR(K));
               CALL DCHXC02;
               CMTS = 'DATA';
               CALL RITEOUT;
               K=K+1;
          END;

     /* ENTER LITERALS INTO UP MEMORY */

          K = 1;
          DO WHILE (LT_NTRY(K) ¬= ' ');
               MDCON = LT_NTRY(K);
               IF (VERIFY(MDCON,'0123456789 .')) ¬= 0 THEN GO TO C009;
               CALL DCHXC02;
               CMTS = 'CONSTANT';
               CALL RITEOUT;
               LIT_ADR(K)=HEXADR;
 C009:         K = K + 1;
          END;

     /* RESERVE UP MEMORY LOCATIONS FOR IDENTIFIERS */

          K = 1;
          DO WHILE (ID_NTRY(K) ¬= ' ');
               HEXCON = 'XX';
               MDCON = 'XXXXXXXXX';
               CMTS = ID_NTRY(K);
               CALL RITEOUT;
               IDN_ADR (K) = HEXADR;
               K = K + 1;
          END;

     /* RESERVE UP MEMORY LOCATIONS FOR TEMPORARY STORAGE */

          DO K=1 TO MAXTEMP BY 1;
               HEXCON = 'XX';
               MDCON='XXXXXXXXX';
               CMTS = 'TEMPORARY STORAGE LOCATION';
               CALL RITEOUT;
               TMP_ADR(K) = HEXADR;
          END;

     /* JUMP TO HEX ADDRESS 0100 */
          DECADR = 256;

     /* INITIALIZE INDEX REGISTER (DATA TABLE POINTER) */

          CALL RESTRCG;

     /* PREPARE TO READ MATRIX INFORMATION */

          GT_IDX = 1;
          K = 1;
          CLOSE FILE (MATRIX);
```

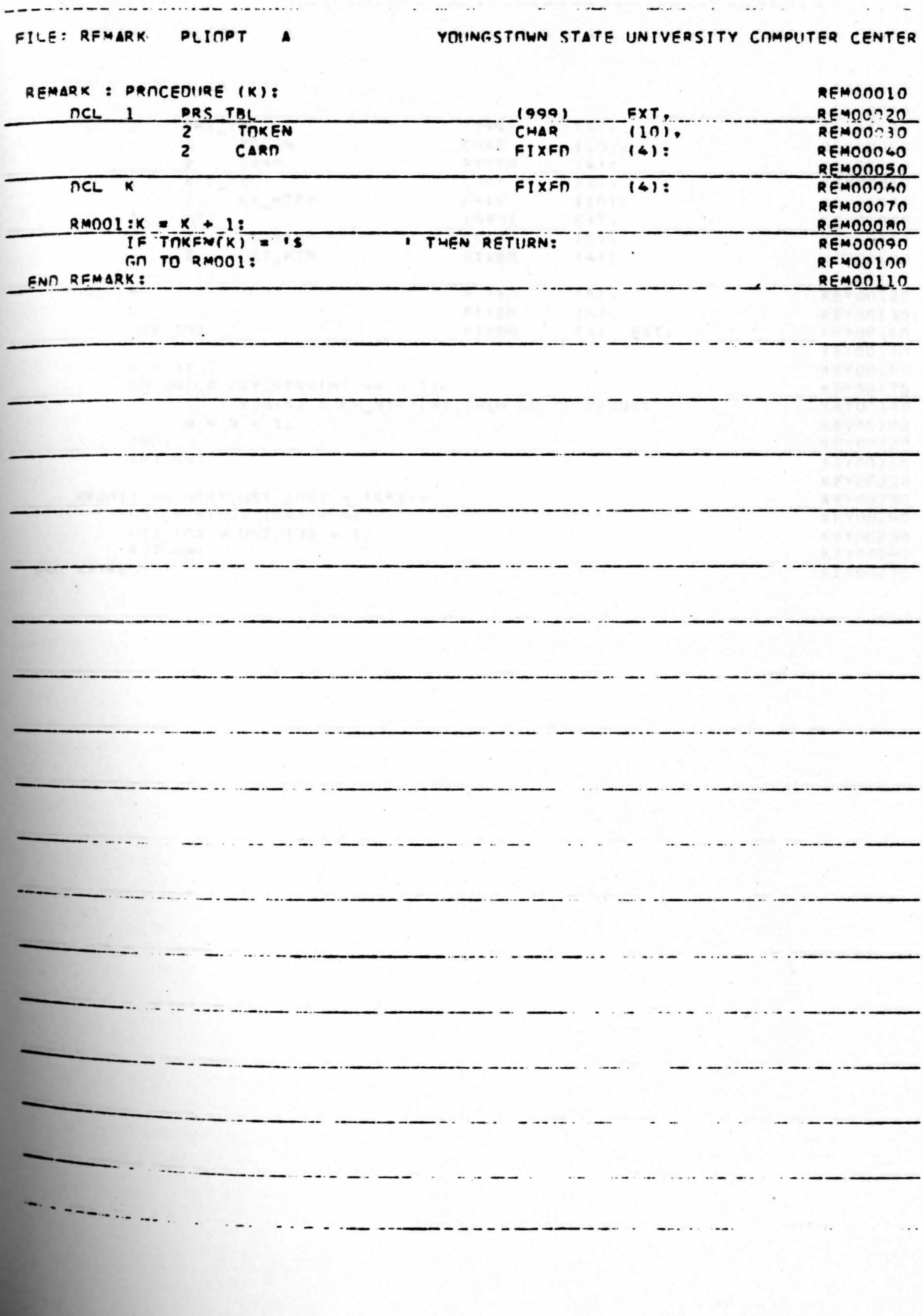
```
        ON ENDFILE (MATRIX) GO TO C010;                                 CON04960
        OPEN FILE (MATRIX) INPUT;                                       CON04970
                                                                        CON04980
    /* READ A MATRIX LINE */                                            CON04990
                                                                        CON05000
READ_MATRIX:                                                            CON05010
        READ FILE (MATRIX) INTO (LINE);                                 CON05020
        LIN_PTR = SUBSTR (LINE, 1, 4);                                  CON05030
        OPR = SUBSTR (LINE, 5, 3);                                      CON05040
        OPR_PTR = SUBSTR (LINE, 8, 4);                                  CON05050
        OP1 = SUBSTR (LINE, 12, 3);                                     CON05060
        OP1_PTR = SUBSTR (LINE, 15, 4);                                 CON05070
        OP2 = SUBSTR (LINE, 19, 3);                                     CON05080
        OP2_PTR = SUBSTR (LINE, 22, 4);                                 CON05090
                                                                        CON05100
    /* IS THIS THE FIRST ENCOUNTER WITH THIS LINE NUMBER ? */           CON05110
                                                                        CON05120
        IF LIN_PTR ¬= LBL_PTR(K) THEN                                   CON05130
            DO;                                                         CON05140
                TMP_IDX = 1;                                            CON05150
                K = K + 1;                                              CON05160
                LBL_PTR(K) = LIN_PTR;                                   CON05170
                CALL DCHXCD;                                            CON05180
                LBL_ADR(K) = HEXADR;                                    CON05190
            END;                                                        CON05200
                                                                        CON05210
    /* CALL APPROPRIATE CODE GENERATION SUBROUTINE */                   CON05220
                                                                        CON05230
        /* READ */                                                      CON05240
                                                                        CON05250
        IF((OPR = 'KEY')&(OPR_PTR = 13)) THEN                           CON05260
            DO;                                                         CON05270
                CALL READCG;                                            CON05280
                GO TO READ_MATRIX;                                      CON05290
            END;                                                        CON05300
                                                                        CON05310
        /* RESTORE */                                                   CON05320
                                                                        CON05330
        IF ((OPR = 'KEY')&(OPR_PTR = 14)) THEN                          CON05340
            DO;                                                         CON05350
                CALL RESTRCG;                                           CON05360
                GO TO READ_MATRIX;                                      CON05370
            END;                                                        CON05380
                                                                        CON05390
        /* RETURN */                                                    CON05400
                                                                        CON05410
        IF ((OPR = 'KEY')&(OPR_PTR = 9)) THEN                           CON05420
            DO;                                                         CON05430
                CALL RTRNCG;                                            CON05440
                GO TO READ_MATRIX;                                      CON05450
            END;                                                        CON05460
                                                                        CON05470
        /* RDIN */                                                      CON05480
                                                                        CON05490
        IF ((OPR = 'KEY')&(OPR_PTR = 19)) THEN                          CON05500
        DO:
            CALL RDINCG:
            GO TO READ_MATRIX:
        END:

    /* WRTOUT */

    IF ((OPR = 'KEY')&(OPR_PTR = 18)) THEN
        DO:
            CALL WTOUTCG:
            GO TO READ_MATRIX:
        END:

    /* + */

    IF ((OPR = 'TRM')&(OPR_PTR = 2)) THEN
        DO:
            CALL PLUSCG:
            GO TO READ_MATRIX:
        END:

    /* - */

    IF ((OPR = 'TRM')&(OPR_PTR = 1)) THEN
        DO:
            CALL MINUSCG:
            GO TO READ_MATRIX:
        END:

    /* * */

    IF ((OPR = 'TRM')&(OPR_PTR = 4)) THEN
        DO:
            CALL MTPLYCG:
            GO TO READ_MATRIX:
        END:

    /* / */

    IF ((OPR = 'TRM')&(OPR_PTR = 3)) THEN
        DO:
            CALL DIVIDCG:
            GO TO READ_MATRIX:
        END:

    /* = */

    IF ((OPR = 'TRM')&(OPR_PTR = 6)) THEN
        DO:
            CALL EQUALCG:
            GO TO READ_MATRIX:
        END:

    /* CONDITIONAL = */
    IF ((OPR = 'TRM')&(OPR_PTR = 27)) THEN
         DO;
              CALL CONFOCG;
              GO TO READ_MATRIX;
         END;

    /* > */

    IF ((OPR = 'TRM')&(OPR_PTR = 9)) THEN
         DO;
              CALL GTCG;
              GO TO READ_MATRIX;
         END;

    /* < */

    IF ((OPR = 'TRM')&(OPR_PTR = 10)) THEN
         DO;
              CALL LTCG;
              GO TO READ_MATRIX;
         END;

    /* LEQ */

    IF ((OPR = 'TRM')&(OPR_PTR = 24)) THEN
         DO;
              CALL LEQCG;
              GO TO READ_MATRIX;
         END;

    /* GEQ */

    IF ((OPR = 'TRM')&(OPR_PTR = 25)) THEN
         DO;
              CALL GEQCG;
              GO TO READ_MATRIX;
         END;

    /* NEQ */

    IF ((OPR = 'TRM')&(OPR_PTR = 26)) THEN
         DO;
              CALL NEQCG;
              GO TO READ_MATRIX;
         END;

    /* SQUARE_ROOT */

    IF ((OPR = 'TRM')&(OPR_PTR = 21)) THEN
         DO;
              CALL SQRCG;
              GO TO READ_MATRIX;
         END;

    /* ¬ */
```

```

      IF ((OPR = 'TRM')&(OPR_PTR = 5)) THEN
          DO:
               CALL EXPCG:
               GO TO READ_MATRIX:
          END:

      /* STOP */

      IF ((OPR = 'KEY')&(OPR_PTR = 16)) THEN
          DO:
               CALL STOPCG:
               GO TO READ_MATRIX:
          END:

      /* END */

      IF ((OPR = 'KEY')&(OPR_PTR = 17)) THEN
          DO:
               CALL ENDCG:
               GO TO READ_MATRIX:
          END:

      /* GOTO */

      IF ((OPR = 'KEY')&(OPR_PTR = 2)) THEN
          DO:
               CALL GOTOCG:
               GO TO READ_MATRIX:
          END:

      /* GOSUB */

      IF ((OPR = 'KEY')&(OPR_PTR = 8)) THEN
          DO:
               CALL GOSUBCG:
               GO TO READ_MATRIX:
          END:

      GO TO READ_MATRIX:

C010:
      MEMORY_CNT = DECADR:

  /* COMPLETE COMPILATION BY ASSIGNING DESTINATION ADDRESSES TO
     BRANCH STATEMENTS */

      CALL BRANCHG:

  /* WRITE TOTAL UP MEMORY REQUIRED */

      PUT FILE (DATAOUT) SKIP:
      PUT FILE (DATAOUT) SKIP EDIT ('PROGRAM STARTING ADDRESS',
          '0100') (X(5), A(24), X(1), A(4)):
      PUT FILE (DATAOUT) SKIP:
          PUT FILE (DATAOUT) SKIP EDIT ('TOTAL MEMORY USED :',
           MEMORY_CNT, 'BYTES') (X(5), A(19), X(2), F(5), X(1), A(5));

          /*****************************************************
          ************   SEND REPORTS TO PRINTER   ************
          *****************************************************/

      /* WRITE SOURCE LISTING WITH CARD NUMBER */

 CABND : PUT FILE (DATAOUT) PAGE;
          PUT FILE(DATAOUT) SKIP EDIT ('SOURCE LISTING')
              (X(11), A(14));
          PUT FILE(DATAOUT) SKIP;
          DO WHILE (SUB2 <= SRS_CNT);
              PUT FILE(DATAOUT) SKIP EDIT (COMMAND(SUB2), SUB2)
                  (X(3), A(72), F(3));
              SUB2 = SUB2 + 1;
          END;

      /* WRITE ERROR TABLE */

          TKN_CNT = CNT;
          CNT = 1;
          PUT FILE(DATAOUT) PAGE;
          PUT FILE(DATAOUT) SKIP EDIT ('ERROR TABLE')
              (X(11), A(11));
          PUT FILE(DATAOUT) SKIP;
          PUT FILE(DATAOUT) SKIP EDIT ('ERROR', 'SOURCE LINE')
              (X(3), A(5), X(35), A(11));
          PUT FILE(DATAOUT) SKIP;
          DO WHILE (CNT < TKN_CNT);
          PUT FILE(DATAOUT) SKIP EDIT (ERROR(CNT), SRS_LIN(CNT))
              (X(3), A(40), F(3));
          CNT = CNT + 1;
          END;


      /* WRITE KEYWORD TABLE */

          PUT FILE (DATAOUT) PAGE;
          PUT FILE (DATAOUT) SKIP EDIT ('KEYWORD TABLE')
              (X(11), A(13));
          PUT FILE (DATAOUT) SKIP;
          SUB = 1;
          DO WHILE (KY_NTRY(SUB) ¬= ' ');
              PUT FILE (DATAOUT) SKIP EDIT (KY_NTRY(SUB), SUB)
                  (X(3), A(10), X(3), F(3));
              SUB = SUB + 1;
          END;

      /* WRITE TERMINAL TABLE */

          PUT FILE (DATAOUT) PAGE;
          PUT FILE (DATAOUT) SKIP EDIT ('TERMINAL TABLE')
              (X(11), A(14));
```

```
        PUT FILE (DATAOUT) SKIP;
        SUB = 1;
        DO WHILE (TM_NTRY(SUB) ¬= ' ');
              PUT FILE (DATAOUT) SKIP EDIT (TM_NTRY(SUB), SUB)
                    (X(3), A(3), X(3), F(3));
              SUB = SUB + 1;
        END;

   /* WRITE IDENTIFIER TABLE */

        PUT FILE (DATAOUT) PAGE;
        PUT FILE (DATAOUT) SKIP EDIT ('IDENTIFIER TABLE')
              (X(11), A(16));
        PUT FILE (DATAOUT) SKIP;
        SUB = 1;
        DO WHILE (ID_NTRY(SUB) ¬= ' ');
              PUT FILE (DATAOUT) SKIP EDIT (ID_NTRY(SUB), SUB)
                    (X(3), A(2), X(3), F(3));
              SUB = SUB + 1;
        END;

   /* WRITE LITERAL TABLE */

        PUT FILE (DATAOUT) PAGE;
        PUT FILE (DATAOUT) SKIP EDIT ('LITERAL TABLE')
              (X(11), A(13));
        PUT FILE (DATAOUT) SKIP;
        SUB = 1;
        DO WHILE (LT_NTRY(SUB) ¬= ' ');
              PUT FILE (DATAOUT) SKIP EDIT (LT_NTRY(SUB), SUB)
                    (X(3), A(9), X(3), F(3));
              SUB = SUB + 1;
        END;

   /* WRITE UNIFORM SYMBOL TABLE */

        PUT FILE(DATAOUT) PAGE;
        PUT FILE(DATAOUT) SKIP EDIT ('UNIFORM SYMBOL TABLE')
              (X(11), A(20));
        PUT FILE(DATAOUT) SKIP;
        PUT FILE(DATAOUT) SKIP EDIT ('SYMBOL', 'POINTER')
              (X(3), A(6), X(3), A(7));
        SUB = 1;
        DO WHILE(SUB < UST_IDX);
              PUT FILE(DATAOUT) SKIP EDIT (US_NTRY(SUB), UST_PTR(SUB))
                    (X(3), A(3), X(6), F(4));
              SUB = SUB + 1;
        END;

   /* WRITE DATA TABLE */

        PUT FILE (DATAOUT) PAGE;
        PUT FILE (DATAOUT) SKIP EDIT ('DATA TABLE')
              (X(11),A(10));
        PUT FILE (DATAOUT) SKIP;
        PUT FILE (DATAOUT) SKIP EDIT ('LIT TBL POINTER')                CON08260
             (X(3),A(15));                                              CON08270
        SUB = 1;                                                        CON08280
        DO WHILE (DT_PTR(SUB)¬=0);                                      CON08290
           PUT FILE (DATAOUT) SKIP EDIT (DT_PTR(SUB))                   CON08300
                (X(3),F(3));                                            CON08310
           SUB = SUB + 1;                                               CON08320
        END;                                                            CON08330
                                                                        CON08340
     /* CLOSE FILES */                                                  CON08350
                                                                        CON08360
        CLOSE FILE(SOURCE);                                             CON08370
        CLOSE FILE (MATRIX);                                            CON08380
        CLOSE FILE (KEYINP);                                            CON08390
        CLOSE FILE(TRMINP);                                             CON08400
  END CONTROL;                                                          CON08410
```

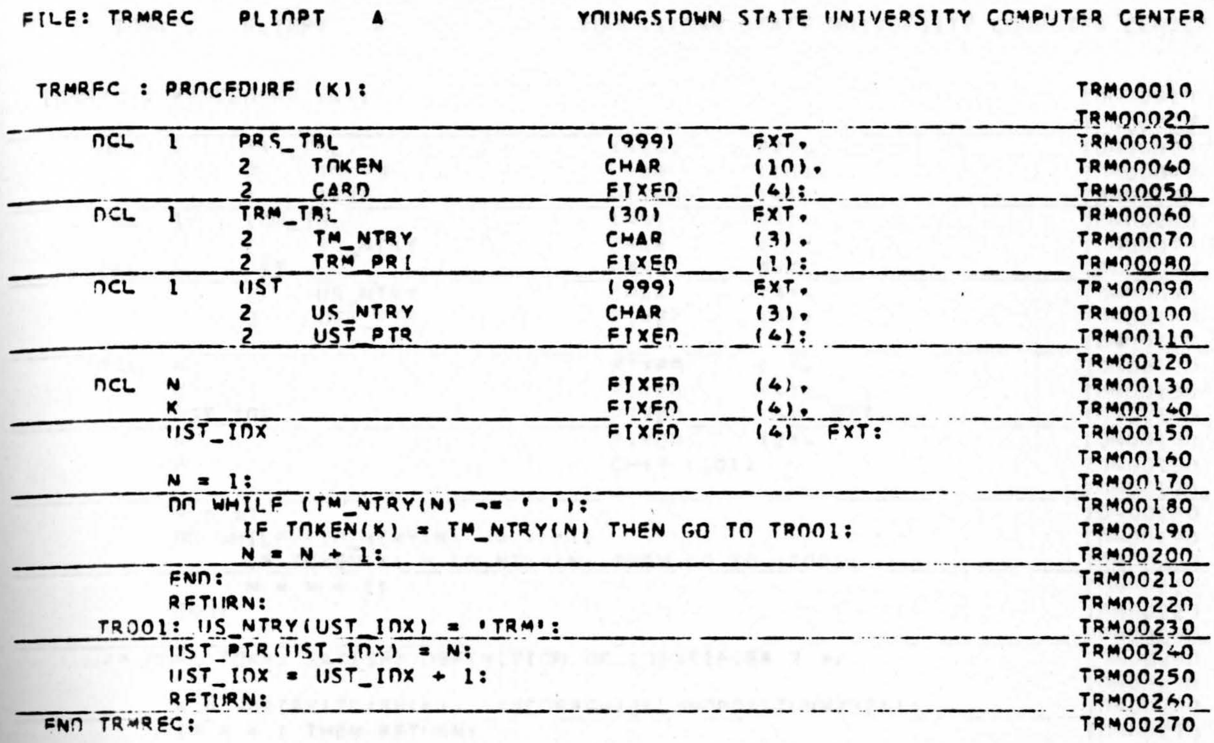
```
 PARSE2 : PROCEDURE;

     DCL  1    PRS_TBL                         (999)      EXT.
               2     TOKEN                     CHAR       (10).
               2     CARD                      FIXED      (4);

     DCL   SUB                                 FIXED      (3)   EXT.
           N                                   FIXED      (2).
           P                                   FIXED      (2).
           LINE                                CHAR       (80)   EXT.
           CRD_CTR                             FIXED      (3)   EXT.
           BREAK                               CHAR       (80).
           BRK_CHR                             CHAR       (12)
                INIT(' -+/*@=()><.').
           ALF_CHR                             CHAR       (37)
                INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.');

     /* CHECK FOR LEADING BLANKS */

           N = VERIFY (LINE, BRK_CHR);
           IF N=0 THEN N=73;
           LINE = SUBSTR(LINE, N);

           DO WHILE (LINE ¬= ' ');
                N = VERIFY (LINE, ALF_CHR);
                TOKEN(SUB) = SUBSTR (LINE, 1, N-1);
                CARD(SUB) = CRD_CTR;
                SUB = SUB + 1;
                LINE = SUBSTR (LINE, N);
                N = VERIFY (LINE, BRK_CHR);
                IF N = 0 THEN N = 73;
                BREAK = SUBSTR (LINE, 1, N-1);
      T005:     P = VERIFY (BREAK, ' ');
                IF P = 0 THEN GO TO T004;
                TOKEN(SUB) = SUBSTR (BREAK, P, 1);
                CARD(SUB) = CRD_CTR;
                SUB = SUB + 1;
                BREAK = SUBSTR (BREAK, P + 1);
                GO TO T005;
      T004:     LINE = SUBSTR (LINE, N);
           END;

           TOKEN(SUB) = '$         ';
           CARD(SUB) = CRD_CTR;
           SUB = SUB + 1;
           RETURN;
   END PARSE2;
```

```
ERRCHK : PROCEDURE:                                                          ERR00010
                                                                             ERR00020
      DCL 1 ERR_TBL                              (25)       EXT.             ERR00030
              2    ERROR                         CHAR(40).                   ERR00040
              2    SRS_LIN                       FIXED(3):                   ERR00050
                                                                             ERR00060
      DCL LINE                                   CHAR(80)   EXT.             ERR00070
          CNT                                    FIXED(2)   EXT.             ERR00080
          CRD_CTR                                FIXED(3)   EXT.             ERR00090
          CHK                                    CHAR(1).                    ERR00100
          ERR1                                   FIXED(1).                   ERR00110
          LABEL                                  FIXED(2).                   ERR00120
          TAIL                                   CHAR(8):                    ERR00130
                                                                             ERR00140
          CHK = SUBSTR (LINE. 1. 1):                                         ERR00150
          ERR1 = VERIFY (CHK. '0123456789'):                                 ERR00160
          IF ERR1 ¬= 0 THEN GO TO EC001:                                     ERR00170
                                                                             ERR00180
      EC002: LABEL = VERIFY (LINE. '0123456789'):                            ERR00190
          IF LABEL > 6 THEN GO TO EC003:                                     ERR00200
      EC004: TAIL = SUBSTR (LINE. 73. 8):                                    ERR00210
          IF TAIL ¬= ' ' THEN GO TO EC005:                                   ERR00220
      EC006: RETURN:                                                         ERR00230
                                                                             ERR00240
      EC001: ERROR(CNT) = 'NUMERIC NOT FOUND IN COLUMN ONE':                 ERR00250
          SRS_LIN(CNT) = CRD_CTR:                                            ERR00260
          CNT = CNT + 1:                                                     ERR00270
          GO TO EC002:                                                       ERR00280
                                                                             ERR00290
      EC003: ERROR(CNT) = 'LINE NUMBER EXCEEDS FIVE DIGITS':                 ERR00300
          SRS_LIN(CNT) = CRD_CTR:                                            ERR00310
          CNT = CNT + 1:                                                     ERR00320
          GO TO EC004:                                                       ERR00330
      EC005: ERROR(CNT) = 'CHARACTERS FOUND BEYOND COLUMN 72':               ERR00340
          SRS_LIN(CNT) = CRD_CTR:                                            ERR00350
          CNT = CNT + 1:                                                     ERR00360
          GO TO EC006:                                                       ERR00370
   END ERRCHK:                                                               ERR00380
```



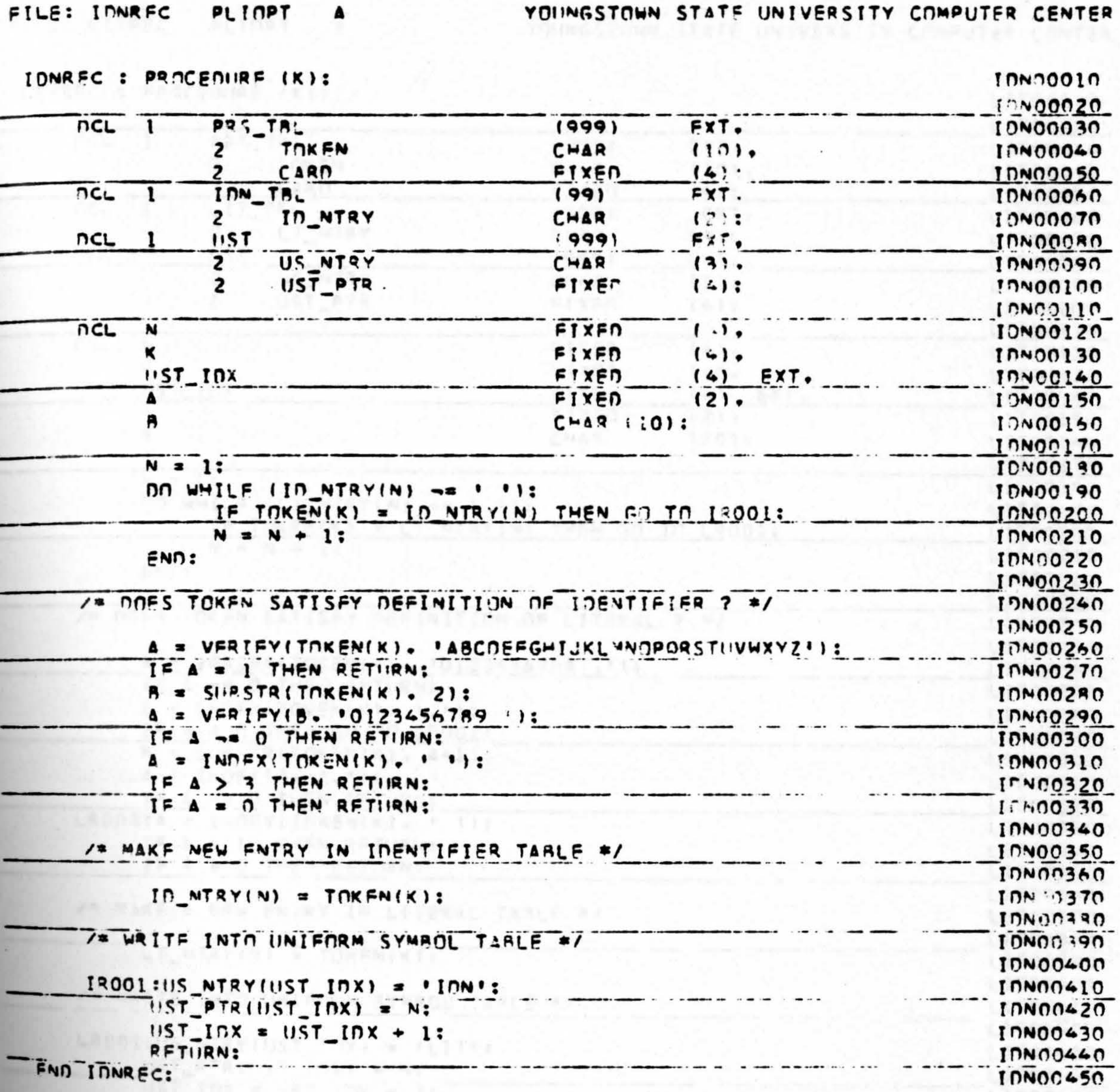
```
FILE: REMARK   PLIOPT   A                          YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

REMARK : PROCEDURE (K);                                                          REM00010
    DCL  1      PRS_TBL                       (999)      EXT,                    REM00020
                2     TOKEN                   CHAR       (10),                   REM00030
                2     CARD                    FIXED      (4);                    REM00040
                                                                                 REM00050
    DCL  K                                    FIXED      (4);                    REM00060
                                                                                 REM00070
    RM001:K = K + 1;                                                             REM00080
          IF TOKEN(K) = '$        ' THEN RETURN;                                 REM00090
          GO TO RM001;                                                           REM00100
 END REMARK;                                                                     REM00110
```

```
FILE: KEYREC   PLIOPT   A                 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 KEYREC : PROCEDURE (K);

      DCL  1      PRS_TBL                    (999)      EXT,
                  2     TOKEN                CHAR       (10),
                  2     CARD                 FIXED      (4);
      DCL  1      KEY_TBL                    (30)       EXT,
                  2     KY_NTRY              CHAR       (10);
      DCL  1      UST                        (999)      EXT,
                  2     US_NTRY              CHAR       (3),
                  2     UST_PTR              FIXED      (4);

      DCL  N                                 FIXED      (4),
           K                                 FIXED      (4),
           UST_IDX                           FIXED      (4)  EXT;

           N = 1;
           DO WHILE (KY_NTRY(N) ¬= ' ');
                  IF TOKEN(K) = KY_NTRY(N) THEN GO TO KR001;
                  N = N + 1;
           END;
           RETURN;

