NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A CROSS COMPILER AND PROGRAMMING SUPPORT SYSTEM FOR THE HP41CV CALCULATOR

by

James Norman Richmann

September 1981

Thesis Advisors: S. H. Parry
R. H. Shudde

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Cross Compiler and Programming Support System for the HP41CV Calculator		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1981
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James Norman Richmann		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE September 1981
		13. NUMBER OF PAGES 242
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Calculator, Cross Compiler, HP41CV Programmable Calculator, Optical Bar Code

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

With growing Army-wide use of programmable calculators, a system is needed to support the programming and testing of calculator software. This thesis provides a FORTRAN IV program to enable an operations research analyst to more efficiently write and document HP41CV calculator programs. Optical bar code readable by the HP41CV is generated by the program. Also given is an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

IBM EXEC II program which provides an interactive programming environment including on-line, self contained instructions. To illustrate the use of the system and the quality of the finished bar code and calculator program listings, examples are given including single variable statistics and linear programming. A final example provides a set of short utility routines which illustrate how programs can be developed for use in a calculator read-only-memory.

A Cross Compiler and Programming Support System for the HP41CV Calculator

by

James Norman Richmann
Captain, United States Army
B.S., Iowa State University, 1971

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September, 1981

Author: James Richmann

Approved by: D. H. Parry
Thesis Advisor

Roy H. Stucke
Co-Advisor

Kneale T. Marshall
Chairman, Department of Operations Research

W. M. Woods
Dean of Information and Policy Sciences

ABSTRACT

With growing Army-wide use of programmable calculators, a system is needed to support the programming and testing of calculator software. This thesis provides a Fortran IV program to enable an operations research analyst to more efficiently write and document HP41CV calculator programs. Optical bar code readable by the HP41CV is generated by the program. Also given is an IBM EXEC II program which provides an interactive programming environment including on-line, self contained instructions. To illustrate the use of the system and the quality of the finished bar code and calculator program listings, examples are given including single variable statistics and linear programming. A final example provides a set of short utility routines which illustrate how programs can be developed for use in a calculator read-only-memory.

TABLE OF CONTENTS

I. INTRODUCTION - 8

II. THE PROGRAMMING ENVIRONMENT - - - - - - - - - - - - - - 23

A. CHAPTER OVERVIEW - - - - - - - - - - - - - - - - - - 23

B. STRUCTURED PROGRAMMING WITH THE HP41CV - - - - - - 24

1. The Need for Structure - - - - - - - - - - - - - 24

2. Fundamental Limitations of Calculators - - - 25

3. Modular Design - - - - - - - - - - - - - - - - - - 27

4. Control of Program Flow - - - - - - - - - - - - - 28

5. Clarification of Program Structure - - - - - - 29

6. Data Types and Indirect Addressing - - - - - - 31

C. ADDITIONAL CRITERIA FOR PROGRAM EVALUATION - - - 33

1. User Friendliness - - - - - - - - - - - - - - - - 35

2. Execution Speed - - - - - - - - - - - - - - - - - 37

D. A PROGRAMMING SUPPORT SYSTEM - - - - - - - - - - - 39

1. A Cross Compiler and Bar Code Generator - - - 40

2. A Calculator Emulator - - - - - - - - - - - - - - 41

3. A Higher Level Language Compiler - - - - - - - 42

APPENDIX A: SINGLE VARIABLE STATISTICS EXAMPLE - - - - - 44

APPENDIX B: LINEAR PROGRAMMING EXAMPLE - - - - - - - - - 72

APPENDIX C: SUBROUTINES FOR READ ONLY MEMORY - - - - - 107

APPENDIX D: THE CROSS COMPILER PROGRAM AND COMMAND PROCESSOR - - - - - - - - - - - - - - - - - 128

LIST OF REFERENCES - - - - - - - - - - - - - - - - - - 237

INITIAL DISTRIBUTION LIST - - - - - - - - - - - - - - - 238

LIST OF FIGURES

1. Programming Environment Command Menu - - - - - - - - 12

2. List of Commands - 13

3. On-Line Introductory Material - - - - - - - - - - - - - 15

4. Components of Program Documentation - - - - - - - - - 21

5. Example Program to Add n Numbers - - - - - - - - - - - 30

6. Program to Recall an Element of a Matrix - - - - - 34

I. INTRODUCTION

For the Army to fight effectively in a resource scarce environment, the quantitative decision making techniques of operations research are important skills for Army staff officers. Staff officers are expected to be able to put numbers in their estimates when briefing commanders. They are expected to be able to measure and evaluate complex operations and subordinate units. They are expected to be frugal managers of time and money. And above all, staff officers must be able to apply sound, quantified reasoning in planning how to win the air-land battle.

The use of hand-held programmable calculators by Army staff officers has the potential for improving the use of quantitative decision making techniques throughout the Army. Faster and more accurate than paper and pencil, the calculator is less expensive and more portable than larger computers. Even when compared to the latest micro-computer systems or to portable terminals used for distributed data processing, the hand-held programmable calculator offers advantages in cost, reliability, power consumption and emission of electromagnetic radiation. Hand-held programmable calculators have already been successfully used by soldiers in the field for applications in artillery fire direction, surveying, and navigation. In addition, large numbers of Army officers own their own pocket calculators and routinely use them for staff planning and reporting functions.

In January of 1981 the U. S. Army Command and General Staff College at Fort Leavenworth, Kansas selected a programmable calculator for the Combined Arms and Services Staff School (CAS ^{3} .) Using both resident and non-resident instruction, this course is designed to teach all Army captains staff techniques and procedures. As a significant part of the curriculum, the students are introduced to subjects such as statistics and regression, decision theory, combat modeling and linear programming. Considering the large number of officers projected to attend this course in future years, this course represents the most widespread training in operations research techniques ever attempted by the Army. The decision to provide a sophisticated calculator to these students on an experimental basis was made for two fundamental reasons. First, the availability of a calculator with immediate field utility should motivate the student to apply the quantitative techniques as compared to the student who would be forced to do all calculations by hand. Second, the power of the calculator permits classroom discussion of techniques such as linear programming and regression which are very difficult and time consuming to perform manually.

This thesis documents the author's work to support the use of a calculator in the Combined Arms and Services Staff School. Initially, the intent was to produce a series of lesson materials incorporating the use of the calculator on a series of operations research topics which have immediate application for the Army division level staff officer. Instead, the work accomplished focused on the design and construction of a system to make the programming and testing of calculator programs easier and more efficient. Except for the introduction, this thesis is written for the person wishing to implement the programming support system described. The implementor must have a detailed knowledge of the instruction set and programming characteristics of the HP41CV calculator as described in Wickes [Ref. 1: pp. 6-20]. For the eventual user of the system, as compared to the implementor, the system itself provides on-line documentation on how to use the system and what commands and options are available. Figure 1 shows the command menu displayed on the terminal screen by this interactive program;

Figure 2 gives a more detailed explanation of each of the commands; and Figure 3 displays the on line introductory material that is provided to new users of the system. For the user, a knowledge of the information contained in the calculator owner's handbook [Ref. 2] is sufficient to begin writing calculator programs using the support system described.

The calculator selected by the Command and General Staff College, the Hewlett-Packard HP41CV, typifies the state of the art in off-the-shelf calculator technology. While not without disadvantages, this calculator was selected because of its power and features which make it easier for Army staff officers to use. First and most important of these features is the ability of the calculator to manipulate alphabetic characters in addition to numeric data. The calculator can display the name of a variable when input data is required or label output when the calculation is completed. With this feature, the calculator helps the user know what data to input or what action to take next. It also helps alleviate the need for constant reference to printed instructions which are difficult to use under field conditions.

```
HP41C CROSS COMPILER ......... program name ..........EDITION=17 SEP 81
SELECT DESIRED COMMAND FROM THE FOLLOWING:

PF-KEY   COMMAND  CODE  ACTION TAKEN BY PROGRAMMING  COMMAND SYSTEM

PF13      STOP      S    GETS YOU OUT OF THE HP41C CROSS COMPILER
PF14      HELP      H    SHORT EXPLANATION OF HOW TO USE THE CROSS COMPILER
PF15      ENTER     E    INTERACTIVE PROGRAM ENTRY (NO FILE CREATED)
PF16      BAR       B    SUBMIT JOB FOR PHYSICAL PRODUCTION OF BAR CODE
PF17      NEW       N    BEGIN WORK ON A NEW PROGRAM OR NAMED SUBROUTINE
PF18      DIREC     D    DIRECTORY OF COMMANDS
PF19      LIST      L    DISPLAY  NAMES   OF HP41C PROGRAMS ON DISK
PF20      OCOMP    C O   OFFLINE COMPILE AND AUTO GENERATION OF BAR CODE
PF21      PRINT     P    PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM
PF22       *        *    RESERVED FOR FUTURE USE BY HP41 EMMULATOR
PF23      COMP      C    COMPILE A SOURCE LISTING ON CMS DISK
PF24      XEDIT     X    EDIT THE PROGRAM USING THE CMS FULL-SCREEN EDITOR
          ERASE          ERASE THE SOURCE FILE,LISTING FILE AND TEXT FILE
          CMS            ALLOWS EXECUTION OF ANY VALID CMS COMMAND
          CP             ALLOWS EXECUTION OF ANY VALID CP COMMAND
INPUT COMMAND:
```

Figure 1: Programming Environment Command Menu

PF-KEY	CMD	CODE	ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM
PF13	STOP	S	THIS COMMAND IS USED WHEN YOU WISH TO STOP PROCESSING HP41C PROGRAMS AND RETURN TO CMS. IF YOU ARE EXECUTING A FUNCTION THAT WAS INVOKED FROM THE COMMAND MENU, IN MOST CASES PF13 WILL RETURN YOU TO THE MENU, AND BY PRESSING PF13 AGAIN YOU WILL RETURN TO CMS.
PF14	HELP	H	THIS COMMAND IS USED TO DISPLAY THE DETAILED EXPLANATION OF THE MENU COMMAND PROCESSOR AND ITS AVAILABLE COMMANDS. IF YOU HAVE QUESTIONS ABOUT THE PROCESS OF WRITING ACTUAL HP41C PROGRAMS YOU SHOULD CONSULT THE HP41 OWNER'S HANDBOOK.
PF15	ENTER	E	THIS COMMAND IS USED TO ENTER A PROGRAM USING THE CROSS-COMPILER IN AN INTERACTIVE MODE. THE ADVANTAGE OF THIS MODE IS THAT ANY SYNTACTICAL ERRORS IN THE HP41C PROGRAM ARE IMMEDIATELY IDENTIFIED BY THE CROSS-COMPILER AND AN ERROR MESSAGE IS SHOWN ON THE SCREEN. THE DISADVANTAGE IS THAT THE USER IS TOTALLY RESPONSIBLE FOR UPPER AND LOWER CASE BEING ENTERED PROPERLY.
PF16	BAR	B	THIS COMMAND IS USED ONCE THE HP41C PROGRAM IS WRITTEN AND COMPILED WITHOUT ERRORS. IT SUBMITS A JOB TO MVS BATCH FOR THE PHYSICAL PRODUCTION OF THE BAR CODE.
PF17	NEW	N	THIS COMMAND IS USED TO DIRECT THE ATTENTION OF THE COMMAND PROCESSOR TO A NEW HP41 PROGRAM SOURCE FILE. WHEN USED TO INITIATE NEW HP41C PROGRAMS, IT AUTOMATICALLY INSURES THAT A NEW FILE IS CREATED WITH FILETYPE "HP41" AND PROMPTS THE USER FOR THE PROGRAM TITLE WHICH IS THE MANDATORY FIRST LINE OF EVERY HP41C SOURCE CODE FILE.
PF18	DIREC	D	THIS COMMAND DISPLAYS THE FULL COMMAND MENU. IT HAS PRIMARY USE WHEN YOU FINISH AN OPERATION THAT FILLS THE SCREEN WITH TEXTUAL MATTER AND YOU RECEIVE ONLY THE PROMPT "INPUT COMMAND".

Figure 2: List of Commands

PF-KEY	CMD	CODE	ACTION TAKEN BY PROGRAMMING COMMAND SYSTEM
PF19	LIST	L	THIS COMMAND DISPLAYS "FLIST" FOR THOSE HP41C PROGRAMS THAT ARE ACTIVE ON YOUR A DISK. FROM THIS LIST, YOU CAN ERASE OLD PROGRAMS TO RELEASE DISK STORAGE, CHANGE THE NAME OF PROGRAMS, OR EXAMINE THE CONTENTS OF ANY PROGRAM.
PF20	OCOMP	C O	THIS COMMAND IS USED TO PRODUCE AN "OFFLINE" COMPILE. THE PROGRAM LISTING IS AUTOMATICALLY PRINTED IN HARD COPY ON THE HIGH SPEED PRINTER. IF THE COMPILE WAS WITHOUT ERROR THE BAR CODE IS AUTOMATICALLY PRODUCED.
PF21	PRINT	P	THIS COMMAND PRINTS A COPY OF THE "LISTING" FILE ON THE HIGH SPEED PRINTER. IF YOU WISH TO HAVE A PRINTED COPY OF THE SOURCE CODE WITHOUT THE CROSS-COMPILER'S FEEDBACK, IT IS BEST TO SIMPLY PRINT THE SOURCE CODE CMS FILE BY ISSUING THE CMS PRINT COMMAND.
PF22	GO	G	THIS COMMAND IS USED TO INVOKE THE HP41C EMMULATOR PROGRAM WHICH ALLOWS YOU TO TEST EXECUTION OF THE PROGRAM ON THE LARGE COMPUTER. THE EMULATION PROGRAM WILL EXECUTE THE PROGRAM EXACTLY AS YOUR CALCULATOR WOULD. THIS COMMAND HAS NOT BEEN IMPLEMENTED AS OF 17 SEP 81.
PF23	COMP	C	THIS COMMAND IS USED TO INVOKE THE CROSS COMPILER TO TRANSLATE AN HP41C PROGRAM WRITTEN ON CMS DISK IN SOURCE CODE FORM. AFTER THE COMPILE THE USER IS AUTOMATICALLY PLACED IN THE CMS BROWSE MODE FOR THE OUTPUT "LISTING" FILE THAT RESULTED FROM THE COMPILE.
PF24	XEDIT	X	THIS COMMAND IS USED TO INVOKE THE FULL-SCREEN EDITOR TO MAKE MODIFICATIONS TO THE HP41C SOURCE CODE FILE.

Figure 2 (Continued)

```
                HP41C CROSS COMPILER COMMAND PROCESSOR

YOU ARE CURRENTLY EXECUTING A CMS EXEC FILE THAT MAKES IT EASY TO INVOKE
THE HP41C CROSS COMPILER AND WRITE  PROGRAMS USING CMS AND THE IBM 3278
DISPLAY TERMINAL.  COMMON PROGRAMMING REQUIREMENTS SUCH AS EDITING CAN
BE ACCOMPLISHED IN THREE WAYS:

      --USING THE PROGRAMMED FUNCTION KEYS  (PF KEYS)
      --USING A SHORT COMMAND WORD
      --USING A ONE OR TWO LETTER MNEMONIC CODE

THE COMMAND ACTIONS AND THEIR ASSOCIATED PF KEYS AND CODES ARE ALL GIVEN
IN A DIRECTORY WHICH IS DISPLAYED WHEN THE COMMAND PROCESSOR IS WAITING
FOR YOUR INPUT.

IN ORDER TO  GO FROM A PROGRAM IN  YOUR HEAD TO THE  FINISHED BAR CODE
THERE ARE THREE MAIN STEPS:

      (1) EDIT.  THE PROGRAM MUST BE PREPARED AS INPUT TO THE CROSS
                 COMPILER.  THE EASIEST WAY TO DO THIS IS WITH THE
                 CMS XEDIT FACILITY.

      (2) COMPILE.  THE PROGRAM MUST BE PROCESSED BY THE CROSS-COMPILER.
                    THE CROSS-COMPILER IS ACTUALLY A FORTRAN PROGRAM
                    WHICH PRODUCES TWO CMS FILES AS OUTPUT.   BOTH
                    THESE FILES HAVE THE SAME NAME AS YOUR PROGRAM NAME,
                    BUT HAVE DIFFERENT FILE TYPES.  THE "LISTING" FILE
                    SHOWS THE RESULTS OF THE COMPILE STEP INCLUDING ANY
                    ERRORS, AND THE "DATA" FILE IS A FILE OFZERO'S AND
                    ONE'S USED BY THE BAR CODE GENERATOR.

      (3) BAR.  THE "DATA" FILE FROM THE COMPILE STEP IS USED AS INPUT
                TO PRODUCE THE ACTUAL BAR CODE.  YOU SHOULD NEVER PER-.
                FORM THIS STEP UNTIL YOUR PROGRAM HAS SUCCESSFULLY
                COMPILED WITHOUT ERRORS.  THIS STEP IS DONE BY THE
                BATCH PROCESSOR AND IT MAY TAKE SEVERAL HOURS TO GET
                YOUR FINISHED BAR CODE.
```

Figure 3: On-Line Introductory Material

A second important feature is the multiplicity of means by which programs can be entered into the calculator. Magnetic cards, read only memory, and optical bar code are all available and each has advantages depending on the situation. For the long term, read only memory offers the ability to retain very large programs (in excess of 8000 bytes) and the simplest and most reliable means of entering programs into the calculator under field conditions. For the short term, optical bar code offers the least expensive method of reproducing and distributing calculator software that has not been subject to extensive field testing. In addition, as shown in this thesis, the optical bar code can provide an important link between a main-frame computer and the hand-held calculator.