      KR001: US_NTRY(UST_IDX) = 'KEY';
           UST_PTR(UST_IDX) = N;
           UST_IDX = UST_IDX + 1;
           RETURN;
 END KEYREC;
```



FILE: TRMREC   PLIOPT   A   YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
 TRMREC : PROCEDURE (K);

      DCL  1      PRS_TBL                  (999)      EXT,
                  2     TOKEN              CHAR       (10),
                  2     CARD               FIXED      (4);
      DCL  1      TRM_TBL                  (30)       EXT,
                  2     TM_NTRY            CHAR       (3),
                  2     TRM_PRI            FIXED      (1);
      DCL  1      UST                      (999)      EXT,
                  2     US_NTRY            CHAR       (3),
                  2     UST_PTR            FIXED      (4);

      DCL  N                               FIXED      (4),
           K                               FIXED      (4),
           UST_IDX                         FIXED      (4)   EXT;

           N = 1;
           DO WHILE (TM_NTRY(N) ¬= ' ');
              IF TOKEN(K) = TM_NTRY(N) THEN GO TO TR001;
              N = N + 1;
           END;
           RETURN;
     TR001: US_NTRY(UST_IDX) = 'TRM';
           UST_PTR(UST_IDX) = N;
           UST_IDX = UST_IDX + 1;
           RETURN;
  END TRMREC;
```



```
FILE: IDNREC    PLIOPT    A              YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 IDNREC : PROCEDURE (K);

      DCL  1     PRS_TBL                        (999)      EXT,
                 2     TOKEN                    CHAR       (10),
                 2     CARD                     FIXED      (4);
      DCL  1     IDN_TBL                        (99)       EXT,
                 2     ID_NTRY                  CHAR       (2);
      DCL  1     UST                            (999)      EXT,
                 2     US_NTRY                  CHAR       (3),
                 2     UST_PTR                  FIXED      (4);

      DCL  N                                    FIXED      (4),
           K                                    FIXED      (4),
           UST_IDX                              FIXED      (4)  EXT,
           A                                    FIXED      (2),
           B                                    CHAR (10);

           N = 1;
           DO WHILE (ID_NTRY(N) ¬= ' ');
                IF TOKEN(K) = ID_NTRY(N) THEN GO TO IR001;
                N = N + 1;
           END;

      /* DOES TOKEN SATISFY DEFINITION OF IDENTIFIER ? */

           A = VERIFY(TOKEN(K), 'ABCDEFGHIJKLMNOPQRSTUVWXYZ');
           IF A = 1 THEN RETURN;
           B = SUBSTR(TOKEN(K), 2);
           A = VERIFY(B, '0123456789 ');
           IF A ¬= 0 THEN RETURN;
           A = INDEX(TOKEN(K), ' ');
           IF A > 3 THEN RETURN;
           IF A = 0 THEN RETURN;

      /* MAKE NEW ENTRY IN IDENTIFIER TABLE */

           ID_NTRY(N) = TOKEN(K);

      /* WRITE INTO UNIFORM SYMBOL TABLE */

      IR001:US_NTRY(UST_IDX) = 'IDN';
           UST_PTR(UST_IDX) = N;
           UST_IDX = UST_IDX + 1;
           RETURN;
   END IDNREC;
```

```
FILE: LITREC   PLIOPT    A                  YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

LITREC : PROCEDURE (K);

     DCL  1    PRS_TBL                  (999)      EXT,
               2     TOKEN              CHAR       (10),
               2     CARD               FIXED      (4);
     DCL  1    LIT_TBL                  (500)      EXT,
               2     LT_NTRY            CHAR       (9);
     DCL 1     UST                      (999)      EXT,
               2     US_NTRY            CHAR       (3),
               2     UST_PTR            FIXED      (4);

     DCL  N                             FIXED      (4),
          K                             FIXED      (4),
          UST_IDX                       FIXED      (4)  EXT,
          A                             FIXED      (2),
          B                             CHAR       (20);

          N = 1;
          DO WHILE (LT_NTRY(N) ¬= ' ');
               IF TOKEN(K) = LT_NTRY(N) THEN GO TO LR001;
               N = N + 1;
          END;

     /* DOES TOKEN SATISFY DEFINITION OF LITERAL ? */

          A = VERIFY(TOKEN(K), '0123456789 .');
          IF A ¬= 0 THEN RETURN;
          A = INDEX(TOKEN(K), '.');
          IF A = 0 THEN GO TO LR002;
          B = SUBSTR(TOKEN(K), A+1);
          A = INDEX(B, '.');
          IF A ¬= 0 THEN RETURN;
     LR002:A = INDEX(TOKEN(K), ' ');
          IF A > 10 THEN RETURN;
          IF A = 0 THEN RETURN;

     /* MAKE A NEW ENTRY IN LITERAL TABLE */

          LT_NTRY(N) = TOKEN(K);

     /* WRITE INTO UNIFORM SYMBOL TABLE */

     LR001:US_NTRY(UST_IDX) = 'LIT';
           UST_PTR(UST_IDX) = N;
           UST_IDX = UST_IDX + 1;
           RETURN;
  END LITREC;
```

```
FILE: HEXREC    PLIOPT    A                    YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

HEXREC : PROCEDURE (K);

    DCL  1    PRS_TBL                  (999)      EXT,
              2    TOKEN               CHAR       (10),
              2    CARD                FIXED      (4);
    DCL  1    LIT_TBL                  (500)       EXT,
              2    LT_NTRY             CHAR       (9):
    DCL  1    UST                      (999)      EXT,
              2    US_NTRY             CHAR       (3).
              2    UST_PTR             FIXED      (4):

    DCL  K                             FIXED      (4),
         UST_IDX                       FIXED      (4)  EXT,
         N                             FIXED      (4),
         A                             FIXED      (2):

         N = 1:
         DO WHILE (LT_NTRY(N) ¬= ' ');
              IF TOKEN(K) = LT_NTRY(N) THEN GO TO HR001:
              N = N + 1:
         END:

    /* DOES TOKEN SATISFY DEFINITION OF HEXADECIMAL LITERAL ? */

         A = VERIFY(TOKEN(K), '0123456789ABCDEF '):
         IF A ¬= 0 THEN RETURN:
         A = INDEX(TOKEN(K), '.'):
         IF A ¬= 0 THEN RETURN:
         A = INDEX(TOKEN(K), ' '):
         IF A ¬= 5 THEN RETURN:

    /* MAKE A NEW ENTRY IN LITERAL TABLE */

         LT_NTRY(N) = TOKEN(K):

    /* WRITE INTO UNIFORM SYMBOL TABLE */

    HR001:US_NTRY(UST_IDX) = 'LIT':
         UST_PTR(UST_IDX) = N:
         UST_IDX = UST_IDX + 1:
         RETURN:
 END HEXREC:
```

```
FILE: TGTCHK    PLIOPT    A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 TGTCHK : PROCEDURE;
      DCL   1      UST                        (999)       EXT,
                   2     US_NTRY              CHAR        (3),
                   2     UST_PTR              FIXED       (4);
      DCL   1      ERR_TBL                    (75)        EXT,
                   2     ERROR                CHAR        (40),
                   2     SRS_LIN              FIXED       (3);
      DCL   1      LIT_TBL                    (500)        EXT,
                   2     LT_NTRY             CHAR        (9);
      DCL   UST_IDX                           FIXED       (4)   EXT,
            CNT                               FIXED       (2)   EXT,
            CRD_CTR                           FIXED       (3)   EXT,
            TGT_PTR                           FIXED       (4)   EXT;

    /* IS THE FIRST UNIFORM SYMBOL A VALID LINE NUMBER ? */

          IF US_NTRY(UST_IDX) ¬= 'LIT' THEN GO TO TC001;

    /* CHECK FOR ALL NUMERICS */

          IF(VERIFY(LT_NTRY(UST_PTR(UST_IDX)),'0123456789 ')¬=0)
               THEN GO TO TC001;

    /* LINE NUMBER OK */
          TGT_PTR = UST_PTR(UST_IDX);
          UST_IDX = UST_IDX + 1;
          RETURN;
    TC001: ERROR(CNT) = 'LINE DOES NOT START WITH A VALID NUMBER';
          SRS_LIN(CNT) = CRD_CTR;
          CNT = CNT + 1;
          DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
               UST_IDX = UST_IDX + 1;
          END;
          UST_IDX = UST_IDX + 1;
          CRD_CTR = CRD_CTR + 1;
          RETURN;
 END TGTCHK;
```

```
FILE: LETAR     PLIOPT   A               YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 LETAR : PROCEDURE:
    DCL  1   UST                        (999)       EXT,
             2    US_NTRY               CHAR        (3),
             2    UST_PTR               FIXED       (4):
    DCL  1   ERR_TBL                    (75)        EXT,
             2    ERROR                 CHAR        (40),
             2    SRS_LIN               FIXED       (3):
    DCL  1   TRM_TBL                    (30)        EXT,
             2    TM_NTRY               CHAR        (3),
             2 TRM_PRI                  FIXED       (1):
    DCL  1   MTX_FIL,
             2    LBL_PTR               CHAR        (4),
             2    OPR                   CHAR        (3),
             2    OPR_PTR               CHAR        (4),
             2    OP1                   CHAR        (3),
             2    OP1_PTR               CHAR        (4),
             2    OP2                   CHAR        (3),
             2    OP2_PTR               CHAR        (4),
             2    FILL                  CHAR        (55) INIT((55)' '):
    DCL  1   OPND                       CONTROLLED,
             2    OPND_STACK_TBL        CHAR        (3),
             2    OPND_STACK_PTR        FIXED       (4):
    DCL  1   OPTR                       CONTROLLED,
             2    OPTR_STACK_TBL        CHAR        (3),
             2    OPTR_STACK_PTR        FIXED       (4),
             2    OPTR_PRIORITY         FIXED       (4):
    DCL  1   FCN_OPND                   CONTROLLED,
             2    FCN_OPND_STACK_TBL    CHAR        (3),
             2    FCN_OPND_STACK_PTR    FIXED       (4):
    DCL  1   FCN_OPTR                   CONTROLLED,
             2    FCN_OPTR_STACK_TBL    CHAR        (3),
             2    FCN_OPTR_STACK_PTR    FIXED       (4),
             2    FCN_OPTR_PRIORITY     FIXED       (4):
    DCL  UST_IDX                        FIXED       (4)   EXT,
         CNT                            FIXED       (2)   EXT,
         AREA                           CHAR        (7),
         ZEROO                          CHAR        (1)   INIT('0'),
         BLANK                          CHAR        (1)   INIT(' '),
         CRD_CTR                        FIXED       (3)   EXT,
         TGT_PTR                        FIXED       (4)   EXT,
         TMP_STR                        FIXED       (3)   EXT,
         NO_FLG                         FIXED       (1),
         PAR_CT                         FIXED       (3),
         PRIORITY                       FIXED       (3),
         FCN_PAR_CT                     FIXED       (3),
         FCN_FLG                        FIXED       (1),
         ASGN                           FIXED       (4),
         FCN_PTR                        FIXED       (4):
    DCL  MATRIX                         FILE        RECORD:

    NO_FLG, PAR_CT, PRIORITY, FCN_PAR_CT, FCN_FLG, TMP_STR = 0:

    /* NEXT SYMBOL MUST BE AN IDENTIFIER */

         UST_IDX = UST_IDX + 1:
```

```
         IF US_NTRY(UST_IDX)¬='IDN' THEN
               DO: ERROR(CNT)= 'IDENTIFIER MUST FOLLOW LET':
                    GO TO LA001_ERROR:
               END:
         ASGN = UST_PTR(UST_IDX):

    /* NEXT SYMBOL MUST BE '=' */

         UST_IDX = UST_IDX + 1:
         IF(US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=6) THEN
               DO: ERROR(CNT)='EQUAL SIGN IS NOT IN PROPER POSITION':
                    GO TO LA001_ERROR:
               END:

    /* CHECK FOR LEFT PARENTHESIS, RIGHT PARENTHESIS, LINE END */

LA002:   UST_IDX = UST_IDX + 1:
         IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=7) THEN
               DO:
                    IF FCN_FLG = 1 THEN FCN_PAR_CT = FCN_PAR_CT + 10:
                    PAR_CT = PAR_CT + 10:
                    GO TO LA002;
               END:
LA003:   IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=8) THEN
               DO:
                    IF FCN_FLG=1 THEN FCN_PAR_CT = FCN_PAR_CT-10:
                    PAR_CT = PAR_CT - 10:
                    UST_IDX = UST_IDX + 1:
                    GO TO LA003:
               END:
         IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=23) THEN
               DO:
                    NO_FLG = 1:
                    GO TO LA004_POP;
               END:

    /* IS SYMBOL A FUNCTION NAME ? */

         IF(US_NTRY(UST_IDX) ='TRM')&(UST_PTR(UST_IDX)>12) &
          (UST_PTR(UST_IDX)<23) THEN
               DO:
                    FCN_FLG = 1:
                    FCN_PTR = UST_PTR(UST_IDX):
                    UST_IDX = UST_IDX + 1:
                    IF(US_NTRY(UST_IDX)¬='TRM')|
                         (UST_PTR(UST_IDX)¬=7) THEN
                         DO:
               ERROR(CNT)='PARENTHESIS MUST ENCLOSE FUNCTION ARG':
                              GO TO LA001_ERROR:
                         END:
                    UST_IDX = UST_IDX - 1:
                    GO TO LA002:
               END:

    /* IS SYMBOL A LITERAL OR IDENTIFIER ? */
```

```
                                                                       LET01110
        IF(US_NTRY(UST_IDX)¬='LIT')&(US_NTRY(UST_IDX)¬='IDN') THEN     LET01120
            DO:                                                        LET01130
                ERROR(CNT)='INVALID SYNTAX':                           LET01140
                GO TO LA001_ERROR:                                     LET01150
            END:                                                       LET01160
                                                                       LET01170
     /* PUSH SYMBOL ONTO APPROPRIATE STACK */                          LET01180
                                                                       LET01190
        IF FCN_FLG = 1 THEN                                            LET01200
            DO:                                                        LET01210
                ALLOCATE FCN_OPND:                                     LET01220
                FCN_OPND_STACK_TBL = US_NTRY(UST_IDX):                 LET01230
                FCN_OPND_STACK_PTR = UST_PTR(UST_IDX):                 LET01240
                GO TO LA005:                                           LET01250
            END:                                                       LET01260
        ALLOCATE OPND:                                                 LET01270
        OPND_STACK_TBL = US_NTRY(UST_IDX):                             LET01280
        OPND_STACK_PTR = UST_PTR(UST_IDX):                             LET01290
                                                                       LET01300
     /* CHECK FOR LEFT PARENTHESIS, RIGHT PARENTHESIS, LINE END */     LET01310
                                                                       LET01320
LA005:  UST_IDX = UST_IDX + 1:                                         LET01330
        IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=7) THEN           LET01340
            DO:                                                        LET01350
                IF FCN_FLG = 1 THEN FCN_PAR_CT = FCN_PAR_CT+10:        LET01360
                PAR_CT = PAR_CT + 10:                                  LET01370
                GO TO LA005:                                           LET01380
            END:                                                       LET01390
LA006:  IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=8) THEN           LET01400
            DO:                                                        LET01410
                IF FCN_FLG = 1 THEN FCN_PAR_CT = FCN_PAR_CT-10:        LET01420
                PAR_CT = PAR_CT -10:                                   LET01430
                IF(FCN_PAR_CT=0)&(FCN_FLG=1) THEN                      LET01440
                    DO:                                                LET01450
                        IF(ALLOCATION(FCN_OPTR)=0) THEN                LET01460
                            GO TO LA007_POP_FCN:                       LET01470
                        GO TO LA004_POP:                               LET01480
                    END:                                               LET01490
                UST_IDX = UST_IDX + 1:                                 LET01500
                GO TO LA006:                                           LET01510
            END:                                                       LET01520
        IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=23) THEN          LET01530
            DO:                                                        LET01540
                ND_FLG = 1:                                            LET01550
                IF ALLOCATION(OPTR)=0 THEN GO TO LA009_FINISH:         LET01560
                GO TO LA004_POP:                                       LET01570
            END:                                                       LET01580
                                                                       LET01590
     /* IS SYMBOL AN OPERATOR ? */                                     LET01600
        IF(US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)>5) THEN          LET01610
            DO:                                                        LET01620
                ERROR(CNT)='INVALID SYNTAX':                           LET01630
                GO TO LA001_ERROR:                                     LET01640
            END:                                                       LET01650
```

```
         PRIORITY=TRM_PRI(UST_PTR(UST_IDX))+PAR_CT;

    /* IS OPERATOR STACK EMPTY ? */
         IF FCN_FLG = 1 THEN
              DO;
                   IF ALLOCATION(FCN_OPTR)=0 THEN
                        GO TO LA008_PUSH_OPTR;
                   GO TO LA010;
              END;
         IF ALLOCATION(OPTR)=0 THEN GO TO LA008_PUSH_OPTR;

    /* IS TOP OF STACK GREATER THAN PRIORITY ? */

LA010:   IF FCN_FLG = 1 THEN
              DO;
                   IF FCN_OPTR_PRIORITY > PRIORITY THEN
                        GO TO LA004_POP;
                   GO TO LA008_PUSH_OPTR;
              END;
         IF OPTR_PRIORITY > PRIORITY THEN
              GO TO LA004_POP;
         GO TO LA008_PUSH_OPTR;

    /* POP STACKS AND WRITE INTO MATRIX */

LA004_POP:
         IF FCN_FLG = 1 THEN
              DO;
                   AREA=TGT_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   LBL_PTR=SUBSTR(AREA, 4, 4);
                   OPR=FCN_OPTR_STACK_TBL;
                   AREA = FCN_OPTR_STACK_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   OPR_PTR = SUBSTR(AREA, 4, 4);
                   FREE FCN_OPTR;
                   OP2=FCN_OPND_STACK_TBL;
                   AREA=FCN_OPND_STACK_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   OP2_PTR = SUBSTR(AREA, 4, 4);
                   FREE FCN_OPND;
                   OP1=FCN_OPND_STACK_TBL;
                   AREA=FCN_OPND_STACK_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   OP1_PTR= SUBSTR(AREA, 4, 4);
                   FREE FCN_OPND;
                   WRITE FILE (MATRIX) FROM (MTX_FIL);
                   GO TO LA011;
              END;
         AREA=TGT_PTR;
         AREA=TRANSLATE(AREA, ZEROO, BLANK);
         LBL_PTR=SUBSTR(AREA, 4, 4);
         OPR=OPTR_STACK_TBL;
         AREA=OPTR_STACK_PTR;
         AREA=TRANSLATE(AREA, ZEROO, BLANK);
```

```
        OPR_PTR=SUBSTR(AREA, 4, 4);
        FREE OPTR;
        OP2=OPND_STACK_TBL;
        AREA=OPND_STACK_PTR;
        AREA=TRANSLATE(AREA, ZEROO, BLANK);
        OP2_PTR=SUBSTR(AREA, 4, 4);
        FREE OPND;
        OP1=OPND_STACK_TBL;
        AREA=OPND_STACK_PTR;
        AREA=TRANSLATE(AREA, ZEROO, BLANK);
        OP1_PTR=SUBSTR(AREA, 4, 4);
        FREE OPND;
        WRITE FILE (MATRIX) FROM (MTX_FIL);

    /* ASSIGN TEMPORARY VALUE AND PUSH ONTO APPROPRIATE STACK*/

 LA011:TMP_STR=TMP_STR + 1;
        IF FCN_FLG = 1 THEN
           DO;
                 ALLOCATE FCN_OPND;
                 FCN_OPND_STACK_TBL = 'TMP';
                 FCN_OPND_STACK_PTR=TMP_STR;
                 GO TO LA012;
           END;
        ALLOCATE OPND;
        OPND_STACK_TBL = 'TMP';
        OPND_STACK_PTR=TMP_STR;

    /* CHECK FOR END OF FUNCTION */

 LA012:IF(FCN_PAR_CT=0)&(FCN_FLG=1) THEN
           DO;
                 IF ALLOCATION(FCN_OPTR)=0 THEN
                    GO TO LA007_POP_FCN;
                 GO TO LA004_POP;
           END;

    /* IS OPERATOR STACK EMPTY AND END FLAG SET ? */

        IF (ALLOCATION(OPTR)=0)&(NO_FLG=1) THEN
           GO TO LA009_FINISH;

    /* CHECK FOR END OF LINE */

        IF(NO_FLG=1) THEN
           GO TO LA004_POP;

    /* CHECK FOR EMPTY OPERATOR STACK */

        IF FCN_FLG = 1 THEN
           DO;
                 IF ALLOCATION(FCN_OPTR)=0 THEN
                    GO TO LA008_PUSH_OPTR;
                 GO TO LA010;
           END;
```

```
         IF ALLOCATION(OPTR)=0 THEN
              GO TO LA008_PUSH_OPTR;
         GO TO LA010;

    /* PUSH OPERATOR ONTO APPROPRIATE STACK */

LA008_PUSH_OPTR:
         IF FCN_FLG = 1 THEN
              DO;
                   ALLOCATE FCN_OPTR;
                   FCN_OPTR_STACK_TBL = US_NTRY(UST_IDX);
                   FCN_OPTR_STACK_PTR = UST_PTR(UST_IDX);
                   FCN_OPTR_PRIORITY = PRIORITY;
                   GO TO LA002;
              END;
         ALLOCATE OPTR;
         OPTR_STACK_TBL = US_NTRY(UST_IDX);
         OPTR_STACK_PTR = UST_PTR(UST_IDX);
         OPTR_PRIORITY = PRIORITY;
         GO TO LA002;

    /* FINISH FUNCTION PROCESSING */

LA007_POP_FCN:
         AREA=TGT_PTR;
         AREA=TRANSLATE(AREA, ZEROO, BLANK);
         LBL_PTR = SUBSTR (AREA, 4, 4);
         OPR='TRM';
         AREA = FCN_PTR;
         AREA=TRANSLATE(AREA, ZEROO, BLANK);
         OPR_PTR=SUBSTR(AREA, 4, 4);
         OP1=FCN_OPND_STACK_TBL;
         AREA=FCN_OPND_STACK_PTR;
         AREA=TRANSLATE(AREA, ZEROO, BLANK);
         OP1_PTR=SUBSTR(AREA, 4, 4);
         FREE FCN_OPND;
         OP2='    ';
         OP2_PTR='0000';
         WRITE FILE (MATRIX) FROM (MTX_FIL);
         FCN_FLG=0;
         TMP_STR=TMP_STR + 1;
         ALLOCATE OPND;
         OPND_STACK_TBL = 'TMP';
         OPND_STACK_PTR = TMP_STR;
         GO TO LA005;

    /* FINISH ASSIGNMENT STATEMENT AND RETURN TO CONTROL */

 LA009_FINISH:

    /* CHECK FOR AGREEMENT OF PARENTHESES */

         IF PAR_CT¬=0 THEN
              DO;
               ERROR(CNT)='UNEQUAL NO. OF LEFT AND RIGHT PAREN.';
```

```
          GO TO LA001_ERROR;                                   LET03310
          END;                                                 LET03320
      AREA=TGT_PTR;                                            LET03330
      AREA=TRANSLATE(AREA, ZEROO, BLANK);                      LET03340
      LBL_PTR=SUBSTR(AREA, 4, 4);                              LET03350
      OPR='TRM';                                               LET03360
      OPR_PTR='0006';                                          LET03370
      OP2=OPND_STACK_TBL;                                      LET03380
      AREA=OPND_STACK_PTR;                                     LET03390
      AREA=TRANSLATE(AREA, ZEROO, BLANK);                      LET03400
      OP2_PTR=SUBSTR(AREA, 4, 4);                              LET03410
      FREE OPND;                                               LET03420
      OP1='IDN';                                               LET03430
      AREA=ASGN;                                               LET03440
      AREA=TRANSLATE(AREA, ZEROO, BLANK);                      LET03450
      OP1_PTR=SUBSTR(AREA, 4, 4);                              LET03460
      WRITE FILE (MATRIX) FROM (MTX_FIL);                      LET03470
      UST_IDX = UST_IDX + 1;                                   LET03480
      CRD_CTR = CRD_CTR + 1;                                   LET03490
      RETURN;                                                  LET03500
                                                               LET03510
   /* ERROR ROUTINE */                                         LET03520
                                                               LET03530
LA001_ERROR:                                                   LET03540
      SRS_LIN(CNT)=CRD_CTR;                                    LET03550
      CNT=CNT+1;                                               LET03560
      DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));  LET03570
          UST_IDX = UST_IDX + 1;                               LET03580
      END;                                                     LET03590
      UST_IDX = UST_IDX + 1;                                   LET03600
      CRD_CTR = CRD_CTR + 1;                                   LET03610
                                                               LET03620
   /* CLEAR STACKS */                                          LET03630
                                                               LET03640
      DO WHILE(ALLOCATION(OPND)¬=0);                           LET03650
          FREE OPND;                                           LET03660
      END;                                                     LET03670
      DO WHILE(ALLOCATION(OPTR)¬=0);                           LET03680
          FREE OPTR;                                           LET03690
      END;                                                     LET03700
      DO WHILE(ALLOCATION(FCN_OPND)¬=0);                       LET03710
          FREE FCN_OPND;                                       LET03720
      END;                                                     LET03730
      DO WHILE(ALLOCATION(FCN_OPTR)¬=0);                       LET03740
          FREE FCN_OPTR;                                       LET03750
      END;                                                     LET03760
      RETURN;                                                  LET03770
                                                               LET03780
END LETAR;                                                     LET03790
```

FILE: RDINAR   PLIOPT   A   YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
 RDINAR : PROCEDURE;

     DCL   1  UST                      (999)          EXT,
              2     US_NTRY            CHAR           (3),
              2     UST_PTR            FIXED          (4);
     DCL   1  ERR_TBL                  (75)           EXT,
              2     ERROR              CHAR           (40),
              2     SRS_LIN            FIXED          (3);
     DCL   1     LIT_TBL               (500)          EXT,
              2     LT_NTRY            CHAR           (9);
     DCL   1     MTX_FIL,
              2     LBL_PTR            CHAR (4),
              2     OPR                CHAR           (3),
              2     OPR_PTR            CHAR         (4),
              2     OP1                CHAR           (3),
              2     OP1_PTR            CHAR         (4),
              2     OP2                CHAR           (3),
              2     OP2_PTR            CHAR         (4),
              2     FILL               CHAR         (55)  INIT((55)' ');
     DCL   UST_IDX                     FIXED        (4)   EXT,
           AREA                        CHAR         (7),
           ZEROO                       CHAR         (1)   INIT('0'),
           BLANK                       CHAR         (1)   INIT(' '),
           CNT                         FIXED        (2)   EXT,
           CRD_CTR                     FIXED        (3)   EXT,
           TGT_PTR                     FIXED        (4)   EXT;
     DCL   MATRIX               FILE    RECORD;

   /* IS NEXT UNIFORM SYMBOL AN IDENTIFIER ? */

        UST_IDX = UST_IDX + 1;
        IF US_NTRY(UST_IDX)¬='IDN' THEN
             DO;
                  ERROR(CNT)='VARIABLE NAME MUST FOLLOW RDIN';
                  SRS_LIN(CNT)=CRD_CTR;
                  CNT=CNT+1;
                  DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                        (UST_PTR(UST_IDX)¬=23));
                        UST_IDX = UST_IDX + 1;
                  END;
                  UST_IDX = UST_IDX + 1;
                  CRD_CTR = CRD_CTR + 1;
                  RETURN;
             END;

   /* WRITE IDENTIFIER INTO MATRIX */

        AREA = TGT_PTR;
        AREA = TRANSLATE (AREA, ZEROO, BLANK);
        LBL_PTR = SUBSTR (AREA, 4, 4);
        OPR = 'KEY';
        OPR_PTR = '0019';
        OP1 = 'IDN';
        AREA = UST_PTR(UST_IDX);
        AREA = TRANSLATE (AREA, ZEROO, BLANK);
        OP1_PTR = SUBSTR (AREA, 4, 4);

    /* IS NEXT UNIFORM SYMBOL 'FROM' ? */

        UST_IDX = UST_IDX + 1;
        IF(US_NTRY(UST_IDX)¬='KEY')|(UST_PTR (UST_IDX)¬=15) THEN
            DO;
                ERROR(CNT)='FROM MUST FOLLOW VARIABLE NAME';
                SRS_LIN(CNT) = CRD_CTR;
                CNT = CNT + 1;
                DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|
                    (UST_PTR(UST_IDX)¬=23));
                    UST_IDX = UST_IDX + 1;
                END;
                UST_IDX = UST_IDX + 1;
                CRD_CTR = CRD_CTR + 1;
                RETURN;
            END;

    /* IS NEXT UNIFORM SYMBOL A HEX ADDRESS ? */

        UST_IDX=UST_IDX+1;
        IF(US_NTRY(UST_IDX)¬='LIT')|
            (INDEX(LT_NTRY(UST_PTR(UST_IDX)),'.')¬=0)|
            (INDEX(LT_NTRY(UST_PTR(UST_IDX)),' ')¬=5) THEN
            DO;
                ERROR(CNT)='HEX ADDRESS MUST FOLLOW FROM';
                SRS_LIN(CNT)=CRD_CTR;
                CNT=CNT+1;
                DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                    (UST_PTR(UST_IDX)¬=23));
                    UST_IDX=UST_IDX+1;
                END;
                UST_IDX=UST_IDX+1;
              CRD_CTR=CRD_CTR+1;
                RETURN;
            END;

    /* WRITE HEX ADDRESS INTO MATRIX */

        OP2='LIT';
        AREA = UST_PTR(UST_IDX);
        AREA = TRANSLATE (AREA, ZEROO, BLANK);
        OP2_PTR = SUBSTR (AREA, 4, 4);
        WRITE FILE (MATRIX) FROM (MTX_FIL);

    /* IS REMAINDER OF SOURCE LINE BLANK ? */

        UST_IDX=UST_IDX+1;
        IF (US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23) THEN
            DO;
                ERROR(CNT)='MULTIPLE ENTRIES FOR HEX ADDRESS';
                SRS_LIN(CNT)=CRD_CTR;
                CNT=CNT+1;
                DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|
                        (UST_PTR(UST_IDX)¬=23)):                                RDI01110
                        UST_IDX=UST_IDX+1;                                      RDI01120
                  END;                                                          RDI01130
                  UST_IDX=UST_IDX+1:                                            RDI01140
                  CRD_CTR=CRD_CTR+1;                                            RDI01150
                  RETURN:                                                       RDI01160
               END:                                                             RDI01170
                                                                                RDI01180
      /*NORMAL RETURN TO CONTROL*/                                              RDI01190
                                 .                                              RDI01200
         UST_IDX=UST_IDX+1:                                                     RDI01210
         CRD_CTR=CRD_CTR+1:                                                     RDI01220
         RETURN:                                                                RDI01230
 END RDINAR:                                                                    RDI01240
```

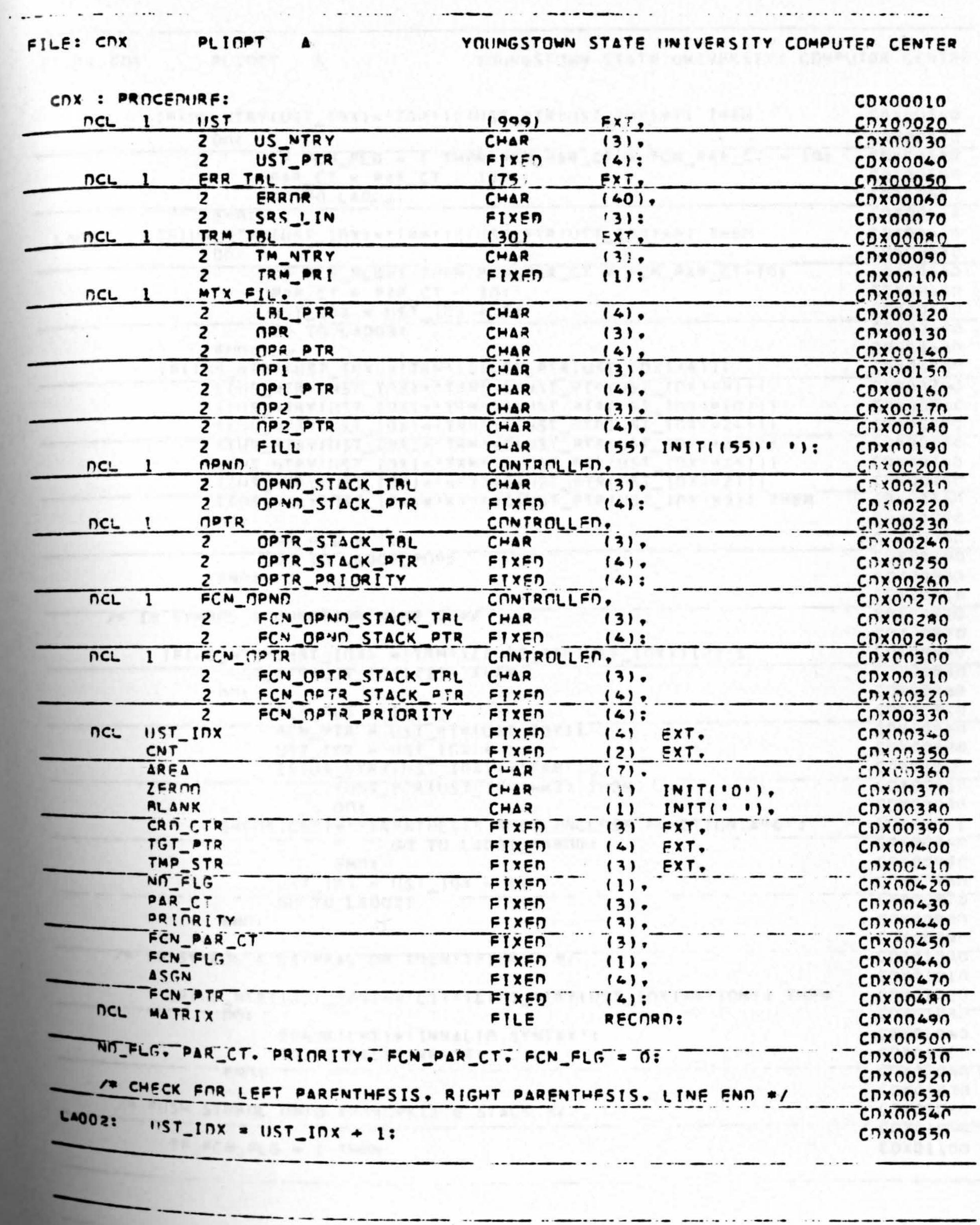
```
FILE: IFAR     PLIOPT   A               YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

  IFAR : PROCEDURE;                                                        IFA00010
       DCL  1    UST                      (999)        EXT,                IFA00020
                 2    US_NTRY             CHAR         (3),                IFA00030
                 2    UST_PTR             FIXED        (4);                IFA00040
       DCL  1    ERR_TBL                  (75)         EXT,                IFA00050
                 2    ERROR               CHAR         (40),               IFA00060
                 2    SRS_LIN             FIXED        (3);                IFA00070
       DCL  1    TRM_TBL                  (30)         EXT,                IFA00080
                 2    TM_NTRY             CHAR         (3),                IFA00090
                 2    TRM_PRI             FIXED        (1);                IFA00100
       DCL  1    LIT_TBL                  (500)        EXT,                IFA00110
                 2    LT_NTRY             CHAR         (9);                IFA00120
       DCL  1    MTX_FIL,                                                  IFA00130
                 2    LBL_PTR             CHAR         (4),                IFA00140
                 2    OPR                 CHAR         (3),                IFA00150
                 2    OPR_PTR             CHAR         (4),                IFA00160
                 2    OP1                 CHAR         (3),                IFA00170
                 2    OP1_PTR             CHAR         (4),                IFA00180
                 2    OP2                 CHAR         (3),                IFA00190
                 2    OP2_PTR             CHAR         (4),                IFA00200
                 2    FILL                CHAR         (55)  INIT((55)' '); IFA00210
       DCL  UST_IDX                       FIXED        (4)   EXT,          IFA00220
            ORIGINAL_IDX                  FIXED        (4),                IFA00230
            CNT                           FIXED        (2)   EXT,          IFA00240
            AREA                          CHAR         (7),                IFA00250
            F_LIN                         FIXED        (6),                IFA00260
            ZEROO                         CHAR         (1)   INIT('0'),    IFA00270
            BLANK                         CHAR         (1)   INIT(' '),    IFA00280
            CRD_CTR                       FIXED        (3)   EXT,          IFA00290
            TGT_PTR                       FIXED        (4)   EXT,          IFA00300
            TMP_STR                       FIXED        (3)   EXT,          IFA00310
            NXT_LIN                       CHAR         (9),                IFA00320
            N                             FIXED        (4),                IFA00330
            CXL                           FIXED        (4),                IFA00340
            CXR                           FIXED        (4),                IFA00350
            RELOP                         FIXED        (4);                IFA00360
       DCL  CDX                           ENTRY,                           IFA00370
            LETAR                         ENTRY,                           IFA00380
            RETRNAR                       ENTRY,                           IFA00390
            READAR                        ENTRY,                           IFA00400
            RESTRAR                       ENTRY,                           IFA00410
            STOPAR                        ENTRY,                           IFA00420
            ENDAR                         ENTRY,                           IFA00430
            WTOUTAR                       ENTRY,                           IFA00440
            RDINAR                        ENTRY,                           IFA00450
            GOTOAR                        ENTRY,                           IFA00460
            GOSUBAR                       ENTRY;                           IFA00470
       DCL  MATRIX                        FILE         RECORD;             IFA00480
                                                                           IFA00490
            TMP_STR=0;                                                     IFA00500
                                                                           IFA00510
     /* PROCESS LEFT HAND SIDE OF CONDITION */                             IFA00520
                                                                           IFA00530
            CALL CDX;                                                      IFA00540
            CXL=TMP_STR;                                                   IFA00550
```

```

   /* CHECK FOR RELATIONAL OPERATOR AFTER FIRST ARGUMENT */

         IF UST_PTR(UST_IDX) = 6  THEN DO;
              RELOP = 27;
              GO TO IF000;
              END;
         IF UST_PTR(UST_IDX) = 9  THEN DO;
              RELOP = 9;
              GO TO IF000;
              END;
         IF UST_PTR(UST_IDX) = 10 THEN DO;
              RELOP = 10;
              GO TO IF000;
              END;
         IF UST_PTR(UST_IDX) = 24 THEN DO;
              RELOP = 24;
              GO TO IF000;
              END;
         IF UST_PTR(UST_IDX)= 25 THEN DO;
              RELOP = 25;
              GO TO IF000;
              END;
         IF UST_PTR(UST_IDX) = 26 THEN DO;
              RELOP = 26;
              GO TO IF000;
              END;
         ERROR(CNT)='REL. OP. MUST FOLLOW FIRST CONDITION';
         GO TO IF003;

IF000:  /* PROCESS RIGHT HAND SIDE OF CONDITION */

         CALL CDX;
         CXR=TMP_STR;

    /* WRITE MATRIX LINE */

         AREA=TGT_PTR;
         AREA = TRANSLATE(AREA, ZEROO, BLANK);
         LBL_PTR=SUBSTR(AREA, 4, 4);
         OPR='TRM';
         AREA=RELOP;
         AREA = TRANSLATE(AREA, ZEROO, BLANK);
         OPR_PTR = SUBSTR(AREA, 4 , 4);
         OP1 ='TMP';
         AREA=CXL;
         AREA = TRANSLATE(AREA, ZEROO, BLANK);
         OP1_PTR=SUBSTR(AREA, 4, 4);
         OP2='TMP';
         AREA=CXR;
         AREA = TRANSLATE(AREA, ZEROO, BLANK);
         OP2_PTR=SUBSTR(AREA, 4, 4);
         WRITE FILE (MATRIX) FROM (MTX_FIL);

    /* BRANCH TO ACTION 2 */
```

```

        OPR='KEY';
        OPR_PTR='0002';
        OP1='LIT';

        ORIGINAL_IDX = UST_IDX;
        DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
              UST_IDX = UST_IDX + 1;
        END;

        AREA = UST_PTR(UST_IDX+1);
        AREA = TRANSLATE(AREA, ZEROO, BLANK);
        OP1_PTR = SUBSTR(AREA, 4, 4);

        OP2 = ' ';
        OP2_PTR = '0000';
        WRITE FILE (MATRIX) FROM (MTX_FIL);

        UST_IDX = ORIGINAL_IDX;

  /* IS NEXT SYMBOL 'THEN' ? */

        IF(US_NTRY(UST_IDX)¬='KEY')|(UST_PTR(UST_IDX)¬=5) THEN
              GO TO IF001;

  /* IS NEXT SYMBOL A KEYWORD ? */

        UST_IDX = UST_IDX + 1;
        IF US_NTRY(UST_IDX)¬='KEY' THEN GO TO IF002;

  /* CALL ACTION ROUTINE */

        IF (UST_PTR(UST_IDX)=3)|
           (UST_PTR(UST_IDX)=4)|
           (UST_PTR(UST_IDX)=5)|
           (UST_PTR(UST_IDX)=6)|
           (UST_PTR(UST_IDX)=7)|
           (UST_PTR(UST_IDX)=10)|
           (UST_PTR(UST_IDX)=11)|
           (UST_PTR(UST_IDX)=12)|
           (UST_PTR(UST_IDX)=15)
           THEN DO;
              ERROR(CNT)='INVALID ACTION SPECIFIED';
              GO TO IF003;
           END;
        IF UST_PTR(UST_IDX)=1 THEN DO;
              CALL LETAR;
              GO TO IF004;
              END;
        IF UST_PTR(UST_IDX)=2 THEN DO;
              CALL GOTOAR;
              GO TO IF004;
              END;
        IF UST_PTR(UST_IDX)=8 THEN DO;
              CALL GOSUBAR;
```

```
             GO TO IF004;
             END;
        IF UST_PTR(UST_IDX)=9 THEN DO;
             CALL RETRNAR;
             GO TO IF004;
             END;
        IF UST_PTR(UST_IDX)=13 THEN DO;
             CALL READAR;
             GO TO IF004;
             END;
        IF UST_PTR(UST_IDX)=14 THEN DO;
             CALL RESTRAR;
             GO TO IF004;
             END;
        IF UST_PTR(UST_IDX)=16 THEN DO;
             CALL STOPAR;
             GO TO IF004;
             END;
        IF UST_PTR(UST_IDX)=17 THEN DO;
             CALL ENDAR;
             GO TO IF004;
             END;
        IF UST_PTR(UST_IDX)=18 THEN DO;
             CALL WTOUTAR;
             GO TO IF004;
             END;
        IF UST_PTR(UST_IDX)=19 THEN DO;
             CALL RDINAR;
             GO TO IF004;
             END;

IF001:  /* IS SYMBOL A GOTO ? */

        IF(US_NTRY(UST_IDX)='KEY')&(UST_PTR(UST_IDX)=2) THEN
             DO;
                  CALL GOTOAR;
                  GO TO IF004;
             END;
        ERROR(CNT)='THEN OR GOTO MUST FOLLOW CONDITION';
        GO TO IF003;

IF002:  /* IS SYMBOL A LINE NUMBER ? */

        IF(US_NTRY(UST_IDX)¬='LIT')|
             (VERIFY(LT_NTRY(UST_PTR(UST_IDX)),'0123456789')>7)|
             (VERIFY(LT_NTRY(UST_PTR(UST_IDX)),' 0123456789')¬=0)
             THEN DO;
                  ERROR(CNT)='INVALID COMMAND FOLLOWS CONDITION';
                  GO TO IF003;
             END;
        UST_IDX=UST_IDX-1;
        CALL GOTOAR;
        GO TO IF004;

IF003:  /* ERROR ROUTINE */
```

```
FILE: IFAR     PLIOPT   A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

                                                                       IFA02210
         SRS_LIN(CNT)=CRD_CTR;                                         IFA02220
         CNT=CNT + 1;                                                  IFA02230
         DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));  IFAC2240
            UST_IDX=UST_IDX + 1;                                       IFA02250
         END;                                                          IFA02260
         UST_IDX = UST_IDX + 1;                                        IFA02270
         CRD_CTR = CRD_CTR + 1;                                        IFA02280
         RETURN;                                                       IFA02290
                                                                       IFA02300
IF004:   /* NORMAL RETURN TO CONTROL */                                IFA02310
   /* UST_IDX & CRD_CTR HAVE BEEN UPDATED IN ACTION ROUTINES */        IFA02320
                                                                       IFA02330
         RETURN;                                                       IFA02340
                                                                       IFA02350
END IFAR;                                                              IFA02360
```



```
FILE: CDX      PLIOPT   A             YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