A third important feature of the HP41CV is its relatively large memory capacity as compared to programmable calculators such as the Texas Instruments TI-59. A large amount of memory permits the solution of larger, often more realistic problems than could previously be solved on a hand-held device. A demonstration program given in this thesis for linear programming is an example of an application where the full memory capability of the HP41CV is required to be able to solve realistic problems.

To take advantage of the calculator's unequalled economy and portability, the operations research analyst is challenged to overcome its limits of speed and memory capacity. The preparation of calculator software is as difficult, if not more so, than the preparation of software for larger computers. To accomplish the most possible with the handheld device, the calculator programmer is often forced to write programs which are very difficult to comprehend when examined by other programmers. As Dahl, Dijkstra and Hoare [Ref. 3: pp. 1-10] point out there are limits to human competence which interfere with the programming process. In the past, with less mature calculators which constrained the typical program to a few hundred program steps, these limits to human competence were neither as apparent nor as economically important as they are with the HP41CV. Accordingly, it is not envisioned that the average Army officer who uses the HP41CV on real world problems which push the calculator to the limits of its capability would write their own programs. In particular, it was never intended that the students in the Combined Arms and Services Staff School would be taught calculator programming. It is a tribute to the power of the device and the quality of the calculator software when a relatively inexperienced user can run complex programs using little more than the digit entry keys and the run-stop key on the calculator. This does not mean that the user must not have a clear understanding of his problem or the solution technique, but rather it means that the calculator should not require programming skill or extensive training prior to application.

The growing complexity of calculator programs described above and the realization that calculator programs for Army field use are not programmed in the field, suggest the need for a system to support the development, distribution and maintenance of calculator software. An operations research analyst or other professional programmer must be able to more efficiently prepare calculator programs than by keying them into the hand-held device. By preparing the programs initially on a larger computer, such as the IBM 3033, the programmer can use the speed and storage capability of the larger machine to great advantage. In addition, the availability of a full-screen video text editor speeds the process of program revision and maintenance. By providing a capability to integrate comments directly into the source code on the larger computer, program documentation is more easily provided. Essentially the idea is that a programmer would write the calculator program using a terminal connected to a large computer. After the calculator program is entered into the large computer, a compiler program running on the large computer would check the calculator program for errors and convert the mnemonic instructions into the "key codes" which are the numeric instructions actually executed by the calculator. Then an emulator program running on the large computer would take the numeric instructions from the compiler and execute the program--in effect making the large computer produce the same effects as the calculator only much faster and more efficiently for the programmer. Finally, when the program has been written and tested on the large computer, optical bar code is produced which allows for the economical distribution and use of the program in the field. To encourage the calculator programmer to use the system described, this process should occur in an interactive programming environment in which the user can move from one step to another by issuing simple commands such as those listed and described in Figure 2 and receive help or on line instruction whenever desired. Under this proposed system, the advantages of both the larger computer and the hand-held calculator are used appropriately in a mutually supporting manner. This thesis presents two of the components of this proposed system. First, an IBM EXEC II program is given which provides an interactive programming environment for users operating under IBM's Conversational Monitor System (CMS.) A short discussion of the design of this program and a complete copy of the source code is contained in Appendix D to this thesis. Secondly, a cross compiler written in IBM standard FORTRAN IV is provided for translating calculator mnemonic instructions into the key codes necessary for use by the emulator and also for the production of optical bar code. The term cross compiler refers to the fact that the program runs on one machine (the larger computer) but compiles programs for another machine (the calculator.) A discussion of the design of this program and a complete copy of the source code is contained in Appendix D. To make the program easier to understand and adapt to new requirements, it is modularized into 24 subroutines and is heavily commented.

To illustrate the use of the system, two of the six example programs originally planned are provided in this thesis. Revised plans now call for the remaining four example programs to be issued at a later date as Naval Postgraduate School technical reports. Because the reasons for the delay constitute some of the most important lessons learned from this thesis research, Chapter 2 documents the process with a technical discussion of the factors involved. The major conclusions described in Chapter 2 are the need for a prioritized list of criteria with which to evaluate calculator programs and the need for more structure in the programming process. Chapter 2 is technically oriented and assumes the reader is familiar with the concepts of structured programming.

Each of the calculator program examples is described in a separate appendix in which the documentation listed in

1. Program Description
2. Sample Problem
3. User's Guide
4. Source Code Listing with Comments
5. Bar Code

Figure 4: Components of Program Documentation

Figure 4 is provided. The first example on single variable statistics is documented in Appendix A and uses the calculator in an area where calculators have long been used, but does so in a way that shows the unique capabilities of the

HP41CV. A second example on linear programming is documented in Appendix B and illustrates an area where calculators have not received widespread application. Most calculator linear programs which have been published to date have been either incomplete algorithms or have been limited to very simple problems.

A third example, which by its nature does not conform to the documentation standards outlined above, describes a set of utility routines which could be distributed in read only memory. Programs for read only memory have different characteristics from other calculator programs and Appendix C is provided to illustrate some of these differences.

II. THE PROGRAMMING ENVIRONMENT

A. CHAPTER OVERVIEW

This Chapter examines calculator programming within the context of the author's experience in preparing HP41CV programs in support of the Combined Arms and Services Staff School. With the advanced capabilities and features of the HP41CV, it was hoped that a complete package of software could be prepared quickly. To document why this did not occur, this chapter will examine strengths and weaknesses of the calculator in relationship to a collection of techniques referred to in computer science as structured programming. For the reader unfamiliar with this term, the previously cited work by Dahl, Dijkstra, and Hoare [Ref. 3] is recommended. This chapter is technically oriented and does assume familiaritiy with structured programming concepts.

When programming calculator programs for personal use, most programmers, including the author, do not find the task difficult. Programming a hand-held calculator with the capabilities and features of the HP41CV can be a rewarding experience. It is rewarding to master the algorithm of an operations research technique on a hand-held device. The educational value in programming the calculator has been recognized by many educators, including Hamming [Ref. 4: pp. 2-3] and Weir [Ref. 5: pp. xii-xiii]. Providing a program for general distribution which makes optimum use of the calculator is quite a different situation. It was the author's experience that programs, which gave correct answers when used by the author, often had to be completely re-written several times before being acceptable. This problem became more acute as the size of the programs grew beyond 400 program steps, for at that size it became increasingly difficult to modify programs without affecting the total design. The major conclusions described in this chapter are the need for a prioritized list of criteria with which to evaluate calculator programs and the need for more structure in the programming process.

B. STRUCTURED PROGRAMMING WITH THE HP41CV

1. The Need for Structure

To increase the efficiency of the programming process, a collection of techniques known as structured programming has received widespread attention in the computer science community. While there is no one definition of structured programming, it does require three essential characteristics. First, there must be a logical structure to the program which reflects the nature of the problem to be solved and any constraints imposed upon the solution. Second, the systematic process of stepwise refinement is used to limit the complexity of program segments. Third, the programming language must reflect the logical structure of the program and assist in stepwise refinement. These three characteristics represent not so much a detailed recipe for program development as they do a philosophy of how programs can be more efficiently written. It was with this philosophy in mind, that a calculator programming support system was proposed which could take into account the strengths and weaknesses of the calculator; balance the structured programming philosophy with the other criteria listed below; and thereby solve the problems encountered in writing calculator software for the Combined Arms and Services Staff School.

2. Fundamental Limitations of Calculators

Writing programs to solve complex problems on a hand-held calculator is difficult both because of inherent limitations in the calculation speed and memory capacity of the machine and also the inability of the calculator's native programming language to directly support structured programming constructs. In many respects, the task is similar to writing assembly level language programs for larger computers. Calculator programming features a powerful instruction set including advanced mathematical functions but lacks any ability to refer to variables by name instead of storage address. Like assembly language, the calculator's programming language consists of short mnemonic instructions typically followed by the storage location of the data to which the operation is to be applied. While a large amount of computer programming is still done in assembly language, it is generally accepted that programming in a higher level language such as FORTRAN is preferable. Programs written in an assembly language take more time to write and are not as easily changed as higher level language programs. Also, because they depend on the instruction set of a particular machine, they can not be easily transfered from one computer to another. These same disadvantages apply to calculator programming. In addition, because the hand-held device does not have the speed and memory capability of the larger machine, the calculator programmer must be even more mindful of the need to optimize his program to save program steps and execution time.

3. Modular Design

The HP41CV supports structured programming as well or better than any other hand-held calculator. As described in the owner's manual [Ref. 2: pp. 177-196], the machine primitive instruction XEQ encourages the construction of modular programs using calculator subroutines. Each subroutine can be a self-contained unit capable of being written and tested independently and used by multiple programs. This modularity is most strongly encouraged when routines in read only memory are used, for then the application programmer can significantly reduce the number of program steps in his own program. This modularity, however, is not complete, since all variables are globally referenced and can be changed deliberately or inadvertently by any subroutine. This problem is no more apparent than with the use of read only memory, since one of the most limiting factors in using the read only memory programs as subroutines is conflict in the use of common registers. Also, unlike the modularity required in truly structured programs, there is no restriction limiting a subroutine to a single entry and a single exit point. In structured programs, such limits on entry and exit serve to define the fundamental building blocks by which stepwise refinement is made possible. With the calculator, however, multiple entry and exit points are most useful for allowing a common routine to handle a duplicity of problem conditions. In this thesis, for example, programs are given for which two standard entry points are provided. One entry point uses an alpha-numeric label and an audio prompt to speed data entry, while a second entry point uses the alpha-numeric label but suppresses the audio tone. After data entry, the value entered is displayed, and the user is required to verify the accuracy of the data entered. By using the same subroutine with different entry points, memory space is saved overall at the sacrifice of the structured programming philosophy.

4. Control of Program Flow

A basic deficiency prohibiting the HP41CV from directly supporting structured programming is the way in which program flow is controlled. Programming languages which support structured programming typically have instruction constructs such as WHILE--ENDWHILE, REPEAT--UNTIL, or LOOP--QUIT--ENDLOOP which make programming loops clear and concise. Constructs such as IF--THEN--ELSEIF--ELSE--ENDIF and the CASE statement make the evaluation of conditional expressions efficient and relatively error free. Also, structured programming languages typically discourage the use of GOTO unconditional transfers because they lead to confusing code. In contrast, the HP41CV programmer must write his own looping constructs and his own conditional evaluation constructs using machine primitive instructions which somewhat obscure the program's basic objective and flow of control. In addition, it is difficult to avoid disturbing pending operations in the stack registers when a conditional statement must be evaluated. As can be seen by the short program shown in Figure 1, the notation of the programming language does not permit structured program flow.

5. Clarification of Program Structure

Because no calculator, including the HP41CV, supports named variables, the use of comments as an integral part of the calculator program is vital if the logical structure of the program is to be made clear as required by structured programming. Comments should provide the variable names when storing and recalling data; they should provide clarification of program flow; and they should mark subroutine boundaries and entry and exit points to make it easier to identify segments of the program. With the HP41CV's stack oriented architecture, it is also frequently useful to display the names of the contents of each of the

Given the number n in the x-register, this program fragment will sum the data values stored in memory locations 1 through n.

Instruction	Comment
LBL "SUM	To execute press "XEQ SUM".
1E3	
/	
1	
+	Establishes a loop counter.
0	Clears x and pushes loop
LBL 00	counter into y.
RCL IND Y	Recall the next data value.
+	Accumulate the sum.
ISG Y	Increment the loop counter.
GTO 00	If more data remains, branch;
RTN	else, quit and display sum.

Figure 5: Example Program to Add n Numbers

stack registers. In Appendix C on common subroutines with read only memory application, a shell sort [Ref. 6: pp. 84-95] routine is given which employs the technique of using comments to display the names of the variables on the stack register.

6. Data Types and Indirect Addressing

Calculator programs represent more than a sequence of keystrokes; they also represent the manipulation and transformation of data. For maximum efficiency, the manipulation of data should be structured so as to prevent common programming errors. For this reason, most computer languages which directly support structured programming enforce data type correspondence between data and operations. Frequently the formal declaration and initialization of variables is also required. The HP41CV handles two types of data--real numbers and alphanumeric characters. While no formal declaration of variables is required, type checking is done automatically and is transparent to the user. Any attempt to perform an arithmetic operation on alpha-numeric data will result in the message "ALPHA DATA" and the program will halt.

Because there is no formal declaration of variables, the programmer writing programs for the HP41CV must use extreme caution in managing his data set and insuring that the numbers stored and recalled by the calculator program are in fact the data elements desired. A typical example of an improper data reference occurs when a program is using indirect addressing and attempts to store or recall data from a non-existent data register. This programming error is so common that a special error message "NONEXISTENT" is provided by the calculator when this error is detected. Indirect addressing is an important feature which gives the calculator a considerable amount of power and flexibility, but also represents an additional responsibility for the programmer to explicitly control. On the HP41CV all indirect addressing calculations must be specifically provided by the application program--there are no vector or array data types such as usually found with higher level languages. In an attempt to make indirect addressing more transparent to the programmer, an experimental subroutine was prepared to recall an arbitrary element of a matrix stored as a two dimensional array. This subroutine, which is shown in Figure 2, was used in a simultaneous differential equation combat model and the results evaluated. It accomplished the task, but slowed the execution of the program considerably (resulting in an overhead of 10.5 seconds of extra execution time for every 100 subroutine calls) and did not significantly improve the size or legibility of the application program. Accordingly, this technique is not recommended and indirect addressing remains a task that must be treated explicitly by the application programmer.

C. ADDITIONAL CRITERIA FOR PROGRAM EVALUATION

Calculator programming in many respects resembles a multi-criteria decision problem. On the surface the criteria for program effectiveness are quite straight forward--the program must yield the correct answer, run quickly, require the fewest possible memory registers and be user friendly. Unfortunately, these objectives often conflict and can not always be simultaneously achieved. In particular, the principles of structured programming are often in conflict with the desire to reduce the size of programs and increase their execution speed. It is also true that the objectives of structured programming concern the process of writing programs, whereas the additional criteria listed concern the final program product itself and are therefore logically considered separately. Attempting to achieve all criteria at once can lead to failure, and some tradeoffs must be considered to evaluate programs and guide program development. The following criteria represent

Entry to this routine assumes the x register contains the column number and the y register contains the row number. The base address must be stored in R04 and the dimension of the matrix must be stored in R05.

```
 1 LBL "RCLM
 2 RCL 04          (BASE ADDRESS REGISTER
 3 +               (ADD BASE TO COLUMN NUMBER
 4 X<>Y            (RECALL THE ROW NUMBER
 5 1
 6 -
 7 RCL 05          (DIMENSION OF THE MATRIX
 8 *
 9 +               (ADDRESS IS NOW IN X REG
10 RCL IND X       (RECALL THE DATA DESIRED
11 RTN
12 END
```

Figure 6: Program to Recall an Element of a Matrix

"lessons learned" in developing application programs as examples for this thesis.

1. User Friendliness

User friendly programs consider the application environment and do not task the user to be all knowing or without error in entering data. While individuals differ greatly with experience, the average user will make frequent errors in entering data with the hand-held calculator's small keyboard. In talking with officers who had used the TI-59 calculator in the field for fire direction, it was discovered that most preferred to use the printer with the calculator because it allowed data to be checked after entry. This was in spite of the fact that the printer and calculator combination is more costly, less portable and less suitable for use in the field than the calculator alone. In short, user friendliness was more important than these other criteria. For this reason, it should be mandatory that any calculator programs intended for Army use in the field must allow the verification of data after entry. Because the use of the printer obviates many of the advantages of the hand-held calculator, the printer should not be required for this verification. One of the considerable advantages of the HP41CV is that the large amount of program memory makes it possible to store the input values and perform this verification. However, programs written with this criteria in mind may not appear be most efficient to the casual observer.

Another important aspect of user friendliness is limiting the complexity of the calculator and the actions required to get results. The typical Army officer has little appreciation for the multitude of scientific and mathematical functions labeling the keys of the HP41CV. Yet the common programming practice of using the top two rows of calculator keys to indicate the identity of a variable either upon input or output increases confusion over the use of the function keys. This works as follows: When local alphabetic labels are used in a program to represent entry points by which a user indicates the identity of an input variable or requests a particular output variable, then the first two rows of keys on the HP41CV become subroutine execution keys pointing to these local labels when the calculator is in user mode. This feature was very important on the HP67 and TI-59 where the lack of alpha-numeric capability required this method of program execution in order to most easily determine the identity of the input or output value, but it is less important on the HP41CV. It is almost always true that a program which requires the use of local labels is harder to use, and requires more frequent reference to the user instructions than a program which uses only the run-stop key and properly prompts the user and labels output values.

2. Execution Speed

The second most important criteria for a calculator program is that it must yield results relatively quickly. In preparing example programs for this thesis, this point became very clear when testing two particular programs. One program, a simultaneous differential equation combat model, required in excess of 150 data values in order to yield results. It should be noted that it was only with the introduction of the HP41CV that it became feasible to consider such large problems on a hand-held device. To accomodate the size of the model, the program was written so as to economize on program steps at the expense of increased execution time. It became immediately obvious upon initial testing that this had been the wrong priority--for users of the program were not impressed with either the use of the calculator or the utility of the combat model. If such user acceptance is not present, then the calculator program will remain unused, no matter how elegant the design to conserve memory. In contrast, the linear programming example given in Appendix B was written so as to emphasize speed even if it meant including code redundancy. This program has been well received in part because it is so much faster than paper and pencil methods.

The easiest and most effective technique that is useful in increasing speed is to decrease the number of program steps that the calculator must process inside program loops. For example, if two different program options require similar but slightly different actions within a program loop, it is tempting to insert a program flag check and branching instructions within a loop so as to use the same loop for both conditions. But this means that the calculator must test the flag and branch inside the loop even though the program is probably shorter overall.[1]

Instead, if the application permits, the memory capacity of the HP41CV can be used to best advantage by testing the flag once and then providing separate program loops for the two conditions. Again, this does not appear elegant to the casual observer, but it may result in a more successful program overall. This principle was discovered while

----

[1] Branching is required when the flag tests either set or clear if more than one instruction is required to account for the differences in the two conditions.

programming the single variable statistics program given in Appendix A. Initially, this program used a common loop for all data input and output operations, including reviewing the input data and making individual corrections. By providing a separate, somewhat redundant loop for data correction, the time required to input data points was reduced.

D. A PROGRAMMING SUPPORT SYSTEM

Considering the structured programming philosophy discussed above in paragraph B and the additional criteria for evaluating programs listed in paragraph C, it becomes immediately obvious that programming with the calculator alone will never meet even a majority of these objectives. It must be recognized that the problem under consideration is not how the average person who owns a calculator should proceed to program it for his own personal use, but rather how the Army can best provide the most cost-effective computational resource for field use. For these reasons, a comprehensive programming support system is required. The programming support system outlined here will consider only the requirement for cost-effective preparation and maintenance of the calculator programs and not the broader issues of distribution and logistic support for the entire calculator system to include hardware, training materials and printed references.

1. A Cross Compiler and Bar Code Generator

The first requirement for an operational support package is to free the programmer from the limitations of the hand-held calculator itself. Even with the printer and other peripherals, the calculator is no match for the larger machine when large programs must be examined or edited. In addition, the calculator is not currently capable of producing its own optical bar code as required for economic reproduction and distribution of the software. Accordingly, a cross compiler for the HP41CV was listed as the first requirement of the programming support system. Such a cross compiler has been written and is the major outcome of this thesis effort. This cross compiler accepts an HP41CV program written in the language of the calculator and returns the finished bar code as output. Any valid HP41CV program will be processed without need for modification by the cross compiler. In addition to the basic language of the calculator, the user is allowed to inject comments directly into the source code with the use of the left parenthesis as a comment indicator mark. The ability to make comments directly in the source code makes the calculator programs more legible and more easily modified at a later date or by another programmer. Often, well placed comments can make up for a lack of structure in the program itself as far as legibility and maintainability are concerned. Having the comments directly in the source code facilitates their use and helps insure that they are as up to date as the program. For the average programmer, use of unmodified HP41CV source code augmented with a comment indicator will represent the most common use of the cross compiler. The cross compiler is described in more detail in Appendix D including a complete listing of the source code.

2. A Calculator Emulator

After the calculator source code has been processed by the cross compiler, a need exists to be able to run the program without the wait for the generation of bar code. In addition, for the future development of read only memories for the calculator, an emulator program is required because the calculator itself can store only up to 2000 instructions in active random access memory. The read only memory can store up to four times this amount. Thus, the calculator by itself may not be capable of testing extremely large programs or programs with large amounts of constant data also stored in the read only memory. Although an emulator was not written for this thesis, the design of the cross compiler reflects the need for such a program. For example, the cross compiler generates an intermediate array of decimal integers which repesent the machine language of the HP41CV prior to conversion to binary. It was intended that these decimal integers could be used without modification or further translation within a FORTRAN computed goto statement. Thus, with the difficult translation, instruction parsing and syntax recognition already performed by the cross compiler routines, the emulator could consist of one large FORTRAN loop wherein a decimal integer was addressed in the instruction array by a program pointer variable. The integer is then immediately sent to a computed goto statement which would branch to the appropriate line of FORTRAN code which would simulate the referenced instruction, including updating the stack and the program pointer as appropriate.

3. A Higher Level Language Compiler

The final component in the calculator programming support system would be a program that would translate a higher level language such as PASCAL into HP41CV language which could then be sent to the cross compiler for verification and generation of the bar code and intermediate calculator language listings. It is the higher level language compiler that would most directly make up for the weakness of the calculator in supporting structured programming. It would be able to increase the modularity of programs, provide for named variables, make indirect addressing transparent and provide structured statements such as WHILE--ENDWHILE and IF--THEN--ELSE. Again, the design of the cross compiler anticipates this requirement and provides a considerable number of subroutines that would also be required by a higher level language compiler. These subroutines include a complete set of string functions for manipulating character data in FORTRAN and an instruction parser. Because it was envisioned that the higher level language compiler would also be able to process statements entered directly as HP41CV instructions, the cross compiler is constructed so that the routine which compiles individual lines of HP41CV source code could be called as a subroutine by the higher level language compiler. Thus, all three major components of the proposed calculator programming support system would work together efficiently.

APPENDIX A

SINGLE VARIABLE STATISTICS EXAMPLE

INTRODUCTION:

Calculating single variable statistics is one of the most frequently used applications of programmable calculators. Army division level staff officers use single variable statistics to summarize and describe data for command briefings and periodic reports. The text by Mendenhall, Scheaffer and Wackerly [Ref. 7: pp. 3-13] is recommended as an introduction to the statistical measures calculated by the program given in this appendix. This program automatically calculates:

- Mean and Median
- Sample Standard Deviation
- Sum of the Squared Deviations about the Mean
- Coefficients of Skewness and Kurtosis
- Minimum, Maximum and Range
- Histogram Cell Frequencies

A single variable statistics program has been given as an example because of its immediate utility to the staff officer and to illustrate several features of the HP41CV which make it a superior device for Army field use. The most important of the these features is alphanumeric prompting for input data values. The program given in this appendix provides an alphanumeric prompt for every input and output value and requires only the digit entry keys and run/stop key for data entry. Another important feature of the HP41CV used by this program is its large memory capacity. This program retains up to 219 data points in the calculator's memory to allow the user to review the input data and make corrections during data entry. The large amount of memory allows the calculator to sort the data and calculate the order statistics including the minimum, maximum and median. Calculation of the median is a feature of this program which distinguishes it from other calculator statistics programs. In addition, without having to re-enter the data, the histogram may be calculated with a varying number of cells or a varying cell width.

PROGRAM DESCRIPTION:

The single variable statistics program has entry points for two different techniques of data input. The fastest method, which provides both an alphanumeric prompt and an audio tone to speed data entry, may be called by execution of the program from entry point "STAT1." A slower method, which provides greater accuracy and suppresses the audio tone for classroom use, may be called by execution of the program from entry point "S1." When called from "S1," the program requires the verification of each data point after entry. The sequence of actions is as follows:

1. The calculator displays an alphanumeric prompt. As an example, "X1?" is the prompt for the first point.
2. The user enters the data value with the digit entry keys and presses the run/stop key.
3. The calculator displays the data entered with a label derived from the alphanumeric prompt. For example, "x1=3.1415" is a typical calculator response. This display is prompting the user to verify the correctness of the data displayed.
4. If the value is correct, then the user simply presses the run/stop key and the calculator advances to the next point.
5. If the value is erroneous, the user enters the correct value with the digit entry keys and then presses the run/stop key. Then the calculator will again repeat step 3 and ask the user to verify the data value. This process will continue until the user makes no modification to the data value.

To run the program from either entry point the user may use the XEQ key, or assign the entry point label ("STAT1" or "S1") to a key and execute it by pressing that key in the USER mode. Further instructions on running programs and making key assignments are contained in the calculator owner's manual [Ref. 2: pp 114-116].

In addition to the two initial entry points described above, several other alphabetic labels provide the user with functions that are called outside the normal sequence of program execution. Label "SR" provides the user with the capability to review the data stored in calculator memory, either before or after the data has been sorted. When used before the sort, the "SR" function is most useful in verifying the entire data set at one time. If used for this purpose, it should be called after all of the data has been entered and the mean of the data set is displayed with the "XBAR" label. If flag 21, the printer enable flag, is set "on" during this data review, then the calculator will stop as each point is displayed and the user may make corrections in the same manner as described above for the point-by-point verification associated with the "S1" entry point. When used after the sort, the "SR" function is most useful for displaying the order statistics for the data set. If used for this purpose, it should be called after the histogram is output--when the "CMD" prompt is displayed. If the user presses run/stop after the "CMD" prompt, the order statistics will automatically be displayed.

The design of the program, especially the data entry loop, reflects the need for calculation speed. Code redundancy exists at several points in order to reduce the need for extra flags, labels and goto statements which would slow execution during data entry. In spite of this need for speed, the summary totals needed for calculation of mean, standard deviation, skewness and kurtosis are accumulated during data entry. This is done so that these summary statistics are available with little or no wait following data entry.

A complete listing of the program registers and flags used by this program is shown at the end of the program listing.

SAMPLE PROBLEM:

In order to establish a training standard for an obstacle course, a division assistant G3 randomly selects 10 soldiers and records the time it takes each to complete the course. The following times in minutes were recorded:

2.1	2.4	2.2	2.7	2.5
2.4	2.6	2.6	2.3	2.9

Determine the summary statistics and cell frequencies necessary to plot a histogram of this data.

SOLUTION:

1. First, set the size of the calculator's data memory large enough to retain the data values. This requires at least 16 registers plus 1 for each data point, or a total of 26 in this example. Alternatively, the size of data memory may be set arbitrarily large, up to a maximum of 235 provided the user has no other programs in the calculator he wishes to retain. For this example press:

XEQ   ALPHA   SIZE   ALPHA   26

2. To call the program, determine the appropriate method of data entry and select the corresponding entry point.
Press:

XEQ	ALPHA	STAT1	ALPHA	(quick entry)
		-or-		
XEQ	ALPHA	S1	ALPHA	(classroom use)

3. The calculator will respond with the prompt "N?" asking for the number of data points. Press:

10   R/S

4. The calculator will respond with the prompt "X1?" asking for the first data point. Press:

2.1   R/S

5. If you called the program via "S1" the calculator will respond with "X1=2.100" asking for verification that the first point is correct. If not correct enter the correct value, else press run/stop.

6. The calculator will continue in the same way as steps 4 and 5 for the remaining data points until all the data has been entered. If at any time you discover that you have made an error in data entry for any point, press:

XEQ ALPHA SC ALPHA

The calculator will respond with the prompt "POINT?" asking for the number of the point in error. For example, if point number 5 were in error, you would then press:

5 R/S

Assuming you had just input a 5 as the point in error, the calculator would then respond with the prompt "X5?" asking for the correct value of point 5. Respond with the correct value and press run/stop. The calculator will then go back to the place in the data entry sequence where it left off or it will go to the calculation of the summary statistics if data entry was previously completed.

7. When data entry has been completed, the calculator will respond with the mean of the data sample labeled as follows:

XBAR=2.470

At this point, you have the option of reviewing the entire data set or continuing to calculate the remainder of the statistics. To review the entire data set, press:

XEQ ALPHA SR ALPHA

Note that if flag 21 is set on (press SF 21), the calculator will stop after each data point is displayed, permitting you to change any value simply by entering the new value and pressing run/stop.

8. After the mean is displayed with the "XBAR" label, if you simply press the run/stop key, the calculator will calculate the following statistics with the label shown: After each press R/S.

Display	Meaning
SSQD=0.521	Sum of Squared Deviations About the Mean
SX=0.241	Sample Standard Deviation
SKEW=0.170	Skewness
KURTO=2.302	Kurtosis

9. At this point the calculator will automatically sort the data. This may take from several seconds to several minutes

depending on the number of points in the data set. After the data set has been sorted, the calculator will display the median as follows:

```
MED=2.400 TO          (Press R/S)
.. 2.500
```

Two data values are displayed because when the number of data points is even, the median is not unique, but rather spans an interval from the one point listed above to the other. Many users may wish to simply take the middle of this interval as the median, but any point is technically correct in the interval. When the number of data points is odd, the median is unique and only one value will be displayed by the calculator.

10. After the median is displayed as described in step 9, the calculator will display the following statistics labeled as shown:

Display	Meaning
MIN=2.100	Minimum Value
MAX=2.900	Maximum value
RNG=0.800	Range

11. At this point the calculator will respond with "CELL?" asking for the number of cells the user desires in the

histogram. If the number of cells is not significant at this point, the calculator will pick an appropriate number if the user simply presses run/stop. For this example, press:

R/S

12. Next the calculator responds with "WIDTH" asking for the width of the cells. Simply press run/stop if you do not wish to establish the width manually. Again, you may see the width the calculator will use by pressing the clear arrow key (Unless the width is an integer, you will also need to press FIX 3 to display the decimal properly if you wish to examine the width.) For this example, press:

R/S

13. The calculator will now display the cell frequency counts as an integer count followed by the next cell boundary. The leftmost cell boundary is set equal to the minimum value and is not explicitly output. If a data point falls exactly on a cell boundary, it is counted in the left cell.

For this example, the display will show:

Display	Meaning
CNT=2 xx=2.260	Two observations between 2.1 (the minimum) and 2.26 (the cell boundary)
CNT=3 xx=2.420	Three observations between 2.26 (see above) and 2.42 (the next boundary)
CNT=1 xx=2.580	One observation between 2.42 (see above) and 2.58 (the next boundary)
CNT=3 xx=2.740	Three observations between 2.58 (see above) and 2.74 (the next boundary)
CNT=1 xx=2.900	One observation between 2.74 (see above) and 2.90 (the maximum)

14. After the last cell boundary is displayed, the calculator will display "CMD" asking the user for the next command. Frequently, the user will wish to modify the histogram by changing the number of cells or the cell width. To recalculate the histogram cell frequencies without re-entering the data press:

XEQ ALPHA AGAIN ALPHA

If no further work with the histogram is desired, the user may view the order statistics simply by pressing run/stop.

USER INSRUCTIONS: SINGLE VARIABLE STATISTICS

STEP	EXPLANATION	SEE	PRESS	RESULT
1	SET SIZE (nnn=16+NUMBER OF DATA POINTS)		XEQ "SIZE NNN	UP TO nnn = 235
2	CALL THE PROGRAM ("STAT1 IS FOR REGULAR USE) ("S1 IS FOR CLASSROOM USE)		XEQ "STAT1 -or- "S1	
3	ENTER THE NUMBER DATA POINTS.	N?	input R/S	
4	ENTER THE DATA For mistakes or to review the data see last two steps below. WHEN VERIFY MODE IS SET ON (SET BY FLAG 05 ON), AFTER EACH DATA POINT IS ENTERED, THE VALUE WILL BE ECHOED BACK BY THE CALCULATOR.	X1?, X2? ETC. x1=xx etc.	input R/S R/S -or- correct value	
5	SUMMARY STATISTICS ARE CALCULATED WHEN ALL DATA HAS BEEN ENTERED. STANDARD DEVIATION SKEWNESS KURTOSIS	XBAR=xx SSQD=xx SX=xx SKEW=xx KURT=xx	R/S R/S R/S R/S R/S	mean sum of sq dev from mean
6	CALCULATOR WILL AUTOMATICALLY SORT DATA POINTS. AND THEN DISPLAY: MEDIAN (note if N is even the median is not unique and an interval is displayed) MINIMUM MAXIMUM RANGE	PRGM MED=xx MIN=xx MAX=xx RNG=xx	 R/S R/S R/S R/S	STANDBY

USER INSRUCTIONS: SINGLE VARIABLE STATISTICS

STEP	EXPLANATION	SEE	PRESS	RESULT
7	USER OPTION TO ENTER NUMBER OF HISTOGRAM CELLS. NO INPUT IS REQUIRED.	CELL?	R/S -or- INPUT N R/S	
8	USER OPTION TO ENTER WIDTH OF HISTOGRAM CELLS. (HAS PRECEDENCE OVER NUMBER OF CELLS IF A WIDTH IS ENTERED.)	WIDTH?	R/S -OR- INPUT R/S	
9	CALCULATE HISTOGRAM (OUTPUT DATA ABOUT EACH CELL FROM LEFT TO RIGHT.)	CNT=II XX=xx	R/S R/S	CELL FREQ COUNT UPPER X-VALUE LIMIT
10	ACCEPT NEXT COMMAND	CMD	ENTER NEXT CMD	
11	RECALCULATE HISTOGRAM		XEQ "AGAIN	
12	EDIT AN INPUT VALUE AT ANY TIME PRIOR TO DATA SORT. AFTER INPUT OF NEW VALUE CALCULATOR WILL RETURN TO DATA INPUT OR CALCULATION OF SUMMARY STATS AS APPROPRIATE.	POINT? X?	XEQ "SC INPUT POINT NUMBR INPUT CORRECT VALUE	WILL REMOVE POINT