  CDX : PROCEDURE;
     DCL  1    UST                      (999)       EXT,
               2  US_NTRY               CHAR        (3),
               2     UST_PTR            FIXED       (4);
     DCL  1    ERR_TBL                  (75)        EXT,
               2     ERROR              CHAR        (40),
               2  SRS_LIN               FIXED       (3);
     DCL  1    TRM_TBL                  (30)        EXT,
               2     TM_NTRY            CHAR        (3),
               2     TRM_PRI            FIXED       (1);
     DCL  1    MTX_FIL,
               2     LBL_PTR            CHAR        (4),
               2     OPR                CHAR        (3),
               2     OPR_PTR            CHAR        (4),
               2     OP1                CHAR        (3),
               2     OP1_PTR            CHAR        (4),
               2     OP2                CHAR        (3),
               2     OP2_PTR            CHAR        (4),
               2     FILL               CHAR        (55)  INIT((55)' ');
     DCL  1    OPND                     CONTROLLED,
               2     OPND_STACK_TBL     CHAR        (3),
               2     OPND_STACK_PTR     FIXED       (4);
     DCL  1    OPTR                     CONTROLLED,
               2     OPTR_STACK_TBL     CHAR        (3),
               2     OPTR_STACK_PTR     FIXED       (4),
               2     OPTR_PRIORITY      FIXED       (4);
     DCL  1    FCN_OPND                 CONTROLLED,
               2     FCN_OPND_STACK_TBL CHAR        (3),
               2     FCN_OPND_STACK_PTR FIXED       (4);
     DCL  1    FCN_OPTR                 CONTROLLED,
               2     FCN_OPTR_STACK_TBL CHAR        (3),
               2     FCN_OPTR_STACK_PTR FIXED       (4),
               2     FCN_OPTR_PRIORITY  FIXED       (4);
     DCL  UST_IDX                       FIXED       (4)   EXT,
          CNT                           FIXED       (2)   EXT,
          AREA                          CHAR        (7),
          ZEROO                         CHAR        (1)   INIT('0'),
          BLANK                         CHAR        (1)   INIT(' '),
          CRD_CTR                       FIXED       (3)   EXT,
          TGT_PTR                       FIXED       (4)   EXT,
          TMP_STR                       FIXED       (3)   EXT,
          NO_FLG                        FIXED       (1),
          PAR_CT                        FIXED       (3),
          PRIORITY                      FIXED       (3),
          FCN_PAR_CT                    FIXED       (3),
          FCN_FLG                       FIXED       (1),
          ASGN                          FIXED       (4),
          FCN_PTR                       FIXED       (4);
      DCL MATRIX                        FILE        RECORD;

      NO_FLG, PAR_CT, PRIORITY, FCN_PAR_CT, FCN_FLG = 0;

      /* CHECK FOR LEFT PARENTHESIS, RIGHT PARENTHESIS, LINE END */
LA002:  UST_IDX = UST_IDX + 1;
```

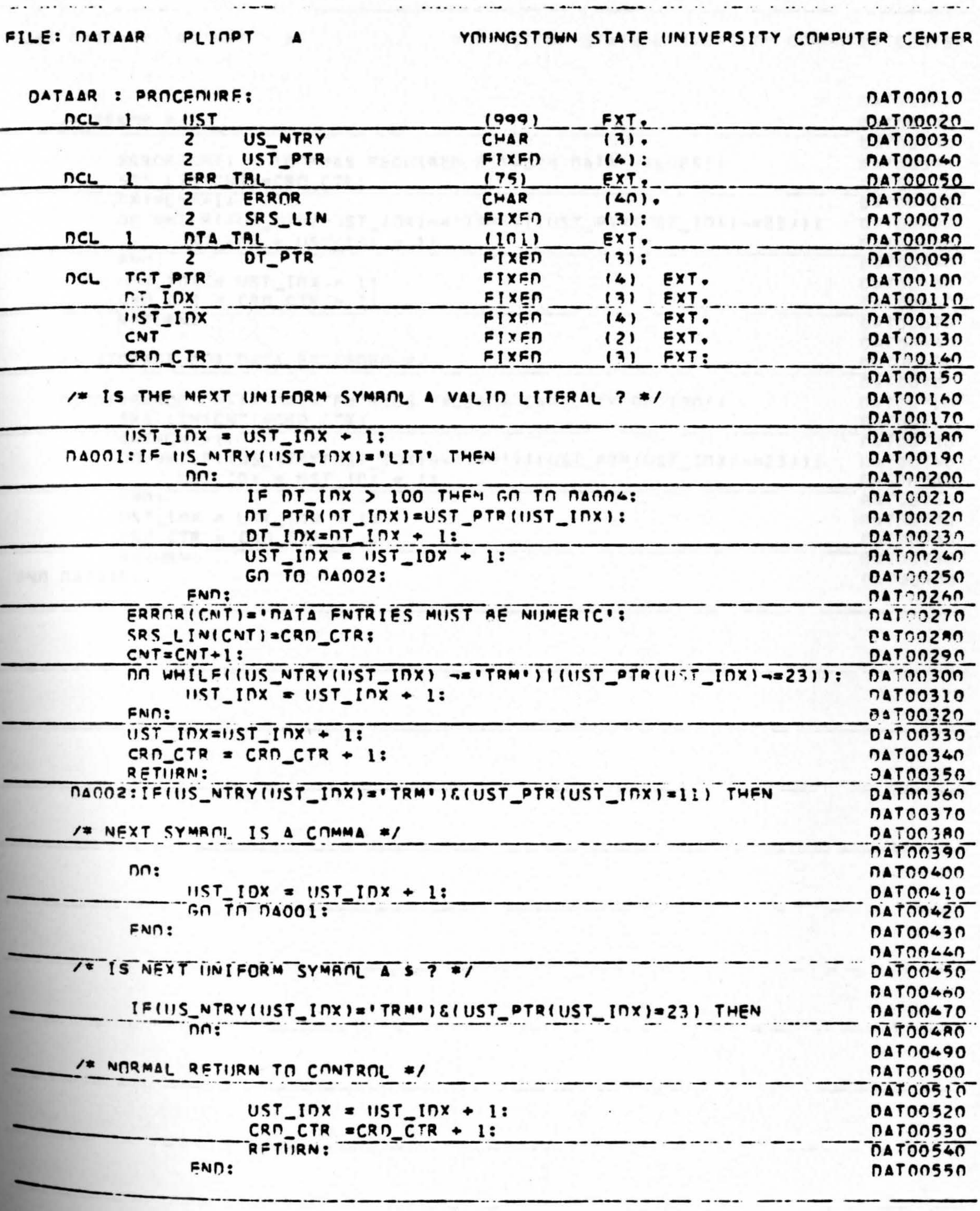
```
        IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=7) THEN
            DO:
                IF FCN_FLG = 1 THEN FCN_PAR_CT = FCN_PAR_CT + 10:
                PAR_CT = PAR_CT + 10:
                GO TO LA002:
            END:
 LA003: IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=8) THEN
            DO:
                IF FCN_FLG=1 THEN FCN_PAR_CT = FCN_PAR_CT-10:
                PAR_CT = PAR_CT - 10:
                UST_IDX = UST_IDX + 1:
                GO TO LA003:
            END:
        IF((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=6))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=9))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=10))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=24))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=25))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=26))|
            ((US_NTRY(UST_IDX)='KEY')&(UST_PTR(UST_IDX)=2))|
            ((US_NTRY(UST_IDX)='KEY')&(UST_PTR(UST_IDX)=5)) THEN
            DO:
                NO_FLG = 1:
                GO TO LA004_POP:
            END:

    /* IS SYMBOL A FUNCTION NAME ? */

        IF(US_NTRY(UST_IDX) ='TRM')&(UST_PTR(UST_IDX)>12) &
            (UST_PTR(UST_IDX)<23) THEN
            DO:
                FCN_FLG = 1:
                FCN_PTR = UST_PTR(UST_IDX):
                UST_IDX = UST_IDX + 1:
                IF(US_NTRY(UST_IDX)¬='TRM')|
                    (UST_PTR(UST_IDX)¬=7) THEN
                    DO:
            ERROR(CNT)='PARENTHESIS MUST ENCLOSE FUNCTION ARG':
                        GO TO LA001_ERROR:
                    END:
                UST_IDX = UST_IDX - 1:
                GO TO LA002:
            END:

    /* IS SYMBOL A LITERAL OR IDENTIFIER ? */

        IF(US_NTRY(UST_IDX)¬='LIT')&(US_NTRY(UST_IDX)¬='IDN') THEN
            DO:
                ERROR(CNT)='INVALID SYNTAX':
                GO TO LA001_ERROR:
            END:

    /* PUSH SYMBOL ONTO APPROPRIATE STACK */

        IF FCN_FLG = 1 THEN
```

```
            DO;
                    ALLOCATE FCN_OPND;
                    FCN_OPND_STACK_TBL = US_NTRY(UST_IDX);
                    FCN_OPND_STACK_PTR = UST_PTR(UST_IDX);
                    GO TO LA005;
            END;
        ALLOCATE OPND;
        OPND_STACK_TBL = US_NTRY(UST_IDX);
        OPND_STACK_PTR = UST_PTR(UST_IDX);

    /* CHECK FOR LEFT PARENTHESIS, RIGHT PARENTHESIS, LINE_END */

LA005:  UST_IDX = UST_IDX + 1;
        IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=7) THEN
            DO;
                    IF FCN_FLG = 1 THEN FCN_PAR_CT = FCN_PAR_CT+10;
                    PAR_CT = PAR_CT + 10;
                    GO TO LA005;
            END;
LA006:  IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=8) THEN
            DO;
                    IF FCN_FLG = 1 THEN FCN_PAR_CT = FCN_PAR_CT-10;
                    PAR_CT = PAR_CT -10;
                    IF(FCN_PAR_CT=0)&(FCN_FLG=1) THEN
                        DO;
                                IF(ALLOCATION(FCN_OPTR)=0) THEN
                                    GO TO LA007_POP_FCN;
                            GO TO LA004_POP;
                        END;
                    UST_IDX = UST_IDX + 1;
                    GO TO LA006;
            END;
        IF((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=6))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=9))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=10))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=24))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=25))|
            ((US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=26))|
            ((US_NTRY(UST_IDX)='KEY')&(UST_PTR(UST_IDX)=2))|
            ((US_NTRY(UST_IDX)='KEY')&(UST_PTR(UST_IDX)=5)) THEN
            DO;
                    ND_FLG = 1;
                    IF ALLOCATION(OPTR)=0 THEN GO TO LA009_FINISH;
                    GO TO LA004_POP;
            END;

    /* IS SYMBOL AN OPERATOR ? */
        IF(US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)>5) THEN
            DO;
                    ERROR(CNT)='INVALID SYNTAX';
                    GO TO LA001_ERROR;
            END;
        PRIORITY=TRM_PRI(UST_PTR(UST_IDX))+PAR_CT;

    /* IS OPERATOR STACK EMPTY ? */
```

```
         IF FCN_FLG = 1 THEN
              DO;
                   IF ALLOCATION(FCN_OPTR)=0 THEN
                        GO TO LA008_PUSH_OPTR;
                   GO TO LA010;
              END;
         IF ALLOCATION(OPTR)=0 THEN GO TO LA008_PUSH_OPTR;

    /* IS TOP OF STACK GREATER THAN PRIORITY ? */

LA010:   IF FCN_FLG = 1 THEN
              DO;
                   IF FCN_OPTR_PRIORITY > PRIORITY THEN
                        GO TO LA004_POP;
                   GO TO LA008_PUSH_OPTR;
              END;
         IF OPTR_PRIORITY > PRIORITY THEN
              GO TO LA004_POP;
         GO TO LA008_PUSH_OPTR;

    /* POP STACKS AND WRITE INTO MATRIX */

LA004_POP:
         IF FCN_FLG = 1 THEN
              DO;
                   AREA=TGT_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   LBL_PTR=SUBSTR(AREA, 4, 4);
                   OPR=FCN_OPTR_STACK_TBL;
                   AREA = FCN_OPTR_STACK_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   OPR_PTR = SUBSTR(AREA, 4, 4);
                   FREE FCN_OPTR;
                   OP2=FCN_OPND_STACK_TBL;
                   AREA=FCN_OPND_STACK_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   OP2_PTR = SUBSTR(AREA, 4, 4);
                   FREE FCN_OPND;
                   OP1=FCN_OPND_STACK_TBL;
                   AREA=FCN_OPND_STACK_PTR;
                   AREA=TRANSLATE(AREA, ZEROO, BLANK);
                   OP1_PTR= SUBSTR(AREA, 4, 4);
                   FREE FCN_OPND;
                   WRITE FILE (MATRIX) FROM (MTX_FIL);
                   GO TO LA011;
              END;
         AREA=TGT_PTR;
         AREA=TRANSLATE(AREA, ZEROO, BLANK);
         LBL_PTR=SUBSTR(AREA, 4, 4);
         OPR=OPTR_STACK_TBL;
         AREA=OPTR_STACK_PTR;
         AREA=TRANSLATE(AREA, ZEROO, BLANK);
         OPR_PTR=SUBSTR(AREA, 4, 4);
         FREE OPTR;
         OP2=OPND_STACK_TBL;
```

```
       AREA=OPND_STACK_PTR;
       AREA=TRANSLATE(AREA, ZEROO, BLANK);
       OP2_PTR=SUBSTR(AREA, 4, 4);
       FREE OPND;
       OP1=OPND_STACK_TBL;
       AREA=OPND_STACK_PTR;
       AREA=TRANSLATE(AREA, ZEROO, BLANK);
       OP1_PTR=SUBSTR(AREA, 4, 4);
       FREE OPND;
       WRITE FILE (MATRIX) FROM (MTX_FIL);

    /* ASSIGN TEMPORARY VALUE AND PUSH ONTO APPROPRIATE STACK*/

LA011:TMP_STR=TMP_STR + 1;
       IF FCN_FLG = 1 THEN
            DO;
                 ALLOCATE FCN_OPND;
                 FCN_OPND_STACK_TBL = 'TMP';
                 FCN_OPND_STACK_PTR=TMP_STR;
                 GO TO LA012;
            END;
       ALLOCATE OPND;
       OPND_STACK_TBL = 'TMP';
       OPND_STACK_PTR=TMP_STR;

    /* CHECK FOR END OF FUNCTION */

LA012:IF(FCN_PAR_CT=0)&(FCN_FLG=1) THEN
            DO;
                 IF ALLOCATION(FCN_OPTR)=0 THEN
                      GO TO LA007_POP_FCN;
                 GO TO LA004_POP;
            END;

    /* IS OPERATOR STACK EMPTY AND END FLAG SET ? */

       IF (ALLOCATION(OPTR)=0)&(NO_FLG=1) THEN
            GO TO LA009_FINISH;

    /* CHECK FOR END OF LINE */

       IF(NO_FLG=1) THEN
            GO TO LA004_POP;

    /* CHECK FOR EMPTY OPERATOR STACK */

       IF FCN_FLG = 1 THEN
            DO;  IF ALLOCATION(FCN_OPTR)=0 THEN
                      GO TO LA008_PUSH_OPTR;
                 GO TO LA010;
            END;
       IF ALLOCATION(OPTR)=0 THEN
            GO TO LA008_PUSH_OPTR;
       GO TO LA010;
```

```

   /* PUSH OPERATOR ONTO APPROPRIATE STACK */

 LA008_PUSH_OPTR:
       IF FCN_FLG = 1 THEN
          DO:
             ALLOCATE FCN_OPTR:
             FCN_OPTR_STACK_TBL = US_NTRY(UST_IDX):
             FCN_OPTR_STACK_PTR = UST_PTR(UST_IDX):
             FCN_OPTR_PRIORITY = PRIORITY:
             GO TO LA002:
          END:
       ALLOC TE OPTR:
       OPTR_STACK_TBL = US_NTRY(UST_IDX):
       OPTR_STACK_PTR = UST_PTR(UST_IDX):
       OPTR_PRIORITY = PRIORITY;
       GO TO LA002;

   /* FINISH FUNCTION PROCESSING */

 LA007_POP_FCN:
       AREA=TGT_PTR:
       AREA=TRANSLATE(AREA, ZEROO, BLANK):
       LBL_PTR = SUBSTR (AREA, 4, 4):
       OPR='TRM':
       AREA = FCN_PTR:
       AREA=TRANSLATE(AREA, ZEROO, BLANK):
       OPR_PTR=SUBSTR(AREA, 4, 4):
       OP1=FCN_OPND_STACK_TBL:
       AREA=FCN_OPND_STACK_PTR:
       AREA=TRANSLATE(AREA, ZEROO, BLANK):
       OP1_PTR=SUBSTR(AREA, 4, 4);
       FREE FCN_OPND:
       OP2='   ':
       OP2_PTR='0000':
       WRITE FILE (MATRIX) FROM (MTX_FIL):
       FCN_FLG=0:
       TMP_STR=TMP_STR + 1:
       ALLOCATE OPND:
       OPND_STACK_TBL = 'TMP':
       OPND_STACK_PTR = TMP_STR:
       GO TO LA005:

   /* FINISH ASSIGNMENT STATEMENT AND RETURN TO CONTROL */

 LA009_FINISH:

   /* CHECK FOR AGREEMENT OF PARENTHESES */

       IF PAR_CT¬=0 THEN
          DO:
             ERROR(CNT)='UNEQUAL NO. OF LEFT AND RIGHT PAREN.':
             GO TO LA001_ERROR:
          END:
       AREA=TGT_PTR:
          AREA=TRANSLATE(AREA, ZERO, BLANK);
          LBL_PTR=SUBSTR(AREA, 4, 4);
          OPR='TRM';
          OPR_PTR='0006';
          OP2=OPND_STACK_TBL;
          AREA=OPND_STACK_PTR;
          AREA=TRANSLATE(AREA, ZERO, BLANK);
          OP2_PTR=SUBSTR(AREA, 4, 4);
          FREE OPND;
          OP1='TMP';
          TMP_STR=TMP_STR + 1;
          AREA=TMP_STR;
          AREA=TRANSLATE(AREA, ZERO, BLANK);
          OP1_PTR=SUBSTR(AREA, 3, 4);
          WRITE FILE (MATRIX) FROM (MTX_FIL);
          RETURN;

     /* ERROR ROUTINE */

  LA001_ERROR:
          SRS_LIN(CNT)=CRD_CTR;
          CNT=CNT+1;
          DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
               UST_IDX = UST_IDX + 1;
          END;
          UST_IDX = UST_IDX + 1;
          CRD_CTR = CRD_CTR + 1;

     /* CLEAR STACKS */

          DO WHILE(ALLOCATION(OPND)¬=0);
               FREE OPND;
          END;
          DO WHILE(ALLOCATION(OPTR)¬=0);
               FREE OPTR;
          END;
          DO WHILE(ALLOCATION(FCN_OPND)¬=0);
               FREE FCN_OPND;
          END;
          DO WHILE(ALLOCATION(FCN_OPTR)¬=0);
               FREE FCN_OPTR;
          END;
          RETURN;

  END CDX;
```



```
FILE: DATAAR    PLIOPT    A             YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 DATAAR : PROCEDURE;
    DCL  1    UST                         (999)     EXT,
              2    US_NTRY                CHAR      (3),
              2    UST_PTR                FIXED     (4);
    DCL  1    ERR_TBL                     (75)      EXT,
              2    ERROR                  CHAR      (40),
              2    SRS_LIN                FIXED     (3);
    DCL  1    DTA_TBL                     (101)     EXT,
              2    DT_PTR                 FIXED     (3);
    DCL  TGT_PTR                          FIXED     (4)  EXT,
         DT_IDX                           FIXED     (3)  EXT,
         UST_IDX                          FIXED     (4)  EXT,
         CNT                              FIXED     (2)  EXT,
         CRD_CTR                          FIXED     (3)  EXT;

    /* IS THE NEXT UNIFORM SYMBOL A VALID LITERAL ? */

         UST_IDX = UST_IDX + 1;
    DA001:IF US_NTRY(UST_IDX)='LIT' THEN
              DO;
                   IF DT_IDX > 100 THEN GO TO DA004;
                   DT_PTR(DT_IDX)=UST_PTR(UST_IDX);
                   DT_IDX=DT_IDX + 1;
                   UST_IDX = UST_IDX + 1;
                   GO TO DA002;
              END;
         ERROR(CNT)='DATA ENTRIES MUST BE NUMERIC';
         SRS_LIN(CNT)=CRD_CTR;
         CNT=CNT+1;
         DO WHILE((US_NTRY(UST_IDX) ¬='TRM')|(UST_PTR(UST_IDX)¬=23));
              UST_IDX = UST_IDX + 1;
         END;
         UST_IDX=UST_IDX + 1;
         CRD_CTR = CRD_CTR + 1;
         RETURN;
    DA002:IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=11) THEN

    /* NEXT SYMBOL IS A COMMA */

         DO;
              UST_IDX = UST_IDX + 1;
              GO TO DA001;
         END;

    /* IS NEXT UNIFORM SYMBOL A $ ? */

         IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=23) THEN
              DO;

    /* NORMAL RETURN TO CONTROL */

                   UST_IDX = UST_IDX + 1;
                   CRD_CTR =CRD_CTR + 1;
                   RETURN;
              END;
```

```
   /* ERROR */

         ERROR(CNT) = 'COMMAS REQUIRED BETWEEN DATA VALUES';
         SRS_LIN(CNT)=CRD_CTR;
         CNT=CNT+1;
         DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
              UST_IDX = UST_IDX + 1;
         END;
         UST_IDX = UST_IDX + 1;
         CRD_CTR = CRD_CTR + 1;
         RETURN;

   /* STORAGE FOR DATA EXCEEDED */

   DA004:ERROR(CNT)='DATA ENTRIES EXCEEDS CAPACITY OF 100';
         SRS_LIN(CNT)=CRD_CTR;
         CNT=CNT+1;
         DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
              UST_IDX = UST_IDX + 1;
         END;
         UST_IDX = UST_IDX + 1;
         CRD_CTR = CRD_CTR + 1;
         RETURN;
 END DATAAR;
```

FILE: RETRNAR   PLIOPT   A   YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
 RETRNAR : PROCEDURE;
    DCL  1     UST                            (999)      EXT,
                 2     US_NTRY                CHAR       (3),
                 2     UST_PTR                FIXED      (4);
    DCL  1     ERR_TBL                        (75)       EXT,
                 2     ERROR                  CHAR       (40),
                 2     SRS_LIN                FIXED      (3);
    DCL  1     MTX_FIL,
                 2     LBL_PTR                CHAR       (4),
                 2     OPR                    CHAR       (3),
                 2     OPR_PTR                CHAR       (4),
                 2     OP1                    CHAR       (3),
                 2     OP1_PTR                CHAR       (4),
                 2     OP2                    CHAR       (3),
                 2     OP2_PTR                CHAR       (4),
                 2     FILL                   CHAR       (55) INIT((55)' ');
    DCL  UST_IDX                              FIXED      (4)  EXT,
         CNT                                  FIXED      (2)  EXT,
         CRD_CTR                              FIXED      (3)  EXT,
         AREA                                 CHAR       (7),
         ZERO0                                CHAR       (1)  INIT('0'),
         BLANK                                CHAR       (1)  INIT(' '),
         TGT_PTR                              FIXED      (4)  EXT;
    DCL  MATRIX                               FILE       RECORD;

    /* THE NEXT UNIFORM SYMBOL MUST BE '$' */

         UST_IDX = UST_IDX + 1;
         IF(US_NTRY(UST_IDX) ¬= 'TRM')|(UST_PTR(UST_IDX)¬=23) THEN
              GO TO RT001;
         AREA = TGT_PTR;
         AREA = TRANSLATE (AREA, ZERO0, BLANK);
         LBL_PTR = SUBSTR (AREA, 4, 4);
         OPR = 'KEY';
         OPR_PTR = '0009';
         OP1 = '   ';
         OP1_PTR = '0000';
         OP2 = '   ';
         OP2_PTR = '0000';
         WRITE FILE (MATRIX) FROM (MTX_FIL);
         UST_IDX = UST_IDX + 1;
         CRD_CTR = CRD_CTR + 1;
         RETURN;
   RT001:ERROR(CNT) = 'CHARACTERS APPEAR AFTER RETURN STMT';
         SRS_LIN(CNT) = CRD_CTR;
         CNT = CNT + 1;
         DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
              UST_IDX = UST_IDX + 1;
         END;
         UST_IDX = UST_IDX + 1;
         CRD_CTR = CRD_CTR + 1;
         RETURN;
 END RETRNAR;
```

```
FILE: STOPAR   PLIOPT   A                    YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

  STOPAR : PROCEDURE;
     DCL  1     UST                         (999)      EXT,
                2     US_NTRY               CHAR       (3),
                2     UST_PTR               FIXED      (4);
     DCL  1     ERR_TBL                     (75)       EXT,
                2     ERROR                 CHAR       (40),
                2     SRS_LIN               FIXED      (3);
     DCL  1     MTX_FIL,
                2     LBL_PTR               CHAR       (4),
                2     OPR                   CHAR       (3),
                2     OPR_PTR               CHAR       (4),
                2     OP1                   CHAR       (3),
                2     OP1_PTR               CHAR       (4),
                2     OP2                   CHAR       (3),
                2     OP2_PTR               CHAR       (4),
                2     FILL                  CHAR       (55) INIT((55)' '):
     DCL  UST_IDX                           FIXED      (4)  EXT,
          CNT                               FIXED      (2)  EXT,
          AREA                              CHAR       (7), INIT((55)' '));
          ZERO0                             CHAR       (1)  INIT('0'),
          BLANK                             CHAR       (1)  INIT(' '),
          CRD_CTR                           FIXED      (3)  EXT,
          TGT_PTR                           FIXED      (4)  EXT;
     DCL  MATRIX                            FILE       RECORD;

     /* THE NEXT UNIFORM SYMBOL MUST BE 'S' */

          UST_IDX = UST_IDX + 1;
          IF(US_NTRY(UST_IDX) ¬= 'TRM')|(UST_PTR(UST_IDX)¬=23) THEN
             GO TO SA001;
          AREA = TGT_PTR;
          AREA = TRANSLATE (AREA, ZERO0, BLANK);
          LBL_PTR = SUBSTR (AREA, 4, 4);
          OPR = 'KEY';
          OPR_PTR = '0016';
          OP1 = '   ';
          OP1_PTR = '0000';
          OP2 = '   ';
          OP2_PTR = '0000';
          WRITE FILE (MATRIX) FROM (MTX_FIL);
          UST_IDX = UST_IDX + 1;
          CRD_CTR = CRD_CTR + 1;
          RETURN;
    SA001:ERROR(CNT) = 'CHARACTERS APPEAR AFTER STOP STMT';
          SRS_LIN(CNT) = CRD_CTR;
          CNT = CNT + 1;
          DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
             UST_IDX = UST_IDX + 1;
          END;
          UST_IDX = UST_IDX + 1;
          CRD_CTR = CRD_CTR + 1;
          RETURN;
  END STOPAR;
```

```
 WTOUTAR : PROCEDURE;

     DCL  1    UST                      (999)      EXT,
               2    US_NTRY             CHAR       (3),
               2    UST_PTR             FIXED      (4);
     DCL  1    ERR_TBL                  (75)       EXT,
               2    ERROR               CHAR       (40),
               2    SRS_LIN             FIXED      (3);
     DCL  1    LIT_TBL                  (500)      EXT,
               2    LT_NTRY             CHAR       (9);
     DCL  1    MTX_FIL,
               2    LBL_PTR             CHAR  (4),
               2    OPR                 CHAR       (3),
               2    OPR_PTR             CHAR      (4),
               2    OP1                 CHAR       (3),
               2    OP1_PTR             CHAR      (4),
               2    OP2                 CHAR       (3),
               2    OP2_PTR             CHAR      (4),
               2    FILL                CHAR       (55) INIT((55)' ');
     DCL  UST_IDX                       FIXED      (4)  EXT,
          AREA                          CHAR       (7),
          ZEROO                         CHAR       (1)  INIT('0'),
          BLANK                         CHAR       (1)  INIT(' '),
          CNT                           FIXED      (2)  EXT,
          CRD_CTR                       FIXED      (3)  EXT,
          TGT_PTR                       FIXED      (4)  EXT;
     DCL  MATRIX                        FILE   RECORD;

  /* IS NEXT UNIFORM SYMBOL AN IDENTIFIER ? */

       UST_IDX = UST_IDX + 1;
       IF US_NTRY(UST_IDX)¬='IDN' THEN
               DO;
                  ERROR(CNT)='VARIABLE NAME MUST FOLLOW WTOUT';
                  SRS_LIN(CNT)=CRD_CTR;
                  CNT=CNT+1;
                  DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                        (UST_PTR(UST_IDX)¬=23));
                     UST_IDX = UST_IDX + 1;
                  END;
                  UST_IDX = UST_IDX + 1;
                  CRD_CTR = CRD_CTR + 1;
                  RETURN;
               END;
  /* WRITE IDENTIFIER INTO MATRIX */

       AREA = TGT_PTR;
       AREA = TRANSLATE (AREA, ZEROO, BLANK);
       LBL_PTR = SUBSTR (AREA, 4, 4);
       OPR = 'KEY';
       OPR_PTR = '0018';
       OP1 = 'IDN';
       AREA = UST_PTR(UST_IDX);
       AREA = TRANSLATE (AREA, ZEROO, BLANK);
```

```
        OP1_PTR = SUBSTR (AREA, 4, 4);

    /* IS NEXT UNIFORM SYMBOL 'TO' ? */

        UST_IDX = UST_IDX + 1;
        IF(US_NTRY(UST_IDX)¬='KEY')|(UST_PTR (UST_IDX)¬=4) THEN
            DO;
                ERROR(CNT)='TO MUST FOLLOW VARIABLE NAME';
                SRS_LIN(CNT) = CRD_CTR;
                CNT = CNT + 1;
                DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|
                    (UST_PTR(UST_IDX)¬=23));
                    UST_IDX = UST_IDX + 1;
                END;
                UST_IDX = UST_IDX + 1;
                CRD_CTR = CRD_CTR + 1;
                RETURN;
            END;

    /* IS NEXT UNIFORM SYMBOL A HEX ADDRESS ? */

        UST_IDX=UST_IDX+1;
        IF(US_NTRY(UST_IDX)¬='LIT')|
            (INDEX(LT_NTRY(UST_PTR(UST_IDX)),'.')¬=0)|
            (INDEX(LT_NTRY(UST_PTR(UST_IDX)),' ')¬=5)
            THEN DO;
                ERROR(CNT)='HEX ADDRESS MUST FOLLOW TO';
                SRS_LIN(CNT)=CRD_CTR;
                CNT=CNT+1;
                DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                    (UST_PTR(UST_IDX)¬=23));
                    UST_IDX=UST_IDX+1;
                END;
                UST_IDX=UST_IDX+1;
              CRD_CTR=CRD_CTR+1;
                RETURN;
            END;

    /* WRITE HEX ADDRESS INTO MATRIX */

        OP2='LIT';
        AREA = UST_PTR(UST_IDX);
        AREA = TRANSLATE (AREA, ZERDO, BLANK);
        OP2_PTR = SUBSTR (AREA, 4, 4);
        WRITE FILE (MATRIX) FROM (MTX_FIL);

    /* IS REMAINDER OF SOURCE LINE BLANK ? */

        UST_IDX=UST_IDX+1;
        IF (US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23) THEN
            DO;
                ERROR(CNT)='MULTIPLE ENTRIES FOR HEX ADDRESS';
                SRS_LIN(CNT)=CRD_CTR;
                CNT=CNT+1;
                DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|
                         (UST_PTR(UST_IDX)¬=23));                             WT001110
                         UST_IDX=UST_IDX+1;                                   WT001120
                    END;                                                      WT001130
                    UST_IDX=UST_IDX+1;                                        WT001140
                    CRD_CTR=CRD_CTR+1;                                        WT001150
                    RETURN;                                                   WT001160
                END;                                                          WT001170
                                                                              WT001180
      /*NORMAL RETURN TO CONTROL*/                                            WT001190
                                                                              WT001200
         UST_IDX=UST_IDX+1;                                                   WT001210
         CRD_CTR=CRD_CTR+1;                                                   WT001220
         RETURN;                                                              WT001230
 END WTOUTAR;                                                                 WT001240
```