USER INSRUCTIONS: SINGLE VARIABLE STATISTICS

STEP	EXPLANATION	SEE	PRESS	RESULT
13	REVIEW DATA POINTS (OR REVIEW ORDER STATS AFTER SORT.)		XEQ "SR	

HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
              STAT1

 1 LBL "STAT1               (RECOMMENDED ENTRY POINT
 2 CF 05                    (SET VERIFY MODE OFF
 3 SF 26                    (ENABLE AUDIO
 4 GTO "SS

              S1

 5 LBL "S1                  (ENTRY POINT FOR CLASSROOM USE
 6 SF 05                    (SET VERIFY MODE ON
 7 CF 26                    (DISABLE AUDIO TONES
 8 SF 21                    (SET TO STOP DURING VERIFICATION

              "SS

 9 LBL "SS                  (ENTRY POINT FOR USER SET OPTIONS
10 CF 29                    (NO DIGIT GROUPING
11 ΣREG 10                  (ESTABLISH STATISTICAL REGISTERS
12 CF 06                    (USED BY DATA REVIEW FUNCTION
13 CF 08                    (USED BY DATA EDITING FUNCTION

              00                  DATA ENTRY (F06-CLEAR)
                                         AND
                                  DATA REVIEW (F06-SET)

14 LBL 00
15 15
16 STO 04                   (ESTABLISH INDIRECT ADDRESS BASE REG.
17 STO 00                   (INITIALIZE DATA ENTRY POINTER
18 RCL 15                   (NUMBER DATA POINTS (LAST PROBLEM)
19 CLΣ
20 "N?
21 FC? 06                   (CLEAR MEANS  DATA ENTRY,NOT REVIEW
22 PROMPT
23 1E3
24 /
25 1
26 +
27 STO 01                   (SET UP LOOP COUNTER FOR DATA POINTS
```

HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
{(-----------------------)
{                       }
{           01          }      DATA ENTRY LOOP
{                       }
{                       }
{-----------------------}
LBL 01
ISG 00                        (INCREMENT DATA STORAGE POINTER
LBL 02
RCL IND 00                    (RECALL DATA VALUE
"X
FIX 0
ARCL 01
FIX 3
ASTO 03                       (TEMP STORAGE FOR LABEL
FS? 06                        (IS THIS REVIEW OF DATA PREV. ENTERED?
GTO 03
"⊢=?
TONE 9                        (PROMPT USER FOR NEXT DATA VALUE
PROMPT
STO IND 00                    (STORE THE DATA VALUE
FC? 05                        (NO VERIFICATION OF DATA DESIRED?
GTO 04
LBL 03                        (FOLLOWING IS THE VERIFICATION ROUTINE
CLA
ARCL 03                       (RECALL THE LABEL
"⊢=
ARCL IND 00                   (RECALL THE STORED DATA
CF 22                         (CLEAR DATA ENTRY FLAG
AVIEW                         (WILL STOP FOR DATA ENTRY IF F21 SET
FC? 22                        (WAS THERE NO DATA CHANGE DURING VIEW?
GTO 04
STO IND 00                    (IF THERE WAS A NEW VALUE, THEN RECORD
GTO 03                        (IT AND GOBACK AND RE-VERIFY THE DATA.
LBL 04                        (FOLLOWING IS THE STATISTICAL ACCUM.
ST+ 10                        (STORES SIGMA X
X↑2
ST+ 11                        (STORES SIGMA X-SQUARED
LASTX
*
ST+ 12                        (STORES SIGMA X-CUBED
LASTX
*
ST+ 13                        (STORES SIGMA X-FOURTH-POWER
FS? 08                        (IS THIS A DATA REVIEW?
RTN
ISG 01                        (IF DATA ENTRY, INCREMENT INPUT CNTR.
GTO 01
RCL 01                        (AT END OF DATA ENTRY, RECALL INPUT
   (                           (COUNTER, WHICH IS A NUMBER EQUAL ONE
   (                           (MORE THAN NUMBR POINTS
```

HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
      (------------------------)
      (                        )
      (         "SM            )     CALCULATION OF SUMMARY STATS
      (                        )
      (------------------------)
 71 LBL "SM              (ENTRY ASSUMES X-REGISTER HAS A NUMBR
 72 INT                  (1 MORE THAN NUMBER OF DATA POINTS.
 73 1
 74 -
 75 STO 15               (STORES THE NUMBER OF DATA POINTS
 76 MEAN
 77 "XBAR
 78 XEQ 97               (CALL AN OUTPUT LABELING ROUTINE
 79 STO 03               (TEMP STORE FOR XBAR
 80 RCL 11               (RECALL SIGMA X-SQUARED
 81 RCL 03               (RECALL XBAR
 82 X|2
 83 RCL 15               (RECALL NUMBR POINTS
 84 *
 85 -
 86 STO 09               (TEMP STORE FOR SUM OF SQUARED
 87 "SSQD                   (DEVIATIONS ABOUT THE MEAN
 88 XEQ 97
 89 RCL 15               (NUMBER POINTS
 90 1
 91 -                    (CAN NOT USE SDEV FUNCTION BECAUSE OF
 92 /                       (NON-STANDARD USE OF REGISTERS 12-14
 93 SQRT
 94 "SX                  (STANDARD DEVIATION
 95 XEQ 97
 96 RCL 09               (SUM OF SQ DEVIATION ABOUT MEAN
 97 RCL 15               (NUMBER POINTS
 98 /
 99 STO 05               (SECOND MOMENT
100 RCL 12               (SIGMA X-CUBED
101 RCL 11               (SIGMA X-SQUARED
102 RCL 03               (XBAR
103 *
104 3
105 *
106 -
107 RCL 15               (NUMBER POINTS
108 /
109 RCL 03               (XBAR
110 3
111 Y|X
112 2
113 *
114 +
115 STO 06               (THIRD MOMENT
116 RCL 05               (SECOND MOMENT
117 1.5
118 Y|X
119 /
120 "SKEW
121 XEQ 97               (OUTPUT THE SKEWNESS OF THE DATA
122 RCL 13               (SIGMA X-FOURTH-POWER
123 RCL 12               (SIGMA X-CUBED
124 RCL 03               (XBAR
```

```
125 *
126 4
127 *
128 -
129 RCL 03                 (XBAR
130 X|2
131 RCL 11                 (SIGMA X-SQUARED
132 *
133 6
134 *
135 +
136 RCL 15                 (NUMBER POINTS
137 /
138 RCL 03                 (XBAR
139 4
140 Y|X
141 3
142 *
143 -
144 STO 07                 (FOURTH MOMENT
145 RCL 05                 (SECOND MOMENT
146 X|2
147 /
148 "KURT=
149 XEQ 97                 (OUTPUT THE KURTOSIS
      (FS? 09               (SHORT FORM WOULD NOT COMPUTE STATS
      (RTN                    (WHICH REQUIRE SORTED DATA
150 XEQ 98                 (CALL A DATA SORTING ROUTINE
151 CF 00                  (INITIALIZE TEMP FLAG USED TO CHECK
152 RCL 15                    (EVEN OR ODD NUMBER OF DATA POINTS
153 2
154 /
155 FRC
156 X=0?                   (WAS IT AN EVEN NUMBER OF POINTS?
157 SF 00                  (IF WAS EVEN NUMBER, SET FLAG.
158 LASTX
159 .5                     (COMPUTING ADDRESS OF MEDIAN
160 +
161 RCL 04                 (ADDRESS BASE REGISTER
162 +                      (X-REG NOW HAS ADDRESS OF MEDIAN
163 "MED=
164 ARCL IND X
165 FC? 00                 (EVEN NUMBER POINTS IMPLIES THE MEDIAN
166 GTO 05                   (NOT UNIQUE,BUT SPANS AN INTERVAL
167 "¬ TO
168 PROMPT                 (DISPLAY THE LEFT BOUNDARY OF MEDIAN
169 1
170 +                      (X-REG POINTS TO RIGHT BOUND OF MEDIAN
171 ".. "
172 ARCL IND X
173 LBL 05
174 PROMPT
```

"AGAIN DISPLAY HISTOGRAM

```
LBL "AGAIN
RCL 04                     (ADDRESS BASE REGISTER
1
+
RCL IND X                  (RECALL THE FIRST ORDER STAT
"MIN
XEQ 97                     (CALL AN OUTPUT LABELING ROUTINE
STO 09                     (HOLDS STARTING (LEFTMOST) X BOUNDARY
RCL 04                     (ADDRESS BASE REGISTER
RCL 15                     (NUMBER OF DATA POINTS
+
RCL IND X                  (RECALL THE N-TH ORDER STATISTIC
"MAX
XEQ 97                     (DISPLAY THE MAX VALUE OBSERVED
RCL Z                      (MIN
-
STO 08                     (TEMP STORE FOR THE RANGE
"RNG
XEQ 97                     (DISPLAY THE RANGE
CF 00                      (INITIALIZE TEMP FLAG TO MARK LAST BAR
RCL 15                     (NUMBER DATA POINTS
RCL 04                     (ADDRESS BASE REGISTER
+
1E3
/                          (COMPUTING INDEX LOOP COUNTER
RCL 04                     (ADDRESS BASE REGISTER
+
1
+
STO 01                     (R01 SET TO ADDRESS AND LOOP THRU DATA
RCL 15                     (NUMBER POINTS
LN
2
*                          (DEFAULT NUMBER OF BARS IS 2*LN(N)
FIX 0
RND                        (VALUE IS ROUNDED NOT TRUNCATED
"CELL?
PROMPT                     (USER HAS OPTION TO CHNGE NUMBR CELLS
RCL 08                     (RANGE
X<>Y
/
"WIDTH
PROMPT                     (USER HAS OPTION TO CHANGE CELL WIDTH
STO 08                     (NOW HOLDS CELL WIDTH NOT RANGE
LBL 06
RCL 08                     (CELL WIDTH
ST+ 09                     (UPPER LIMIT OF CURRENT CELL COUNTED
0
STO 02                     (INITIALIZE CELL COUNTER
LBL 07
RCL IND 01                 (NEXT DATA POINT
RCL 09                     (CELL UPPER LIMIT
FIX 3
RND
```

```
X<Y?        (DATA POINT LESS THAN UPPER LIMIT
GTO 08
1
ST+ 02      (INCREMENT THE CELL COUNTER
ISG 01      (PREPARE TO LOOK AT NEXT DATA POINT
GTO 07
SF 00       (SET FLAG FOR OUTPUT OF LAST BAR
LBL 08
"CNT
RCL 02
FIX 0
XEQ 97      (OUTPUT THE CELL FREQUENCY COUNT
FIX 3
"XX
RCL 09
XEQ 97      (OUTPUT  CELL BOUNDARY--LOWER LIMIT
FC?C 00     (IS THIS THE LAST BAR ?
GTO 06
"CMD
AVIEW
RTN
```

```
     ((----------------------)
      (                      )
      (         "SR          )      REVIEW THE DATA
      (                      )
      (----------------------)
250 LBL "SR
251 SF 06                  (SETS MODE FOR REVIEW NOT QUERY
252 GTO 00
      (----------------------)
      (                      )
      (         "SC          )      EDIT THE DATA
      (                      )
      (----------------------)
253 LBL "SC
254 SF 08                  (SET TO EDIT MODE
255 CF 06                  (SET TO QUERY FOR CORRECT VALUE
256 RCL 00                 (CURRENT INPUT ADDRESS POINTER
257 STO 05                 (SAVE TO ENABLE RETURN TO DATA ENTRY
258 RCL 01                 (CURRENT INPUT INDEX LOOP COUNTER
259 STO 06                 (SAVE TO ENABLE RETURN TO DATA ENTRY
260 "POINT?
261 PROMPT
262 STO 01                 (ESTABLISH PSEUDO-INDEX COUNTER
263 RCL 04                 (ADDRESS BASE REGISTER
264 +
265 STO 00                 (COMPUTED ADDRESS OF DATA TO BE EDITED
266 RCL IND 00             (RECALL THE OLD VALUE
267 ST- 10                 (CORRECT SIGMA X
268 X|2
269 ST- 11                 (CORRECT SIGMA X-SQUARED
270 LASTX
271 *
272 ST- 12                 (CORRECT SIGMA X-CUBED
273 LASTX
274 *
275 ST- 13                 (CORRECT SIGMA X-FOURTH-POWER
276 XEQ 02                 (CALL DATA ENTRY AS A SUBROUTINE
277 CF 08                  (FOLLOWING RESTORES DATA ENTRY
278 RCL 05                 (INPUT ADDRESS REGISTER VALUE
279 STO 00
280 RCL 06                 (INPUT LOOP COUNTER
281 STO 01
282 1
283 -
284 ISG X                  (TEST TO SEE IF ALL DATA ALREADY INPUT
285 GTO 02                 (IF NOT, BRANCH TO THE INPUT LOOP
286 GTO "SM                (IF YES, RECOMPUTE THE SUMMARY STATS
```

HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
{(-----------------------)
{                       }
{          97           )     OUTPUT LABELING ROUTINE
{                       }
{                       }
{-----------------------)
LBL 97
"¬=
ARCL X
PROMPT
RTN
```

```
{(----------------------}
{                       }
{                       }
{          98           }           SHELL SORT
{                       }
{                       }
{-----------------------}
LBL 98
RCL 15                 (RECALL NUMBER OF DATA POINTS
STO 01                 (DEFINE A = "MIDPOINT"
LBL 09
RCL 01                 (RECALL MIDPOINT
2
/
INT
STO 01                 (A = INT(A/2)
X=0?                   (TEST TO SEE IF LIST SORTED
RTN
1
STO 02                 (B = 1  -- RESET LEFT SHELL BOUNDARY
LBL 10                 (    STACK TABLE FOLLOWS:
STO 03                 (    C=B         B
LBL 11                 (    C           B
RCL 01                 (    A           C          B
+                      (    D=C+A       B
RCL 04                 (    BASE        D          B
+                      (    ADDR D      B
RCL IND X              (    X(D)        ADDR D     B
RCL 03                 (    C           X(D)       ADDR D ,B
RCL 04                 (    BASE        C          X(D)   ,ADDR D
+                      (    ADDR C      X(D)       ADDR D
X<>Y                   (    X(D)        ADDR C     ADDR D
RCL IND Y              (    X(C)        X(D)       ADDR C ,ADDR D
X<=Y?
GTO 12                 (FOLLOWING INTERCHANGES X(C) AND X(D)
STO IND T              (    X(C)        X(D)       ADDR C ,ADDR D
X<>Y                   (    X(D)        X(C)       ADDR C ,ADDR D
STO IND Z              (    X(D)        X(C)       ADDR C ,ADDR D
RCL 03                 (    C
RCL 01                 (    A           C
-                      (    C=C-A
STO 03                 (    C
X>0?
GTO 11
LBL 12
RCL 15                 (    N
RCL 01                 (    A           N
-                      (    E=N-A
RCL 02                 (    B           E
1                      (    1           B          E
+                      (    B+1         E
STO 02                 (    B=B+1       E
X<=Y?
GTO 10
GTO 09
```

HP41C SOURCE CODE: SINGLE VARIABLE STATISTICS

```
     ((
            THIS PROGRAM USES THE FOLLOWING REGISTERS:

                 R00 -- INPUT DATA ADDRESS POINTER
                 R01 -- LOOP INDEX COUNTER
                        (USED BY DATA ENTRY AND SORT ROUTINES)
                 R02 -- TEMP REGISTER
                        (CELL FREQUENCY COUNT IN HISTO RTN)
                        (AND SHELL BOUNDARY IN SORT ROUTINE)
                 R03 -- TEMP REGISTER
                        (INPUT LABEL IN DATA INPUT ROUTINE)
                        (XBAR IN SUMMARY STAT ROUTINE)
                        (AND  POINTER IN SORT ROUTINE)
                 R04 -- INDIRECT ADDRESS BASE
                 R05 -- SECOND MOMENT (POPULATION VARIANCE)
                 R06 -- THIRD MOMENT
                 R07 -- FOURTH MOMENT
                 R08 -- HISTOGRAM CELL WIDTH
                 R09 -- TEMP REGISTER
                        (HOLDS SUM OF SQ DEVIATION ABOUT MEAN)
                        (AND HISTOGRAM CELL UPPER LIMIT)
                 R10 -- SUM OF X VALUES
                 R11 -- SUM OF X-SQUARED VALUES
                 R12 -- SUM OF X-CUBED VALUES
                 R13 -- SUM OF X RAISED TO THE FOURTH POWER
                 R14 -- NOT USED BUT SET TO ZERO BY CLRΣ
                 R15 -- NUMBER DATA POINTS (SET BY Σ+)
                 R16....R228 RAW DATA POINTS
                          -- IN NATURAL SEQUENCE BEFORE SORT
                          -- AS ORDER STATISTICS AFTER SORT

           THIS PROGRAM USES THE FOLLOWING FLAGS:

                 F00 -- TEMP FLAG (USED IN EDIT AND HISTO RTNS)
                 F05 -- VERIFY MODE (EVERY DATA POINT ECHOED)
                 F06 -- INDICATES REVIEW OF DATA NOT QUERY MODE
                 F08 -- INDICATES EDITING A DATA POINT
                 F21 -- PRINTER ENABLE (STOPS CALCULATOR
                            DURING AVIEW INSTRUCTION)
                 F26 -- AUDIO ENABLE
 340 END
```