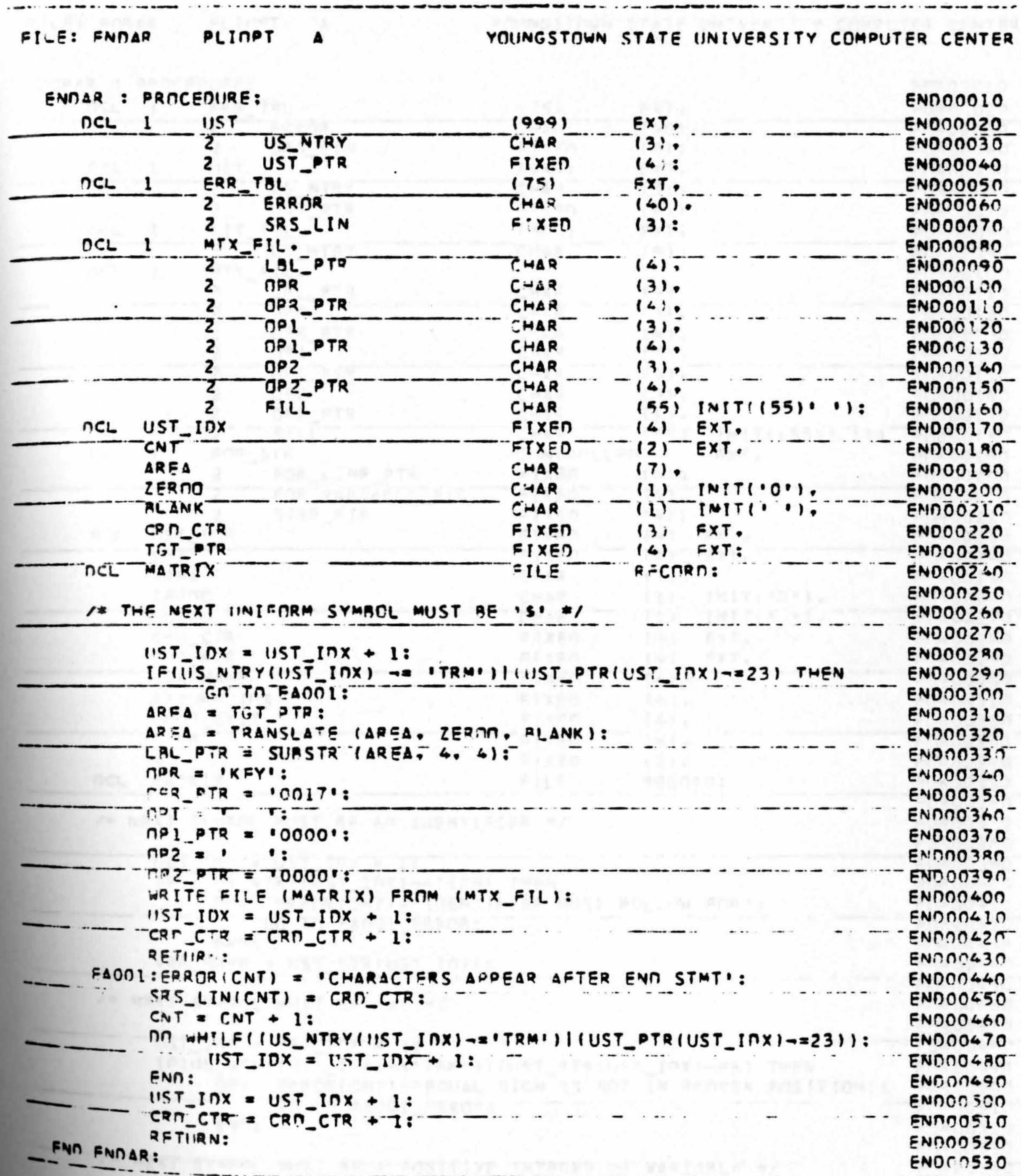
```
FILE: GOTOAR    PLIOPT    A                  YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 GOTOAR : PROCEDURE:
     DCL   1     UST                          (999)        EXT,
                 2     US_NTRY                CHAR         (3),
                 2     UST_PTR                FIXED        (4):
     DCL   1     ERR_TBL                      (75)         EXT,
                 2     ERROR                  CHAR         (40),
                 2     SRS_LIN                FIXED        (3):
     DCL   1     LIT_TBL                      (500)        EXT,
                 2     LT_NTRY                CHAR         (9):
     DCL   1     MTX_FIL,
                 2     LBL_PTR                CHAR         (4),
                 2     OPR                    CHAR         (3),
                 2     OPR_PTR                CHAR         (4),
                 2     OP1                    CHAR         (3),
                 2     OP1_PTR                CHAR         (4),
                 2     OP2                    CHAR         (3),
                 2     OP2_PTR                CHAR         (4),
                 2     FILL                   CHAR         (55)  INIT((55)' '):
     DCL   UST_IDX                            FIXED        (4)   EXT,
           CNT                                FIXED        (2)   EXT,
           CRD_CTR                            FIXED        (3)   EXT,
           TGT_PTR                            FIXED        (4)   EXT,
           AREA                               CHAR         (7),
           ZERDO                              CHAR         (1)   INIT('0'),
           BLANK                              CHAR         (1)   INIT(' '):
     DCL   MATRIX                             FILE         RECORD:

     /* IS NEXT UNIFORM SYMBOL A LITERAL OF < 6 CHARACTERS ? */

           UST_IDX = UST_IDX + 1:
           IF(US_NTRY(UST_IDX)¬='LIT')|
              (VERIFY(LT_NTRY(UST_PTR(UST_IDX)),'0123456789')>0)
              THEN DO:
                   ERROR(CNT)='LINE NUMBER MUST FOLLOW GOTO STMT':
                   SRS_LIN(CNT)=CRD_CTR:
                   CNT = CNT + 1:
                   DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                        (UST_PTR(UST_IDX)¬=23)):
                         UST_IDX = UST_IDX + 1:
                   END:
                   UST_IDX = UST_IDX + 1:
                   CRD_CTR = CRD_CTR + 1:
                   RETURN:
              END:

     /* CHECK FOR ILLEGAL RADIX OR ALPHABETIC */

           IF(VERIFY(LT_NTRY(UST_PTR(UST_IDX)),' 0123456789')¬=0) THEN
              DO:
                   ERROR(CNT)='LINE NUMBER MUST FOLLOW GOTO STMT':
                   SRS_LIN(CNT) = CRD_CTR:
                   CNT = CNT + 1:
                   DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                        (UST_PTR(UST_IDX)¬=23)):
                         UST_IDX = UST_IDX + 1;
```

```
                  END;                                                  GOT00560
                  UST_IDX = UST_IDX + 1;                                GOT00570
                  CRD_CTR = CRD_CTR + 1;                                GOT00580
                  RETURN;                                               GOT00590
             END;                                                       GOT00600
                                                                        GOT00610
     /* IS REMAINDER OF SOURCE LINE BLANK ? */                          GOT00620
                                                                        GOT00630
        UST_IDX = UST_IDX + 1;                                          GOT00640
        IF (US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23) THEN        GOT00650
             DO;                                                        GOT00660
                  ERROR(CNT)='MULTIPLE ARGUMENTS IN GOTO STMT';         GOT00670
                  SRS_LIN(CNT)=CRD_CTR;                                 GOT00680
                  CNT = CNT + 1;                                        GOT00690
                  DO WHILE((US_NTRY(UST_IDX)¬='TRM')|                   GOT00700
                       (UST_PTR(UST_IDX)¬=23));                         GOT00710
                       UST_IDX = UST_IDX + 1;                           GOT00720
                  END;                                                  GOT00730
                  UST_IDX = UST_IDX + 1;                                GOT00740
                  CRD_CTR = CRD_CTR + 1;                                GOT00750
                  RETURN;                                               GOT00760
             END;                                                       GOT00770
                                                                        GOT00780
     /* NORMAL RETURN TO CONTROL */                                     GOT00790
                                                                        GOT00800
        AREA = TGT_PTR;                                                 GOT00810
        AREA = TRANSLATE (AREA, ZEROO, BLANK);                          GOT00820
        LBL_PTR = SUBSTR (AREA, 4, 4);                                  GOT00830
        OPR = 'KEY';                                                    GOT00840
        OPR_PTR = '0002';                                               GOT00850
        OP1 = 'LIT';                                                    GOT00860
        AREA = UST_PTR(UST_IDX-1);                                      GOT00870
        AREA = TRANSLATE (AREA, ZEROO, BLANK);                          GOT00880
        OP1_PTR = SUBSTR (AREA, 4, 4);                                  GOT00890
        OP2 = '   ';                                                    GOT00900
        OP2_PTR = '0000';                                               GOT00910
        WRITE FILE (MATRIX) FROM (MTX_FIL);                             GOT00920
        UST_IDX = UST_IDX + 1;                                          GOT00930
        CRD_CTR = CRD_CTR + 1;                                          GOT00940
        RETURN;                                                         GOT00950
  END GOTOAR;                                                           GOT00960
```

```
FILE: RESTRAR   PLIOPT   A              YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 RESTRAR : PROCEDURE:
     DCL  1     UST                          (999)     EXT,
                2     US_NTRY                CHAR      (3),
                2     UST_PTR                FIXED     (4):
     DCL  1     ERR_TBL                      (75)      EXT,
                2     ERROR                  CHAR      (40),
                2     SRS_LIN                FIXED     (3):
     DCL  1     MTX_FIL,
                2     LBL_PTR                CHAR      (4),
                2     OPR                    CHAR      (3),
                2     OPR_PTR                CHAR      (4),
                2     OP1                    CHAR      (3),
                2     OP1_PTR                CHAR      (4),
                2     OP2                    CHAR      (3),
                2     OP2_PTR                CHAR      (4),
                2     FILL                   CHAR      (55)  INIT((55)' '):
     DCL  UST_IDX                            FIXED     (4)  EXT,
          CNT                                FIXED     (2)  EXT,
          AREA                               CHAR      (7),
          ZEROO                              CHAR      (1)  INIT('0'),
          BLANK                              CHAR      (1)  INIT(' '),
          CRD_CTR                            FIXED     (3)  EXT,
          TGT_PTR                            FIXED     (4)  EXT:
     DCL  MATRIX                             FILE      RECORD:

     /* THE NEXT UNIFORM SYMBOL MUST BE '$' */

          UST_IDX = UST_IDX + 1:
          IF(US_NTRY(UST_IDX) ¬= 'TRM')|(UST_PTR(UST_IDX)¬=23) THEN
               GO TO RS001:
          AREA = TGT_PTR:
          AREA = TRANSLATE (AREA, ZEROO, BLANK);
          LBL_PTR = SUBSTR (AREA, 4, 4):
          OPR = 'KEY':
          OPR_PTR = '0014':
          OP1 = '   ':
          OP1_PTR = '0000':
          OP2 = '   ':
          OP2_PTR = '0000':
          WRITE FILE (MATRIX) FROM (MTX_FIL):
          UST_IDX = UST_IDX + 1:
          CRD_CTR = CRD_CTR + 1:
          RETURN:
      RS001:ERROR(CNT) = 'CHARACTERS APPEAR AFTER RESTORE STMT':
          SRS_LIN(CNT) = CRD_CTR:
          CNT = CNT + 1:
          DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23)):
               UST_IDX = UST_IDX + 1:
          END:
          UST_IDX = UST_IDX + 1:
          CRD_CTR = CRD_CTR + 1:
          RETURN:
   END RESTRAR:
```

```
 READAR : PROCEDURE;
    DCL  1   UST                        (999)     EXT,
             2   US_NTRY                CHAR      (3),
             2   UST_PTR                FIXED     (4);
    DCL  1   ERR_TBL                    (75)      EXT,
             2   ERROR                  CHAR      (40),
             2   SRS_LIN                FIXED     (3);
    DCL  1   MTX_FIL,
             2   LBL_PTR                CHAR      (4),
             2   OPR                    CHAR      (3),
             2   OPR_PTR                CHAR      (4),
             2   OP1                    CHAR      (3),
             2   OP1_PTR                CHAR      (4),
             2   OP2                    CHAR      (3),
             2   OP2_PTR                CHAR      (4),
             2   FILL                   CHAR      (55) INIT((55)' ');
    DCL  UST_IDX                        FIXED     (4)  EXT,
         CNT                            FIXED     (2)  EXT,
         AREA                           CHAR      (7),
         ZEROO                          CHAR      (1)  INIT('0'),
         BLANK                          CHAR      (1)  INIT(' '),
         CRD_CTR                        FIXED     (3)  EXT,
         TGT_PTR                        FIXED     (4)  EXT;
    DCL  MATRIX                         FILE      RECORD;

    /* IS THE NEXT UNIFORM SYMBOL AN IDENTIFIER ? */

         UST_IDX = UST_IDX + 1;
    RD001:IF US_NTRY(UST_IDX)¬='IDN' THEN GO TO RD002;

    /* ENTER IDENTIFIER INTO MATRIX */

         AREA = TGT_PTR;
         AREA = TRANSLATE (AREA, ZEROO, BLANK);
         LBL_PTR = SUBSTR (AREA, 4, 4);
         OPR='KEY';
         OPR_PTR='0013';
         OP1='IDN';
         AREA = UST_PTR(UST_IDX);
         AREA = TRANSLATE (AREA, ZEROO, BLANK);
         OP1_PTR = SUBSTR (AREA, 4, 4);
         OP2='   ';
         OP2_PTR='0000';
         WRITE FILE (MATRIX) FROM (MTX_FIL);
         UST_IDX=UST_IDX+1;

    /* IS NEW UNIFORM SYMBOL A COMMA ? */

         IF (US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=11)THEN
              GO TO RD003;
         UST_IDX=UST_IDX+1;
         GO TO RD001;
    RD002:ERROR(CNT) = 'READ ARGUMENT MUST BE A VARIABLE NAME';
         SRS_LIN(CNT)=CRD_CTR;
         CNT=CNT+1;
```

```
        DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
             UST_IDX=UST_IDX + 1;
        END;
        UST_IDX=UST_IDX+1;
        CRD_CTR=CRD_CTR+1;
        RETURN;
   RD003:IF (US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23) THEN
        DO:  ERROR(CNT)='COMMAS REQUIRED BETWEEN READ ARGUMENTS';
             SRS_LIN(CNT)=CRD_CTR;
             CNT=CNT+1;
             DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=
                  23));
                  UST_IDX=UST_IDX+1;
             END;
             UST_IDX=UST_IDX+1;
             CRD_CTR=CRD_CTR+1;
             RETURN;
        END;

   /* NORMAL RETURN TO CONTROL */

        UST_IDX = UST_IDX + 1;
        CRD_CTR = CRD_CTR + 1;
        RETURN;
END READAR;
```



```
FILE: ENDAR     PLIOPT   A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

  ENDAR : PROCEDURE:
     DCL  1     UST                           (999)      EXT,
                2   US_NTRY                   CHAR       (3),
                2   UST_PTR                   FIXED      (4):
     DCL  1     ERR_TBL                       (75)       EXT,
                2    ERROR                    CHAR       (40).
                2   SRS_LIN                   FIXED      (3):
     DCL  1     MTX_FIL.
                2    LBL_PTR                  CHAR       (4),
                2    OPR                      CHAR       (3),
                2    OPR_PTR                  CHAR       (4),
                2    OP1                      CHAR       (3),
                2    OP1_PTR                  CHAR       (4),
                2    OP2                      CHAR       (3),
                2    OP2_PTR                  CHAR       (4),
                2    FILL                     CHAR       (55)  INIT((55)' '):
     DCL  UST_IDX                             FIXED      (4)   EXT,
          CNT                                 FIXED      (2)   EXT.
          AREA                                CHAR       (7),
          ZERO0                               CHAR       (1)   INIT('0'),
          BLANK                               CHAR       (1)   INIT(' '),
          CRD_CTR                             FIXED      (3)   EXT.
          TGT_PTR                             FIXED      (4)   EXT:
     DCL  MATRIX                              FILE       RECORD:

   /* THE NEXT UNIFORM SYMBOL MUST BE '$' */

          UST_IDX = UST_IDX + 1:
          IF(US_NTRY(UST_IDX) ¬= 'TRM')|(UST_PTR(UST_IDX)¬=23) THEN
                GO TO EA001:
          AREA = TGT_PTR:
          AREA = TRANSLATE (AREA, ZERO0, BLANK):
          LBL_PTR = SUBSTR (AREA, 4, 4):
          OPR = 'KEY':
          OPR_PTR = '0017':
          OP1 = '   ':
          OP1_PTR = '0000':
          OP2 = '   ':
          OP2_PTR = '0000':
          WRITE FILE (MATRIX) FROM (MTX_FIL):
          UST_IDX = UST_IDX + 1:
          CRD_CTR = CRD_CTR + 1:
          RETURN:
   EA001:ERROR(CNT) = 'CHARACTERS APPEAR AFTER END STMT':
          SRS_LIN(CNT) = CRD_CTR:
          CNT = CNT + 1:
          DO WHILE((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23)):
                UST_IDX = UST_IDX + 1:
          END:
          UST_IDX = UST_IDX + 1:
          CRD_CTR = CRD_CTR + 1:
          RETURN:
  END ENDAR:
```

```
FORAR : PROCEDURE:
   DCL  1    ERR_TBL                (75)        EXT,
             2    ERROR              CHAR        (40),
             2    SRS_LIN            FIXED       (3):
   DCL  1    UST                    (999)       EXT,
             2    US_NTRY            CHAR        (3),
             2    UST_PTR            FIXED       (4):
   DCL  1    LIT_TBL                (500)       EXT,
             2    LT_NTRY            CHAR        (9):
   DCL  1    MTX_FIL,
             2    LBL_PTR            CHAR        (4),
             2    OPR                CHAR        (3),
             2    OPR_PTR            CHAR        (4),
             2    OP1                CHAR        (3),
             2    OP1_PTR            CHAR        (4),
             2    OP2                CHAR        (3),
             2    OP2_PTR            CHAR        (4),
             2    FILL               CHAR        (55)  INIT((55)' '):
   DCL  1    FOR_STK                CONTROLLED        EXT,
             2    FOR_LINE_PTR       FIXED       (4),
             2    FOR_VARIABLE_PTR   FIXED       (4),
             2    STEP_PTR           FIXED       (4):
   DCL  UST_IDX                     FIXED       (4)  EXT,
        CNT                         FIXED       (2)  EXT,
        AREA                        CHAR        (7),
        ZEROO                       CHAR        (1)  INIT('0'),
        BLANK                       CHAR        (1)  INIT(' '),
        CRD_CTR                     FIXED       (3)  EXT,
        TGT_PTR                     FIXED       (4)  EXT,
        VAR_NAME                    FIXED       (4),
        RETURN_IDX                  FIXED       (4),
        LOOP_EXIT                   FIXED       (4),
        STEP                        FIXED       (4),
        N                           FIXED       (2):
   DCL  MATRIX                      FILE        RECORD:

 /* NEXT SYMBOL MUST BE AN IDENTIFIER */

        UST_IDX = UST_IDX + 1:
        IF    US_NTRY(UST_IDX)¬='IDN' THEN
              DO:  ERROR(CNT)='IDENTIFIER MUST FOLLOW FOR':
                 GO TO FA001_ERROR:
              END:
        VAR_NAME = UST_PTR(UST_IDX):

 /* NEXT SYMBOL MUST BE '=' */

        UST_IDX = UST_IDX + 1:
        IF(US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=6) THEN
              DO:  ERROR(CNT)='EQUAL SIGN IS NOT IN PROPER POSITION':
                   GO TO FA001_ERROR:
              END:

 /* NEXT SYMBOL MUST BE A POSITIVE INTEGER OR VARIABLE */
```

```
        UST_IDX = UST_IDX + 1;
        IF(US_NTRY(UST_IDX)¬='IDN')&(US_NTRY(UST_IDX)¬='LIT') THEN
           DO:  ERROR(CNT)='POS.INTEGER OR VAR. MUST PRECEDE TO':
                GO TO FA001_ERROR:
           END:
        IF(US_NTRY(UST_IDX)='LIT')&
          (VERIFY(LT_NTRY(UST_PTR(UST_IDX)),' 0123456789')¬=0) THEN
           DO:  ERROR(CNT)='LITERAL PRECEDING TO IS NOT INTEGER':
                GO TO FA001_ERROR:
           END:

   /* WRITE INTO MATRIX SETTING VARIABLE EQUAL TO TOKEN */

        AREA = TGT_PTR:
        AREA = TRANSLATE(AREA,ZEROO,BLANK);
        LBL_PTR = SUBSTR(AREA, 4, 4);
        OPR = 'TRM';
        OPR_PTR = '0005';
        OP1 = 'IDN';
        AREA = VAR_NAME;
        AREA = TRANSLATE(AREA,ZEROO,BLANK);
        OP1_PTR = SUBSTR (AREA, 4, 4);
        OP2 = US_NTRY(UST_IDX);
        AREA = UST_PTR(UST_IDX);
        AREA = TRANSLATE(AREA,ZEROO,BLANK);
        OP2_PTR = SUBSTR (AREA, 4, 4):
        WRITE FILE (MATRIX) FROM (MTX_FIL):

   /* NEXT SYMBOL MUST BE 'TO' */

        UST_IDX = UST_IDX + 1:
        IF(US_NTRY(UST_IDX)¬='KEY')|(UST_PTR(UST_IDX)¬=4) THEN
           DO:  ERROR(CNT)='TO IS MISPLACED OR MISSING':
                GO TO FA001_ERROR:
           END:

   /* DETERMINE NUMBER OF LINE FOLLOWING ASSOCIATED 'NEXT' */

        RETURN_IDX=UST_IDX:
        DO WHILE (US_NTRY(UST_IDX)¬=' '):
  FA002:    UST_IDX=UST_IDX+1:
            IF(US_NTRY(UST_IDX)='KEY')&(UST_PTR(UST_IDX)=7)&
                (US_NTRY(UST_IDX+1)='IDN')&(UST_PTR(UST_IDX+1)=
                VAR_NAME) THEN
                  DO:
                        LOOP_EXIT=UST_PTR(UST_IDX+3):
                        UST_IDX = RETURN_IDX:
                        GO TO FA003:
                  END:
            GO TO FA002:
        END:
        ERROR(CNT) = 'MISSING NEXT STATEMENT':
        GO TO FA001_ERROR:
  FA003:
```

```
  /* NEXT SYMBOL MUST BE A POSITIVE INTEGER OR VARIABLE */
       UST_IDX = UST_IDX + 1:
       IF(US_NTRY(UST_IDX)¬='IDN')&(US_NTRY(UST_IDX)¬='LIT') THEN
            DO:  ERROR(CNT)='POS. INTEGER OR VAR. MUST FOLLOW TO':
                 GO TO FA001_ERROR:
            END:
       IF(US_NTRY(UST_IDX)='LIT')&(VERIFY(LT_NTRY(UST_PTR(UST_IDX)),
            ' 0123456789')¬=0) THEN
            DO:  ERROR(CNT)='LITERAL FOLLOWING TO IS NOT INTEGER':
                 GO TO FA001_ERROR:
            END:

  /* WRITE MATRIX LINE TO COMPARE VALUE OF VARIABLE TO LIMIT */

       OPR_PTR='0024':
       OP2=US_NTRY(UST_IDX):
       AREA = UST_PTR(UST_IDX):
       AREA = TRANSLATE(AREA, ZEROO, BLANK):
       OP2_PTR = SUBSTR(AREA, 4, 4):
       WRITE FILE (MATRIX) FROM (MTX_FIL):

  /* WRITE MATRIX LINE TO EXIT LOOP */

       OPR = 'KEY':
       OPR_PTR = '0002':
       OP1 = 'LIT':
       AREA = LOOP_EXIT:
       AREA = TRANSLATE(AREA, ZEROO, BLANK):
       OP1_PTR = SUBSTR(AREA, 4, 4):
       OP2 = '    ':
       OP2_PTR = '0000':
       WRITE FILE (MATRIX) FROM (MTX_FIL):

  /* IS NEXT SYMBOL '$' ? */

       UST_IDX = UST_IDX + 1:
       IF(US_NTRY(UST_IDX)='TRM')&(UST_PTR(UST_IDX)=23) THEN
            DO:  STEP = 1:
                 GO TO FA004:
            END:

  /* IS SYMBOL 'STEP' ? */

       IF(US_NTRY(UST_IDX)¬='KEY')|(UST_PTR(UST_IDX)¬=10) THEN
            DO:  ERROR(CNT)='ONLY STEP MAY FOLLOW INTEGER AFTER TO':
                 GO TO FA001_ERROR:
            END:

  /* IS NEXT SYMBOL A POSITIVE INTEGER OR VARIABLE ? */

       UST_IDX = UST_IDX + 1:
       IF(US_NTRY(UST_IDX)¬='IDN')&(US_NTRY(UST_IDX)¬='LIT') THEN
            DO:  ERROR(CNT)='POSITIVE INTEGER MUST FOLLOW STEP':
                 GO TO FA001_ERROR:
            END:
```

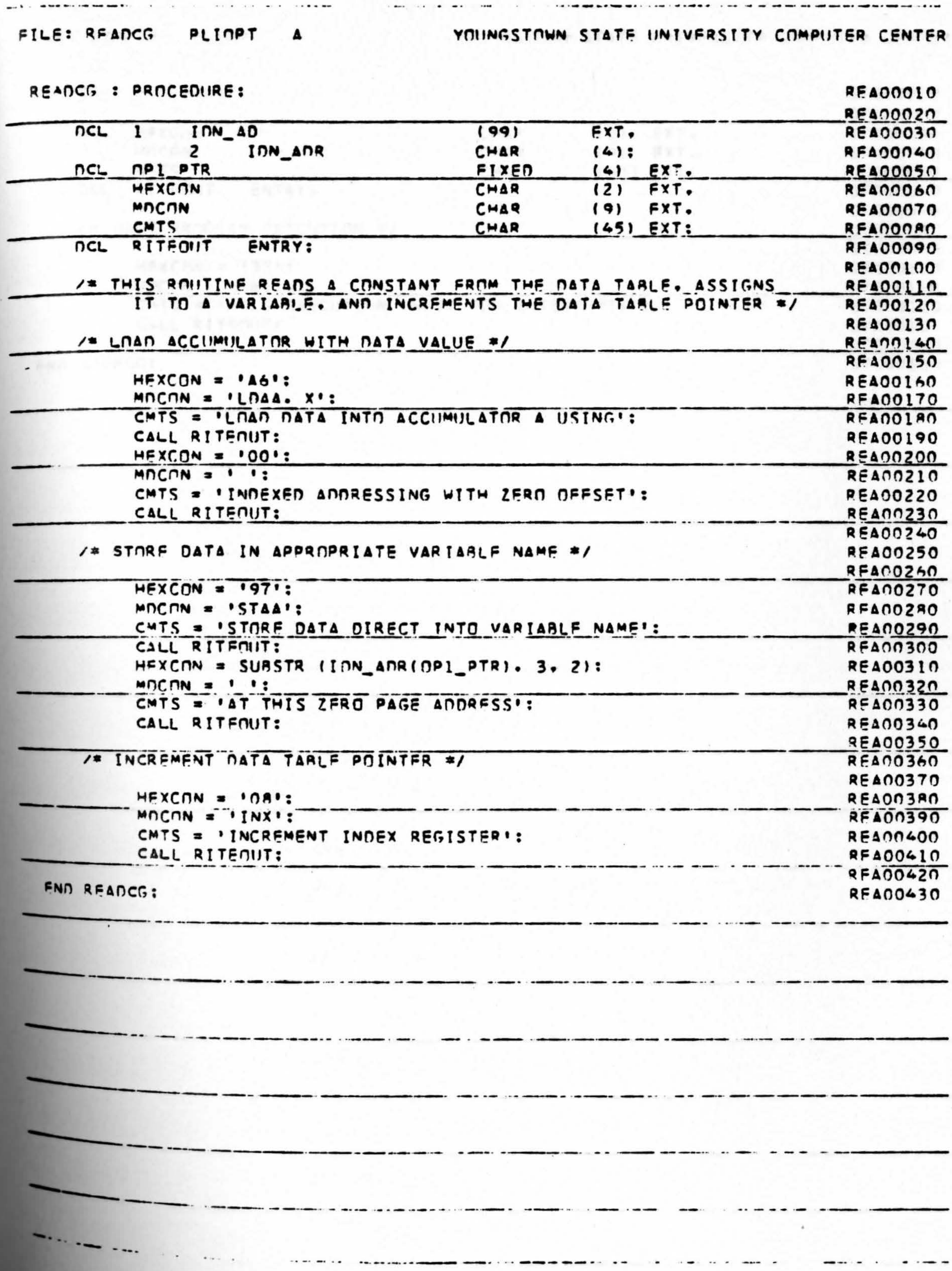
```
        IF(US_NTRY(UST_IDX)='LIT')&
          (VERIFY(LT_NTRY(UST_PTR(UST_IDX)),' 0123456789')¬=0)THEN
             DO:  ERROR(CNT)='LITERAL FOLLOWING STEP IS NOT INTEGER':
                  GO TO FA001_ERROR:
             END:

   /* IS NEXT SYMBOL 'S' ? */

        UST_IDX = UST_IDX + 1;
        IF(US_NTRY(UST_IDX)¬='TRM')&(UST_PTR(UST_IDX)¬=23) THEN
             DO:  ERROR(CNT)='EXTRANEOUS CHARACTERS AT END OF LINE':
                  GO TO FA001_ERROR:
             END:

        STEP = UST_PTR(UST_IDX-1):

   FA004:

   /* PUSH LINE NUMBER, VARIABLE NAME, AND STEP VALUE ONTO STACK */

        ALLOCATE FOR_STK:
        FOR_LINE_PTR = TGT_PTR:
        FOR_VARIABLE_PTR = VAR_NAME:
        STEP_PTR = STEP:

   /* NORMAL RETURN TO CONTROL */

        UST_IDX = UST_IDX + 1:
        CRD_CTR = CRD_CTR + 1:
        RETURN:

   /* ERROR ROUTINE */

   FA001_ERROR:
        SRS_LIN(CNT)=CRD_CTR:
        CNT = CNT + 1:
        DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23)):
             UST_IDX = UST_IDX + 1:
        END:
        UST_IDX = UST_IDX + 1:
        CRD_CTR = CRD_CTR + 1:
        RETURN:
    END FORAR:
```

FILE: NEXTAR   PLIOPT   A   YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
NEXTAR : PROCEDURE:
   DCL  1     UST                             (999)     EXT,
              2     US_NTRY                   CHAR      (3),
              2     UST_PTR                   FIXED     (4):
   DCL  1     ERR_TBL                         (75)      EXT,
              2     ERROR                     CHAR      (40),
              2     SRS_LIN                   FIXED     (3):
   DCL  1     MTX_FIL,
              2     LBL_PTR                   CHAR      (4),
              2     OPR                       CHAR      (3),
              2     OPR_PTR                   CHAR      (4),
              2     OP1                       CHAR      (3),
              2     OP1_PTR                   CHAR      (4),
              2     OP2                       CHAR      (3),
              2     OP2_PTR                   CHAR      (4),
              2     FILL                      CHAR      (55)  INIT((55)' '):
   DCL  UST_IDX                               FIXED     (4)   EXT,
        CNT                                   FIXED     (2)   EXT,
        CRD_CTR                               FIXED     (3)   EXT,
        TGT_PTR                               FIXED     (4)   EXT,
        TMP_STR                               FIXED     (3)   EXT,
        AREA                                  CHAR      (7),
        ZEROO                                 CHAR      (1)   INIT('0'),
        BLANK                                 CHAR      (1)   INIT(' '):
   DCL  MATRIX                                FILE      RECORD:
   DCL  1     FOR_STK                         CONTROLLED      EXT,
              2     FOR_LINE_PTR              FIXED     (4),
              2     FOR_VARIABLE_PTR          FIXED     (4),
              2     STEP_PTR                  FIXED     (4):

   /* IS NEXT UNIFORM SYMBOL AN IDENTIFIER ? */

        UST_IDX = UST_IDX + 1:
        IF US_NTRY(UST_IDX) ¬= 'IDN' THEN
             DO:
                  ERROR(CNT)='NEXT MUST BE FOLLOWED BY A VARIABLE':
                  GO TO NA001_ERROR:
             END:

   /* DOES VARIABLE MATCH TOP OF STACK ? */

        IF UST_PTR(UST_IDX)¬= FOR_VARIABLE_PTR THEN
             DO:  ERROR(CNT) = 'IMPROPER FOR - NEXT PAIR':
                  GO TO NA001_ERROR:
             END:

   /* WRITE MATRIX LINES TO ADD STEP VALUE TO VARIABLE */

                  AREA = TGT_PTR:
                  AREA = TRANSLATE(AREA, ZEROO, BLANK):
                  LBL_PTR = SUBSTR(AREA, 4, 4):
                  OPR='TRM':
                  OPR_PTR='0002':
                  OP1='IDN':
                  AREA = UST_PTR(UST_IDX):
```

```
                 AREA = TRANSLATE(AREA, ZEROO, BLANK);
                 OP1_PTR = SUBSTR(AREA, 4, 4);
                 OP2 = 'LIT';
                 AREA = STEP_PTR;
                 AREA = TRANSLATE(AREA, ZEROO, BLANK);
                 OP2_PTR = SUBSTR(AREA, 4, 4);
                 WRITE FILE (MATRIX) FROM (MTX_FIL);

                 OPR_PTR = '0006';
                 OP2 = 'TMP';
                 OP2_PTR = '0001';
                 WRITE FILE (MATRIX) FROM (MTX_FIL);

   /* WRITE MATRIX LINE TO RETURN EXECUTION TO TOP OF LOOP */

       OPR = 'KEY';
       OPR_PTR = '0002';
       OP1 = 'LIT';
       AREA = FOR_LINE_PTR;
       AREA = TRANSLATE(AREA, ZEROO, BLANK);
       OP1_PTR = SUBSTR(AREA, 4, 4);
       OP2 = 'LIT';
       OP2_PTR = '0002';
       WRITE FILE (MATRIX) FROM (MTX_FIL);

   /* POP UP NEXT 'FOR' INFORMATION */

       FREE FOR_STK;

   /* IS NEXT SYMBOL '$' ? */

       UST_IDX = UST_IDX + 1;
       IF(US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23) THEN
           DO:  ERROR(CNT)='CHARACTERS APPEAR AFTER VARIABLE NAME';
                GO TO NA001_ERROR;
           END;

   /* NORMAL RETURN TO CONTROL */

       UST_IDX = UST_IDX + 1;
       CRD_CTR = CRD_CTR + 1;
       RETURN;

   /* ERROR ROUTINE */

   NA001_ERROR:
       SRS_LIN(CNT) = CRD_CTR;
       CNT = CNT + 1;
       DO WHILE ((US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23));
            UST_IDX=UST_IDX+1;
       END;
       UST_IDX = UST_IDX + 1;
       CRD_CTR = CRD_CTR + 1;
       RETURN;
 END NEXTAR;
 GOSUBAR : PROCEDURE;
    DCL   1      UST                     (999)       EXT,
                 2    US_NTRY            CHAR        (3),
                 2    UST_PTR            FIXED       (4);
    DCL   1      ERR_TBL                 (75)        EXT,
                 2    ERROR              CHAR        (40),
                 2    SRS_LIN            FIXED       (3);
    DCL   1      LIT_TBL                 (500)       EXT,
                 2    LT_NTRY            CHAR        (9);
    DCL   1      MTX_FIL,
                 2    LBL_PTR            CHAR        (4),
                 2    OPR                CHAR        (3),
                 2    OPR_PTR            CHAR        (4),
                 2    OP1                CHAR        (3),
                 2    OP1_PTR            CHAR        (4),
                 2    OP2                CHAR        (3),
                 2    OP2_PTR            CHAR        (4),
                 2    FILL               CHAR        (55)  INIT((55)' ');
     DCL   UST_IDX                       FIXED       (4)   EXT,
           CNT                           FIXED       (2)   EXT,
           CRD_CTR                       FIXED       (3)   EXT,
           TGT_PTR                       FIXED       (4)   EXT,
           AREA                          CHAR        (7),
           ZEROO                         CHAR        (1)   INIT('0'),
           BLANK                         CHAR        (1)   INIT(' ');
     DCL   MATRIX                        FILE        RECORD;

     /* IS NEXT UNIFORM SYMBOL A LITERAL OF < 6 CHARACTERS ? */

           UST_IDX = UST_IDX + 1;
           IF(US_NTRY(UST_IDX)¬='LIT')|
              (VERIFY(LT_NTRY(UST_PTR(UST_IDX)),'0123456789')>6)
              THEN DO;
                    ERROR(CNT)='LINE NUMBER MUST FOLLOW GOSUB STMT';
                    SRS_LIN(CNT)=CRD_CTR;
                    CNT = CNT + 1;
                    DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                          (UST_PTR(UST_IDX)¬=23));
                          UST_IDX = UST_IDX + 1;
                    END;
                    UST_IDX = UST_IDX + 1;
                    CRD_CTR = CRD_CTR + 1;
                    RETURN;
              END;

     /* CHECK FOR ILLEGAL RADIX OR ALPHABETIC */

           IF(VERIFY(LT_NTRY(UST_PTR(UST_IDX)),' 0123456789')¬=0) THEN
              DO;
                    ERROR(CNT)='LINE NUMBER MUST FOLLOW GOSUB STMT';
                    SRS_LIN(CNT)=CRD_CTR;
                    CNT = CNT + 1;
                    DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
                          (UST_PTR(UST_IDX)¬=23));
                          UST_IDX = UST_IDX + 1;
```

```
FILE: GOSUBAR   PLIOPT   A          YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

           END;
           UST_IDX = UST_IDX + 1:
           CRD_CTR = CRD_CTR + 1:
           RETURN:
        END:

  /* IS REMAINDER OF SOURCE LINE BLANK ? */

     UST_IDX = UST_IDX + 1:
     IF (US_NTRY(UST_IDX)¬='TRM')|(UST_PTR(UST_IDX)¬=23) THEN
        DO:
           ERROR(CNT)='MULTIPLE ARGUMENTS IN GOSUB STMT':
           SRS_LIN(CNT)=CRD_CTR:
           CNT = CNT + 1:
           DO WHILE((US_NTRY(UST_IDX)¬='TRM')|
              (UST_PTR(UST_IDX)¬=23)):
              UST_IDX = UST_IDX + 1;
           END:
           UST_IDX = UST_IDX + 1:
           CRD_CTR = CRD_CTR + 1:
           RETURN;
        END:

  /* NORMAL RETURN TO CONTROL */
     AREA = TGT_PTR:
     AREA = TRANSLATE (AREA, ZEROO, BLANK):
     LBL_PTR = SUBSTR (AREA, 4, 4):
     OPR = 'KEY':
     OPR_PTR = '0008':
     OP1 = 'LIT':
     AREA = UST_PTR(UST_IDX-1):
     AREA = TRANSLATE (AREA, ZEROO, BLANK);
     OP1_PTR = SUBSTR (AREA, 4, 4):
     OP2 = '    ':
     OP2_PTR = '0000':
     WRITE FILE (MATRIX) FROM (MTX_FIL):
     UST_IDX = UST_IDX + 1:
     CRD_CTR = CRD_CTR + 1:
     RETURN:
END GOSUBAR;
```



```
FILE: READCG    PLIOPT    A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 READCG : PROCEDURE;

     DCL  1     IDN_AD                        (99)        EXT,
                2     IDN_ADR                 CHAR        (4);
     DCL  OP1_PTR                             FIXED       (4)  EXT,
          HEXCON                              CHAR        (2)  EXT,
          MDCON                               CHAR        (9)  EXT,
          CMTS                                CHAR        (45) EXT;
     DCL  RITEOUT    ENTRY;

     /* THIS ROUTINE READS A CONSTANT FROM THE DATA TABLE, ASSIGNS
          IT TO A VARIABLE, AND INCREMENTS THE DATA TABLE POINTER */

     /* LOAD ACCUMULATOR WITH DATA VALUE */

          HEXCON = 'A6';
          MDCON = 'LDAA. X';
          CMTS = 'LOAD DATA INTO ACCUMULATOR A USING';
          CALL RITEOUT;
          HEXCON = '00';
          MDCON = ' ';
          CMTS = 'INDEXED ADDRESSING WITH ZERO OFFSET';
          CALL RITEOUT;

     /* STORE DATA IN APPROPRIATE VARIABLE NAME */

          HEXCON = '97';
          MDCON = 'STAA';
          CMTS = 'STORE DATA DIRECT INTO VARIABLE NAME';
          CALL RITEOUT;
          HEXCON = SUBSTR (IDN_ADR(OP1_PTR), 3, 2);
          MDCON = ' ';
          CMTS = 'AT THIS ZERO PAGE ADDRESS';
          CALL RITEOUT;

     /* INCREMENT DATA TABLE POINTER */

          HEXCON = '08';
          MDCON = 'INX';
          CMTS = 'INCREMENT INDEX REGISTER';
          CALL RITEOUT;

  END READCG;
```

FILE: STOPCG   PLIOPT   A   YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

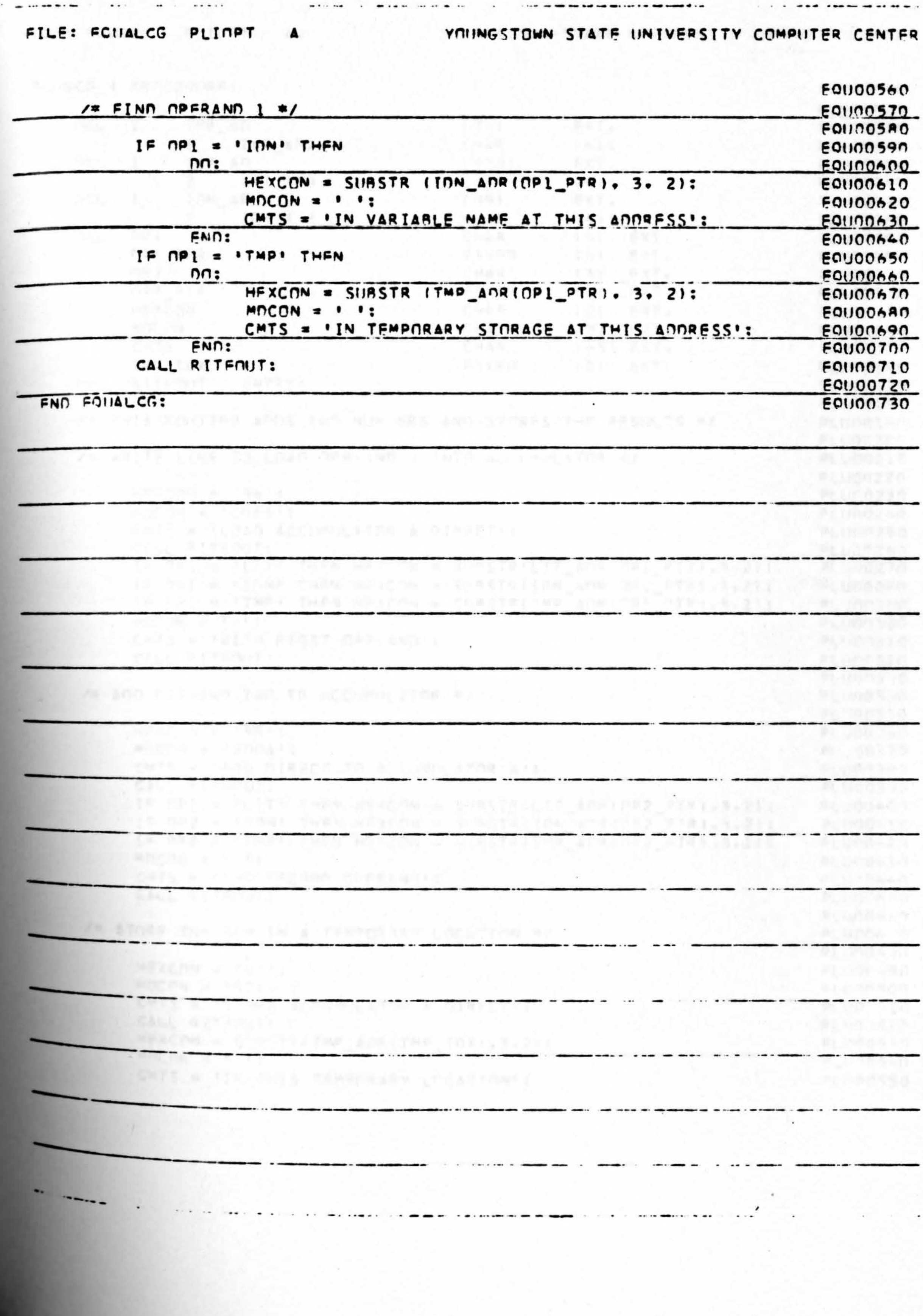
```
 STOPCG : PROCEDURE;

     DCL  HEXCON                           CHAR       (2)   EXT,
          MDCON                            CHAR       (9)   EXT,
          CMTS                             CHAR       (45)  EXT;
     DCL  RITEOUT    ENTRY;

     /* HALT PROGRAM EXECUTION */

          HEXCON = '3E';
          MDCON = 'WAI';
          CMTS = 'HALT EXECUTION (WAIT FOR INTERRUPT)';
          CALL RITEOUT;

 END STOPCG;
```

```
FILE: ENDCG    PLIOPT   A               YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 ENDCG : PROCEDURE;                                                      END00010
                                                                         END00020
      DCL  HEXCON                          CHAR       (2)  EXT,          END00030
           MDCON                           CHAR       (9)  EXT,          END00040
           CMTS                            CHAR       (45) EXT;          END00050
      DCL  RITEOUT    ENTRY;                                             END00060
                                                                         END00070
      /* HALT PROGRAM EXECUTION */                                       END00080
                                                                         END00090
           HEXCON = '3E';                                                END00100
           MDCON = 'WAI';                                                END00110
           CMTS = 'HALT EXECUTION (WAIT FOR INTERRUPT)';                 END00120
           CALL RITEOUT;                                                 END00130
                                                                         END00140
 END ENDCG;                                                              END00150
```

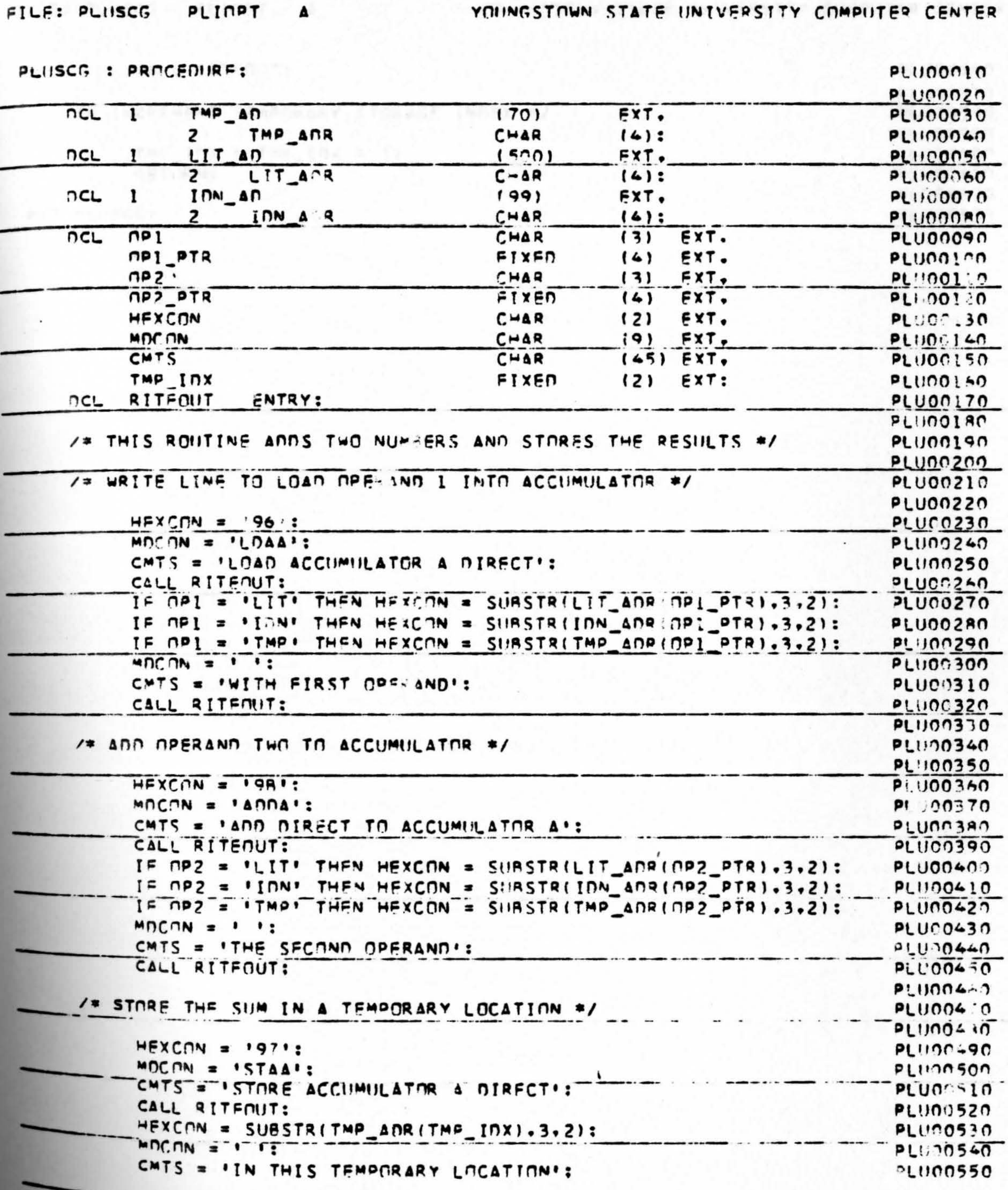
```
FILE: MTPLYCG  PLIOPT   A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

MTPLYCG : PROCEDURE;

    DCL  1   TMP_AD                     (70)       EXT,
             2   TMP_ADR                CHAR       (4);
    DCL  1   LIT_AD                     (500)      EXT,
             2   LIT_ADR                CHAR       (4);
    DCL  1   IDN_AD                     (99)       EXT,
             2   IDN_ADR                CHAR       (4);
    DCL  OP1                            CHAR       (3)  EXT,
         OP1_PTR                        FIXED      (4)  EXT,
         OP2                            CHAR       (3)  EXT,
         OP2_PTR                        FIXED      (4)  EXT,
         HEXCON                         CHAR       (2)  EXT,
         MDCON                          CHAR       (9)  EXT,
         CMTS                           CHAR       (45) EXT,
         TMP_IDX                        FIXED      (2)  EXT;
    DCL  RITEOUT     ENTRY;

    /* THIS ROUTINE MULTIPLIES TWO NUMBERS */

         HEXCON = 'D6';
         MDCON = 'LDAB';
         CMTS = 'LOAD THE B ACCUMULATOR WITH THE';
         CALL RITEOUT;

         IF OP1 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP1_PTR),3,2);
         IF OP1 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP1_PTR),3,2);
         IF OP1 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP1_PTR),3,2);
         MDCON = ' ';
         CMTS = 'MULTIPLIER FROM THIS LOCATION';
         CALL RITEOUT;

         HEXCON = '4F';
         MDCON = 'CLRA';
         CMTS = 'CLEAR ACCUMULATOR A';
         CALL RITEOUT;

         HEXCON = '97';
         MDCON = 'STAA';
         CMTS = 'STORE THE PRODUCT IN';
         CALL RITEOUT;

         HEXCON = SUBSTR(TMP_ADR(TMP_IDX),3,2);
         MDCON = ' ';
         CMTS = 'THIS TEMPORARY LOCATION';
         CALL RITEOUT;

         HEXCON = '17';
         MDCON = 'TBA';
         CMTS = 'TRANSFER ACCUMULATOR B TO A';
         CALL RITEOUT;

         HEXCON = '27';
         MDCON = 'BEQ';
         CMTS = 'IF THE MULTIPLIER IS ZERO, BRANCH';
```

```
        CALL RITEOUT;

        HEXCON = '08';
        MDCON = ' ';
        CMTS = 'TO THE NEXT ALGORITHM';
        CALL RITEOUT;

        HEXCON = '4A';
        MDCON = 'DECA';
        CMTS = 'OTHERWISE, DECREMENT THE MULTIPLIER';
        CALL RITEOUT;

        HEXCON = '16';
        MDCON = 'TAB';
        CMTS = 'TRANSFER ACCUMULATOR A TO B';
        CALL RITEOUT;

        HEXCON = '96';
        MDCON = 'LDAA';
        CMTS = 'LOAD THE ACCUMULATOR WITH THE';
        CALL RITEOUT;

        HEXCON = SUBSTR(TMP_ADR(TMP_IDX),3,2);
        MDCON = ' ';
        CMTS = 'PRODUCT STORED IN THIS TEMPORARY LOCATION';
        CALL RITEOUT;

        HEXCON = '9B';
        MDCON = 'ADDA';
        CMTS = 'ADD TO THE PRODUCT';
        CALL RITEOUT;

        IF OP2 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP2_PTR),3,2);
        IF OP2 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP2_PTR),3,2);
        IF OP2 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP2_PTR),3,2);
        MDCON = ' ';
        CMTS = 'MULTIPLICAND AT THIS ADDRESS';
        CALL RITEOUT;

        HEXCON = '20';
        MDCON = 'BRA';
        CMTS = 'BRANCH BACK TO BEGINNING OF';
        CALL RITEOUT;

        HEXCON = 'F3';
        MDCON = ' ';
        CMTS = 'MULTIPLY ALGORITHM';
        CALL RITEOUT;

     /* INCREMENT TEMPORARY STORAGE INDEX */

        TMP_IDX = TMP_IDX + 1;
        RETURN;

END MTPLYCG;
```

```
FILE: RESTRCG  PLIOPT   A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 RESTRCG : PROCEDURE;

     DCL  HEXCON                           CHAR       (2)  EXT,
          MDCON                            CHAR       (9)  EXT,
          CMTS                             CHAR       (45) EXT;
     DCL  RITEOUT   ENTRY;

     /* THIS ROUTINE RESETS THE INDEX REGISTER TO THE TOP OF THE
          DATA TABLE AT ADDRESS 0000 */

          HEXCON = 'CE';
          MDCON = 'LDX';
          CMTS = 'RESET INDEX REGISTER TO';
          CALL RITEOUT;
          HEXCON = '00';
          MDCON = ' ';
          CMTS = 'TOP OF DATA TABLE STARTING';
          CALL RITEOUT;
          HEXCON = '00';
          MDCON = ' ';
          CMTS = 'AT ADDRESS 0000';
          CALL RITEOUT;
          RETURN;
 END RESTRCG;
 EQUALCG : PROCEDURE;

      DCL   1     TMP_AD                      (70)        EXT,
                  2     TMP_ADR               CHAR        (4);
      DCL   1     LIT_AD                      (500)       EXT,
                  2     LIT_ADR               CHAR        (4);
      DCL   1     IDN_AD                      (99)        EXT,
                  2     IDN_ADR               CHAR        (4);
      DCL   OP1                               CHAR        (3)   EXT,
            OP1_PTR                           FIXED       (4)   EXT,
            OP2                               CHAR        (3)   EXT,
            OP2_PTR                           FIXED       (4)   EXT,
            HEXADR                            CHAR        (4)   EXT,
            HEXCON                            CHAR        (2)   EXT,
            MDCON                             CHAR        (9)   EXT,
            CMTS                              CHAR        (45)  EXT;
      DCL   RITEOUT     ENTRY;

    /* THIS ROUTINE ASSIGNS VALUES TO VARIABLES */

    /* WRITE LINE TO LOAD ACCUMULATOR */

         HEXCON = '96';
         MDCON = 'LDAA';
         CMTS = 'LOAD ACCUMULATOR A DIRECT';
         CALL RITEOUT;

    /* FIND OPERAND 2 */

         IF OP2 = 'TMP' THEN
              DO;
                   HEXCON = SUBSTR (TMP_ADR(OP2_PTR), 3, 2);
                   MDCON = ' ';
                   CMTS = 'WITH TEMP VALUE AT THIS ADDRESS';
              END;
         IF OP2 = 'IDN' THEN
              DO;
                   HEXCON = SUBSTR (IDN_ADR(OP2_PTR), 3, 2);
                   MDCON = ' ';
                   CMTS = 'WITH IDENTIFIER VALUE AT THIS ADDRESS';
              END;
         IF OP2 = 'LIT' THEN
              DO;
                   HEXCON = SUBSTR (LIT_ADR(OP2_PTR), 3, 2);
                   MDCON = ' ';
                   CMTS = 'WITH CONSTANT AT THIS ADDRESS';
              END;
         CALL RITEOUT;

    /* WRITE LINE TO STORE ACCUMULATOR IN MEMORY */

         HEXCON = '97';
         MDCON = 'STAA';
         CMTS = 'STORE ACCUMULATOR A DIRECT';
         CALL RITEOUT;
```

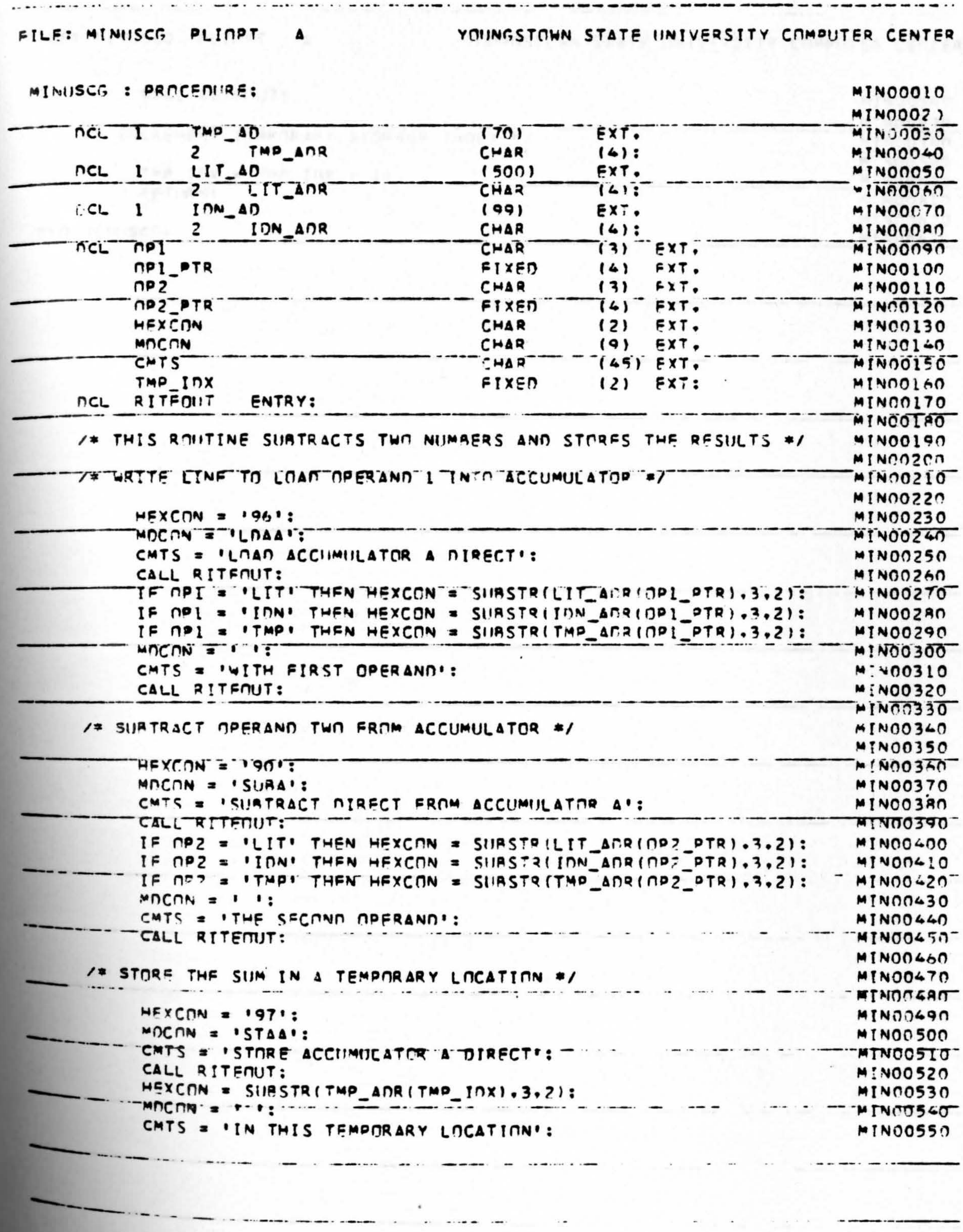


```
FILE: FCUALCG  PLIOPT   A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

                                                                         EQU00560
     /* FIND OPERAND 1 */                                                EQU00570
                                                                         EQU00580
          IF OP1 = 'IDN' THEN                                            EQU00590
               DO;                                                       EQU00600
                    HEXCON = SUBSTR (IDN_ADR(OP1_PTR), 3, 2);            EQU00610
                    MDCON = ' ';                                         EQU00620
                    CMTS = 'IN VARIABLE NAME AT THIS ADDRESS';           EQU00630
               END;                                                      EQU00640
          IF OP1 = 'TMP' THEN                                            EQU00650
               DO;                                                       EQU00660
                    HEXCON = SUBSTR (TMP_ADR(OP1_PTR), 3, 2);            EQU00670
                    MDCON = ' ';                                         EQU00680
                    CMTS = 'IN TEMPORARY STORAGE AT THIS ADDRESS';       EQU00690
               END;                                                      EQU00700
          CALL RITEOUT;                                                  EQU00710
                                                                         EQU00720
  END EQUALCG;                                                           EQU00730
```



```
FILE: PLUSCG    PLIOPT   A              YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

PLUSCG : PROCEDURE;

    DCL  1     TMP_AD                    (70)       EXT,
               2     TMP_ADR             CHAR       (4);
    DCL  1     LIT_AD                    (500)      EXT,
               2     LIT_ADR             CHAR       (4);
    DCL  1     IDN_AD                    (99)       EXT,
               2     IDN_ADR             CHAR       (4);
    DCL  OP1                             CHAR       (3)  EXT,
         OP1_PTR                         FIXED      (4)  EXT,
         OP2                             CHAR       (3)  EXT,
         OP2_PTR                         FIXED      (4)  EXT,
         HEXCON                          CHAR       (2)  EXT,
         MDCON                           CHAR       (9)  EXT,
         CMTS                            CHAR       (45) EXT,
         TMP_IDX                         FIXED      (2)  EXT;
    DCL  RITEOUT     ENTRY;

    /* THIS ROUTINE ADDS TWO NUMBERS AND STORES THE RESULTS */

    /* WRITE LINE TO LOAD OPERAND 1 INTO ACCUMULATOR */

         HEXCON = '96';
         MDCON = 'LDAA';
         CMTS = 'LOAD ACCUMULATOR A DIRECT';
         CALL RITEOUT;
         IF OP1 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP1_PTR),3,2);
         IF OP1 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP1_PTR),3,2);
         IF OP1 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP1_PTR),3,2);
         MDCON = ' ';
         CMTS = 'WITH FIRST OPERAND';
         CALL RITEOUT;

     /* ADD OPERAND TWO TO ACCUMULATOR */

         HEXCON = '9B';
         MDCON = 'ADDA';
         CMTS = 'ADD DIRECT TO ACCUMULATOR A';
         CALL RITEOUT;
         IF OP2 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP2_PTR),3,2);
         IF OP2 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP2_PTR),3,2);
         IF OP2 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP2_PTR),3,2);
         MDCON = ' ';
         CMTS = 'THE SECOND OPERAND';
         CALL RITEOUT;

     /* STORE THE SUM IN A TEMPORARY LOCATION */

         HEXCON = '97';
         MDCON = 'STAA';
         CMTS = 'STORE ACCUMULATOR A DIRECT';
         CALL RITEOUT;
         HEXCON = SUBSTR(TMP_ADR(TMP_IDX),3,2);
         MDCON = ' ';
         CMTS = 'IN THIS TEMPORARY LOCATION';
```

```
        CALL RITEOUT:                                                   PLU00560
                                                                        PLU00570
     /* INCREMENT TEMPORARY STORAGE INDEX */                            PLU00580
                                                                        PLU00590
        TMP_IDX = TMP_IDX + 1:                                          PLU00600
        RETURN:                                                         PLU00610
                                                                        PLU00620
 END PLUSCG:                                                            PLU00630
```



FILE: MINUSCG PLIOPT A   YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

```
MINUSCG : PROCEDURE;

       DCL  1   TMP_AD                       (70)        EXT,
                2    TMP_ADR                 CHAR        (4);
       DCL  1   LIT_AD                       (500)       EXT,
                2    LIT_ADR                 CHAR        (4);
       DCL  1   IDN_AD                       (99)        EXT,
                2    IDN_ADR                 CHAR        (4);
       DCL  OP1                              CHAR        (3)  EXT,
            OP1_PTR                          FIXED       (4)  EXT,
            OP2                              CHAR        (3)  EXT,
            OP2_PTR                          FIXED       (4)  EXT,
            HEXCON                           CHAR        (2)  EXT,
            MOCON                            CHAR        (9)  EXT,
            CMTS                             CHAR        (45) EXT,
            TMP_IDX                          FIXED       (2)  EXT;
       DCL  RITEOUT    ENTRY;

    /* THIS ROUTINE SUBTRACTS TWO NUMBERS AND STORES THE RESULTS */

    /* WRITE LINE TO LOAD OPERAND 1 INTO ACCUMULATOR */

            HEXCON = '96';
            MOCON = 'LDAA';
            CMTS = 'LOAD ACCUMULATOR A DIRECT';
            CALL RITEOUT;
            IF OP1 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP1_PTR),3,2);
            IF OP1 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP1_PTR),3,2);
            IF OP1 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP1_PTR),3,2);
            MOCON = ' ';
            CMTS = 'WITH FIRST OPERAND';
            CALL RITEOUT;

    /* SUBTRACT OPERAND TWO FROM ACCUMULATOR */

            HEXCON = '90';
            MOCON = 'SUBA';
            CMTS = 'SUBTRACT DIRECT FROM ACCUMULATOR A';
            CALL RITEOUT;
            IF OP2 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP2_PTR),3,2);
            IF OP2 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP2_PTR),3,2);
            IF OP2 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP2_PTR),3,2);
            MOCON = ' ';
            CMTS = 'THE SECOND OPERAND';
            CALL RITEOUT;

    /* STORE THE SUM IN A TEMPORARY LOCATION */

            HEXCON = '97';
            MOCON = 'STAA';
            CMTS = 'STORE ACCUMULATOR A DIRECT';
            CALL RITEOUT;
            HEXCON = SUBSTR(TMP_ADR(TMP_IDX),3,2);
            MOCON = ' ';
            CMTS = 'IN THIS TEMPORARY LOCATION';
       CALL RITEOUT;                                             MIN00560
                                                                 MIN00570
    /* INCREMENT TEMPORARY STORAGE INDEX */                      MIN00580
                                                                 MIN00590
       TMP_IDX = TMP_IDX + 1;                                    MIN00600
       RETURN;                                                   MIN00610
                                                                 MIN00620
  END MINUSCG;                                                   MIN00630
```

```
FILE: DIVIDCG  PLIOPT    A                 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

DIVIDCG : PROCEDURE;

     DCL  1   TMP_AD                     (70)      EXT,
               2   TMP_ADR               CHAR      (4);
     DCL  1   LIT_AD                     (500)     EXT,
               2  LIT_ADR                CHAR      (4);
     DCL  1   IDN_AD                     (99)      EXT,
               2       IDN_ADR           CHAR      (4);
     DCL  OP1                            CHAR      (3)   EXT,
          OP1_PTR                        FIXED     (4)   EXT,
          OP2                            CHAR      (3)   EXT,
          OP2_PTR                        FIXED     (4)   EXT,
          HEXCON                         CHAR      (2)   EXT,
          MDCON                          CHAR      (9)   EXT,
          CMTS                           CHAR      (45)  EXT,
          TMP_IDX                        FIXED     (2)   EXT;
     DCL  RITEOUT     ENTRY;

     /* THIS ROUTINE PERFORMS DIVISION OF TWO NUMBERS */

          HEXCON = 'D6';
          MDCON = 'LDAB';
          CMTS = 'LOAD THE B ACCUMULATOR WITH THE';
          CALL RITEOUT;

          IF OP1 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP1_PTR),3,2);
          IF OP1 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP1_PTR),3,2);
          IF OP1 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP1_PTR),3,2);
          MDCON = ' ';
          CMTS = 'DIVIDEND FROM THIS LOCATION';
          CALL RITEOUT;

          HEXCON = '4F';
          MDCON = 'CLRA';
          CMTS = 'CLEAR ACCUMULATOR A';
          CALL RITEOUT;

          HEXCON = '97';
          MDCON = 'STAA';
          CMTS = 'STORE THE QUOTIENT IN';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(TMP_IDX),3,2);
          MDCON = ' ';
          CMTS = 'THIS TEMPORARY LOCATION';
          CALL RITEOUT;

          HEXCON = '17';
          MDCON = 'TBA';
          CMTS = 'TRANSFER ACCUMULATOR B TO A';
          CALL RITEOUT;

          HEXCON = '90';
          MDCON = 'SUBA';
          CMTS = 'SUBTRACT FROM THE DIVIDEND THE';
```

```
      CALL RITEOUT;

      IF OP2 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP2_PTR),3,2);
      IF OP2 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP2_PTR),3,2);
      IF OP2 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP2_PTR),3,2);
      MDCON = ' ';
      CMTS = 'DIVISOR AT THIS ADDRESS';
      CALL RITEOUT;

      HEXCON = '2B';
      MDCON = 'BMI';
      CMTS = 'IF THE DIFFERENCE IS NEGATIVE,';
      CALL RITEOUT;

      HEXCON = '06';
      MDCON = ' ';
      CMTS = 'BRANCH TO THE NEXT ALGORITHM';
      CALL RITEOUT;

      HEXCON = '16';
      MDCON = 'TAB';
      CMTS = 'OTHERWISE, TRANSFER ACCUMULATOR A TO B';
      CALL RITEOUT;

      HEXCON = '96';
      MDCON = 'LDAA';
      CMTS = 'LOAD THE ACCUMULATOR WITH THE';
      CALL RITEOUT;

      HEXCON = SUBSTR(TMP_ADR(TMP_IDX),3,2);
      MDCON = ' ';
      CMTS = 'QUOTIENT STORED IN THIS TEMPORARY LOCATION';
      CALL RITEOUT;

      HEXCON = '4C';
      MDCON = 'INCA';
      CMTS = 'INCREMENT THE QUOTIENT';
      CALL RITEOUT;

      HEXCON = '20';
      MDCON = 'BRA';
      CMTS = 'BRANCH BACK TO BEGINNING OF';
      CALL RITEOUT;

      HEXCON = 'F3';
      MDCON = ' ';
      CMTS = 'DIVIDE ALGORITHM';
      CALL RITEOUT;

   /* INCREMENT TEMPORARY STORAGE INDEX */

      TMP_IDX = TMP_IDX + 1;
      RETURN;

END DIVIDCG;
```

```
EXPCG : PROCEDURE;

       DCL  1    TMP_AD                          (70)       EXT.
                 2    TMP_ADR                    CHAR       (4);
       DCL  1    LIT_AD                          (500)      EXT.
                 2    LIT_ADR                    CHAR       (4);
       DCL 1     IDN_AD                          (99)       EXT.
                 2    IDN_ADR                    CHAR       (4);
       DCL  OP1                                  CHAR       (3)   EXT.
            OP1_PTR                              FIXED      (4)   EXT.
            OP2                                  CHAR       (3)   EXT.
            OP2_PTR                              FIXED      (4)   EXT.
            HEXCON                               CHAR       (2)   EXT.
            MDCON                                CHAR       (9)   EXT.
            CMTS                                 CHAR       (45)  EXT.
            HEXADR                               CHAR       (4)   EXT.
            TMP_IDX                              FIXED      (2)   EXT.
            SCRATCH1                             CHAR       (4),
            SCRATCH2                             CHAR       (4);
       DCL  RITEOUT    ENTRY;

       /* THIS ROUTINE RAISES A NUMBER TO A POWER */

            HEXCON = '20';
            MDCON = 'BRA';
            CMTS = 'CREATE A SCRATCHPAD';
            CALL RITEOUT;

            HEXCON = '01';
            MDCON = ' ';
            CMTS = 'LOCATION FOR';
            CALL RITEOUT;

            HEXCON = 'XX';
            MDCON = ' ';
            CMTS = 'EXPONENT';
            CALL RITEOUT;
            SCRATCH1 = HEXADR;

            HEXCON = '86';
            MDCON = 'LDAA';
            CMTS = 'SET RESULT OF';
            CALL RITEOUT;

            HEXCON = '01';
            MDCON = ' ';
            CMTS = 'EXPONENTIATION';
            CALL RITEOUT;

            HEXCON = '97';
            MDCON = 'STAA';
            CMTS = 'TO';
            CALL RITEOUT;

            HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 3, 2);
```

```
        MOCON = ' ';
        CMTS = 'ONE';
        CALL RITEOUT;

        HEXCON = '96';
        MOCON = 'LDAA';
        CMTS = 'LOAD VALUE OF';
        CALL RITEOUT;

        IF OP2 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP2_PTR), 3, 2);
        IF OP2 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP2_PTR), 3, 2);
        IF OP2 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP2_PTR), 3, 2);
        MOCON = ' ';
        CMTS = 'EXPONENT';
        CALL RITEOUT;

        HEXCON = 'B7';
        MOCON = 'STAA';
        CMTS = 'INTO';
        CALL RITEOUT;

        HEXCON = SUBSTR(SCRATCH1, 1, 2);
        MOCON = ' ';
        CMTS = 'SCRATCHPAD';
        CALL RITEOUT;

        HEXCON = SUBSTR(SCRATCH1, 3, 2);
        MOCON = ' ';
        CMTS = 'LOCATION';
        CALL RITEOUT;

        HEXCON = '27';
        MOCON = 'BEQ';
        CMTS = 'IF EXPONENT IS ZERO';
        CALL RITEOUT;

        HEXCON = '1F';
        MOCON = ' ';
        CMTS = 'GO ON TO NEXT ALGORITHM';
        CALL RITEOUT;

        HEXCON = '20';
        MOCON = 'BRA';
        CMTS = 'CREATE A SCRATCHPAD';
        CALL RITEOUT;

        HEXCON = '01';
        MOCON = ' ';
        CMTS = 'LOCATION FOR';
        CALL RITEOUT;

        HEXCON = 'XX';
        MOCON = ' ';
        CMTS = 'PRODUCT';
        CALL RITEOUT;
```

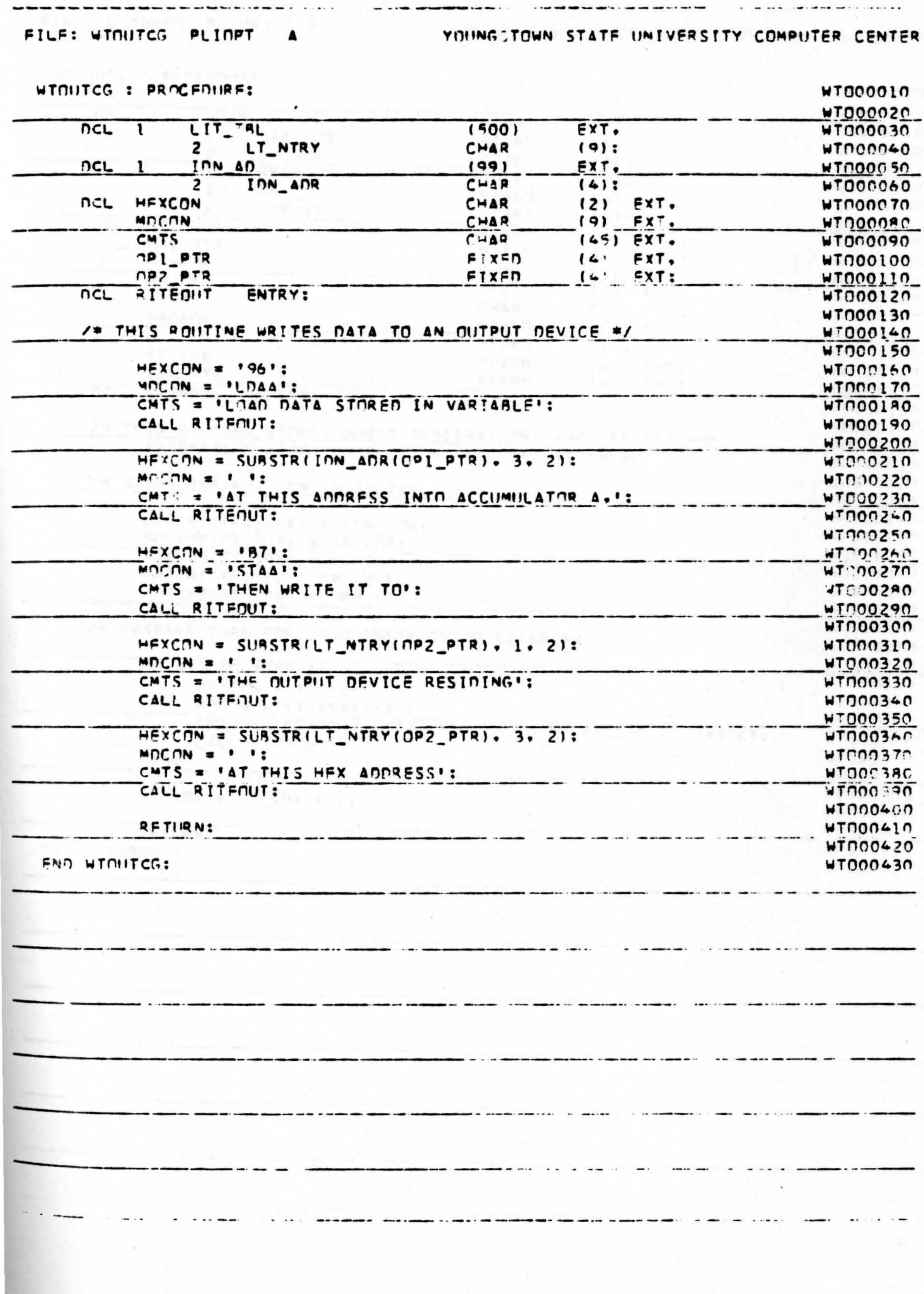
```
         SCRATCH2 = HEXADR:                                             EXP01110
                                                                        EXP01120
         HEXCON = '06':                                                 EXP01130
         MOCON = 'LDAB':                                                EXP01140
         CMTS = 'LOAD ACCUMULATOR B WITH THE':                          EXP01150
         CALL RITEOUT:                                                  EXP01160
                                                                        EXP01170
         IF OP1 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP1_PTR), 3, 2):   EXP01180
         IF OP1 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP1_PTR), 3, 2):   EXP01190
         IF OP1 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP1_PTR), 3, 2):   EXP01200
         MOCON = ' ':                                                   EXP01210
         CMTS = 'MULTIPLIER FROM THIS LOCATION':                        EXP01220
         CALL RITEOUT:                                                  EXP01230
                                                                        EXP01240
         HEXCON = '4F':                                                 EXP01250
         MOCON = 'CLRA':                                                EXP01260
         CMTS = 'CLEAR ACCUMULATOR A':                                  EXP01270
         CALL RITEOUT:                                                  EXP01280
                                                                        EXP01290
         HEXCON = 'B7':                                                 EXP01300
         MOCON = 'STAA':                                                EXP01310
         CMTS = 'STORE THE PRODUCT IN':                                 EXP01320
         CALL RITEOUT:                                                  EXP01330
                                                                        EXP01340
         HEXCON = SUBSTR(SCRATCH2. 1. 2):                               EXP01350
         MOCON = ' ':                                                   EXP01360
         CMTS = 'THIS':                                                 EXP01370
         CALL RITEOUT:                                                  EXP01380
                                                                        EXP01390
         HEXCON = SUBSTR(SCRATCH2. 3. 2):                               EXP01400
         MOCON = ' ':                                                   EXP01410
         CMTS = 'LOCATION':                                             EXP01420
         CALL RITEOUT:                                                  EXP01430
                                                                        EXP01440
         HEXCON = '17':                                                 EXP01450
         MOCON = 'TBA':                                                 EXP01460
         CMTS = 'TRANSFER ACCUMULATOR B TO A':                          EXP01470
         CALL RITEOUT:                                                  EXP01480
                                                                        EXP01490
         HEXCON = '27':                                                 EXP01500
         MOCON = 'BEQ':                                                 EXP01510
         CMTS = 'IF THE MULTIPLIER IS ZERO, EXIT':                      EXP01520
         CALL RITEOUT:                                                  EXP01530
                                                                        EXP01540
         HEXCON = '09':                                                 EXP01550
         MOCON = ' ':                                                   EXP01560
         CMTS = 'THE MULTIPLIER LOOP':                                  EXP01570
         CALL RITEOUT:                                                  EXP01580
                                                                        EXP01590
         HEXCON = '4A':                                                 EXP01600
         MOCON = 'DECA':                                                EXP01610
         CMTS = 'OTHERWISE, DECREMENT THE MULTIPLIER':                  EXP01620
         CALL RITEOUT:                                                  EXP01630
                                                                        EXP01640
         HEXCON = '16':                                                 EXP01650
```