SINGLE VARIABLE STATISTICS

1

2

3

4

5

6

7

8

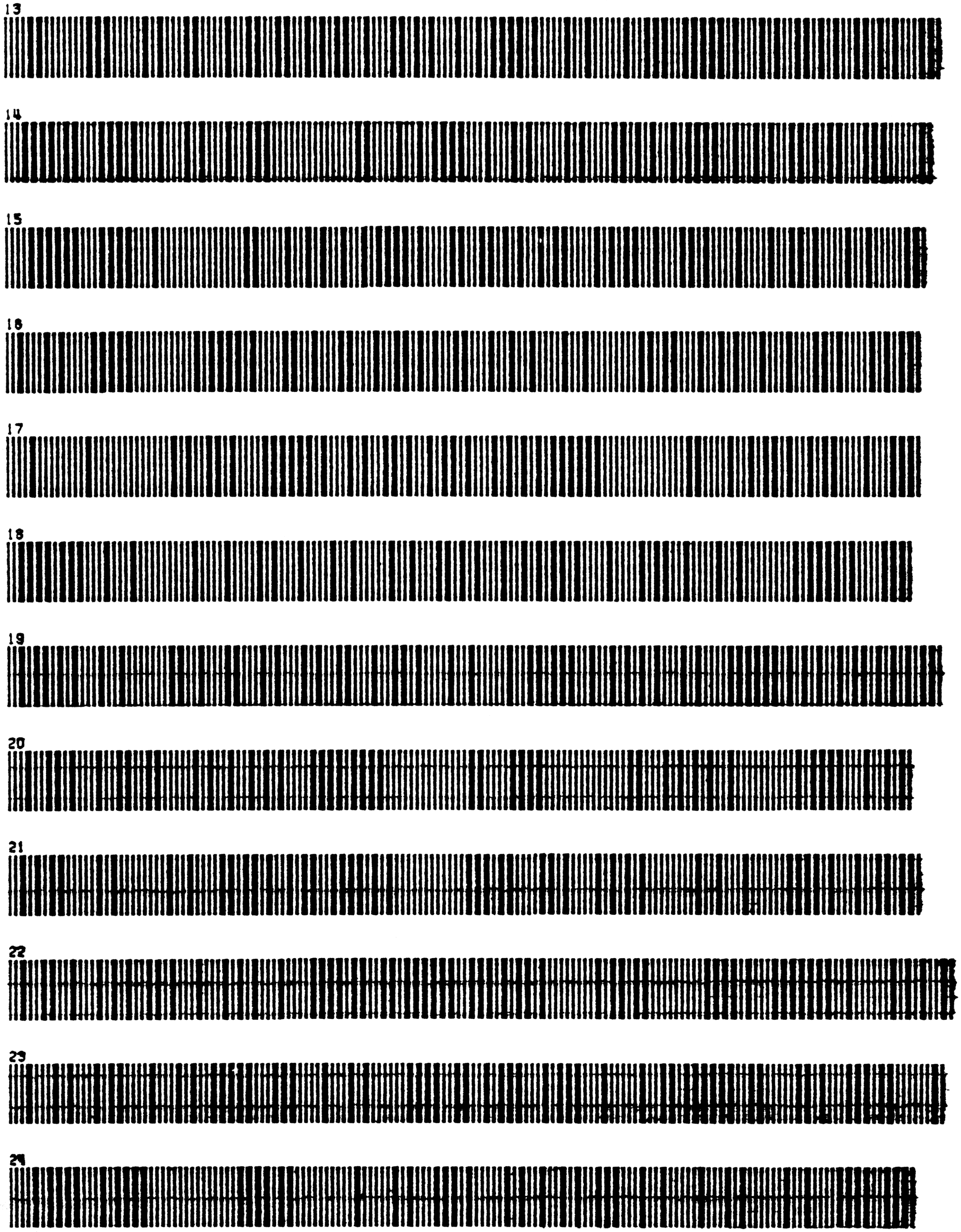
9

10

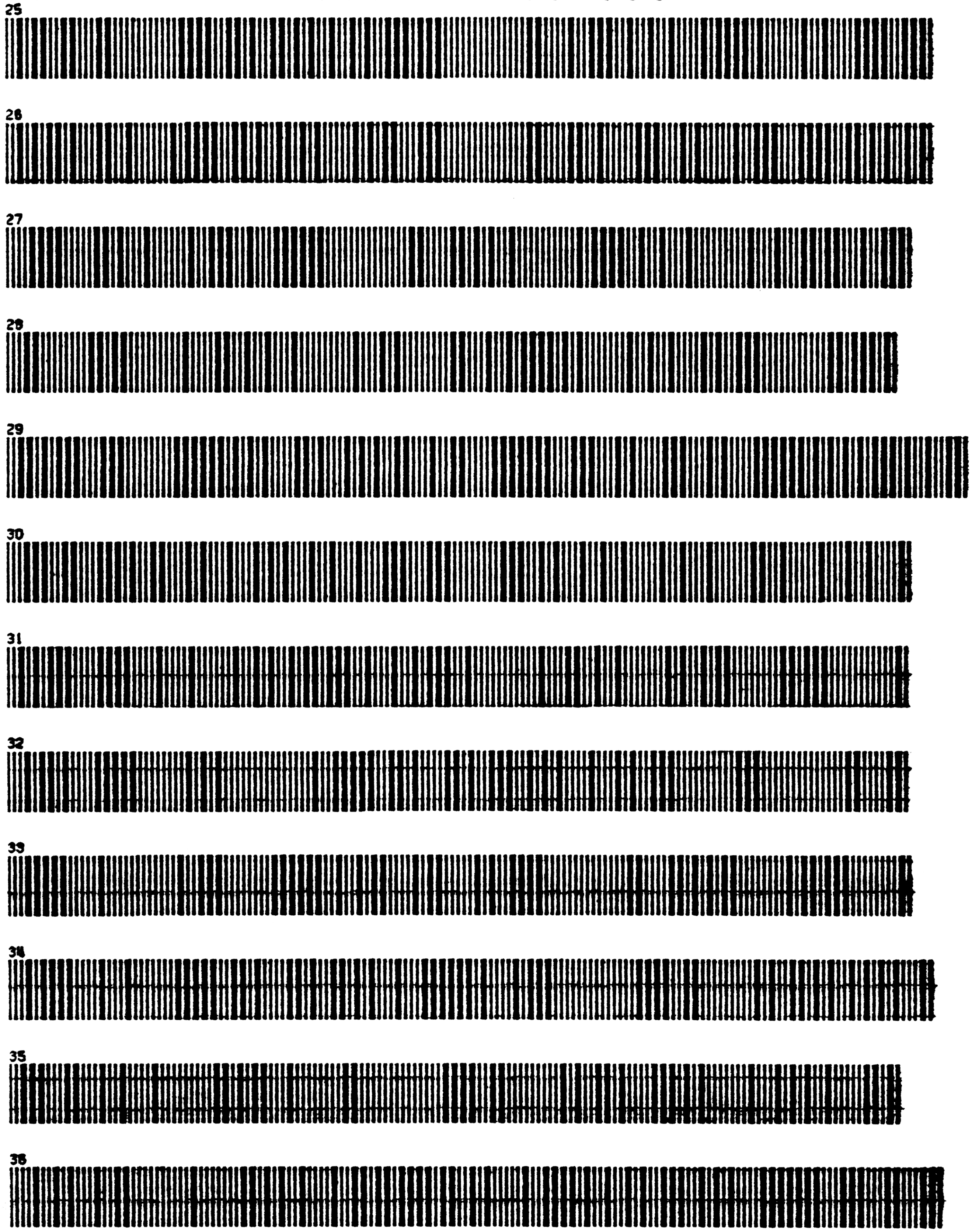
11

12

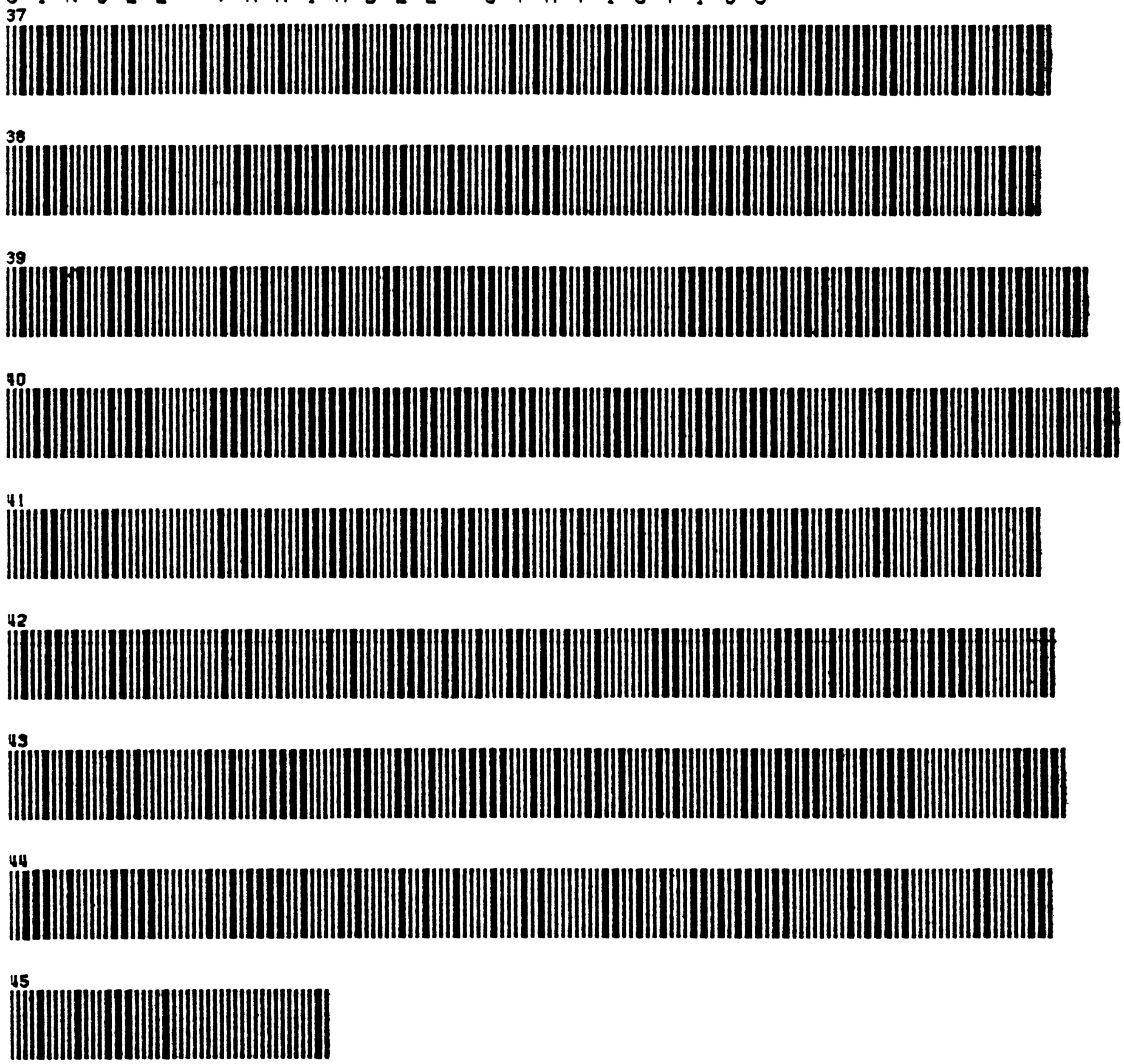
SINGLE VARIABLE STATISTICS



SINGLE VARIABLE STATISTICS



SINGLE VARIABLE STATISTICS



APPENDIX B

LINEAR PROGRAMMING EXAMPLE

INTRODUCTION:

Linear programming is an operations research technique normally associated with computerized data bases and the largest computers. Because of the complexity of the computer programs for linear programming and the large amount of data associated with most real world problems, calculators have not been widely used for this application. With the increased memory capacity of the HP41CV, however, it is now possible to offer a calculator program which can solve interesting small scale linear programs. Of value primarily as an educational aid, this program will also be able to solve many small scale problems found at Army division, brigade and battalion level. The text by Hillier and Lieberman [Ref. 8: pp. 16-66] is recommended as an introduction to the theory of linear programming as used by the program given in this appendix. Use of the program requires the user to formulate the linear programming problem; set up a Simplex tableau in standard form including adding slack, surplus and artificial variables as required; and interpret the final tableau including the calculation of the values associated with the variables in the final basis. Using the tableau form of the Simplex algorithm, the calculator performs both phase I (to obtain a feasible solution) and phase II (to obtain an optimal solution) to solve the linear programming problem. The calculator automatically determines the pivot column and pivot row for each pivot step. Infeasible and unbounded problems are automatically identified for the user by the program. There is no explicit handling of variables with upper bounds.

PROGRAM DESCRIPTION:

The program is written as a series of subroutines, each of which performs a major step in the Simplex algorithm. To provide clarity to the user, alphabetic labels have been retained to identify the subroutines in lieu of faster and more memory efficient numeric labels. The alphabetic labels have not been retained for use as program entry points and may be changed to numeric labels at the option of the user. The program has two entry points, "LP" for running a new problems and "ALP" for reviewing data previously entered.

Subroutine "FINDQ" determines the pivot column by selecting the variable in the objective function with the most negative "price." If "FINDQ" discovers at least one negative value in the objective function, then the tableau column number associated with the most negative value will be stored in register 05. Upon return from "FINDQ," the main routine tests register 05 to see if it contains a non-zero entry. If the entry is zero, it means that no further pivots will improve the value of the objective function, and the Simplex algorithm halts. If the program was in phase I (flag 11 clear) and the value of the phase I objective function is reduced to zero, then a feasible solution has been found and the program will automatically proceed to phase II to discover an optimal solution.

Subroutine "FINDP" determines which variable will leave the current basis by performing a minimum positive ratio test along the pivot column. In this way, the pivot row is determined. The row number of the pivot row is stored in register 06. Upon return from subroutine "FINDP," the main routine tests register 06 to see if it contains a non-zero entry. If the entry is zero, it means that the problem is unbounded and the Simplex algorithm halts. Such an unbounded condition is most likely caused by an error in the problem formulation.

Having determined the pivot column and the pivot row, subroutine "PIVOT" performs the actual Simplex pivot operation. To speed calculation register 00 is used as a temporary register to hold the reciprocal of the pivot element. Note that the pivot row is handled separately from the other rows in the tableau. Flag 04 is used to provide the option of stopping calculation after every pivot. When this flag is set, the program will halt to allow the user to review the status of the tableau with the "ALP" function.

Subroutine "CHECK" has two primary functions. First, it is used to verify that the designated basic variables are in row elimination form prior to the start of the Simplex algorithm. This means that the basic variable must have a coefficient of 1 in the row in which it is basic and zero's in all other rows. The second function of check is to prepare the objective function for phase I, if the initial basis contains artificial variables as indicated by one or more minus signs in the "JB" vector.

Three subroutines are used to query the user for input data. Subroutine "READMN" queries the user for the number of constraints and decision variables in the problem and verifies the calculator memory is set to contain all the data necessary to solve the problem. Subroutine "READJB" queries the user for a column vector of pointers which indicate which variable is currently basic in each row. When entering this vector of pointers, the user indicates artificial variables with a minus sign. Subroutine "READA" queries the user for the values in the initial Simplex tableau including the slack and surplus variables and the right hand side and objective function.

Several other service routines also are provided in this program. Memory size verification is done by subroutine "SIZE," which is called from within "READMN." Subroutine "IN" is used to query the user for data entry and is called by all of the data input routines. Subroutine "NXT" initializes registers which contain frequently used quantities such as the the total size of the tableau for phase I and phase II. Subroutine "INIT" clears the calculator memory and sets flags and program constants appropriately for input of a new problem. Subroutine "SETL" establishes the loop counters used repeatedly within almost every other subroutine. Subroutine "ERR1" displays an appropriate error message if a data entry error is detected.

SAMPLE PROBLEM:

A division assistant G4 is planning an ammunition upload plan. There are four types of tank munitions to consider, including:

A = Discarding Sabot Rounds
B = High Explosive Anti-Tank Rounds
C = Phosphorous Munitions
D = Machine Gun Ammunition

Based on the Commander's guidance the assistant G4 is to consider the sabot rounds as twice as important as the HEAT rounds, which in turn are themselves twice as important as a unit amount of phosphorous munitions and machine gun ammunition. His mission then, is to maximize:

$$Z = 4A + 2B + C + D$$

He is, however, constrained by the following factors:

1. There can be no more than 30 units of both sabot and HEAT munitions combined.
2. There can be no more than 50 units of all types of ammunition combined.
3. There must be at least 30 units of HEAT and phosphorous munitions combined.
4. There must be at least 5 units of machine gun ammunition.

These constraints may be expressed as:

$$\begin{aligned} A + B \quad\quad\quad\quad &\le 30 \\ A + B + C + D &\le 50 \\ B + C \quad\quad\; &\ge 30 \\ D &\ge 5 \end{aligned}$$

Based on the Commander's guidance and the constraints listed above, formulate an optimum load plan. Fractional units are permitted.