```
        MDCON = 'TAB';                                                  EXP01660
        CMTS = 'TRANSFER ACCUMULATOR A TO B';                           EXP01670
        CALL RITEOUT;                                                   EXP01680
                                                                        EXP01690
        HEXCON = 'B6';                                                  EXP01700
        MDCON = 'LDAA';                                                 EXP01710
        CMTS = 'LOAD ACCUMULATOR';                                      EXP01720
        CALL RITEOUT;                                                   EXP01730
                                                                        EXP01740
        HEXCON = SUBSTR(SCRATCH2, 1, 2);                                EXP01750
        MDCON = ' ';                                                    EXP01760
        CMTS = 'WITH PRODUCT STORED';                                   EXP01770
        CALL RITEOUT;                                                   EXP01780
                                                                        EXP01790
        HEXCON = SUBSTR(SCRATCH2, 3, 2);                                EXP01800
        MDCON = ' ';                                                    EXP01810
        CMTS = 'IN THIS LOCATION';                                      EXP01820
        CALL RITEOUT;                                                   EXP01830
                                                                        EXP01840
        HEXCON = '9B';                                                  EXP01850
        MDCON = 'ADDA';                                                 EXP01860
        CMTS = 'ADD TO THE PRODUCT THE';                                EXP01870
        CALL RITEOUT;                                                   EXP01880
                                                                        EXP01890
        HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 3, 2);                        EXP01900
        MDCON = ' ';                                                    EXP01910
        CMTS = 'MULTIPLICAND AT THIS ADDRESS';                          EXP01920
        CALL RITEOUT;                                                   EXP01930
                                                                        EXP01940
        HEXCON = '20';                                                  EXP01950
        MDCON = 'BRA';                                                  EXP01960
        CMTS = 'BRANCH BACK TO BEGINNING';                              EXP01970
        CALL RITEOUT;                                                   EXP01980
                                                                        EXP01990
        HEXCON = 'F1';                                                  EXP02000
        MDCON = ' ';                                                    EXP02010
        CMTS = 'OF MULTIPLY ALGORITHM';                                 EXP02020
        CALL RITEOUT;                                                   EXP02030
                                                                        EXP02040
        HEXCON = 'B6';                                                  EXP02050
        MDCON ='LDAA';                                                  EXP02060
        CMTS = 'LOAD ACCUMULATOR A';                                    EXP02070
        CALL RITEOUT;                                                   EXP02080
                                                                        EXP02090
        HEXCON = SUBSTR(SCRATCH2, 1, 2);                                EXP02100
        MDCON = ' ';                                                    EXP02110
        CMTS = 'WITH PRODUCT STORED';                                   EXP02120
        CALL RITEOUT;                                                   EXP02130
                                                                        EXP02140
        HEXCON = SUBSTR(SCRATCH2, 3, 2);                                EXP02150
        MDCON = ' ';                                                    EXP02160
        CMTS = 'IN THIS LOCATION';                                      EXP02170
        CALL RITEOUT;                                                   EXP02180
                                                                        EXP02190
        HEXCON = '97';                                                  EXP02200
        MDCON = 'STAA';
        CMTS = 'STORE RESULT IN THIS';
        CALL RITEOUT:

        HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 3, 2):
        MDCON = ' ':
        CMTS = 'TEMPORARY STORAGE LOCATION';
        CALL RITEOUT:

        HEXCON = '7A':
        MDCON = 'DEC':
        CMTS = 'DECREMENT':
        CALL RITEOUT:

        HEXCON = SUBSTR(SCRATCH1, 1, 2):
        MDCON = ' ':
        CMTS = 'EXPONENT':
        CALL RITEOUT:

        HEXCON = SUBSTR(SCRATCH1, 3, 2):
        MDCON = ' ':
        CMTS = 'LOCATION':
        CALL RITEOUT:

        HEXCON = '20':
        MDCON = 'BRA':
        CMTS = 'BRANCH BACK TO BEGINNING';
        CALL RITEOUT:

        HEXCON = 'DF':
        MDCON = ' ':
        CMTS = 'OF EXPONENTIATION ALGORITHM';
        CALL RITEOUT:

     /* INCREMENT TEMPORARY STORAGE INDEX */

        TMP_IDX = TMP_IDX + 1:
        RETURN:

 END EXPCG:
```

```
FILE: SQRCG     PLIOPT   A                 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 SQRCG : PROCEDURE;                                                             SQR00010
                                                                                SQR00020
       DCL  1     TMP_AD                      (70)        EXT,                  SQR00030
                  2    TMP_ADR                CHAR        (4);                  SQR00040
       DCL  1     LIT_AD                      (500)       EXT,                  SQR00050
                  2    LIT_ADR                CHAR        (4);                  SQR00060
       DCL 1      IDN_AD                      (99)        EXT,                  SQR00070
                  2    IDN_ADR                CHAR        (4);                  SQR00080
       DCL   OP1                              CHAR        (3)   EXT,            SQR00090
             OP1_PTR                          FIXED       (4)   EXT,            SQR00100
             OP2                              CHAR        (3)   EXT,            SQR00110
             OP2_PTR                          FIXED       (4)   EXT,            SQR00120
             HEXCON                           CHAR        (2)   EXT,            SQR00130
             MDCON                            CHAR        (9)   EXT,            SQR00140
             CMTS                             CHAR        (45)  EXT,            SQR00150
             HEXADR                           CHAR        (4)   EXT,            SQR00160
             TMP_IDX                          FIXED       (2)   EXT,            SQR00170
             SCRATCH                          CHAR        (4);                  SQR00180
       DCL   RITEOUT     ENTRY;                                                 SQR00190
                                                                                SQR00200
       /* THIS ROUTINE COMPUTES SQUARE ROOT TO THE NEAREST WHOLE NUMBER */      SQR00210
                                                                                SQR00220
             HEXCON = '7E';                                                     SQR00230
             MDCON = 'CLR';                                                     SQR00240
             CMTS = 'INITIALIZE';                                               SQR00250
             CALL RITEOUT;                                                      SQR00260
                                                                                SQR00270
             HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 1, 2);                           SQR00280
             MDCON = ' ';                                                       SQR00290
             CMTS = 'ANSWER';                                                   SQR00300
             CALL RITEOUT;                                                      SQR00310
                                                                                SQR00320
             HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 3, 2);                           SQR00330
             MDCON = ' ';                                                       SQR00340
             CMTS = 'AT ZERO';                                                  SQR00350
             CALL RITEOUT;                                                      SQR00360
                                                                                SQR00370
             HEXCON = '20';                                                     SQR00380
             MDCON = 'BRA';                                                     SQR00390
             CMTS = 'CREATE A SCRATCHPAD';                                      SQR00400
             CALL RITEOUT;                                                      SQR00410
                                                                                SQR00420
             HEXCON = '01';                                                     SQR00430
             MDCON = ' ';                                                       SQR00440
             CMTS = 'LOCATION FOR';                                             SQR00450
             CALL RITEOUT;                                                      SQR00460
                                                                                SQR00470
             HEXCON = 'XX';                                                     SQR00480
             MDCON = ' ';                                                       SQR00490
             CMTS = 'SQUARE OF ANSWER';                                         SQR00500
             CALL RITEOUT;                                                      SQR00510
             SCRATCH = HEXADR;                                                  SQR00520
                                                                                SQR00530
             HEXCON = 'D6';                                                     SQR00540
             MDCON = 'LDAB';                                                    SQR00550
       CMTS = 'LOAD THE B ACCUMULATOR WITH';
       CALL RITEOUT;

       HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 3, 2);
       MDCON = ' ';
       CMTS = 'THE MULTIPLIER FROM THIS LOCATION';
       CALL RITEOUT;

       HEXCON = '4F';
       MDCON = 'CLRA';
       CMTS = 'CLEAR ACCUMULATOR A';
       CALL RITEOUT;

       HEXCON = 'B7';
       MDCON = 'STAA';
       CMTS = 'STORE THE PRODUCT IN';
       CALL RITEOUT;

       HEXCON = SUBSTR(SCRATCH, 1, 2);
       MDCON = ' ';
       CMTS = 'THIS SCRATCHPAD';
       CALL RITEOUT;

       HEXCON = SUBSTR(SCRATCH, 3, 2);
       MDCON = ' ';
       CMTS = 'LOCATION';
       CALL RITEOUT;

       HEXCON = '17';
       MDCON = 'TBA';
       CMTS = 'TRANSFER ACCUMULATOR B TO A';
       CALL RITEOUT;

       HEXCON = '27';
       MDCON = 'BEQ';
       CMTS = 'IF THE MULTIPLIER IS ZERO,';
       CALL RITEOUT;

       HEXCON = '09';
       MDCON = ' ';
       CMTS = 'CONTINUE WITH ALGORITHM';
       CALL RITEOUT;

       HEXCON = '4A';
       MDCON = 'DECA';
       CMTS = 'OTHERWISE, DECREMENT MULTIPLIER';
       CALL RITEOUT;

       HEXCON = '16';
       MDCON = 'TAB';
       CMTS = 'TRANSFER ACCUMULATOR A TO B';
       CALL RITEOUT;

       HEXCON = 'B6';
       MDCON = 'LDAA';
```

```
          CMTS = 'LOAD THE ACCUMULATOR WITH';
          CALL RITEOUT;

          HEXCON = SUBSTR(SCRATCH, 1, 2);
          MOCON = ' ';
          CMTS = 'THE PRODUCT STORED IN';
          CALL RITEOUT;

          HEXCON = SUBSTR(SCRATCH, 3, 2);
          MOCON = ' ';
          CMTS = 'THIS SCRATCHPAD LOCATION';
          CALL RITEOUT;

          HEXCON = '9B';
          MOCON = 'ADDA';
          CMTS = 'ADD TO THE PRODUCT THE';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 3, 2);
          MOCON = ' ';
          CMTS = 'MULTIPLICAND AT THIS ADDRESS';
          CALL RITEOUT;

          HEXCON = '20';
          MOCON = 'BRA';
          CMTS ='BRANCH BACK TO BEGINNING';
          CALL RITEOUT;

          HEXCON = 'F1';
          MOCON = ' ';
          CMTS = 'OF MULTIPLY SEQUENCE';
          CALL RITEOUT;

          HEXCON = 'B6';
          MOCON = 'LDAA';
          CMTS = 'LOAD ANSWER';
          CALL RITEOUT;

          HEXCON = SUBSTR(SCRATCH, 1, 2);
          MOCON = '';
          CMTS = 'SQUARED INTO';
          CALL RITEOUT;

          HEXCON = SUBSTR(SCRATCH, 3, 2);
          MOCON = ' ';
          CMTS = 'ACCUMULATOR A';
          CALL RITEOUT;

          HEXCON = '90';
          MOCON = 'SUBA';
          CMTS = 'SUBTRACT THE ORIGINAL';
          CALL RITEOUT;

          IF OP1 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP1_PTR), 3, 2);
          IF OP1 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP1_PTR), 3, 2);
          IF OP1 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP1_PTR), 3, 2);
          MDCON = ' ';
          CMTS = 'NUMBER FROM ACCUMULATOR A';
          CALL RITEOUT;

          HEXCON = '2C';
          MDCON = 'BGE';
          CMTS = 'IF ANSWER SQUARED IS GREATER THAN';
          CALL RITEOUT;

          HEXCON = '05';
          MDCON = ' ';
          CMTS = 'OR EQUAL TO ARGUMENT, GO TO NEXT ALGORITHM';
          CALL RITEOUT;

          HEXCON = '7C';
          MDCON = 'INC';
          CMTS = 'OTHERWISE, INCREMENT THE';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 1, 2);
          MDCON = ' ';
          CMTS = 'ANSWER AND';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(TMP_IDX), 3, 2);
          MDCON = ' ';
          CMTS = 'RETURN TO';
          CALL RITEOUT;

          HEXCON = '20';
          MDCON = 'BRA';
          CMTS = 'SQUARING';
          CALL RITEOUT;

          HEXCON = 'E2';
          MDCON = ' ';
          CMTS = 'ROUTINE';
          CALL RITEOUT;

      /* INCREMENT TEMPORARY STORAGE INDEX */

          TMP_IDX = TMP_IDX + 1;

          RETURN;

  END SORCG;
```

```
FILE: RDINCG    PLIOPT    A                 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 RDINCG : PROCEDURE;

     DCL   1      LIT_TBL                        (500)      EXT,
                  2     LT_NTRY                  CHAR          (9);
     DCL   1      IDN_AD                         (99)       EXT,
                  2     IDN_ADR                  CHAR          (4);
     DCL   HEXCON                                CHAR          (2)   EXT.
           MDCON                                 CHAR          (9)   EXT.
           CMTS                                  CHAR          (45)  EXT.
           OP1_PTR                               FIXED         (4)   EXT.
           OP2_PTR                               FIXED         (4)   EXT;
     DCL   RITEOUT    ENTRY;

     /* THIS ROUTINE READS DATA FROM AN INPUT DEVICE */

           HEXCON = 'B6';
           MDCON = 'LDAA';
           CMTS = 'LOAD DATA FROM INPUT';
           CALL RITEOUT;

           HEXCON = SUBSTR(LT_NTRY(OP2_PTR), 1, 2);
           MDCON = ' ';
           CMTS = 'DEVICE AT THIS ADDRESS';
           CALL RITEOUT;

           HEXCON = SUBSTR(LT_NTRY(OP2_PTR), 3, 2);
           MDCON = ' ';
           CMTS = 'INTO ACCUMULATOR A, THEN';
           CALL RITEOUT;

           HEXCON = '97';
           MDCON = 'STAA';
           CMTS = 'STORE DATA IN VARIABLE';
           CALL RITEOUT;

           HEXCON = SUBSTR(IDN_ADR(OP1_PTR), 3, 2);
           MDCON = ' ';
           CMTS = 'NAME AT THIS ADDRESS';
           CALL RITEOUT;

           RETURN;

 END RDINCG;
```



```
FILE: WTOUTCG   PLIOPT    A              YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 WTOUTCG : PROCEDURE;

      DCL  1     LIT_TBL                      (500)      EXT,
                 2    LT_NTRY                 CHAR       (9);
      DCL  1     IDN_AD                       (99)       EXT,
                 2    IDN_ADR                 CHAR       (4);
      DCL  HEXCON                             CHAR       (2)   EXT,
           MDCON                              CHAR       (9)   EXT,
           CMTS                               CHAR       (45)  EXT,
           OP1_PTR                            FIXED      (4)   EXT,
           OP2_PTR                            FIXED      (4)   EXT;
      DCL  RITEOUT     ENTRY;

      /* THIS ROUTINE WRITES DATA TO AN OUTPUT DEVICE */

           HEXCON = '96';
           MDCON = 'LDAA';
           CMTS = 'LOAD DATA STORED IN VARIABLE';
           CALL RITEOUT;

           HEXCON = SUBSTR(IDN_ADR(OP1_PTR), 3, 2);
           MDCON = ' ';
           CMTS = 'AT THIS ADDRESS INTO ACCUMULATOR A.';
           CALL RITEOUT;

           HEXCON = 'B7';
           MDCON = 'STAA';
           CMTS = 'THEN WRITE IT TO';
           CALL RITEOUT;

           HEXCON = SUBSTR(LT_NTRY(OP2_PTR), 1, 2);
           MDCON = ' ';
           CMTS = 'THE OUTPUT DEVICE RESIDING';
           CALL RITEOUT;

           HEXCON = SUBSTR(LT_NTRY(OP2_PTR), 3, 2);
           MDCON = ' ';
           CMTS = 'AT THIS HEX ADDRESS';
           CALL RITEOUT;

           RETURN;

 END WTOUTCG;
```

```
FILE: GOSUBCG  PLIOPT   A                    YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 GOSUBCG : PROCEDURE;

     DCL  1     GOTBL                       (100)      EXT.
                2     LAB_PTR               FIXED      (4),
                2     GO_ADR                FIXED      (5),
                2     BRANCH                CHAR       (5),
                2     DESTN                 FIXED      (4),
                2     OFFSET                FIXED      (4);
     DCL  OP1_PTR                           FIXED      (4)  EXT,
          OP2_PTR                           FIXED      (4)  EXT,
          HEXCON                            CHAR       (2)  EXT,
          MDCON                             CHAR       (9)  EXT,
          CMTS                              CHAR       (45) EXT,
          DECADR                            FIXED      (5)  EXT,
          J                                 FIXED      (1),
          GT_IDX                            FIXED      (4)  EXT,
          LIN_PTR                           FIXED      (4)  EXT;
     DCL  RITEOUT    ENTRY;

     /* THIS ROUTINE STORES GOSUB DESTINATIONS AND OFFSETS FOR
          PROCESSING AFTER COMPLETION OF COMPILATION */

     /* STORE INFORMATION IN GO TABLE */

          LAB_PTR(GT_IDX) = LIN_PTR;
          GO_ADR(GT_IDX) = DECADR;
          BRANCH(GT_IDX) = 'GOSUB';
          DESTN(GT_IDX) = OP1_PTR;
          OFFSET(GT_IDX) = OP2_PTR;

     /* RESERVE CORE FOR JUMP INSTRUCTION LATER */

          DO J=1 TO 3 BY 1;
               HEXCON = 'XX';
               MDCON = 'XXXXXXXXX';
               CMTS = 'BRANCH INSTRUCTION - TO BE FILLED IN LATER';
               CALL RITEOUT;
          END;

          GT_IDX = GT_IDX + 1;

          RETURN;
 END GOSUBCG;
 GOTOCG : PROCEDURE;

     DCL  1    GOTBL                      (100)     EXT,
               2      LAB_PTR              FIXED     (4),
               2      GO_ADR               FIXED     (5),
               2      BRANCH               CHAR      (5),
               2      DESTN                FIXED     (4),
               2      OFFSET               FIXED     (4);
     DCL  OP1_PTR                          FIXED     (4)   EXT,
          OP2_PTR                          FIXED     (4)   EXT,
          HEXCON                           CHAR      (2)   EXT,
          MDCON                            CHAR      (9)   EXT,
          CMTS                             CHAR      (45)  EXT,
          DECADR                           FIXED     (5)   EXT,
          J                                FIXED     (1),
          GT_IDX                           FIXED     (4)   EXT,
          LIN_PTR                          FIXED     (4)   EXT;
     DCL  RITEOUT   ENTRY;

     /* THIS ROUTINE STORES GOTO DESTINATIONS AND OFFSETS FOR
          PROCESSING AFTER COMPLETION OF COMPILATION */

     /* STORE INFORMATION IN GO TABLE */

          LAB_PTR(GT_IDX) = LIN_PTR;
          GO_ADR(GT_IDX) = DECADR;
          BRANCH(GT_IDX) = 'GOTO ';
          DESTN(GT_IDX) = OP1_PTR;
          OFFSET(GT_IDX) = OP2_PTR;

     /* RESERVE CORE FOR JUMP INSTRUCTION LATER */

          DO J=1 TO 3 BY 1;
                HEXCON = 'XX';
                MDCON = 'XXXXXXXXX';
                CMTS = 'BRANCH INSTRUCTION - TO BE FILLED IN LATER';
                CALL RITEOUT;
          END;

          GT_IDX = GT_IDX + 1;

          RETURN;
 END GOTOCG;
```

```
FILE: RTRNCG   PLIOPT   A                    YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 RTRNCG : PROCEDURE:

     DCL  HEXCON                           CHAR       (2)   EXT.
          MDCON                            CHAR       (9)   EXT.
          CMTS                             CHAR       (45)  EXT:
     DCL  RITEOUT    ENTRY:

          HEXCON = '39':
          MDCON = 'RTS':
          CMTS = 'RETURN FROM SUBROUTINE':
          CALL RITEOUT:

          RETURN:
 END RTRNCG:
```

```
 BRANCHG : PROCEDURE;

    DCL  1      LBL_AD                      (500)        EXT,
                2     LBL_PTR               FIXED        (4),
                2     LBL_ADR               CHAR         (4);
    DCL  1      GOTBL                       (100)        EXT,
                2     LAB_PTR               FIXED        (4),
                2     GO_ADR                FIXED        (5),
                2     BRANCH                CHAR         (5),
                2     DESTN                 FIXED        (4),
                2     OFFSET                FIXED        (4);
    DCL  HEXCON                             CHAR         (2)    EXT,
         MOCON                              CHAR         (9)    EXT,
         CMTS                               CHAR         (45)   EXT,
         DECADR                             FIXED        (5)    EXT,
         GT_IDX                             FIXED        (4)    EXT,
         DECDEST                            FIXED        (5)    EXT,
         NEXT_ADR                           CHAR         (2),
         DECADR_SAVE                        FIXED        (5),
         HEXADR                             CHAR         (4)    EXT,
         LBL_IDX                            FIXED        (4)    EXT;
    DCL  RITEOUT     ENTRY;
    DCL  DCHXCO      ENTRY;
    DCL  HXDCCON     ENTRY;

    /* THIS ROUTINE ASSIGNS DESTINATIONS TO BRANCH STATEMENTS */

       GT_IDX = 0;

    /* READ A GO TABLE ENTRY */

 B001:   GT_IDX = GT_IDX + 1;

    /* IF BRANCH ENTRY IS BLANK, RETURN TO MAIN */

       IF BRANCH(GT_IDX) = ' ' THEN RETURN;
       IF BRANCH(GT_IDX) = 'GOSUB' THEN
            DO;
                 DECADR = GO_ADR(GT_IDX);
                 HEXCON = 'BD';
                 MOCON = 'JSR';
                 CMTS = 'JUMP TO SUBROUTINE';
                 CALL RITEOUT;

                 LBL_IDX = 1;

                 DO WHILE (DESTN(GT_IDX)¬=LBL_PTR(LBL_IDX));
                      LBL_IDX = LBL_IDX + 1;
                 END;

                 HEXCON = SUBSTR(LBL_ADR(LBL_IDX), 1, 2);
                 MOCON = ' ';
                 CMTS = 'BEGINNING AT THIS';
                 CALL RITEOUT;
                 HEXCON = SUBSTR(LBL_ADR(LBL_IDX), 3, 2);
                MDCON = ' ';                                    BRA00560
                CMTS = 'MEMORY LOCATION';                       BRA00570
                CALL RITEOUT;                                   BRA00580
                GO TO B001;                                     BRA00590
            END;                                                BRA00600
                                                                BRA00610
        IF OFFSET(GT_IDX) = 0 THEN                              BRA00620
            DO;                                                 BRA00630
                DECADR = GO_ADR(GT_IDX);                        BRA00640
                HEXCON = '7E';                                  BRA00650
                MDCON = 'JMP';                                  BRA00660
                CMTS = 'JUMP TO';                               BRA00670
                CALL RITEOUT;                                   BRA00680
                                                                BRA00690
                LBL_IDX = 1;                                    BRA00700
                                                                BRA00710
                DO WHILE (DESTN(GT_IDX)¬=LBL_PTR(LBL_IDX));     BRA00720
                    LBL_IDX = LBL_IDX + 1;                      BRA00730
                END;                                            BRA00740
                                                                BRA00750
                HEXCON = SUBSTR(LBL_ADR(LBL_IDX), 1, 2);        BRA00760
                MDCON = ' ';                                    BRA00770
                CMTS = 'THIS';                                  BRA00780
                CALL RITEOUT;                                   BRA00790
                HEXCON = SUBSTR(LBL_ADR(LBL_IDX), 3, 2);        BRA00800
                MDCON = ' ';                                    BRA00810
                CMTS = 'HEX ADDRESS';                           BRA00820
                CALL RITEOUT;                                   BRA00830
                GO TO B001;                                     BRA00840
            END;                                                BRA00850
                                                                BRA00860
        DECADR = GO_ADR(GT_IDX);                                BRA00870
        HEXCON = '7E';                                          BRA00880
        MDCON = 'JMP';                                          BRA00890
        CMTS = 'JUMP TO';                                       BRA00900
        CALL RITEOUT;                                           BRA00910
                                                                BRA00920
        LBL_IDX = 1;                                            BRA00930
                                                                BRA00940
        DO WHILE (DESTN(GT_IDX)¬=LBL_PTR(LBL_IDX));             BRA00950
            LBL_IDX = LBL_IDX + 1;                              BRA00960
        END;                                                    BRA00970
                                                                BRA00980
        DECADR_SAVE = DECADR;                                   BRA00990
        CALL HXDCCON;                                           BRA01000
        DECADR = DECDEST + 4;                                   BRA01010
        CALL DCHXCO;                                            BRA01020
        HEXCON = SUBSTR(HEXADR, 1, 2);                          BRA01030
        NEXT_ADR = SUBSTR(HEXADR, 3, 2);                        BRA01040
        DECADR = DECADR_SAVE;                                   BRA01050
        MDCON = ' ';                                            BRA01060
        CMTS = 'THIS';                                          BRA01070
        CALL RITEOUT;                                           BRA01080
        HEXCON = NEXT_ADR;                                      BRA01090
        MDCON = ' ';                                            BRA01100
```

```
FILE: BRANCHG  PLIOPT   A                  YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

        CMTS = 'HEX ADDRESS';                                                  BRA01110
        CALL RITEOUT;                                                          BRA01120
        GO TO B001;                                                            BRA01130
                                                                               BRA01140
 END BRANCHG;                                                                  BRA01150
```

```
FILE: NEQCG     PLIOPT    A                 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

NEQCG : PROCEDURE;

    DCL  1    TMP_AD                    (70)      EXT,
            2    TMP_ADR                CHAR      (4);
    DCL  OP1_PTR                        FIXED     (4)  EXT,
         OP2_PTR                        FIXED     (4)  EXT,
         HEXCON                         CHAR      (2)  EXT,
         MDCON                          CHAR      (9)  EXT,
         CMTS                           CHAR      (45) EXT;
    DCL  RITEOUT     ENTRY;

    /* THIS ROUTINE GENERATES CODE FOR CONDITIONAL BRANCHING */

    /* COMPARE ARGUMENTS */

         HEXCON = '96';
         MDCON = 'LDAA';
         CMTS = 'LOAD FIRST ARGUMENT';
         CALL RITEOUT;

         HEXCON = SUBSTR(TMP_ADR(OP1_PTR), 3, 2);
         MDCON = ' ';
         CMTS = 'INTO ACCUMULATOR A';
         CALL RITEOUT;

         HEXCON = '90';
         MDCON = 'SUBA';
         CMTS = 'SUBTRACT SECOND ARGUMENT';
         CALL RITEOUT;

         HEXCON = SUBSTR(TMP_ADR(OP2_PTR), 3, 2);
         MDCON = ' ';
         CMTS = 'FROM FIRST ARGUMENT';
         CALL RITEOUT;

    /* COMPARE AND BRANCH */

         HEXCON = '26';
         MDCON = 'BNE';
         CMTS = 'IF RESULT OF COMPARISON IS TRUE';
         CALL RITEOUT;

         HEXCON = '03';
         MDCON = ' ';
         CMTS = 'BRANCH TO NEXT STATEMENT';
         CALL RITEOUT;

         RETURN;

 END NEQCG;
```

```
FILE: CONEQCG  PLIOPT   A                    YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 CONEQCG : PROCEDURE;

     DCL  1     TMP_AD                          (70)        EXT,
                2     TMP_ADR                   CHAR        (4);
     DCL  OP1_PTR                               FIXED       (4)  EXT,
          OP2_PTR                               FIXED       (4)  EXT,
          HEXCON                                CHAR        (2)  EXT,
          MDCON                                 CHAR        (9)  EXT,
          CMTS                                  CHAR        (45) EXT;
     DCL  RITEOUT    ENTRY;

     /* THIS ROUTINE GENERATES CODE FOR CONDITIONAL BRANCHING */

     /* COMPARE ARGUMENTS */

          HEXCON = '96';
          MDCON = 'LDAA';
          CMTS = 'LOAD FIRST ARGUMENT';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(OP1_PTR), 3, 2);
          MDCON = ' ';
          CMTS = 'INTO ACCUMULATOR A';
          CALL RITEOUT;

          HEXCON = '90';
          MDCON = 'SUBA';
          CMTS = 'SUBTRACT SECOND ARGUMENT';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(OP2_PTR), 3, 2);
          MDCON = ' ';
          CMTS = 'FROM FIRST ARGUMENT';
          CALL RITEOUT;

     /* COMPARE AND BRANCH */

          HEXCON = '27';
          MDCON = 'BEQ';
          CMTS = 'IF RESULT OF COMPARISON IS TRUE';
          CALL RITEOUT;

          HEXCON = '03';
          MDCON = ' ';
          CMTS = 'BRANCH TO NEXT STATEMENT';
          CALL RITEOUT;

          RETURN;

 END CONEQCG;
```

```
FILE: LTCG      PLIOPT    A                 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 LTCG : PROCEDURE;

     DCL  1     TMP_AD                      (70)        EXT,
                  2    TMP_ADR              CHAR        (4);
     DCL  OP1_PTR                           FIXED       (4)  EXT,
          OP2_PTR                           FIXED       (4)  EXT,
          HEXCON                            CHAR        (2)  EXT,
          MDCON                             CHAR        (9)  EXT,
          CMTS                              CHAR        (45) EXT;
     DCL  RITEOUT    ENTRY;

     /* THIS ROUTINE GENERATES CODE FOR CONDITIONAL BRANCHING */

     /* COMPARE ARGUMENTS */

          HEXCON = '96';
          MDCON = 'LDAA';
          CMTS = 'LOAD FIRST ARGUMENT';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(OP1_PTR), 3, 2);
          MDCON = ' ';
          CMTS = 'INTO ACCUMULATOR A';
          CALL RITEOUT;

          HEXCON = '90';
          MDCON = 'SUBA';
          CMTS = 'SUBTRACT SECOND ARGUMENT';
          CALL RITEOUT;

          HEXCON = SUBSTR(TMP_ADR(OP2_PTR), 3, 2);
          MDCON = ' ';
          CMTS = 'FROM FIRST ARGUMENT';
          CALL RITEOUT;

     /* COMPARE AND BRANCH */

          HEXCON = '20';
          MDCON = 'BLT';
          CMTS = 'IF RESULT OF COMPARISON IS TRUE';
          CALL RITEOUT;

          HEXCON = '03';
          MDCON = ' ';
          CMTS = 'BRANCH TO NEXT STATEMENT';
          CALL RITEOUT;

          RETURN;

  END LTCG;
```

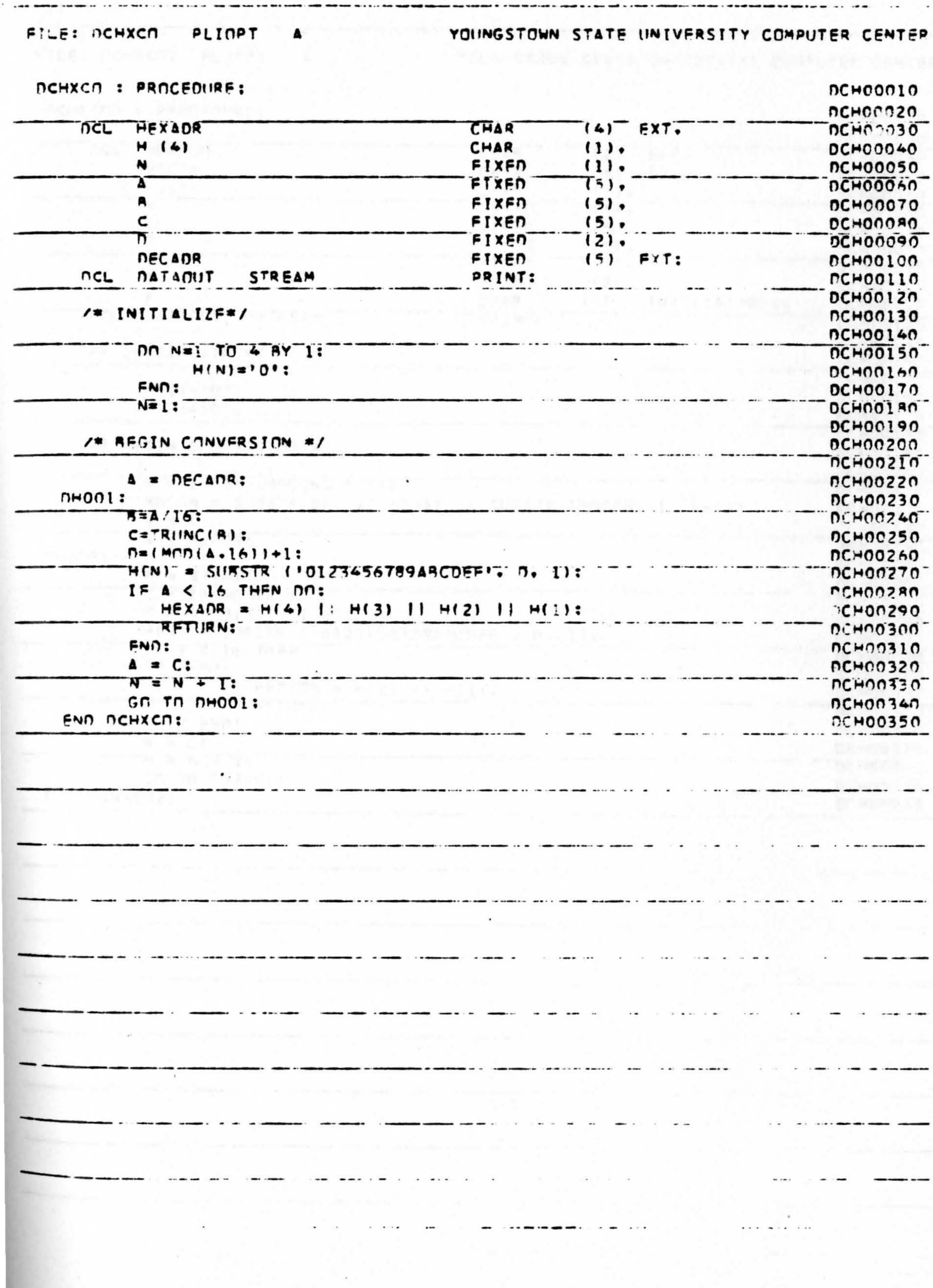
```
FILE: GTCG      PLIOPT    A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 GTCG : PROCEDURE;

      DCL  1     TMP_AD                        (70)        EXT,
                 2     TMP_ADR                 CHAR        (4);
      DCL  OP1_PTR                             FIXED       (4)   EXT,
           OP2_PTR                             FIXED       (4)   EXT,
           HEXCON                              CHAR        (2)   EXT,
           MDCON                               CHAR        (9)   EXT,
           CMTS                                CHAR        (45)  EXT;
      DCL  RITEOUT     ENTRY;

      /* THIS ROUTINE GENERATES CODE FOR CONDITIONAL BRANCHING */

      /* COMPARE ARGUMENTS */

           HEXCON = '96';
           MDCON = 'LDAA';
           CMTS = 'LOAD FIRST ARGUMENT';
           CALL RITEOUT;

           HEXCON = SUBSTR(TMP_ADR(OP1_PTR), 3, 2);
           MDCON = ' ';
           CMTS = 'INTO ACCUMULATOR A';
           CALL RITEOUT;

           HEXCON = '90';
           MDCON = 'SUBA';
           CMTS = 'SUBTRACT SECOND ARGUMENT';
           CALL RITEOUT;

           HEXCON = SUBSTR(TMP_ADR(OP2_PTR), 3, 2);
           MDCON = ' ';
           CMTS = 'FROM FIRST ARGUMENT';
           CALL RITEOUT;

      /* COMPARE AND BRANCH */

           HEXCON = '2E';
           MDCON = 'BGT';
           CMTS = 'IF RESULT OF COMPARISON IS TRUE';
           CALL RITEOUT;

           HEXCON = '03';
           MDCON = ' ';
           CMTS = 'BRANCH TO NEXT STATEMENT';
           CALL RITEOUT;

           RETURN;

 END GTCG;
```

```
FILE: GEOCG     PLIOPT    A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 GEOCG : PROCEDURE;                                                                GE000010
                                                                                   GE000020
     DCL  1      TMP_AD                          (70)        EXT,                  GE000030
                 2     TMP_ADR                   CHAR        (4);                  GE000040
     DCL  OP1_PTR                                FIXED       (4)   EXT,            GE000050
          OP2_PTR                                FIXED       (4)   EXT,            GE000060
          HEXCON                                 CHAR        (2)   EXT,            GE000070
          MDCON                                  CHAR        (9)   EXT,            GE000080
          CMTS                                   CHAR        (45)  EXT;            GE000090
     DCL  RITEOUT     ENTRY;                                                       GE000100
                                                                                   GE000110
     /* THIS ROUTINE GENERATES CODE FOR CONDITIONAL BRANCHING */                   GE000120
                                                                                   GE000130
     /* COMPARE ARGUMENTS */                                                       GE000140
                                                                                   GE000150
          HEXCON = '96';                                                           GE000160
          MDCON = 'LDAA';                                                          GE000170
          CMTS = 'LOAD FIRST ARGUMENT';                                            GE000180
          CALL RITEOUT;                                                            GE000190
                                                                                   GE000200
          HEXCON = SUBSTR(TMP_ADR(OP1_PTR), 3, 2);                                 GE000210
          MDCON = ' ';                                                             GE000220
          CMTS = 'INTO ACCUMULATOR A';                                             GE000230
          CALL RITEOUT;                                                            GE000240
                                                                                   GE000250
          HEXCON = '90';                                                           GE000260
          MDCON = 'SUBA';                                                          GE000270
          CMTS = 'SUBTRACT SECOND ARGUMENT';                                       GE000280
          CALL RITEOUT;                                                            GE000290
                                                                                   GE000300
          HEXCON = SUBSTR(TMP_ADR(OP2_PTR), 3, 2);                                 GE000310
          MDCON = ' ';                                                             GE000320
          CMTS = 'FROM FIRST ARGUMENT';                                            GE000330
          CALL RITEOUT;                                                            GE000340
                                                                                   GE000350
     /* COMPARE AND BRANCH */                                                      GE000360
                                                                                   GE000370
          HEXCON = '2C';                                                           GE000380
          MDCON = 'BGE';                                                           GE000390
          CMTS = 'IF RESULT OF COMPARISON IS TRUE';                                GE000400
          CALL RITEOUT;                                                            GE000410
                                                                                   GE000420
          HEXCON = '03';                                                           GE000430
          MDCON = ' ';                                                             GE000440
          CMTS = 'BRANCH TO NEXT STATEMENT';                                       GE000450
          CALL RITEOUT;                                                            GE000460
                                                                                   GE000470
          RETURN;                                                                  GE000480
                                                                                   GE000490
 END GEOCG;                                                                        GE000500
 LEOCG : PROCEDURE;

      DCL  1     TMP_AD                        (70)       EXT,
                 2    TMP_ADR                  CHAR       (4);
      DCL  1     LIT_AD                        (500)      EXT,
                 2    LIT_ADR                  CHAR       (4);
      DCL  1     IDN_AD                        (99)       EXT,
                 2    IDN_ADR                  CHAR       (4);
      DCL  OP1_PTR                             FIXED      (4)   EXT,
           OP2_PTR                             FIXED      (4)   EXT,
           OP1                                 CHAR       (3)   EXT,
           OP2                                 CHAR       (3)   EXT,
           HEXCON                              CHAR       (2)   EXT,
           MDCON                               CHAR       (9)   EXT,
           CMTS                                CHAR       (45)  EXT;
      DCL  RITEOUT    ENTRY;

      /* THIS ROUTINE GENERATES CODE FOR CONDITIONAL BRANCHING */

      /* COMPARE ARGUMENTS */

           HEXCON = '96';
           MDCON = 'LDAA';
           CMTS = 'LOAD FIRST ARGUMENT';
           CALL RITEOUT;

           IF OP1 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP1_PTR),3,2);
           IF OP1 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP1_PTR),3,2);
           IF OP1 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP1_PTR),3,2);
           MDCON = ' ';
           CMTS = 'INTO ACCUMULATOR A';
           CALL RITEOUT;

           HEXCON = '90';
           MDCON = 'SUBA';
           CMTS = 'SUBTRACT SECOND ARGUMENT ';
           CALL RITEOUT;

           IF OP2 = 'TMP' THEN HEXCON = SUBSTR(TMP_ADR(OP2_PTR),3,2);
           IF OP2 = 'IDN' THEN HEXCON = SUBSTR(IDN_ADR(OP2_PTR),3,2);
           IF OP2 = 'LIT' THEN HEXCON = SUBSTR(LIT_ADR(OP2_PTR),3,2);
           MDCON = ' ';
           CMTS = 'FROM FIRST ARGUMENT';
           CALL RITEOUT;

      /* COMPARE AND BRANCH */

           HEXCON = '2F';
           MDCON = 'BLE';
           CMTS = 'IF RESULT OF COMPARISON IS TRUE';
           CALL RITEOUT;

           HEXCON = '03';
           MDCON = ' ';
           CMTS = 'BRANCH TO NEXT STATEMENT';
```

```
         CALL RITFOUT:                                                  LE000560
                                                                        LE000570
         RETURN:                                                        LE000580
                                                                        LE000590
 END LEQCG:                                                             LE000600
```

```
FILE: HXDCCON  PLIOPT   A                 YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 HXDCCON : PROCEDURE;

      DCL   1    LBL_AD                          (500)       EXT,
                 2    LBL_PTR                    FIXED       (4),
                 2    LBL_ADR                    CHAR        (4);
      DCL   DECDEST                              FIXED       (5)  EXT,
            L                                    FIXED       (1),
            A                                    CHAR        (1),
            X                                    FIXED       (2),
            B                                    FIXED       (5),
            FACTOR                               FIXED       (2),
            LBL_IDX                              FIXED       (4)  EXT;

    /* THIS ROUTINE CONVERTS A HEX CHARACTER FIELD TO A DECIMAL */

         DECDEST = 0;
         DO L = 0 TO 3 BY 1;
              A = SUBSTR(LBL_ADR(LBL_IDX), 4-L, 1);
              X = VERIFY(A, '0123456789');
              IF X = 0 THEN FACTOR = A;
              IF A = 'A' THEN FACTOR = 10;
              IF A = 'B' THEN FACTOR = 11;
              IF A = 'C' THEN FACTOR = 12;
              IF A = 'D' THEN FACTOR = 13;
              IF A = 'E' THEN FACTOR = 14;
              IF A = 'F' THEN FACTOR = 15;
              B = FACTOR * (16 ** L);
              DECDEST = DECDEST + B;
         END;
         RETURN;

 END HXDCCON;
```

```
FILE: HEADING  PLIOPT   A              YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 HEADING : PROCEDURE;

      DCL DATAOUT    STREAM     PRINT;

         PUT FILE (DATAOUT) PAGE;
         PUT FILE (DATAOUT) SKIP EDIT ('MNEMONICS/') (X(25), A(10));
         PUT FILE (DATAOUT) SKIP EDIT ('HEX', 'HEX', 'DECIMAL')
            (X(5), A(3), X(7), A(3), X(7), A(7));
         PUT FILE (DATAOUT) SKIP EDIT ('ADDRESS', 'CONTENTS',
            'CONTENTS', 'COMMENTS') (X(5), A(7), X(3), A(8), X(2),
             A(8), X(7), A(8));
         PUT FILE (DATAOUT) SKIP;
         RETURN;
 END HEADING;
```



```
FILE: DCHXCO   PLIOPT   A              YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 DCHXCO : PROCEDURE;

    DCL  HEXADR                          CHAR        (4)   EXT,
         H (4)                           CHAR        (1),
         N                               FIXED       (1),
         A                               FIXED       (5),
         B                               FIXED       (5),
         C                               FIXED       (5),
         D                               FIXED       (2),
         DECADR                          FIXED       (5)   EXT;
    DCL  DATAOUT    STREAM               PRINT;

    /* INITIALIZE*/

         DO N=1 TO 4 BY 1;
              H(N)='0';
         END;
         N=1;

    /* BEGIN CONVERSION */

        A = DECADR;
  DH001:
        B=A/16;
        C=TRUNC(B);
        D=(MOD(A,16))+1;
        H(N) = SUBSTR ('0123456789ABCDEF', D, 1);
        IF A < 16 THEN DO;
           HEXADR = H(4) || H(3) || H(2) || H(1);
           RETURN;
        END;
        A = C;
        N = N + 1;
        GO TO DH001;
  END DCHXCO;
```

```
FILE: DCHXCO2  PLIOPT    A                  YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

 DCHXCO2 : PROCEDURE;

      DCL  HEXCON                            CHAR        (2)  EXT,
           MDCON                             CHAR        (9)  EXT,
           H (2)                             CHAR        (1),
           N                                 FIXED       (1),
           A                                 FIXED       (5),
           B                                 FIXED       (5),
           C                                 FIXED       (5),
           D                                 FIXED       (2),
           F                                 CHAR      (9)     INIT((9)'0'):
      DCL  DATAOUT   STREAM                  PRINT:

     /* INITIALIZF */

           H(1)='0':
           H(2)='0':

     /* BEGIN CONVFRSION */

           A = INDEX (MDCON, ' '):
           MDCON = SUBSTR (F, 1, 10-A) || SUBSTR (MDCON, 1, A-1):
           N = 1:
           A = MDCON:
 DH2001:
           B = A/16:
           C = TRUNC (B):
           D = (MOD(A,16)) + 1:
           H(N) = SUBSTR ('0123456789ABCDFF', D, 1):
           IF A < 16 THEN
                DO:
                      HEXCON = H(2) || H(1):
                      RETURN:
                END:
           A = C:
           N = N + 1:
           GO TO DH2001:
 END DCHXCO2:
```

```
FILE: RITEOUT  PLIOPT   A               YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

RITEOUT : PROCEDURE;

     DCL  HEXADR                              CHAR      (4)  EXT,
          HEXCON                              CHAR      (2)  EXT,
          MDCON                               CHAR      (9)  EXT,
          CMTS                                CHAR      (45) EXT,
          PGCNT                               FIXED     (2)  EXT,
          DECADR                              FIXED     (5)  EXT;
     DCL  DCHXCO                              ENTRY;
     DCL HEADING                              ENTRY;
     DCL DATAOUT      STREAM                  PRINT;

     /* CONVERT DECIMAL ADDRESS TO HEXADECIMAL ADDRESS */

          CALL DCHXCO;

     /* WRITE TO OUTPUT FILE */

          PUT FILE (DATAOUT) SKIP EDIT (HEXADR, HEXCON, MDCON, CMTS)
                (X(5), A(4), X(6), A(2), X(8), A(10), X(6), A(45));

     /* UPDATE MEMORY ADDRESS AND OUTPUT PAGE LINE COUNTER */

          PGCNT = PGCNT + 1;
          DECADR = DECADR + 1;
          IF PGCNT > 49 THEN
                DO;
                     CALL HEADING;
                     PGCNT = 0;
                END;
          RETURN;
 END RITEOUT;
```

APPENDIX C

Sample Programs

```
                          MNEMONICS/
  HEX         HEX         DECIMAL
  ADDRESS     CONTENTS    CONTENTS        COMMENTS

  0000        02          000000002       DATA
  0001        04          000000004       DATA
  0002        06          000000006       DATA
  0003        01          000000001       CONSTANT
  0004        63          000000099       CONSTANT
  0005        02          000000002       CONSTANT
  0006        03          000000003       CONSTANT
  0007        04          000000004       CONSTANT
  0008        06          000000006       CONSTANT
  0009        05          000000005       CONSTANT
  000A        A4          000004004       CONSTANT
  000B        07          000000007       CONSTANT
  000C        0D          000000013       CONSTANT
  000D        08          000000008       CONSTANT
  000E        0F          000000015       CONSTANT
  000F        09          000000009       CONSTANT
  0010        A6          000004006       CONSTANT
  0011        0A          000000010       CONSTANT
  0012        0B          000000011       CONSTANT
  0013        0C          000000012       CONSTANT
  0014        0E          000000014       CONSTANT
  0015        10          000000016       CONSTANT
  0016        11          000000017       CONSTANT
  0017        XX          XXXXXXXXX       K
  0018        XX          XXXXXXXXX       T
  0019        XX          XXXXXXXXX       A
  001A        XX          XXXXXXXXX       B
  001B        XX          XXXXXXXXX       C
  001C        XX          XXXXXXXXX       X
  001D        XX          XXXXXXXXX       TEMPORARY STORAGE LOCATION
  001E        XX          XXXXXXXXX       TEMPORARY STORAGE LOCATION
  001F        XX          XXXXXXXXX       TEMPORARY STORAGE LOCATION
  0020        XX          XXXXXXXXX       TEMPORARY STORAGE LOCATION
  0021        XX          XXXXXXXXX       TEMPORARY STORAGE LOCATION
  0100        CE          LDX             RESET INDEX REGISTER TO
  0101        00                          TOP OF DATA TABLE STARTING
  0102        00                          AT ADDRESS 0000
  0103        96          LDAA            LOAD ACCUMULATOR A DIRECT
  0104        03                          WITH CONSTANT AT THIS ADDRESS
  0105        97          STAA            STORE ACCUMULATOR A DIRECT
  0106        17                          IN VARIABLE NAME AT THIS ADDRESS
  0107        96          LDAA            LOAD FIRST ARGUMENT
  0108        17                          INTO ACCUMULATOR A
  0109        90          SUBA            SUBTRACT SECOND ARGUMENT
  010A        08                          FROM FIRST ARGUMENT
  010B        2F          BLE             IF RESULT OF COMPARISON IS TRUE
  010C        03                          BRANCH TO NEXT STATEMENT
  010D        XX          XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
  010E        XX          XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
  010F        XX          XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
```

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ DECIMAL CONTENTS	COMMENTS
0110	86	LDAA	LOAD DATA FROM INPUT
0111	40		DEVICE AT THIS ADDRESS
0112	04		INTO ACCUMULATOR A, THEN
0113	97	STAA	STORE DATA IN VARIABLE
0114	18		NAME AT THIS ADDRESS
0115	A6	LDAA, X	LOAD DATA INTO ACCUMULATOR A USING
0116	00		INDEXED ADDRESSING WITH ZERO OFFSET
0117	97	STAA	STORE DATA DIRECT INTO VARIABLE NAME
0118	19		AT THIS ZERO PAGE ADDRESS
0119	08	INX	INCREMENT INDEX REGISTER
011A	A6	LDAA, X	LOAD DATA INTO ACCUMULATOR A USING
011B	00		INDEXED ADDRESSING WITH ZERO OFFSET
011C	97	STAA	STORE DATA DIRECT INTO VARIABLE NAME
011D	1A		AT THIS ZERO PAGE ADDRESS
011E	08	INX	INCREMENT INDEX REGISTER
011F	A6	LDAA, X	LOAD DATA INTO ACCUMULATOR A USING
0120	00		INDEXED ADDRESSING WITH ZERO OFFSET
0121	97	STAA	STORE DATA DIRECT INTO VARIABLE NAME
0122	1B		AT THIS ZERO PAGE ADDRESS
0123	08	INX	INCREMENT INDEX REGISTER
0124	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0125	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0126	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0127	96	LDAA	LOAD ACCUMULATOR A DIRECT
0128	17		WITH IDENTIFIER VALUE AT THIS ADDRESS
0129	97	STAA	STORE ACCUMULATOR A DIRECT
012A	1D		IN TEMPORARY STORAGE AT THIS ADDRESS
012B	96	LDAA	LOAD ACCUMULATOR A DIRECT
012C	09		WITH CONSTANT AT THIS ADDRESS
012D	97	STAA	STORE ACCUMULATOR A DIRECT
012E	1E		IN TEMPORARY STORAGE AT THIS ADDRESS
012F	96	LDAA	LOAD FIRST ARGUMENT
0130	1D		INTO ACCUMULATOR A
0131	90	SUBA	SUBTRACT SECOND ARGUMENT
0132	1E		FROM FIRST ARGUMENT
0133	27	BEQ	IF RESULT OF COMPARISON IS TRUE
0134	03		BRANCH TO NEXT STATEMENT
0135	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0136	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0137	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0138	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0139	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
013A	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
013B	96	LDAA	LOAD DATA STORED IN VARIABLE
013C	1C		AT THIS ADDRESS INTO ACCUMULATOR A.
013D	B7	STAA	THEN WRITE IT TO
013E	40		THE OUTPUT DEVICE RESIDING
013F	06		AT THIS HEX ADDRESS
0140	CE	LDX	RESET INDEX REGISTER TO
0141	00		TOP OF DATA TABLE STARTING

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ DECIMAL CONTENTS	COMMENTS
0142	00		AT ADDRESS 0000
0143	96	LDAA	LOAD ACCUMULATOR A DIRECT
0144	17		WITH FIRST OPERAND
0145	9B	ADDA	ADD DIRECT TO ACCUMULATOR A
0146	03		THE SECOND OPERAND
0147	97	STAA	STORE ACCUMULATOR A DIRECT
0148	1D		IN THIS TEMPORARY LOCATION
0149	96	LDAA	LOAD ACCUMULATOR A DIRECT
014A	1D		WITH TEMP VALUE AT THIS ADDRESS
014B	97	STAA	STORE ACCUMULATOR A DIRECT
014C	17		IN VARIABLE NAME AT THIS ADDRESS
014D	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
014E	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
014F	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0150	3E	WAI	HALT EXECUTION (WAIT FOR INTERRUPT)
0151	96	LDAA	LOAD ACCUMULATOR A DIRECT
0152	19		WITH FIRST OPERAND
0153	9B	ADDA	ADD DIRECT TO ACCUMULATOR A
0154	1A		THE SECOND OPERAND
0155	97	STAA	STORE ACCUMULATOR A DIRECT
0156	1D		IN THIS TEMPORARY LOCATION
0157	7F	CLR	INITIALIZE
0158	00		ANSWER
0159	1F		AT ZERO
015A	20	BRA	CREATE A SCRATCHPAD
015B	01		LOCATION FOR
015C	XX		SQUARE OF ANSWER
015D	D6	LDAB	LOAD THE B ACCUMULATOR WITH
015E	1E		THE MULTIPLIER FROM THIS LOCATION
015F	4F	CLRA	CLEAR ACCUMULATOR A
0160	B7	STAA	STORE THE PRODUCT IN
0161	01		THIS SCRATCHPAD
0162	5C		LOCATION
0163	17	TBA	TRANSFER ACCUMULATOR B TO A
0164	27	BEQ	IF THE MULTIPLIER IS ZERO,
0165	09		CONTINUE WITH ALGORITHM
0166	4A	DECA	OTHERWISE, DECREMENT MULTIPLIER
0167	16	TAB	TRANSFER ACCUMULATOR A TO B
0168	B6	LDAA	LOAD THE ACCUMULATOR WITH
0169	01		THE PRODUCT STORED IN
016A	5C		THIS SCRATCHPAD LOCATION
016B	9B	ADDA	ADD TO THE PRODUCT THE
016C	1F		MULTIPLICAND AT THIS ADDRESS
016D	20	BRA	BRANCH BACK TO BEGINNING
016E	F1		OF MULTIPLY SEQUENCE
016F	B6	LDAA	LOAD ANSWER
0170	01		SQUARED INTO
0171	5C		ACCUMULATOR A
0172	90	SUBA	SUBTRACT THE ORIGINAL
0173	1D		NUMBER FROM ACCUMULATOR A

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ DECIMAL CONTENTS	COMMENTS
0174	2C	BGE	IF ANSWER SQUARED IS GREATER THAN
0175	05		OR EQUAL TO ARGUMENT, GO TO NEXT ALGORITHM
0176	7C	INC	OTHERWISE, INCREMENT THE
0177	00		ANSWER AND
0178	1F		RETURN TO
0179	20	BRA	SQUARING
017A	E2		ROUTINE
017B	96	LDAA	LOAD ACCUMULATOR A DIRECT
017C	17		WITH FIRST OPERAND
017D	90	SUBA	SUBTRACT DIRECT FROM ACCUMULATOR A
017E	18		THE SECOND OPERAND
017F	97	STAA	STORE ACCUMULATOR A DIRECT
0180	1F		IN THIS TEMPORARY LOCATION
0181	96	LDAA	LOAD ACCUMULATOR A DIRECT
0182	1E		WITH FIRST OPERAND
0183	9B	ADDA	ADD DIRECT TO ACCUMULATOR A
0184	1F		THE SECOND OPERAND
0185	97	STAA	STORE ACCUMULATOR A DIRECT
0186	20		IN THIS TEMPORARY LOCATION
0187	D6	LDAB	LOAD THE B ACCUMULATOR WITH THE
0188	20		MULTIPLIER FROM THIS LOCATION
0189	4F	CLRA	CLEAR ACCUMULATOR A
018A	97	STAA	STORE THE PRODUCT IN
018B	21		THIS TEMPORARY LOCATION
018C	17	TBA	TRANSFER ACCUMULATOR B TO A
018D	27	BEQ	IF THE MULTIPLIER IS ZERO, BRANCH
018E	08		TO THE NEXT ALGORITHM
018F	4A	DECA	OTHERWISE, DECREMENT THE MULTIPLIER
0190	16	TAB	TRANSFER ACCUMULATOR A TO B
0191	96	LDAA	LOAD THE ACCUMULATOR WITH THE
0192	21		PRODUCT STORED IN THIS TEMPORARY LOCATION
0193	9B	ADDA	ADD TO THE PRODUCT
0194	05		MULTIPLICAND AT THIS ADDRESS
0195	20	BRA	BRANCH BACK TO BEGINNING OF
0196	F3		MULTIPLY ALGORITHM
0197	96	LDAA	LOAD ACCUMULATOR A DIRECT
0198	21		WITH TEMP VALUE AT THIS ADDRESS
0199	97	STAA	STORE ACCUMULATOR A DIRECT
019A	1C		IN VARIABLE NAME AT THIS ADDRESS
019B	39	RTS	RETURN FROM SUBROUTINE
019C	20	BRA	CREATE A SCRATCHPAD
019D	01		LOCATION FOR
019E	XX		EXPONENT
019F	86	LDAA	SET RESULT OF
01A0	01		EXPONENTIATION
01A1	97	STAA	TO
01A2	1D		ONE
01A3	96	LDAA	LOAD VALUE OF
01A4	19		EXPONENT
01A5	B7	STAA	INTO

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ DECIMAL CONTENTS	COMMENTS
01A6	01		SCRATCHPAD
01A7	9F		LOCATION
01A8	27	BEQ	IF EXPONENT IS ZERO
01A9	1F		GO ON TO NEXT ALGORITHM
01AA	20	BRA	CREATE A SCRATCHPAD
01AB	01		LOCATION FOR
01AC	XX		PRODUCT
01AD	D6	LDAB	LOAD ACCUMULATOR B WITH THE
01AE	1A		MULTIPLIER FROM THIS LOCATION
01AF	4F	CLRA	CLEAR ACCUMULATOR A
01B0	B7	STAA	STORE THE PRODUCT IN
01B1	01		THIS
01B2	AC		LOCATION
01B3	17	TBA	TRANSFER ACCUMULATOR B TO A
01B4	27	BEQ	IF THE MULTIPLIER IS ZERO, EXIT
01B5	09		THE MULTIPLIER LOOP
01B6	4A	DECA	OTHERWISE, DECREMENT THE MULTIPLIER
01B7	16	TAB	TRANSFER ACCUMULATOR A TO B
01B8	B6	LDAA	LOAD ACCUMULATOR
01B9	01		WITH PRODUCT STORED
01BA	AC		IN THIS LOCATION
01BB	9B	ADDA	ADD TO THE PRODUCT THE
01BC	1D		MULTIPLICAND AT THIS ADDRESS
01BD	20	BRA	BRANCH BACK TO BEGINNING
01BE	F1		OF MULTIPLY ALGORITHM
01BF	B6	LDAA	LOAD ACCUMULATOR A
01C0	01		WITH PRODUCT STORED
01C1	AC		IN THIS LOCATION
01C2	97	STAA	STORE RESULT IN THIS
01C3	1D		TEMPORARY STORAGE LOCATION
01C4	7A	DEC	DECREMENT
01C5	01		EXPONENT
01C6	9F		LOCATION
01C7	20	BRA	BRANCH BACK TO BEGINNING
01C8	DE		OF EXPONENTIATION ALGORITHM
01C9	D6	LDAB	LOAD THE B ACCUMULATOR WITH THE
01CA	1D		DIVIDEND FROM THIS LOCATION
01CB	4F	CLRA	CLEAR ACCUMULATOR A
01CC	97	STAA	STORE THE QUOTIENT IN
01CD	1F		THIS TEMPORARY LOCATION
01CE	17	TBA	TRANSFER ACCUMULATOR B TO A
01CF	90	SUBA	SUBTRACT FROM THE DIVIDEND THE
01D0	05		DIVISOR AT THIS ADDRESS
01D1	2B	BMI	IF THE DIFFERENCE IS NEGATIVE,
01D2	06		BRANCH TO THE NEXT ALGORITHM
01D3	16	TAB	OTHERWISE, TRANSFER ACCUMULATOR A TO B
01D4	96	LDAA	LOAD THE ACCUMULATOR WITH THE
01D5	1F		QUOTIENT STORED IN THIS TEMPORARY LOCATION
01D6	4C	INCA	INCREMENT THE QUOTIENT
01D7	20	BRA	BRANCH BACK TO BEGINNING OF

```
                        MNEMONICS/
HEX          HEX        DECIMAL
ADDRESS      CONTENTS   CONTENTS       COMMENTS
0108         E3                        DIVIDE ALGORITHM
0109         96         LDAA           LOAD ACCUMULATOR A DIRECT
010A         1E                        WITH TEMP VALUE AT THIS ADDRESS
010B         97         STAA           STORE ACCUMULATOR A DIRECT
01DC         1C                        IN VARIABLE NAME AT THIS ADDRESS
01DD         39         RTS            RETURN FROM SUBROUTINE
01DE         3E         WAI            HALT EXECUTION (WAIT FOR INTERRUPT)
01DD         7E         JMP            JUMP TO
01CE         01                        THIS
01DF         50                        HEX ADDRESS
0124         BD         JSR            JUMP TO SUBROUTINE
0125         01                        BEGINNING AT THIS
0126         51                        MEMORY LOCATION
0135         7E         JMP            JUMP TO
0136         01                        THIS
0137         3B                        HEX ADDRESS
0138         7E         JMP            JUMP TO
0139         01                        THIS
013A         9C                        HEX ADDRESS
014D         7E         JMP            JUMP TO
014E         01                        THIS
014F         07                        HEX ADDRESS

PROGRAM STARTING ADDRESS 0100

TOTAL MEMORY USED :    479 BYTES
```

```
      SOURCE LISTING

1     REM THIS PROGRAM DEMONSTRATES COMPILER OPERATION.
2     REM NO SOURCE PROGRAM ERRORS RESULTS IN THE GENERATION OF HEX CODE.
3     DATA 2.4.6
4     FOR K=1 TO 6
5     RDIN T FROM 4004
6     READ A,B.C
7     GOSUB 13
8     IF K=5 GOTO 15
9     WRTOUT X TO 4006
10    RESTORE
11    NEXT K
12    STOP
13    LET X = (SQR(A+B)+K-T) * 2
14    RETURN
15    LET X = (B@A)/2
16    RETURN
17    END
```

```
        ERROR TABLE

ERROR                                        SOURCE LINE

NO DIAGNOSTICS GENERATED                       0
                                               0
MAXIMUM TEMPORARY STORAGE USED -               5
```

```
          KEYWORD TABLE

LET               1
GOTO              2
IF                3
TO                4
THEN              5
FOR               6
NEXT              7
GOSUB             8
RETURN            9
STEP             10
REM              11
DATA             12
READ             13
RESTORE          14
FROM             15
STOP             16
END              17
WRTOUT           18
RDIN             19
```

```
        TERMINAL TABLE

-       1
+       2
/       3
*       4
@       5
=       6
(       7
)       8
>       9
<       10
;       11
,       12
ABS     13
ATN     14
COS     15
DEF     16
EXP     17
INT     18
LOG     19
SIN     20
SQR     21
TAN     22
$       23
LEQ     24
GEQ     25
NEQ     26
CEQ     27
```

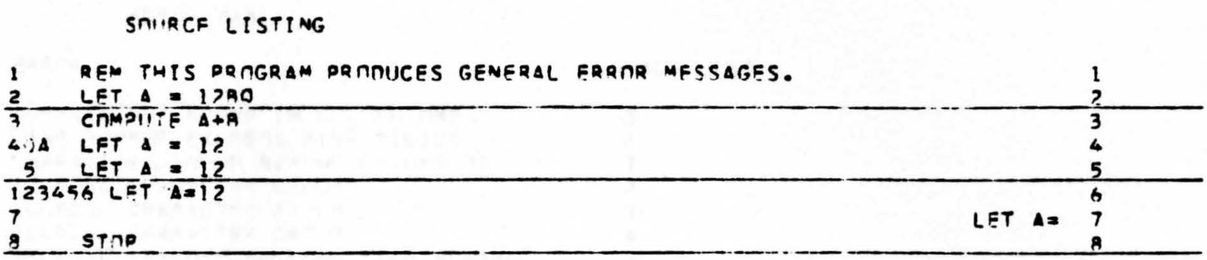
```
        IDENTIFIER TABLE

K       1
T       2
A       3
B       4
C       5
X       6
```

```
LITERAL TABLE

 1              1
 99             2
 2              3
 3              4
 4              5
 6              6
 5              7
 4004           8
 7              9
 13            10
 8             11
 15            12
 9             13
 4006          14
 10            15
 11            16
 12            17
 14            18
 16            19
 17            20
```

```
          UNIFORM SYMBOL TABLE

SYMBOL     POINTER
LIT           1
TRM          23
LIT           3
TRM          23
LIT           4
KEY          12
LIT           3
TRM          11
LIT           5
TRM          11
LIT           6
TRM          23
LIT           5
KEY           6
IDN           1
TRM           6
LIT           1
KEY           4
LIT           6
TRM          23
LIT           7
KEY          19
IDN           2
KEY          15
LIT           8
TRM          23
LIT           6
KEY          13
IDN           3
TRM          11
IDN           4
TRM          11
IDN           5
TRM          23
LIT           9
KEY           8
LIT          10
TRM          23
LIT          11
KEY           3
IDN           1
TRM           6
LIT           7
KEY           2
LIT          12
TRM          23
LIT          13
KEY          18
IDN           6
KEY           4
LIT          14
TRM          23
LIT          15
KEY          14
TRM          23
LIT          16
```

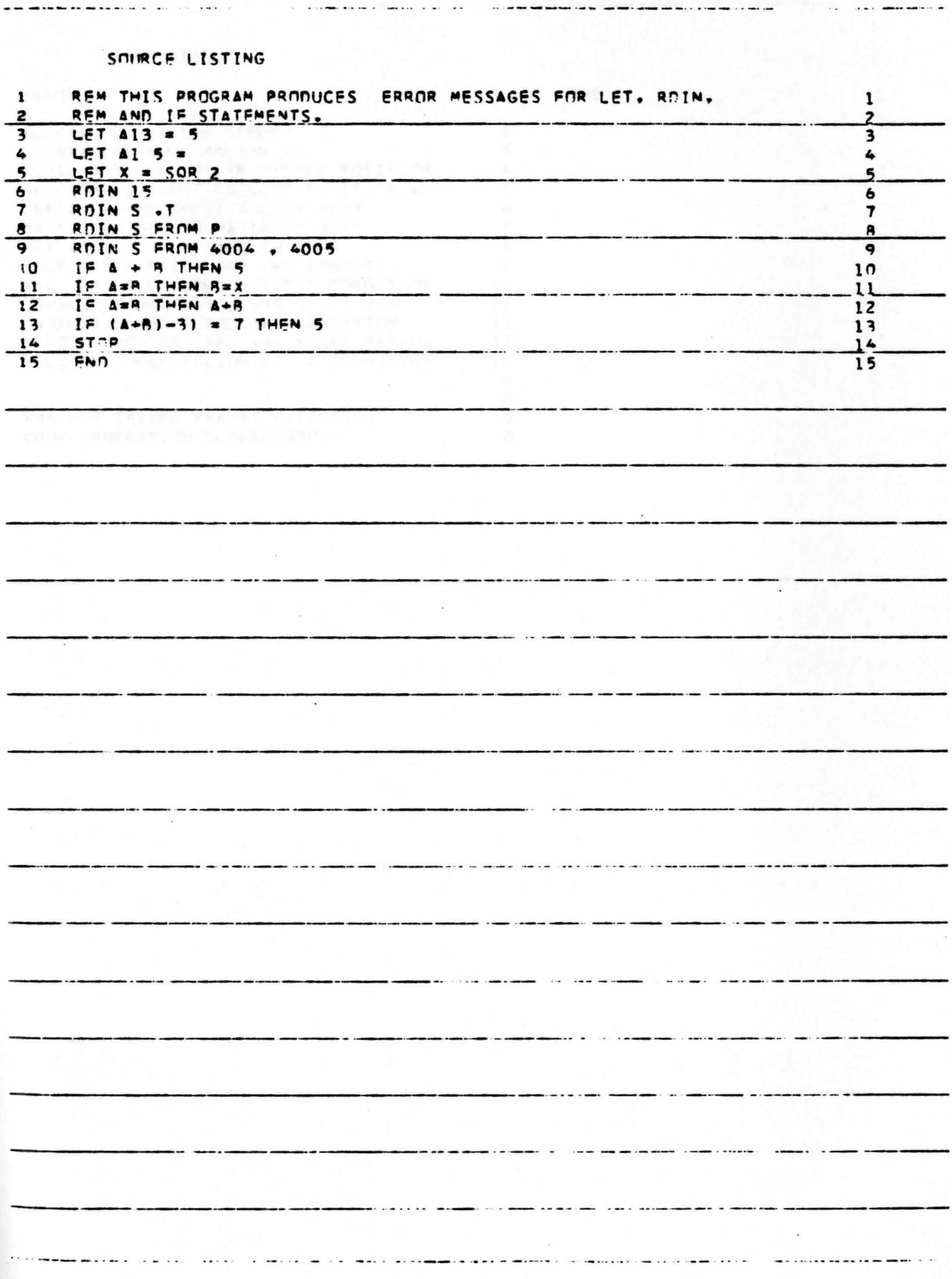
```
KEY         7
IDN         1
TRM        23
LIT        17
KEY        16
TRM        23
LIT        10
KEY         1
IDN         6
TRM         6
TRM         7
TRM        21
TRM         7
IDN         3
TRM         2
IDN         4
TRM         8
TRM         2
IDN         1
TRM         1
IDN         2
TRM         8
TRM         4
LIT         3
TRM        23
LIT        18
KEY         9
TRM        23
LIT        12
KEY         1
IDN         6
TRM         6
TRM         7
IDN         4
TRM         5
IDN         3
TRM         8
TRM         3
LIT         3
TRM        23
LIT        19
KEY         9
TRM        23
LIT        20
KEY        17
TRM        23
```

```
         DATA TABLE

  LIT TBL POINTER
      3
      5
      6
```



```
        SOURCE LISTING

1      REM THIS PROGRAM PRODUCES GENERAL ERROR MESSAGES.                 1
2      LET A = 12B0                                                      2
3      COMPUTE A+B                                                       3
4)A    LET A = 12                                                        4
 5     LET A = 12                                                        5
123456 LET A=12                                                          6
7                                                               LET A=   7
8      STOP                                                              8
```

```
        ERROR TABLE

ERROR                                          SOURCE LINE
NUMERIC NOT FOUND IN COLUMN ONE                 5
LINE NUMBER EXCEEDS FIVE DIGITS                 6
CHARACTERS FOUND BEYOND COLUMN 72               7
ILLEGAL CHARACTER GROUP                         2
ILLEGAL CHARACTER GROUP                         3
ILLEGAL CHARACTER GROUP                         4
ILLEGAL KEYWORD FOLLOWS LINE NUMBER             3
LINE DOES NOT START WITH A VALID NUMBER         4
LAST PROGRAM LINE MUST BE END STMT              9
                                                0
ABEND - SEVERE ERRORS DETECTED.                 0
CODE GENERATION SUPPRESSED.                     0
```



```
     SOURCE LISTING

1    REM THIS PROGRAM PRODUCES  ERROR MESSAGES FOR LET. RDIN,
2    REM AND IF STATEMENTS.
3    LET A13 = 5
4    LET A1 5 =
5    LET X = SQR 2
6    RDIN 15
7    RDIN S .T
8    RDIN S FROM P
9    RDIN S FROM 4004 , 4005
10   IF A + B THEN 5
11   IF A=B THEN B=X
12   IF A=B THEN A+B
13   IF (A+B)-3) = 7 THEN 5
14   STOP
15   END
```

```
       ERROR TABLE

 ERROR                                        SOURCE LINE

 ILLEGAL CHARACTER GROUP                        3
 IDENTIFIER MUST FOLLOW LET                     3
 EQUAL SIGN IS NOT IN PROPER POSITION           4
 PARENTHESIS MUST ENCLOSE FUNCTION ARG          5
 VARIABLE NAME MUST FOLLOW RDIN                 6
 FROM MUST FOLLOW VARIABLE NAME                 7
 HEX ADDRESS MUST FOLLOW FROM                   8
 MULTIPLE ENTRIES FOR HEX ADDRESS               9
 REL. OP. MUST FOLLOW FIRST CONDITION          10
 INVALID COMMAND FOLLOWS CONDITION             11
 INVALID COMMAND FOLLOWS CONDITION             12
 UNEQUAL NO. OF LEFT AND RIGHT PAREN.          13
 REL. OP. MUST FOLLOW FIRST CONDITION          14
                                                0
                                                0
 ABEND - SEVERE ERRORS DETECTED.                0
 CODE GENERATION SUPPRESSED.                    0
```