SOLUTION:

1. Before beginning with the calculator, the first step is to layout the tableau in standard form. This step and the last step of interpreting the final tableau require working knowledge of linear programming as explained in Hillier and Liberman [Ref. 8: pp. 16-66]. The standard form of the tableau is:

1	2	3	4	5	6	7	8	9	10	11
A	B	C	D	H1	H2	S1	S2	A1	A2	RHS
1	1					1				30
1	1	1	1				1			50
	1	1		-1				1		30
			1		-1				1	5
-4	-2	-1	-1							0

In this tableau, H1 and H2 are surplus variables; S1 and S2 are slack variables; and A1 and A2 are artificial variables.

2. The first step with the calculator is to set the size of the calculator's data memory. This program requires 20 registers for temporary storage, 1 register for each tableau element, and 1 register for each row to hold the pointer to the basic variable for that row. Thus, if M is the number of constraints and N is the number of variables including slack, surplus and artificial variables, then the total data

storage requirement is:

storage required = 21 + M + ((N + 1) x (M + 2))

As mentioned in the program description, the "SIZE" subroutine will automatically verify that the user has allocated enough data storage to solve the problem. The length of the program is such that 177 data storage registers is the maximum number of data storage registers that can be allocated. Thus, linear programs with 7 constraints and 15 decision variables can be solved with this program. For this example, press:

XEQ ALPHA SIZE ALPHA 175

3. Call for execution of the program with a new data set. Press:

XEQ ALPHA LP ALPHA

4. The calculator will respond with the prompt "NUM ROWS?" asking for the number of constraints in the linear program formulation. In this example, press:

4 R/S

5. The calculator will respond with the prompt "NUM COLS?" asking for the number of variables in the problem. The user

must count the number of slack, surplus and artificial variables in this total. In this example, press:

10 R/S

6. The calculator will respond with the prompt "BASIC 1 ?" asking for the variable number of the variable which is basic in the first row. One of the major features of this program is that the basic variables need not be in the rightmost positions in the tableau. Thus, if a tableau were given in which some pivots had already been performed, the program could resume operation immediately. In this example, press:

7 R/S

In a similar fashion, the calculator will then query the user for the variable number of the variables which are basic in the remaining rows.

For this example:

See	Respond With		
Basic 2 ?	8	R/S	
Basic 3 ?	9	CHS	R/S
Basic 4 ?	10	CHS	R/S

Notice that because the basic variables in rows three and four are artificial variables, the variable number is entered as a negative number. This signals the calculator that these variables must be driven from the basis in order to reach an initial feasible solution.

7. Next, the calculator will respond with "T1,1?" asking for the first element in the tableau. The user should enter the numbers in the tableau using the digit entry keys and pressing run/stop after every entry. Notice that the right hand side and the objective function will be entered with the appropriate index in the tableau as shown in step 1 above. The user must insure that the objective function is in standard form with the appropriate sign for each coefficient--in this example each coefficient is negative.

8. After the last element in the tableau has been entered, the calculator will begin to automatically perform the Simplex algorithm. If the user wishes to stop the calculator after every pivot, he may at any time press:

R/S SF 04 R/S

If this flag is set, the calculator will stop and display the pivot number after every pivot is completed.

9. When the Simplex algorithm can no longer improve the objective function, the calculator will stop and display the value of the objective function. In this example, the

calculator will stop after approximately three minutes and display:

VALUE=110.000

10. At this point, the user must use entry point "ALP" to determine the status of the final tableau. For this example, press:

XEQ ALPHA ALP ALPHA

Then by sequentially pressing the run/stop and clear arrow keys, the basic variables and tableau entries will be displayed. For example, in this problem:

<u>See</u>	<u>Press</u>	<u>See</u>	<u>Meaning</u>
BASIC 1?	CLX	2	Variable 2 is basic in the first row.
BASIC 2?	CLX	1	Variable 1 is basic in the second row.
etc.			

Then for the elements of the tableau:

<u>See</u>	<u>Press</u>	<u>See</u>	<u>Meaning</u>
T1,1?	CLX	0.000	Tableau entry
T1,2?	CLX	1.000	Tableau entry

11. After the calculator is finished, it remains for the user to interpret the final tableau. Again, the reference by Hillier and Lieberman [Ref. 8: pp. 16-66] is of primary value. In particular, the user must be able to determine the value of the final decision variables based upon what

variables are in the basis, and what the final "right hand side" values are for each row. For this example, the final tableau is:

1 A	2 B	3 C	4 D	5 H1	6 H2	7 S1	8 S2	9 A1	10 A2	11 RHS
	1			-1	-1	1	-1	1	1	15
1				1	1		1	-1	-1	15
		1			1		1		-1	15
			1		-1				1	5
0	0	0	0	2	2	1	3	-2	-2	110

Thus, the solution may be interpreted as 15 units each for munitions A,B and C and 5 units for munition D.

USER INSRUCTIONS: LINEAR PROGRAMMING

STEP	EXPLANATICN	SEE	PRESS	RESULT
1	SET SIZE (NNN= 21+M+(N+1)(M + 2) WHERE M=NUM ROWS AND N=NUM COLS		XEQ "SIZE NNN	UP TO MIN = 177
2	CALL THE PRCGRAM		XEQ "LP	
3	ENTER THE NUMBER OF CONSTRAINTS	NUM ROW ?	INPUT M R/S	
4	ENTER THE NUMBER OF VARIABLES (INCLUDE SLACKS, SURPLUS & ARTIF.	NUM COL ?	INPUT N R/S	
5	ENTER CURRENT BASIC VARIABLE NUMBERS BY ROW	BASIC 1 ? ETC.	INPUT VAR # R/S	
6	ENTER TABLEAU VALUES. FOR MISTAKES OR TO REVIEW THE DATA SEE LAST STEP BELOW.	T1,1? ETC.	INPUT R/S	
7	TO FORCE THE CAL-CULATOR TO STOP AFTER EACH PIVOT.		R/S SF 04 R/S	
8	SIMPLEX COMPLETED: OPTIMAL SOLUTION FOUND.	VALUE= XX.XXX		FINAL OBJ. FUNCTION VALUE
9	SIMPLEX COMPLETED: PROBLEM IS INFEASIBLE.	INFEAS		

STEP	EXPLANATICN	SEE	PRESS	RESULT
10	SIMPLEX COMPLETED: PRCBLEM IS UNBCUNDED.	UNBOUND		
11	TO REVIEW VALUES IN TABLEAU AT ANY TIME, INCLUDING FINAL TABLEAU. AS PROMPTS APPEAR, DATA CAN BE CHANGED BY ENTERING NEW VALUE.		XEQ "ALP	
	WHAT IS CURRENTLY BASIC?	BASIC 1 ? ETC.	CLX	PROMPT WILL VANISH LEAVING DATA
	WHAT ARE VALUES IN TABLEAU? OBTAIN VALUES CF FINAL SOLU- TICN FRCM KNCWING WHICH VARS ARE BASIC AND VALUE OF RIGHT-HAND- SIDE FRCM THE TABLEAU.	T1,1? ETC.	CLX	PROMPT WILL VANISH LEAVING DATA

HP41C SOURCE CODE: LINEAR PROGRAMMING

```
"LP

LBL "LP
XEQ "INIT
LBL "ALP
XEQ "READMN
XEQ "READJB
XEQ "READA
XEQ "CHECK
LBL 15
XEQ "FINDQ
RCL 05
X#0?
GTO 35
FS? 11
GTO 25
RCL 04
RCL 10
RCL 12
*
+
RCL 07
RCL IND Y
ABS
X<Y?
GTO 20
"INFEAS
AVIEW
STOP
LBL 20
SF 11
RCL 04
RCL 08
RCL 12
*
+
STO 14
GTO 15
LBL 25
RCL 04
RCL 09
RCL 12
*
+
RCL IND X
STO 00
"VALUE=
ARCL 00
AVIEW
STOP
LBL 35
XEQ "FINDP
RCL 06
X#0?
GTO 40
"UNBOUND
```

```
ARCL 05
STOP
LBL 40
XEQ "PIVOT
GTO 15
```

```
{(------------------------}
{                         }
{         FINDQ           }
{                         }
{------------------------}
LBL "FINDQ
SF 01
0
STO 03
STO 05
1
RCL 11
XEQ "SETL
LBL 31
RCL 14
RCL 01
+
STO 00
RCL IND X
RCL 03
-
RCL 07
CHS
X<Y?
GTO 38
FC? 11
GTO 37
RCL 15
RCL 01
+
RCL IND X
ABS
RCL 07
X<=Y?
GTO 38
LBL 37
RCL IND 00
STO 03
RCL 01
INT
STO 05
LBL 38
ISG 01
GTO 31
CF 01
RTN
```

```
{(------------------------)
{                        }
{         FINDP          }
{                        }
{------------------------)
LBL "FINDP
SF 02
0
STO 06
1E20
STO 03
1
RCL 08
XEQ "SETL
LBL 41
RCL 01
1
-
RCL 12
*
RCL 04
+
STO 00
RCL 05
+
RCL IND X
STO 02
RCL 07
X>Y?
GTO 48
RCL 00
RCL 12
+
RCL IND X
RCL 02
/
STO 00
RCL 03
-
RCL 07
CHS
X<Y?
GTO 48
LBL 47
RCL 00
STO 03
RCL 01
INT
STO 06
LBL 48
ISG 01
GTO 41
CF 02
RTN
```

```
       {(----------------------)
       {                        }
       {         PIVOT          }
       {                        }
       {------------------------}
150 LBL "PIVOT
151 SF 03
152 RCL 06
153 1
154 -
155 RCL 12
156 *
157 RCL 04
158 +
159 STO 03
160 RCL 05
161 +
162 RCL IND X
163 1/X
164 STO 00
165 1
166 RCL 12
167 XEQ "SETL
168 LBL 51
169 RCL 03
170 RCL 01
171 +
172 RCL 00
173 ST* IND Y
174 ISG 01
175 GTO 51
176 1
177 RCL 09
178 FS? 11
179 0
180 FC? 11
181 1
182 +
183 XEQ "SETL
184 LBL 52
185 RCL 06
186 RCL 01
187 INT
188 X=Y?
189 GTO 59
190 1
191 -
192 RCL 12
193 *
194 RCL 04
195 +
196 STO 16
197 RCL 05
198 +
199 RCL IND X
200 STO 00
201 2
202 RCL 12
203 XEQ "SETL
```

```
LBL 53
CF 07
RCL 05
RCL 02
INT
X=Y?
SF 07
RCL 03
+
RCL IND X
RCL 00
*
RCL 16
RCL 02
+
X<>Y
ST- IND Y
FC? 07
GTO 54
0
STO IND Z
LBL 54
ISG 02
GTO 53
LBL 59
ISG 01
GTO 52
RCL 13
RCL 06
+
RCL 05
STO IND Y
1
ST+ 17
CF 03
TONE 7
FC? 04
RTN
"PIVOT "
ARCL 17
PROMPT
RTN
```

```
{(-----------------------)
{                        }
{                        )
{        READMN
{                        }
{                        }
{-----------------------)
LBL "READMN
7
STO 00
"NUM ROWS
XEQ "IN
XEQ "NXT
XEQ "NXT
"NUM COLS
XEQ "IN
XEQ "NXT
RCL 10
*
RCL 04
+
1
ST+ 00
RDN
STO IND 00
RCL 09
+
XEQ "SIZE?
FC?C 25
PROMPT
RTN
```

```
SIZE?

LBL "SIZE?
"SIZE>=
ARCL X
1
-
SF 25
RCL IND X
RTN

"NXT

LBL "NXT
1
ST+ 00
+
STO IND 00
RTN

SETL

LBL "SETL
1E03
/
1
+
STO IND Y
RTN
```

```
{(----------------------)
{                      }
{                      )
{        READJB
{                      }
{----------------------)
LBL "READJB
1
RCL 08
XEQ "SETL
RCL 13
STO 00
FIX 0
LBL 01
"BASIC "
ARCL 01
"├ "
XEQ "IN
ISG 01
GTO 01
FIX 4
RTN
```

READA

```
LBL "READA
SF 10
1
RCL 09
XEQ "SETL
LBL 11
2
RCL 12
XEQ "SETL
LBL 12
FIX 0
"T
ARCL 01
"¬,
ARCL 02
FIX 4
LBL 13
RCL 01
1
-
RCL 12
*
RCL 02
+
RCL 04
+
1
-
STO 00
LBL 14
FC? 10
GTO 16
XEQ "IN
GTO 17
LBL 16
"¬=
ARCL IND 00
AVIEW
LBL 17
ISG 02
GTO 12
ISG 01
GTO 11
CF 10
0
RTN
```

```
{(---------------------)
{                     }
{         IN          }
{                     }
{---------------------}
LBL "IN
CF 22
1
ST+ 00
RCL IND 00
"¬?
PROMPT
STO IND 00
RTN
{---------------------}
{                     }
{        INIT         }
{                     }
{---------------------}
LBL "INIT
CLRG
CF 29
1E-04
STO 07
19
STO 04
RTN
{---------------------}
{                     }
{        ERR1         }
{                     }
{---------------------}
LBL "ERR1
0
LN
XEQ "READJB
```

```
{(----------------------)
{                      }
{        CHECK         }
{                      }
{----------------------)
LBL "CHECK
SF 11
1
RCL 12
XEQ "SETL
RCL 04
RCL 09
RCL 12
*
+
STO 14
STO 15
LBL 91
RCL 14
RCL 01
+
0
STO IND Y
ISG 01
GTO 91
1
RCL 08
XEQ "SETL
LBL 92
CF 07
RCL 13
RCL 01
+
RCL IND X
X<0?
SF 07
ABS
STO 00
X=0?
GTO "ERR1
RCL 12
X<=Y?
GTO "ERR1
2
RCL 09
XEQ "SETL
LBL 93
CF 08
RCL 01
INT
RCL 02
INT
X=Y?
SF 08
1
-
RCL 12
*
RCL 00
```

```
+
RCL 04
+
FC? 08
0
FS? 08
1
STO IND Y
ISG 02
GTO 93
FC? 07
GTO 98
CF 11
RCL 14
RCL 00
+
1
STO IND Y
2
RCL 12
XEQ "SETL
LBL 96
RCL 01
INT
1
-
RCL 12
*
RCL 02
+
RCL 04
+
RCL IND X
CHS
RCL 14
RCL 02
+
X<>Y
RCL IND Y
+
STO IND Y
ISG 02
GTO 96
LBL 98
ISG 01
GTO 92
FC? 11
RTN
RCL 04
RCL 08
RCL 12
*
+
STO 14
RTN
```

```
((
     THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG
     ASSIGNMENTS MADE BY THIS PROGRAM:

         R00 = TEMPORARY REGISTER.  HOLDS RECIPROCAL OF PIVOT
               ELEMENT IN SUBROUTINE PIVOT.
         R01 = LOOP INDEX COUNTER
         R02 = LOOP INDEX COUNTER
         R03 = TEMPORARY REGISTER.  HOLDS MIN VALUE IN FINDQ;
               MAX VALUE IN FINDP; AND IS AN INTERMEDIATE
               ADDRESS CALCULATION VALUE IN PIVOT.
         R04 = BASE REGISTER FOR INDIRECT ADDRESSING
               (SET = 19)
         R05 = THE PIVOT COLUMN NUMBER
         R06 = THE PIVOT ROW NUMBER
         R07 = EFFECTIVE ZERO LEVEL
         R08 = M = NUMBER OF ROWS
         R09 = M PLUS 1
         R10 = M PLUS 2
         R11 = N = NUMBER OF VARIABLES
         R12 = N PLUS 1
         R13 = BASE REGISTER FOR THE LOCATION OF THE
               VECTOR JB WHICH CONTAINS POINTERS TO
               WHICH VARIABLE IS BASIC IN EACH ROW.
         R14 = ROW NUMBER OF THE OBJECTIVE FUNCTION;
               SET TO M PLUS 1 OR M PLUS 2 AS DETERMINED
               BY NEED FOR PHASE I.
         R15 = BASE REGISTER FOR THE LOCATION OF THE
               PHASE I OBJECTIVE FUNCTION, IF NEEDED.
         R16 = TEMPORARY REGISTER.
         R17 = NUMBER OF PIVOTS PERFORMED.
         R18 = RESERVED FOR FUTURE USE.
         R19-R177 = DATA STORAGE REGISTERS FOR ELEMENTS OF
                    THE TABLEAU AND THE JB VECTOR.

                            FLAGS

     F01 - F03 = SUBROUTINE EXECUTION FLAGS.  BECAUSE THESE
                 FLAGS ARE VISIBLE IN THE DISPLAY THEY CAN
                 BE SET WHEN ENTERING A MAJOR SUBROUTINE AND
                 CLEARED WHEN LEAVING -- GIVING THE USER A
                 VISUAL INDICATION OF WHAT IS HAPPENING
                 INSIDE THE CALCULATOR.

                 F01--SUBROUTINE FINDQ
                 F02--SUBROUTINE FINDP
                 F03--SUBROUTINE PIVOT

     F04 = WHEN SET, STOPS CALCULATOR AFTER EACH PIVOT.
     F07 = USED AS TEMPORARY FLAG IN PIVOT AND CHECK ROUTINES.
     F10 = USED AS A TEMPORARY FLAG IN READ ROUTINES.
     F11 = WHEN SET, INDICATES PHASE II IS IN PROGRESS.
     F29 = CONTROLS FORMAT OF DISPLAY SEPARATOR.
483 END
```

"LP" LINEAR PROGRAMMING

1

2

3

4

5

6

7

8

9

10

11

12

"LP" LINEAR PROGRAMMING

13

14

15

16

17

18

19

20

21

22

23

24

"LP" LINEAR PROGRAMMING

25

26

27

28

29

30

31

32

33

34

35

36

"LP" LINEAR PROGRAMMING

37

38

39

40

41

42

43

44

45

46

47

48

"LP" LINEAR PROGRAMMING

49

50

51

52

53

54

55

56

57

58

59

60

"LP" LINEAR PROGRAMMING

61

62

63

64

65

66

67

68

69

70

71

72

"LP" LINEAR PROGRAMMING

73

74

75

76

77

APPENDIX C

SUBROUTINES FOR READ ONLY MEMORY

INTRODUCTION:

The calculator subroutines described in this appendix perform functions which are frequently required by application programs and are therefore ideal candidates for use in a read only memory (ROM.) These routines were written especially to illustrate the differences between read only memory routines and routines designed for individual use via bar code or magnetic cards. These differences include more attention to entry and exit point options, an attempt to keep the size of the routines as compact as possible, and an attempt not to disturb the register stack if at all possible.