```
     SOURCE LISTING

1    REM THIS PROGRAM PRODUCES ERROR MESSAGES FOR DATA. RETURN, STOP,      1
2    REM. WRTOUT. GOTO, RESTORE, READ AND END STATEMENTS                   2
3    DATA A,B.C                                                            3
4    DATA 1 2 3                                                            4
5    RETURN TO A                                                           5
6    STOP EXECUTION                                                        6
7    WRTOUT 15                                                             7
8    WRTOUT A INTO 4004                                                    8
9    WRTOUT A TO B                                                         9
10   WRTOUT A TO 9006 .9007                                                10
11   GOTO A                                                                11
12   GOTO 1 2                                                              12
13   RESTORE A                                                             13
14   READ A12                                                              14
15   READ A B C                                                            15
16   END EXECUTION                                                         16
17   END                                                                   17
```

```
        ERROR TABLE

ERROR                                         SOURCE LINE
ILLEGAL CHARACTER GROUP                          6
ILLEGAL CHARACTER GROUP                          8
ILLEGAL CHARACTER GROUP                         14
ILLEGAL CHARACTER GROUP                         16
DATA ENTRIES MUST BE NUMERIC                     3
COMMAS REQUIRED BETWEEN DATA VALUES              4
CHARACTERS APPEAR AFTER RETURN STMT              5
VARIABLE NAME MUST FOLLOW WTOUT                  7
TO MUST FOLLOW VARIABLE NAME                     8
HEX ADDRESS MUST FOLLOW TO                       9
MULTIPLE ENTRIES FOR HEX ADDRESS                10
LINE NUMBER MUST FOLLOW GOTO STMT               11
MULTIPLE ARGUMENTS IN GOTO STMT                 12
CHARACTERS APPEAR AFTER RESTORE STMT            13
READ ARGUMENT MUST BE A VARIABLE NAME           14
COMMAS REQUIRED BETWEEN READ ARGUMENTS          15
                                                 0
                                                 0
ABEND - SEVERE ERRORS DETECTED.                  0
CODE GENERATION SUPPRESSED.                      0
```

SOURCE LISTING

```
1     REM THIS PROGRAM PRODUCES ERROR MESSAGES FOR NEXT, GOSUB            1
2     REM AND FOR STATEMENTS.                                             2
3     FOR 12 TO A                                                         3
4     FOR L 1 = TO 7                                                      4
5     FOR L=1.5 TO 2.5                                                    5
6     FOR A57 = 1 TO 7                                                    6
7     FOR A = 1  7                                                        7
8     FOR A = 1 TO 7                                                      8
9     FOR A = 1 TO A57                                                    9
10    FOR A = 1 TO 2.5                                                   10
11    FOR A = 1 TO 7 LET K = 1                                           11
12    FOR A = 1 TO 7 STEP A57                                            12
13    FOR A = 1 TO 7 STEP 1.5                                            13
14    FOR A = 1 TO 7 X                                                   14
15    NEXT 12                                                            15
16    NEXT O                                                             16
17    NEXT L 1                                                           17
18    GOSUB A                                                            18
19    GOSUB 1 2                                                          19
20    NEXT A                                                             20
21    END                                                                21
```

```
        ERROR TABLE

 ERROR                                         SOURCE LINE

 ILLEGAL CHARACTER GROUP                         6
 ILLEGAL CHARACTER GROUP                         9
 ILLEGAL CHARACTER GROUP                        12
 IDENTIFIER MUST FOLLOW FOR                      3
 EQUAL SIGN IS NOT IN PROPER POSITION            4
 LITERAL PRECEDING TO IS NOT INTEGER             5
 IDENTIFIER MUST FOLLOW FOR                      6
 TO IS MISPLACED OR MISSING                      7
 POS. INTEGER OR VAR. MUST FOLLOW TO             9
 LITERAL FOLLOWING TO IS NOT INTEGER            10
 ONLY STEP MAY FOLLOW INTEGER AFTER TO          11
 POSITIVE INTEGER MUST FOLLOW STEP              12
 LITERAL FOLLOWING STEP IS NOT INTEGER          13
 ONLY STEP MAY FOLLOW INTEGER AFTER TO          14
 NEXT MUST BE FOLLOWED BY A VARIABLE            15
 IMPROPER FOR - NEXT PAIR                       16
 IMPROPER FOR - NEXT PAIR                       17
 LINE NUMBER MUST FOLLOW GOSUB STMT             18
 MULTIPLE ARGUMENTS IN GOSUB STMT               19
                                                 0
                                                 0
 ABEND - SEVERE ERRORS DETECTED.                 0
 CODE GENERATION SUPPRESSED.                     0
```

APPENDIX D

Model Railroad Control Program

```
                           MNEMONICS/
HEX          HEX           DECIMAL
ADDRESS      CONTENTS      CONTENTS             COMMENTS

0000         00            000000000            DATA
0001         20            000000032            DATA
0002         0A            000000010            DATA
0003         80            000000128            DATA
0004         01            000000001            DATA
0005         40            000000064            DATA
0006         02            000000002            DATA
0007         90            000000144            DATA
0008         04            000000004            DATA
0009         00            000000000            DATA
000A         01            000000001            DATA
000B         40            000000064            DATA
000C         01            000000001            CONSTANT
000D         63            000000099            CONSTANT
000E         02            000000002            CONSTANT
000F         00            000000000            CONSTANT
0010         20            000000032            CONSTANT
0011         0A            000000010            CONSTANT
0012         80            000000128            CONSTANT
0013         40            000000064            CONSTANT
0014         90            000000144            CONSTANT
0015         04            000000004            CONSTANT
0016         03            000000003            CONSTANT
0017         FF            000000255            CONSTANT
0018         05            000000005            CONSTANT
0019         A6            000004006            CONSTANT
001A         06            000000006            CONSTANT
001B         07            000000007            CONSTANT
001C         A5            000004005            CONSTANT
001D         08            000000008            CONSTANT
001E         A7            000004007            CONSTANT
001F         09            000000009            CONSTANT
0020         50            000000080            CONSTANT
0021         0B            000000011            CONSTANT
0022         C9            000000201            CONSTANT
0023         0C            000000012            CONSTANT
0024         0D            000000013            CONSTANT
0025         0E            000000014            CONSTANT
0026         0F            000000015            CONSTANT
0027         10            000000016            CONSTANT
0028         11            000000017            CONSTANT
0029         64            000000100            CONSTANT
002A         12            000000018            CONSTANT
002B         13            000000019            CONSTANT
002C         14            000000020            CONSTANT
002D         C8            000000200            CONSTANT
002E         15            000000021            CONSTANT
002F         16            000000022            CONSTANT
0030         17            000000023            CONSTANT
0031         18            000000024            CONSTANT
```

```
                         MNEMONICS/
HEX         HEX          DECIMAL
ADDRESS     CONTENTS     CONTENTS          COMMENTS

0032        19           000000025         CONSTANT
0033        1A           000000026         CONSTANT
0034        1B           000000027         CONSTANT
0035        44           000004004         CONSTANT
0036        65           000000101         CONSTANT
0037        66           000000102         CONSTANT
0038        CA           000000202         CONSTANT
0039        CB           000000203         CONSTANT
003A        CC           000000204         CONSTANT
003B        CD           000000205         CONSTANT
003C        CE           000000206         CONSTANT
003D        CF           000000207         CONSTANT
003E        D0           000000208         CONSTANT
003F        D1           000000209         CONSTANT
0040        D2           000000210         CONSTANT
0041        D3           000000211         CONSTANT
0042        XX           XXXXXXXXX         D
0043        XX           XXXXXXXXX         C
0044        XX           XXXXXXXXX         S
0045        XX           XXXXXXXXX         P1
0046        XX           XXXXXXXXX         N
0047        XX           XXXXXXXXX         K
0048        XX           XXXXXXXXX         P
0049        XX           XXXXXXXXX         L
004A        XX           XXXXXXXXX         TEMPORARY STORAGE LOCATION
004B        XX           XXXXXXXXX         TEMPORARY STORAGE LOCATION
0100        CE           LDX               RESET INDEX REGISTER TO
0101        00                             TOP OF DATA TABLE STARTING
0102        00                             AT ADDRESS 0000
0103        96           LDAA              LOAD ACCUMULATOR A DIRECT
0104        17                             WITH CONSTANT AT THIS ADDRESS
0105        97           STAA              STORE ACCUMULATOR A DIRECT
0106        42                             IN VARIABLE NAME AT THIS ADDRESS
0107        96           LDAA              LOAD DATA STORED IN VARIABLE
0108        42                             AT THIS ADDRESS INTO ACCUMULATOR A.
0109        B7           STAA              THEN WRITE IT TO
010A        40                             THE OUTPUT DEVICE RESIDING
010B        06                             AT THIS HEX ADDRESS
010C        96           LDAA              LOAD ACCUMULATOR A DIRECT
010D        15                             WITH CONSTANT AT THIS ADDRESS
010E        97           STAA              STORE ACCUMULATOR A DIRECT
010F        43                             IN VARIABLE NAME AT THIS ADDRESS
0110        96           LDAA              LOAD DATA STORED IN VARIABLE
0111        43                             AT THIS ADDRESS INTO ACCUMULATOR A,
0112        B7           STAA              THEN WRITE IT TO
0113        40                             THE OUTPUT DEVICE RESIDING
0114        05                             AT THIS HEX ADDRESS
0115        96           LDAA              LOAD DATA STORED IN VARIABLE
0116        43                             AT THIS ADDRESS INTO ACCUMULATOR A,
0117        B7           STAA              THEN WRITE IT TO
```

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ DECIMAL CONTENTS	COMMENTS
0118	40		THE OUTPUT DEVICE RESIDING
0119	07		AT THIS HEX ADDRESS
011A	96	LDAA	LOAD ACCUMULATOR A DIRECT
011B	20		WITH CONSTANT AT THIS ADDRESS
011C	97	STAA	STORE ACCUMULATOR A DIRECT
011D	44		IN VARIABLE NAME AT THIS ADDRESS
011E	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
011F	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0120	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0121	96	LDAA	LOAD ACCUMULATOR A DIRECT
0122	18		WITH CONSTANT AT THIS ADDRESS
0123	97	STAA	STORE ACCUMULATOR A DIRECT
0124	44		IN VARIABLE NAME AT THIS ADDRESS
0125	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0126	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0127	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0128	96	LDAA	LOAD ACCUMULATOR A DIRECT
0129	0F		WITH CONSTANT AT THIS ADDRESS
012A	97	STAA	STORE ACCUMULATOR A DIRECT
012B	45		IN VARIABLE NAME AT THIS ADDRESS
012C	96	LDAA	LOAD ACCUMULATOR A DIRECT
012D	0C		WITH CONSTANT AT THIS ADDRESS
012E	97	STAA	STORE ACCUMULATOR A DIRECT
012F	46		IN VARIABLE NAME AT THIS ADDRESS
0130	96	LDAA	LOAD FIRST ARGUMENT
0131	46		INTO ACCUMULATOR A
0132	90	SUBA	SUBTRACT SECOND ARGUMENT
0133	23		FROM FIRST ARGUMENT
0134	2F	BLE	IF RESULT OF COMPARISON IS TRUE
0135	03		BRANCH TO NEXT STATEMENT
0136	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0137	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0138	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0139	96	LDAA	LOAD ACCUMULATOR A DIRECT
013A	0C		WITH CONSTANT AT THIS ADDRESS
013B	97	STAA	STORE ACCUMULATOR A DIRECT
013C	47		IN VARIABLE NAME AT THIS ADDRESS
013D	96	LDAA	LOAD FIRST ARGUMENT
013E	47		INTO ACCUMULATOR A
013F	90	SUBA	SUBTRACT SECOND ARGUMENT
0140	29		FROM FIRST ARGUMENT
0141	2F	BLE	IF RESULT OF COMPARISON IS TRUE
0142	03		BRANCH TO NEXT STATEMENT
0143	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0144	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0145	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0146	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0147	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0148	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0149	96	LDAA	LOAD ACCUMULATOR A DIRECT

```
                      MNEMONICS/
HEX         HEX       DECIMAL
ADDRESS     CONTENTS  CONTENTS        COMMENTS

0144        47                        WITH FIRST OPERAND
014B        9B        ADDA            ADD DIRECT TO ACCUMULATOR A
014C        0C                        THE SECOND OPERAND
014D        97        STAA            STORE ACCUMULATOR A DIRECT
014E        4A                        IN THIS TEMPORARY LOCATION
014F        96        LDAA            LOAD ACCUMULATOR A DIRECT
0150        4A                        WITH TEMP VALUE AT THIS ADDRESS
0151        97        STAA            STORE ACCUMULATOR A DIRECT
0152        47                        IN VARIABLE NAME AT THIS ADDRESS
0153        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0154        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0155        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0156        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0157        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0158        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0159        96        LDAA            LOAD ACCUMULATOR A DIRECT
015A        4B                        WITH IDENTIFIER VALUE AT THIS ADDRESS
015B        97        STAA            STORE ACCUMULATOR A DIRECT
015C        4A                        IN TEMPORARY STORAGE AT THIS ADDRESS
015D        96        LDAA            LOAD ACCUMULATOR A DIRECT
015E        0E                        WITH CONSTANT AT THIS ADDRESS
015F        97        STAA            STORE ACCUMULATOR A DIRECT
0160        4B                        IN TEMPORARY STORAGE AT THIS ADDRESS
0161        96        LDAA            LOAD FIRST ARGUMENT
0162        4A                        INTO ACCUMULATOR A
0163        90        SUBA            SUBTRACT SECOND ARGUMENT
0164        4B                        FROM FIRST ARGUMENT
0165        27        BEQ             IF RESULT OF COMPARISON IS TRUE
0166        03                        BRANCH TO NEXT STATEMENT
0167        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0168        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
0169        XX        XXXXXXXXX       BRANCH INSTRUCTION - TO BE FILLED IN LATER
016A        96        LDAA            LOAD ACCUMULATOR A DIRECT
016B        0C                        WITH CONSTANT AT THIS ADDRESS
016C        97        STAA            STORE ACCUMULATOR A DIRECT
016D        45                        IN VARIABLE NAME AT THIS ADDRESS
016E        96        LDAA            LOAD ACCUMULATOR A DIRECT
016F        4B                        WITH IDENTIFIER VALUE AT THIS ADDRESS
0170        97        STAA            STORE ACCUMULATOR A DIRECT
0171        4A                        IN TEMPORARY STORAGE AT THIS ADDRESS
0172        96        LDAA            LOAD ACCUMULATOR A DIRECT
0173        0C                        WITH CONSTANT AT THIS ADDRESS
0174        97        STAA            STORE ACCUMULATOR A DIRECT
0175        4B                        IN TEMPORARY STORAGE AT THIS ADDRESS
0176        96        LDAA            LOAD FIRST ARGUMENT
0177        4A                        INTO ACCUMULATOR A
0178        90        SUBA            SUBTRACT SECOND ARGUMENT
0179        4B                        FROM FIRST ARGUMENT
017A        27        BEQ             IF RESULT OF COMPARISON IS TRUE
017B        03                        BRANCH TO NEXT STATEMENT
```

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ DECIMAL CONTENTS	COMMENTS
017C	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
017D	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
017E	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
017F	96	LDAA	LOAD ACCUMULATOR A DIRECT
0180	0F		WITH CONSTANT AT THIS ADDRESS
0181	97	STAA	STORE ACCUMULATOR A DIRECT
0182	45		IN VARIABLE NAME AT THIS ADDRESS
0183	96	LDAA	LOAD ACCUMULATOR A DIRECT
0184	46		WITH FIRST OPERAND
0185	9B	ADDA	ADD DIRECT TO ACCUMULATOR A
0186	0C		THE SECOND OPERAND
0187	97	STAA	STORE ACCUMULATOR A DIRECT
0188	4A		IN THIS TEMPORARY LOCATION
0189	96	LDAA	LOAD ACCUMULATOR A DIRECT
018A	4A		WITH TEMP VALUE AT THIS ADDRESS
018B	97	STAA	STORE ACCUMULATOR A DIRECT
018C	46		IN VARIABLE NAME AT THIS ADDRESS
018D	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
018E	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
018F	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0190	CE	LDX	RESET INDEX REGISTER TO
0191	00		TOP OF DATA TABLE STARTING
0192	00		AT ADDRESS 0000
0193	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0194	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0195	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
0196	3E	WAI	HALT EXECUTION (WAIT FOR INTERRUPT)
0197	B6	LDAA	LOAD DATA FROM INPUT
0198	40		DEVICE AT THIS ADDRESS
0199	04		INTO ACCUMULATOR A. THEN
019A	97	STAA	STORE DATA IN VARIABLE
019B	48		NAME AT THIS ADDRESS
019C	96	LDAA	LOAD ACCUMULATOR A DIRECT
019D	48		WITH IDENTIFIER VALUE AT THIS ADDRESS
019E	97	STAA	STORE ACCUMULATOR A DIRECT
019F	4A		IN TEMPORARY STORAGE AT THIS ADDRESS
01A0	96	LDAA	LOAD ACCUMULATOR A DIRECT
01A1	45		WITH IDENTIFIER VALUE AT THIS ADDRESS
01A2	97	STAA	STORE ACCUMULATOR A DIRECT
01A3	4B		IN TEMPORARY STORAGE AT THIS ADDRESS
01A4	96	LDAA	LOAD FIRST ARGUMENT
01A5	4A		INTO ACCUMULATOR A
01A6	90	SUBA	SUBTRACT SECOND ARGUMENT
01A7	4B		FROM FIRST ARGUMENT
01A8	26	BNE	IF RESULT OF COMPARISON IS TRUE
01A9	03		BRANCH TO NEXT STATEMENT
01AA	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
01AB	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
01AC	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER
01AD	XX	XXXXXXXXX	BRANCH INSTRUCTION - TO BE FILLED IN LATER

```
                            MNEMONICS/
HEX         HEX             DECIMAL
ADDRESS     CONTENTS        CONTENTS           COMMENTS

01AE        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01AF        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01B0        39              RTS                RETURN FROM SUBROUTINE
01B1        A6              LDAA. X            LOAD DATA INTO ACCUMULATOR A USING
01B2        00                                 INDEXED ADDRESSING WITH ZERO OFFSET
01B3        97              STAA               STORE DATA DIRECT INTO VARIABLE NAME
01B4        44                                 AT THIS ZERO PAGE ADDRESS
01B5        08              INX                INCREMENT INDEX REGISTER
01B6        96              LDAA               LOAD DATA STORED IN VARIABLE
01B7        44                                 AT THIS ADDRESS INTO ACCUMULATOR A.
01B8        B7              STAA               THEN WRITE IT TO
01B9        40                                 THE OUTPUT DEVICE RESIDING
01BA        06                                 AT THIS HEX ADDRESS
01BB        96              LDAA               LOAD ACCUMULATOR A DIRECT
01BC        0C                                 WITH CONSTANT AT THIS ADDRESS
01BD        97              STAA               STORE ACCUMULATOR A DIRECT
01BE        47                                 IN VARIABLE NAME AT THIS ADDRESS
01BF        96              LDAA               LOAD FIRST ARGUMENT
01C0        47                                 INTO ACCUMULATOR A
01C1        90              SUBA               SUBTRACT SECOND ARGUMENT
01C2        10                                 FROM FIRST ARGUMENT
01C3        2F              BLE                IF RESULT OF COMPARISON IS TRUE
01C4        03                                 BRANCH TO NEXT STATEMENT
01C5        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01C6        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01C7        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01C8        96              LDAA               LOAD ACCUMULATOR A DIRECT
01C9        0C                                 WITH CONSTANT AT THIS ADDRESS
01CA        97              STAA               STORE ACCUMULATOR A DIRECT
01CB        49                                 IN VARIABLE NAME AT THIS ADDRESS
01CC        96              LDAA               LOAD FIRST ARGUMENT
01CD        49                                 INTO ACCUMULATOR A
01CE        90              SUBA               SUBTRACT SECOND ARGUMENT
01CF        10                                 FROM FIRST ARGUMENT
01D0        2F              BLE                IF RESULT OF COMPARISON IS TRUE
01D1        03                                 BRANCH TO NEXT STATEMENT
01D2        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01D3        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01D4        XX              XXXXXXXXX          BRANCH INSTRUCTION - TO BE FILLED IN LATER
01D5        96              LDAA               LOAD ACCUMULATOR A DIRECT
01D6        0F                                 WITH CONSTANT AT THIS ADDRESS
01D7        97              STAA               STORE ACCUMULATOR A DIRECT
01D8        44                                 IN VARIABLE NAME AT THIS ADDRESS
01D9        96              LDAA               LOAD ACCUMULATOR A DIRECT
01DA        0F                                 WITH CONSTANT AT THIS ADDRESS
01DB        97              STAA               STORE ACCUMULATOR A DIRECT
01DC        44                                 IN VARIABLE NAME AT THIS ADDRESS
01DD        96              LDAA               LOAD ACCUMULATOR A DIRECT
01DE        49                                 WITH FIRST OPERAND
01DF        9B              ADDA               ADD DIRECT TO ACCUMULATOR A
```

```
                     MNEMONICS/
HEX       HEX        DECIMAL
ADDRESS   CONTENTS   CONTENTS       COMMENTS
_____________________________________________________________________________
01E0      0C                        THE SECOND OPERAND
01E1      97         STAA           STORE ACCUMULATOR A DIRECT
01E2      4A                        IN THIS TEMPORARY LOCATION
_____________________________________________________________________________
01E3      96         LDAA           LOAD ACCUMULATOR A DIRECT
01E4      4A                        WITH TEMP VALUE AT THIS ADDRESS
01E5      97         STAA           STORE ACCUMULATOR A DIRECT
_____________________________________________________________________________
01E6      49                        IN VARIABLE NAME AT THIS ADDRESS
01E7      XX         XXXXXXXXX      BRANCH INSTRUCTION - TO BE FILLED IN LATER
01E8      XX         XXXXXXXXX      BRANCH INSTRUCTION - TO BE FILLED IN LATER
_____________________________________________________________________________
01E9      XX         XXXXXXXXX      BRANCH INSTRUCTION - TO BE FILLED IN LATER
01EA      96         LDAA           LOAD ACCUMULATOR A DIRECT
01EB      47                        WITH FIRST OPERAND
_____________________________________________________________________________
01EC      9B         ADDA           ADD DIRECT TO ACCUMULATOR A
01ED      0C                        THE SECOND OPERAND
01EE      97         STAA           STORE ACCUMULATOR A DIRECT
_____________________________________________________________________________
01EF      4A                        IN THIS TEMPORARY LOCATION
01F0      96         LDAA           LOAD ACCUMULATOR A DIRECT
01F1      4A                        WITH TEMP VALUE AT THIS ADDRESS
_____________________________________________________________________________
01F2      97         STAA           STORE ACCUMULATOR A DIRECT
01F3      47                        IN VARIABLE NAME AT THIS ADDRESS
01F4      XX         XXXXXXXXX      BRANCH INSTRUCTION - TO BE FILLED IN LATER
_____________________________________________________________________________
01F5      XX         XXXXXXXXX      BRANCH INSTRUCTION - TO BE FILLED IN LATER
01F6      XX         XXXXXXXXX      BRANCH INSTRUCTION - TO BE FILLED IN LATER
01F7      96         LDAA           LOAD DATA STORED IN VARIABLE
_____________________________________________________________________________
01F8      44                        AT THIS ADDRESS INTO ACCUMULATOR A.
01F9      B7         STAA           THEN WRITE IT TO
01FA      40                        THE OUTPUT DEVICE RESIDING
_____________________________________________________________________________
01FB      06                        AT THIS HEX ADDRESS
01FC      39         RTS            RETURN FROM SUBROUTINE
01FD      3F         WAI            HALT EXECUTION (WAIT FOR INTERRUPT)
_____________________________________________________________________________
011E      BD         JSR            JUMP TO SUBROUTINE
011F      01                        BEGINNING AT THIS
0120      B6                        MEMORY LOCATION
_____________________________________________________________________________
0125      BD         JSR            JUMP TO SUBROUTINE
0126      01                        BEGINNING AT THIS
0127      B6                        MEMORY LOCATION
_____________________________________________________________________________
0136      7E         JMP            JUMP TO
0137      01                        THIS
0138      90                        HEX ADDRESS
_____________________________________________________________________________
0143      7E         JMP            JUMP TO
0144      01                        THIS
0145      56                        HEX ADDRESS
_____________________________________________________________________________
0146      BD         JSR            JUMP TO SUBROUTINE
0147      01                        BEGINNING AT THIS
0148      97                        MEMORY LOCATION
_____________________________________________________________________________
0153      7E         JMP            JUMP TO
0154      01                        THIS
0155      3D                        HEX ADDRESS
_____________________________________________________________________________
0156      BD         JSR            JUMP TO SUBROUTINE
0157      01                        BEGINNING AT THIS
```

```
                              MNEMONICS/
HEX          HEX              DECIMAL
ADDRESS      CONTENTS         CONTENTS          COMMENTS

0158         B1                                 MEMORY LOCATION
0167         7E               JMP               JUMP TO
0168         01                                 THIS
0169         6E                                 HEX ADDRESS
017C         7E               JMP               JUMP TO
017D         01                                 THIS
017E         83                                 HEX ADDRESS
018D         7E               JMP               JUMP TO
018E         01                                 THIS
018F         30                                 HEX ADDRESS
0193         7E               JMP               JUMP TO
0194         01                                 THIS
0195         2C                                 HEX ADDRESS
01AA         7E               JMP               JUMP TO
01AB         01                                 THIS
01AC         B0                                 HEX ADDRESS
01AD         7E               JMP               JUMP TO
01AE         01                                 THIS
01AF         97                                 HEX ADDRESS
01C5         7E               JMP               JUMP TO
01C6         01                                 THIS
01C7         E7                                 HEX ADDRESS
01D2         7E               JMP               JUMP TO
01D3         01                                 THIS
01D4         EA                                 HEX ADDRESS
01E7         7E               JMP               JUMP TO
01E8         01                                 THIS
01E9         CC                                 HEX ADDRESS
01F4         7E               JMP               JUMP TO
01F5         01                                 THIS
01F6         8E                                 HEX ADDRESS

PROGRAM STARTING ADDRESS 0100

TOTAL MEMORY USED :     510 BYTES
```

SOURCE LISTING

```
1    REM THIS PROGRAM DRIVES HO R.R. SWITCHES
2    DATA 0, 32, 10, 128, 1, 64, 2, 144, 4, 0, 1, 64
3    REM INITIALIZE PIA
4    LET D = 255
5    WRTOUT D TO 4006
6    LET C = 4
7    WRTOUT C TO 4005
8    WRTOUT C TO 4007
9    REM SET ALL SWITCHES STRAIGHT THROUGH
10   LET S = 80
11   GOSUB 201
12   LET S = 5
13   GOSUB 201
14   REM BEGIN MAIN PROGRAM
15   LET P1 = 2
16   FOR N = 1 TO 12
17   FOR K = 1 TO 100
18   GOSUB 100
19   NEXT K
20   GOSUB 200
21   IF P = 2 THEN LET P1 = 1
22   IF P = 1 THEN LET P1 = 2
23   NEXT N
24   RESTORE
25   GOTO 16
26   STOP
27   REM SUBROUTINES
100  RDIN P FROM 4004
101  IF P NEQ P1 GOTO 100
102  RETURN
200  READ S
201  WRTOUT S TO 4006
202  REM DELAY FOR SWITCH RESPONSE
203  FOR K = 1 TO 32
204  FOR L = 1 TO 32
205  LET S = 0
206  LET S = 0
207  NEXT L
208  NEXT K
209  WRTOUT S TO 4006
210  RETURN
211  END
```

```
        ERROR TABLE

ERROR                                    SOURCE LINE

NO DIAGNOSTICS GENERATED                   0
                                           0
MAXIMUM TEMPORARY STORAGE USED -           2
```

```
        KEYWORD TABLE

LET              1
GOTO             2
IF               3
TO               4
THEN             5
FOR              6
NEXT             7
GOSUB            8
RETURN           9
STEP            10
REM             11
DATA            12
READ            13
RESTORE         14
FROM            15
STOP            16
END             17
WRTOUT          18
RDIN            19
```

```
        TERMINAL TABLE

 -         1
 +         2
 /         3
 *         4
 @         5
 =         6
 (         7
 )         8
 >         9
 <        10
 .        11
 .        12
 ABS      13
 ATN      14
 COS      15
 DEF      16
 EXP      17
 INT      18
 LOG      19
 SIN      20
 SQR      21
 TAN      22
 $        23
 LEQ      24
 GEQ      25
 NEQ      26
 CEQ      27
```

```
        IDENTIFIER TABLE

D          1
C          2
S          3
P1         4
N          5
K          6
P          7
L          8
```

```
        LITERAL TABLE

1             1
99            2
2             3
0             4
32            5
10            6
128           7
64            8
144           9
4             10
3             11
255           12
5             13
4006          14
6             15
7             16
4005          17
8             18
4007          19
9             20
80            21
11            22
201           23
12            24
13            25
14            26
15            27
16            28
17            29
100           30
18            31
19            32
20            33
200           34
21            35
22            36
23            37
24            38
25            39
26            40
27            41
4004          42
101           43
102           44
202           45
203           46
204           47
205           48
206           49
207           50
208           51
209           52
210           53
211           54
```

```
          UNIFORM SYMBOL TABLE

SYMBOL    POINTER
LIT          1
TRM         23
LIT          3
KEY         12
LIT          4
TRM         11
LIT          5
TRM         11
LIT          6
TRM         11
LIT          7
TRM         11
LIT          1
TRM         11
LIT          8
TRM         11
LIT          3
TRM         11
LIT          9
TRM         11
LIT         10
TRM         11
LIT          4
TRM         11
LIT          1
TRM         11
LIT          8
TRM         23
LIT         11
TRM         23
LIT         10
KEY          1
IDN          1
TRM          6
LIT         12
TRM         23
LIT         13
KEY         18
IDN          1
KEY          4
LIT         14
TRM         23
LIT         15
KEY          1
IDN          2
TRM          6
LIT         10
TRM         23
LIT         16
KEY         18
IDN          2
KEY          4
LIT         17
TRM         23
LIT         18
KEY         18
```

```
IDN         2
KEY         4
LIT        19
TRM        23
LIT        20
TRM        23
LIT         6
KEY         1
IDN         3
TRM         6
LIT        21
TRM        23
LIT        22
KEY         8
LIT        23
TRM        23
LIT        24
KEY         1
IDN         3
TRM         5
LIT        13
TRM        23
LIT        25
KEY         8
LIT        23
TRM        23
LIT        26
TRM        23
LIT        27
KEY         1
IDN         4
TRM         6
LIT         3
TRM        23
LIT        28
KEY         6
IDN         5
TRM         6
LIT         1
KEY         4
LIT        24
TRM        23
LIT        29
KEY         6
IDN         6
TRM         6
LIT         1
KEY         4
LIT        30
TRM        23
LIT        31
KEY         8
LIT        30
TRM        23
LIT        32
KEY         7
IDN         6
TRM        23
LIT        33
KEY         8
```

```
LIT        34
TRM        23
LIT        35
KEY         3
IDN         7
TRM         6
LIT         3
KEY         5
KEY         1
IDN         4
TRM         6
LIT         1
TRM        23
LIT        36
KEY         3
IDN         7
TRM         6
LIT         1
KEY         5
KEY         1
IDN         4
TRM         6
LIT         3
TRM        23
LIT        37
KEY         7
IDN         5
TRM        23
LIT        38
KEY        14
TRM        23
LIT        39
KEY         2
LIT        28
TRM        23
LIT        40
KEY        16
TRM        23
LIT        41
TRM        23
LIT        30
KEY        19
IDN         7
KEY        15
LIT        42
TRM        23
LIT        43
KEY         3
IDN         7
TRM        26
IDN         4
KEY         2
LIT        30
TRM        23
LIT        44
KEY         9
TRM        23
LIT        34
KEY        13
IDN         3
```

```
TRM       23
LIT       23
KEY       18
IDN        3
KEY        4
LIT       14
TRM       23
LIT       45
TRM       23
LIT       46
KEY        6
IDN        6
TRM        6
LIT        1
KEY        4
LIT        5
TRM       23
LIT       47
KEY        6
IDN        8
TRM        6
LIT        1
KEY        4
LIT        5
TRM       23
LIT       48
KEY        1
IDN        3
TRM        6
LIT        4
TRM       23
LIT       49
KEY        1
IDN        3
TRM        6
LIT        4
TRM       23
LIT       50
KEY        7
IDN        8
TRM       23
LIT       51
KEY        7
IDN        6
TRM       23
LIT       52
KEY       18
IDN        3
KEY        4
LIT       14
TRM       23
LIT       53
KEY        9
TRM       23
LIT       54
KEY       17
TRM       23
```

```
        DATA TABLE

LIT TBL POINTER
      4
      5
      6
      7
      1
      8
      3
      9
     10
      4
      1
      8
```

```
FILE: MATRIX    DATA     A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

0010TRM0006IDN0001LIT0012
0013KFY0018IDN0001LIT0014
0015TRM0004IDN0002LIT0010
0016KFY0018IDN0002LIT0017
0018KFY0018IDN0002LIT0019
0006TRM0006IDN0003LIT0021
0022KFY0008LIT0023    0000
0024TRM0006IDN0003LIT0013
0025KFY0008LIT0023    0000
0027TRM0006IDN0004LIT0003
0028TRM0006IDN0005LIT0001
0028TRM0024IDN0005LIT0024
0028KFY0002LIT0038    0000
0029TRM0006IDN0006LIT0001
0029TRM0024IDN0006LIT0030
0029KFY0002LIT0033    0000
0031KFY0008LIT0030    0000
0032TRM0002IDN0006LIT0001
0032TRM0006IDN0006TMP0001
0032KFY0002LIT0029LIT0002
0033KFY0008LIT0034    0000
0035TRM0006TMP0001IDN0007
0035TRM0006TMP0002LIT0003
0035TRM0027TMP0001TMP0002
0035KFY0002LIT0036    0000
0035TRM0006IDN0004LIT0001
0036TRM0006TMP0001IDN0007
0036TRM0005TMP0002LIT0001
0036TRM0027TMP0001TMP0002
0036KFY0002LIT0037    0000
0036TRM0006IDN0004LIT0003
0037TRM0002IDN0005LIT0001
0037TRM0005IDN0005TMP0001
0037KFY0002LIT0028LIT0002
0038KFY0014    0000    0000
0039KFY0002LIT0028    0000
0040KFY0015    0000    0000
0030KFY0019IDN0007LIT0042
0043TRM0006TMP0001IDN0007
0043TRM0006TMP0002IDN0004
0043TRM0026TMP0001TMP0002
0043KFY0002LIT0044    0000
0043KFY0002LIT0030    0000
0044KFY0009    0000    0000
0034KFY0013IDN0003    0000
0023KFY0018IDN0003LIT0014
0046TRM0006IDN0006LIT0001
0046TRM0024IDN0006LIT0005
0046KFY0002LIT0052    0000
0047TRM0006IDN0008LIT0001
0047TRM0024IDN0008LIT0005
0047KFY0002LIT0051    0000
0048TRM0006IDN0003LIT0004
0049TRM0006IDN0003LIT0004
0050TRM0002IDN0008LIT0001
```

```
FILE: MATRIX   DATA     A                YOUNGSTOWN STATE UNIVERSITY COMPUTER CENTER

0050TRM0006IDN0008TMP0001
0050KEY0002LIT0047LIT0002
0051TRM0002IDN0006LIT0001
0051TRM0006IDN0006TMP0001
0051KEY0002LIT0046LIT0002
0052KEY0018IDN0003LIT0014
0053KEY0009    0000    0000
0054KEY0017    0000    0000
```

APPENDIX E

Glossary of Compiler Terms

GLOSSARY OF COMPILER TERMS [4]

ACTION ROUTINE. Interprets the meaning of basic syntax constructions and generates matrix entries.

CODE GENERATION. Third and final phase of the compiler. Produces appropriate microprocessor code.

COMPILER. Accepts a program written in a higher level language as input and produces its machine equivalent as output.

IDENTIFIER TABLE. Created by lexical analysis. Contains all variables in the program.

KEYWORD TABLE. Permanent table. Contains all keywords for modified BASIC language.

LEXICAL ANALYSIS. Recognition of basic elements and creation of uniform symbols.

LITERAL TABLE. Created by lexical analysis. Contains all constants in the source program.

MATRIX. Intermediate form of the program which is created by action routines, and is then used for code generation.

PARSE TABLE. A list of the tokens as they appear in the source program. Created by lexical analysis.

SOURCE PROGRAM. The higher level language program used as input to the compiler.

SYNTAX ANALYSIS. Recognition of basic constructs and associated meanings according to the rules of syntax for the source language.

TERMINAL TABLE. Permanent table. Lists all the operators and special symbols for the source language.

TOKEN. Basic elements of the source program. They are identifiers, literals, terminal symbols, and keywords which are delineated by blanks, operators, and special symbols.

UNIFORM SYMBOL TABLE. Abbreviated form of the parse table with each entry containing the identification of the table to which it belongs and its index in that table.

REFERENCES

[1]. Individual Learning Program-Microprocessors, Heathkit Continuing Education Series, Heath Company, Benton Harbor, Michigan, 1977.

[2]. Heathkit Manual for the Microprocessor Trainer Model ET-3400, Heath Company, Benton Harbor, Michigan, 1977.

[3]. BASIC-PLUS Language Manual, Digital Equipment Corporation, Maynard, Mass., May 1971; 4th revision May 1975.

[4]. Donovan, John J., Systems Programming, McGraw-Hill Book Company, New York, 1972.

[5]. IBM Virtual Machine Facility/370: CMS User's Guide, International Business Machines Corporation, Poughkeepsie, New York, 1976.