These common subroutines are provided separately from application programs because when more than one application program uses the routines, as is recommended for these programs, the use of a separate block of common functions saves space in the ROM overall. Also, by providing a convenient set of "macro" instructions, application programs can be constructed more quickly and easily. Because these subroutines are used frequently, they have been individually optimized and tested to save memory space and execution time. By using these "macros" within an application program, the application programmer can be reasonably certain of their efficiency and reliability.

Almost all user/calculator interface is handled by these routines. There is one set of subroutines which assumes the user has a printer, and one set which does not. Printer instructions are preceeded in the listings shown in the appendix by an (PRT: label. When not using these routines on read only memory, the user will load one set or the other (but not both), as appropriate for his/her application. In so doing, the user with the printer gets full benefit from it while the user without the printer pays no penalty in execution time or memory space for the calculator's print instructions. Also, to change from use of the printer to use of the calculator without it, the user needs only to read in the new common block--the application programs are retained in memory unchanged. The subroutines appear the same to any application program--giving the added benefit that any application program which uses them for input or output operations will automatically make good use of the printer even if written by a programmer who did not explicitly consider a printer requirement.

When using these common subroutines, a discipline is enforced upon the application program concerning use of the calculator memory registers. This saves the programmer from having to plan his "register map" from scratch for each new program. Also, it insures compatability across different application programs for similar data objects such as matrices and loop index counters. One of the most annoying problems with read only memory programs available from the calculator manufacturer is this lack of cross program compatability. Conflict in the use of memory registers is the rule, rather than the exception; and it is frequently impossible to efficiently use more than one read only memory program as a subroutine within a user written program. A third reason why register assignment standards are advantageous is that they make it easier for the user of the calculator to remember the key register assignments and, if necessary, recall their contents during the execution of an application program.

Another function performed by this set of common subroutines is to simplify the use of indirect addressing--a critical goal on the HP41CV.

Because the common subroutines listed in this appendix are always called by application programs and never from the keyboard by the user, the typical user instructions are inappropriate. Instead, for the benefit of application programmers wishing to use the routines, the basic functions and structure of each are explained in subsequent sections of this appendix.

<u>Subroutine "IN"</u>

Subroutine "IN" is used as a general input and output interface between the user and the calculator. This subroutine has three alternative entry points which when called affect functions as follows:

```
IN--Input mode (displays a question mark query)
IO--Output mode (displays labeled data value)
IX--Direct mode (input of value in x register)
```

In particular, one entry point, "IN", may be called whenever an application program must query the user for a numeric input value. As such, it is a direct replacement for the PROMPT instruction organic to the calculator, but automatically prompts, verifies and stores the received value using an indirect address contained in register 00. The printer version of the subroutine will automatically label and print the final, verified data value recorded.

Subroutine "IN" uses register 00 as a data location pointer and automatically increments this register so that subsequent calls to the subroutine will result in sequential data manipulation. The application programmer must insure that register 00 contains a number equal to the storage register number prior to calling the subroutine. For example, if register 00 contains 17, "IN" will store the data in register 17. One of the major advantages of this routine is that the same subroutine may be used to verify and/or change the data previously recorded. Thus, separate edit routines are usually unnecessary. Pressing the R/S key without touching any other key leaves the value stored unchanged. Pressing "1" and "+" and then "R/S" adds one to the current stored value, and so on for other function keys. Entering a new string of digits results in that new value being stored.

An additional feature of this subroutine is the "verify" mode indicated by flag 05. Flag 05 is reserved for this purpose and is set "on" by a call to subroutine "VR"--another of the subroutines in this common set. The verify mode is intended for use by a novice or other user who wishes to verify every data value as it is keyed into the calculator. The advantage is increased accuracy and confidence in the result.

Subroutine "D2"

Subroutine "D2" is used to set up the index register for a program loop. This subroutine has two alternative entry points which when called, increment different index registers as follows:

```
D2--Establishes register 02 as the index
D1--Establishes register 01 as the index
```

This subroutine is intended for use with the "ISG" loop structure which has the effect most like that of a FORTRAN "DO LOOP." For example, to execute a loop 20 times:

```
...
...
20
XEQ "D1
LBL 00
    ....
    ....
ISG 01
GTO 00
...
...
```

The advantage of this form of loop structure is that register 00 may be used within the loop for address calculations. The first time the loop is addressed the integer portion of the number in register 00 will be 1, the second time it will be 2 and so on. There is no need to truncate the fractional portion of the number because the HP41C ignores the fractional component of a number when calculating a register address. Use of index registers for address calculation makes indirect addressing practical.

Registers 01 and 02 should be reserved for use as index registers by the application programmer. In most cases these two registers should prove sufficient.

<u>Subroutine</u> "<u>VR</u>"

Subroutine "VR" is used as a general purpose calculator initialization routine. This subroutine has three alternative entry points which vary the amount of initialization performed as follows:

```
VR--Sets the verify mode on, and the following:
WR--Suppresses the audio tones, and the following:
WS--Clears all memory,
    Sets the display for integers,
    Assigns statistical regisers,
    Sets "zero" level for equality testing, and
    Sets base address for indirect addressing.
```

In the printer version of this initialization routine, the subroutine prints a banner (usually the program name) which has been stored in the alpha register prior to calling the initialization subroutine.

If flag 13 is set prior to calling the initialization routine, then the calling program must have placed the number of data registers required to execute the program in the x register prior to calling the initialization routine. In this case, a check will automatically be performed using subroutine "SZ" described below.

It is recommended that all application programs provide an alternate entry point which bypasses the initialization step. Then if a data error is encountered, the user may review the data entered into the calculator by simply pressing the return key once after every prompt. This procedure works because subroutine "IN" recalls the stored value prior to prompting the user. When the user presses the clear key, the alphabetic prompt is removed and the existing data value revealed.

Subroutine "SZ"

Subroutine "SZ" is used to test if sufficient numbers of data registers are available to run an application program. This subroutine may be either called directly or as part of the initialization routine described above.

Subroutine "ER"

Subroutine "ER" is called whenever the application program encounters an error--usually in the input data. A prompt is displayed and an audio tone sounded, provided flag 26 (the audio enable flag) has not been cleared by the initialization routine described in paragraph D.

Subroutine "SORT"

This subroutine is included to illustrate the use of a stack register table in the program comments, but it also represents a useful utility routine. The sorting algorithm used is the shell sort [Ref. 6: pp. 84-95] which gives reasonably fast sorting times with a very small program size. All conventions such as base register in R04 and number of data points in R15 are assumed by this subroutine. A complete list of all such register assignments is listed at the end of the program listing.

Subroutines "PUT" and "GET"

These two small routines provide a useful capability to store and recall up to three integers between 0 and 999 in one data register. This means that if you are manipulating a spread sheet of small, positive integers you can store the same data in one third the space. Of course, run time is degraded (about 20 seconds for every 100 data references.) To store a value, assuming the base register has been defined, just press:

value ENTER↑ point-number XEQ "PUT

To recall a value, simply press:

point-number XEQ "GET

```
     ((----------------------)
     (                       )
     (                       )
     (          IN
     (                       )
     (                       )
     (-----------------------)
 1 LBL "IN                  (INPUT MODE--DISPLAY LABEL AND ?
 2 SF 10                    (SET QUERY ONCE FLAG
 3 GTO 05
 4 LBL "IC                  (OUTPUT MODE--DISPLAY LABEL AND DATA
 5 CF 10                    (INSURE NO QUERY
 6 LBL 05
 7 RCL IND 00               (ASSUMES R00 POINTS TO STORAGE REG
 8 LBL "IX                  (DIRECT MODE--ASSUMES X REG HOLDS DATA
 9 ASTO 05                  (ASSUMES LABEL SET UP BY CALLING PROG
10 LBL 01
11 "¬=
12 FS? 10                   (QUERY OR DISPLAY VALUE?
13 "¬?
14 FC? 10
15 ARCL IND 00              (DISPLAY DATA VALUE
16 CF 22                    (DIGIT ENTRY DETECTION FLAG
   (PRT: FS? 10
   (PRT: CF 21                 (DISABLE PRINTER FOR QUERY
17 PROMPT
   (PRT: SF 21
18 STO IND 00               (STORE INPUT VALUE
19 CLA                      (PREPARE ALPHA REG FOR NEXT PROMPT
20 FC? 05                   (NOT VERIFY MODE?
21 GTO 03
   (FOLLOWING IS ACTION TAKEN WHEN IN VERIFY MODE
22 FS? 22                   (ANY INPUT DETECTED?
23 GTO 02                   (REDISPLAY VALUE IF USER INPUT
24 FC? 10                   (DISPLAY VALUE ONCE IF NO INPUT
25 GTO 04
26 LBL 02                   (DISPLAY LABEL AND DATA VALUE
27 CF 10                    (NOTE MUST HAVE QUERIED ONCE
28 ARCL 05
29 GTO 01
   (FOLLOWING IS ACTION TAKEN FOR NON-VERIFY MODE
30 LBL 03
31 FS?C 10                  (INPUT MODE?
32 GTO 04                   (IF INPUT MODE, EXIT ROUTINE
33 FS? 22                   (WAS THERE INPUT ?
34 GTO 02                   (USER CHANGED VALUE, SO VERIFY.
35 LBL 04                   (PREPARE TO EXIT
   (PRT: FS? 55                (PRINTER ATTACHED?
   (PRT: GTO 06
   (PRT: LBL 05
36 ISG 00                   (INCREMENT POINTER FOR NEXT IO OPERATION
37 RTN
38 RTN                      (FINAL VALUE IS IN X REG UPON EXIT
   (PRT: LBL 06                (PRINT THE FINAL VALUE
   (PRT: CLA
   (PRT: ARCL  05
   (PRT: "¬=
   (PRT: ARCL IND 00
   (PRT: PRA
   (PRT: GTO 05
```

```
((-----------------------)
(                       )
(                       )
(          D2
(                       )
(                       )
(-----------------------)
LBL "D2              (SETUP LOOP USING REG 2 AS INDEX COUNTER
2
GTO 07
LBL "D1              (SETUP LOOP USING REG 1 AS INDEX COUNTER
1
LBL 07
X<>Y                 (NUMBER LOOP ITERATIONS MUST BE IN X
1E3                  (PRIOR TO CALLING THIS SUBROUTINE.
/
1
+
STO IND Y
RTN
```

```
      ((------------------------)
      (                         )
      (                         )
      (          VR
      (                         )
      (                         )
      (-------------------------)
52 LBL "VR                (SET VERIFY MODE ON
53 SF 05
54 LBL "WR                (INITIALIZE FOR CLASSROOM USE -- NO AUDIO
55 CF 26
56 LBL "WS                (STANDARD INITIALIZATION ROUTINE FOLLOWS
57 CLRG
58 &REG 10
59 CF 29                  (SETS DISPLAY--NO DECIMAL POINT IF INTEGER
60 1E-04                  (EFFECTIVE ZERO LEVEL
61 STO 03
62 15                     (NORMAL INDIRECT ADDRESS BASE REGISTER.
63 STO 04                 (THIS IS 1 LESS THAN THE NUMBER OF THE
64 1
65 +
66 STO 00                 (FIRST REGISTER WHERE DATA IS STORED.
    (PRT: ADV
    (PRT: SF 12               (DOUBLE WIDE PRINTING
    (PRT: FC? 55              (PRINTER ATTACHED?
    (PRT: PRA
    (PRT: CF 12               (SET BACK TO SINGLE WIDE PRINT
    (PRT: ADV
67 FC? 13                 (HAS INITIALIZATION ROUTINE BEEN ASKED TO
68 RTN                    (VERIFY SIZE?
69 X<> Z
```

```
      ((----------------------)
      (                      )
      (                      )
      (          SZ
      (                      )
      (                      )
      (----------------------)
70 LBL "SZ
71 "SET SZ=
72 ARCL X
73 1
74 -
75 SF 25                 (PREPARE TO IGNORE ERROR
76 RCL IND X             (TEST FLAG 25 TO SEE IF SUFFICIENT
77 FC?C 25
78 PROMPT
79 RTN                   (SIZE EXISTS.
      (----------------------)
      (                      )
      (                      )
      (          ER
      (                      )
      (                      )
      (----------------------)
80 LBL "ER               (DISPLAY "DATA ERROR" PROMPT & SOUND TONE.
81 0
82 LN                    (BEST WAY TO DETERMINE WHERE ERROR
83 TONE 2                (OCCURED IS TO HIT THE SST KEY ONCE,
84 RTN                   (THEN GO INTO PRGM MODE.
```

```
       ((-----------------------)
       (                       )
       (                       )
       (        SORT
       (                       )
       (                       )
       (-----------------------)
 85 LBL "SORT
 86 RCL 15              (RECALL NUMBER OF DATA POINTS
 87 STO 01              (DEFINE A = "MIDPOINT" OF NUMBER POINTS
 88 LBL 00
 89 RCL 01              (RECALL MIDPOINT
 90 2
 91 /
 92 INT
 93 STO 01              (A = INT(A/2)
 94 X=0?                (TEST TO SEE IF LIST SORTED
 95 RTN
 96 1
 97 STO 02              (B = 1  -- RESET LEFT SHELL BOUNDARY
 98 LBL 01           (     STACK TABLE FOLLOWS:
 99 STO 03           (     C=B         B
100 LBL 02           (     C           B
101 RCL 01           (     A           C          B
102 +                (     D=C+A       B
103 RCL 04           (     BASE        D          B
104 +                (     ADDR D      B
105 RCL IND X        (     X(D)        ADDR D     B
106 RCL 03           (     C           X(D)       ADDR D     B
107 RCL 04           (     BASE        C          X(D)       ADDR D
108 +                (     ADDR C      X(D)       ADDR D
109 X<>Y             (     X(D)        ADDR C     ADDR D
110 RCL IND Y        (     X(C)        X(D)       ADDR C     ADDR D
111 X<=Y?
112 GTO 03           (FOLLOWING WILL INTERCHANGE X(C) AND X(D)
113 STO IND T        (     X(C)        X(D)       ADDR C     ADDR D
114 X<>Y             (     X(D)        X(C)       ADDR C     ADDR D
115 STO IND Z        (     X(D)        X(C)       ADDR C     ADDR D
116 RCL 03           (     C
117 RCL 01           (     A           C
118 -                (     C=C-A
119 STO 03           (     C
120 X>0?
121 GTO 02
122 LBL 03
123 RCL 15           (     N
124 RCL 01           (     A           N
125 -                (     E=N-A
126 RCL 02           (     B           E
127 1                (     1           B          E
128 +                (     B+1         E
129 STO 02           (     B=B+1       E
130 X<=Y?
131 GTO 01
132 GTO 00
       (               SORT    USES THE FOLLOWING REGISTERS:
       (                       R01 -- A
       (                       R02 -- B
       (                       R03 -- C
       (                       R04 -- INDIRECT ADDRESS BASE
       (                       R15 -- NUMBER DATA POINTS (SET BY Σ+)
```

```
((-----------------------)
(                       )
(                       )
(          SC
(                       )
(                       )
(-----------------------)
LBL "SC
RCL 00
1
+
STO 05
-1EC3
STO 10
-1
3
/
STO 09
```

```
((----------------------)
(                      )
(                      )
(         PUT           
(                      )
(                      )
(----------------------)
144 LBL "PUT
145 "VALUE
146 11
147 STO 00
148 XEQ "IA
149 SF 02
(----------------------)
(                      )
(                      )
(         GET           
(                      )
(                      )
(----------------------)
150 LBL "GET
151 "BIT REG
152 PROMPT
```

```
((----------------------)
(                      )
(                      )
(         SA
(                      )
(                      )
(----------------------)
LBL "SA
3
/
INT
STO 07
LASTX
-
RCL 09
/
12
+
STO 08
RCL 05
ST+ 07
RCL IND 07
INT
STO 14
LASTX
-
RCL 10
*
INT
STO 13
LASTX
-
RCL 10
*
INT
STO 12
RCL 11
RCL IND 08
STO 11
FC?C 02
RTN
X<>Y
STO IND 08
RCL 12
RCL 10
CHS
/
RCL 13
+
RCL 10
CHS
/
RCL 14
+
STO IND 07
RTN
```

```
       ((
       (
       (
       ( THE FOLLOWING TABLE DESCRIBES THE KEY REGISTER AND FLAG
       ( ASSIGNMENTS MADE BY THIS PROGRAM:
       (
       (     R00 = INDIRECT ADDRESS FOR STORAGE OF INPUT DATA
       (     R01 = LOOP INDEX COUNTER
       (     R02 = LOOP INDEX COUNTER
       (     R03 = EFFECTIVE ZERO LEVEL -- USE AS TEMP IF NA
       (     R04 = BASE REGISTER FOR INDIRECT ADDRESSING (15)
       (     R05 = TEMP REGISTER FOR ALPHA PROMPT
       (     R06 - R09 = APPLICATION PROGRAM TEMP REGISTERS
       (     R10 - R15 = STATISTICAL REGISTERS--USE AS TEMP IF NA
       (     R16....   = STORAGE OF DATA VIA INDIRECT ADDRESSING
       (
       (                      FLAGS
       (
       (     F00-F04 = SUBROUTINE EXECUTION FLAGS.  BECAUSE THESE
       (               FLAGS ARE VISIBLE IN THE DISPLAY THEY CAN
       (               BE SET WHEN ENTERING A MAJOR SUBROUTINE AND
       (               CLEARED WHEN LEAVING -- GIVING THE USER A
       (               VISUAL INDICATION OF WHAT IS HAPPENING
       (               INSIDE THE CALCULATOR.  USE AS TEMPORARY
       (               FLAGS IF THIS IS NOT REQUIRED.
       (
       (     F05  = VERIFY INPUT MODE.  "ON" WHEN SET.  WHEN SET
       (            AFTER EVERY DATA VALUE IS ENTERED, THE CALC.
       (               WILL ECHO THE PROMPT AND DATA VALUE ENTERED.
       (               IF VALUE IS CORRECT, SIMPLY PRESS R/S KEY,
       (               OTHERWISE  ENTER A CORRECTED VALUE AND
       (               CALCULATOR WILL AGAIN ASK FOR VERIFICATION.
       (
       (     F10 = USED AS A TEMPORARY FLAG INSIDE "IC".  INDICATES
       (           NO QUERY PROMPT IS DESIRED.
       (     F11 = AUTOMATIC EXECUTION FLAG -- DON'T USE EVER
       (     F12 = DOUBLE WIDE PRINTING -- LOCAL WITHIN "IO".
       (     F13 = WHEN SET MEANS MAIN ROUTINE ASKING "VR" FOR
       (           AUTOMATIC SIZE CHECK AFTER INITIALIZATION.
       (           APPLICATION PROGRAM MAY USE FREELY AFTER INIT.
       (     F14 - F20 = AVAILABLE FOR APPLICATION PROGRAM USE.
  202 END
```

COMMON SUBROUTINES

1

2

3

4

5

6

7

8

9

10

11

12

COMMON SUBROUTINES

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

APPENDIX D

THE CROSS COMPILER PROGRAM AND COMMAND PROCESSOR

DESIGN METHODOLOGY:

This appendix discusses the design methodology used during construction of both the cross compiler program and the command processor, which is an IBM EXEC II program which provides an interactive programming environment for users of the system.

Blazie's compiler for the HP65 calculator [Ref. 9] represents one of the first attempts to provide a compiler for calculator programs. Both Carvalho [Ref. 10: pp. 25-29] and McNeal [Ref. 11: pp. 148-178] have published BASIC language programs which cross compile HP41CV instructions on a microcomputer for output to a line printer which can print acceptable bar code. While these referenced programs provided valuable insights into the problem, especially into the special characteristics of the HP41CV instruction set, none was exactly suited to the needs of this study. Because the Versatec plotter at the Naval Postgraduate School could be easily used only by FORTRAN programs, FORTRAN seemed the computer language of choice for this project. Both programs were written with limited objectives and neither would have easily supported the extensions desired. Extensions planned for implementation included:

- An extended instruction set.
- In code comments.
- Extensive error checking.
- Compatability with the Emulator.
- Synthetic Instructions [Ref. 1].
- Instruction macro's.

Having decided to code an original cross compiler, a design methodology which would capitalize on the advantages of FORTRAN was planned. FORTRAN's major deficiency for use in constructing a compiler of any type is its lack of alpha-numeric string handling capabilities. Rather than struggle with the lack of string functions, it was decided to code the necessary string functions as separate subroutines. This decision was reinforced countless times throughout the process of writing the compiler. The string function subroutines have been used not only in the cross compiler, but in many other FORTRAN programs since they were originally written. In fact, many persons who have no interest in the HP41CV cross compiler may find the set of string functions listed in this thesis to be a valuable set of utility routines to be used to augment FORTRAN. The general convention used througout the string function subroutines is that an alphabetic string may be represented as a vector of two byte integer variables used to store the characters and a single four byte integer variable used to store the length of the string.

One of the major advantages of the cross compiler is its ability to handle comments integrated within the HP41CV source code. This feature is critical to the clarification of the logical structure of the HP41CV programs. Because the parenthesis is not a valid HP41CV character, it was chosen as the comment indicator character. A comment may occur beginning at the first column on an input line or anywhere after an HP41CV instruction. The comment must follow the instruction because everything after the comment mark out to the end of the input line is considered part of the comment.

The control the user has over the output listing is also one of the advantages of the cross compiler. When two comment indicator marks are placed in positions one and two of the input line, the compiler will force a page eject when printing the output listing. In addition, the user can vary the number of output lines per page and cause useful banners to be placed adjacent to program labels for ease of recognition.

Altogether there are twenty-four subroutines and a main program which constitute the cross compiler. The source code for each of these routines is provided in the second section of this appendix. Each subroutine begins with a statement concerning its function and construction. Accordingly, no general description of each subroutine will be repeated here. However, subroutine COMP deserves special attention, for it is the master lexicographic analyzer for the compiler and would also interface the user with the emulator. Its function is to receive a single line of HP41CV source code and identify it. COMP considers all HP41CV instructions to be of one of three types. The first category are the single byte instructions with no operands that can be compiled by a simple table look up. COMP has been constructed so that the instruction set can be extended at any time simply by increasing the size of this table. In this way abbreviated or altered command names could be easily used. The second category of instructions are the multi-byte instructions which require a table lookup and the translation of one or more operands, including possibly an indirect instruction indicator. The table examined by the compiler is the same as for the category one instructions, and a code is given in the table which indicates to the compiler the number of operands which are required with each instruction. A syntax check is then made in subroutines IONE and ITWO to insure that the number and characteristics of each operand are appropriate for the given instruction. One of the major advantages of the use of the cross compiler is the syntax and error checking that is performed during the compilation process. The third type of instruction represents the exceptional instructions that are so difficult to compile that they require separate subroutines for efficient compilation. These instructions include storage and recall of data, program labels and program flow control statements such as goto and execute.

In order to provide an efficient programming command system for the compiler that would minimize the need to know technical details about the operation of the compiler, an IBM EXEC II program was written. This program not only interfaces the user to the compiler, but it also provides on line user instructions as to how to use the system. Included in this command processor is a command menu which gives the format and short description for each command.

Another command, help, provides more detailed information about each command. When a novice user first enters the exec, or types the name of the exec followed by a question mark, then he receives a four page narative description of what the system is, how it works, and what actions he must take to write a successful HP41CV application program.

```
&TRACE
****************************************************************************
***                                                                       **
***     HP41CV CROSS COMPILER COMMAND PROCESSOR                           **
***                                                                       **
***     THIS IBM EXEC II PROGRAM PROVIDES AN INTERACTIVE                  **
***     PROGRAMMING ENVIRONMENT FOR THE CONSTRUCTION OF                   **
***     HP41CV CALCULATOR PROGRAMS.                                       **
***                                                                       **
***     WITH THE EXCEPTION OF THIS PROGRAM AND THREE OTHERS, ALL OF       **
***     THE SOFTWARE IN THE HP41CV SYSTEM IS DESIGNED TO BE               **
***     TRANSPORTABLE TO OTHER COMPUTER SYSTEMS WITHOUT EXTENSIVE         **
***     PROGRAM MODIFICATION.  THE INSTALLATION UNIQUE COMPONENTS         **
***     ARE IN THE FOLLOWING ROUTINES:                                    **
***                                                                       **
***              -- HP41CV EXEC      (THE COMMAND ENVIRONMENT)            **
***              -- VERSA FORTRAN    (PLCTTING SUBROUTINE)                **
***              -- PLOTPARM JCL     (PLOT CONTROL JCL)                   **
***              -- LBL XEDIT        (EDIT MACRO FOR LOWER CASE LABELS)**
***                                                                       **
***     ANOTHER VERSION OF THIS EXEC FOR USE WITH ASCII TERMINALS         **
***     HAS BEEN PROVIDED.  THIS ASCII ORIENTED EXEC MAY BE USED          **
***     BY ENTERING THE COMMAND "HP41C".  THE PRIMARY DIFFERENCES         **
***     BETWEEN THESE TWO EXECS IS THAT FOR ASCII TERMINALS THE           **
***     PRINTING OF THE COMMAND MENU IS SURPRESSED AFTER ONE PRINT        **
***     AND COMMANDS WHICH HAVE MEANING ONLY FOR VIDEO TERMINALS          **
***     SUCH AS FLIST , BROWSE AND XEDIT HAVE BEEN CHANGED TO THE         **
***     CORRESPONDING TYPEWRITER TERMINAL EQUIVALENTS SUCH AS             **
***     LISTFILE, TYPE AND EDIT.                                          **
***                                                                       **
***                                                                       **
***     FOR THE NEW USER OF THE SYSTEM, IT IS RECOMMENDED THAT            **
***     THIS PROGRAM BE EXECUTED SIMPLY BY TYPING THE COMMAND             **
***                                                                       **
***                    HP41CV                                             **
***                                                                       **
***    FOR EXPERIENCED USERS, WHO HAVE NO NEED FOR THE DESCRIPTIVE        **
***    INSTRUCTIONS, THE FOLLOWING COMMAND IS RECOMMENDED:                **
***                                                                       **
***                    HP41CV (FN) (1ST COMMAND)                          **
***                                                                       **
****************************************************************************
&IF /&2 = /HP41 &GOTO -CALLER
&IF /&2 ¬= / &STACK &2
&IF /&3 ¬= / &STACK &3
-CALLER
&IF /&4 ¬= / &STACK &4
&IF /&5 ¬= / &STACK &5
```

```
CP SET PF01 IMMED PF13
CP SET PF02 IMMED PF14
CP SET PF03 IMMED PF15
CP SET PF04 IMMED PF16
CP SET PF05 IMMED PF17
CP SET PF06 IMMED PF18
CP SET PF07 IMMED PF19
CP SET PF08 IMMED PF20
CP SET PF09 IMMED PF21
CP SET PF10 IMMED PF22
CP SET PF11 IMMED PF23
CP SET PF12 IMMED PF24
CP SET PF13 IMMED PF13
CP SET PF14 IMMED PF14
CP SET PF15 IMMED PF15
CP SET PF16 IMMED PF16
CP SET PF17 IMMED PF17
CP SET PF18 IMMED PF18
CP SET PF19 IMMED PF19
CP SET PF20 IMMED PF20
CP SET PF21 IMMED PF21
CP SET PF22 IMMED PF22
CP SET PF23 IMMED PF23
CP SET PF24 IMMED PF24
CP TERMINAL LINEND OFF
CP SPOOL PRINTER CONT
GLOBAL TXTLIB NONIMSL CMSLIB FORTMOD2 MCD2EEH IMSLSP
STATE INSTR CODES A
&IF &RETCODE ¬= 0 &USERMODE = R
&IF &INDEX GT 0 &GOTO -PROGNAM
CLRSCRN
&BEGTYPE -ENDINTRO
                    ***   HP41C CROSS COMPILER ***

 THIS PROGRAM IS USED TO MAKE IT  EASIER TO WRITE, DOCUMENT AND REVISE
 PROGRAMS FOR  THE HP41C  CALCULATOR.   AS  OUTPUT,  THIS  PROGRAM WILL
 PRODUCE  OPTICAL  BAR CODE FOR  DIRECT ENTRY  OF YOUR PROGRAM  INTO AN
 HP41C OR HP41CV CALCULATOR.


 WARNING    THIS PROGRAM IS PART OF AN CNGOING RESEARCH PROJECT AND AS
 SUCH  IS SUBJECT  TO CONSTANT  REVISION.   WHILE THERE  ARE NO  KNOWN
 ERRORS,  THE PROGRAM HAS NOT BEEN EXTENSIVELY TESTED.   TO INSURE THAT
 ANY ERRORS YOU DETECT ARE PROMPTLY CORRECTED, IT IS IMPORTANT THAT YOU
 SUBMIT AN ERROR REPORT TO THE PROGRAM PROPONENT AS SOON AS POSSIBLE.
```

IN ORDER TO GO FROM A PROGRAM IN YOUR HEAD TO THE FINISHED BAR CODE THERE ARE THREE MAIN STEPS:

(1) EDIT. THE PROGRAM MUST BE PREPARED AS INPUT TO THE CROSS COMPILER. THE EASIEST WAY TO DO THIS IS WITH THE CMS XEDIT FACILITY.

(2) COMPILE. THE PROGRAM MUST BE PROCESSED BY THE CROSS-COMPILER. THE CROSS-COMPILER IS ACTUALLY A FORTRAN PROGRAM WHICH PRODUCES TWO CMS FILES AS OUTPUT. BOTH THESE FILES HAVE THE SAME NAME AS YOUR PROGRAM NAME, BUT HAVE DIFFERENT FILE TYPES. THE "LISTING" FILE SHOWS THE RESULTS OF THE COMPILE STEP INCLUDING ANY ERRORS, AND THE "DATA" FILE IS A FILE OFZERO'S AND ONE'S USED BY THE BAR CODE GENERATOR.

(3) BAR. THE "DATA" FILE FROM THE COMPILE STEP IS USED AS INPUT TO PRODUCE THE ACTUAL BAR CODE. YOU SHOULD NEVER PER-. FORM THIS STEP UNTIL YOUR PROGRAM HAS SUCCESSFULLY COMPILED WITHOUT ERRORS. THIS STEP IS DONE BY THE BATCH PROCESSOR AND IT MAY TAKE SEVERAL HOURS TO GET YOUR FINISHED BAR CODE.

EXECUTION OF THE THREE STEPS NECESSARY TO PRODUCE BAR CODE IS UNDER YOUR CONTROL BY SELECTION OF THE APPROPRIATE STEP FROM A MENU OF COMMANDS WHICH WILL APPEAR AT YOUR TERMINAL SHORTLY.

THE FIRST STEP IN USING THE CROSS-COMPILER IS TO PREPARE THE SOURCE CODE (YOUR PROGRAM) ON CMS DISK. THE FIRST LINE OF A SOURCE CODE FILE MUST CONTAIN THE TITLE OF THE PROGRAM THAT IS TO BE USED AS A LABEL ON THE TOP OF THE BAR CODE. THIS TITLE SHOULD HAVE NC MORE THAN 40 LETTERS. TO HELP YOU REMEMBER THAT THE LABEL CF THE PROGRAM MUST BE THE FIRST LINE, YOU MAY RECEIVE A PROMPT ASKING YOU TO ENTER THE TITLE WHEN YOU FIRST DECLARE A NEW HP41 PROGRAM. AFTER YOU ENTER THE TITLE, YOUR TERMINAL WILL IMMEDIATELY SHIFT TO THE CMS EDITOR AND YOU WILL SEE THE TITLE AS THE FIRST LINE OF THE NEW FILE. THIS IS YOUR CUE TO ENTER THE HP41C PROGRAM THAT YOU HAVE WRITTEN. WHEN YOU EXECUTE A "FILE" COMMAND IN THE EDITOR MODE, THE TERMINAL WILL DISPLAY THE COMMAND MENU. YOU MAY THEN SELECT TO CROSS-COMPILE THE NEW PROGRAM OR ANY OTHER OPTION.

```
WHEN PREPARING  YOUR SOURCE CODE PLEASE  NOTE THAT LOWER  CASE LETTERS
ARE NOT THE SAME AS CAPITALS, AND IN MOST CASES LOWER CASE WILL NOT BE
ACCEPTED. IN ORDER TO MAKE IT EASY TO ENTER THE LOWER CASE ALPHABETIC
LABELS, AN XEDIT MACRO "LBL" HAS BEEN PROVIDED.  TO USE THIS MACRO,
SIMPLY TYPE IN THE XEDIT CCMMAND LINE, FOR EXAMPLE:

              LBL LOWER A   (FOR LOWER CASE "A" LABEL)

NOTE THAT THIS XEDIT MACRO ALSO DOES OTHER HELPFUL THINGS, SUCH AS
PROVIDING A BANNER TO HELP LOCATE LABELS AND DIRECTING THE CROSS-
COMPILER TO START A NEW PAGE (INDICATED BY "((" IN COLUMNS 1 AND 2.)
TO AVOID GOING TO A NEW PAGE WHEN YOU WRITE A LABEL, TYPE THE
OPTION "NOPAGE" AS FOLLOWS:

              LBL DOG NOPAGE (FOR AN ALPHA LABEL "DOG")


IN  THE FUTURE,   YOU  MAY  FIND IT  MORE  CONVIENIENT  TO SKIP  THESE
INSTRUCTIONS AND GO  DIRECTLY TO THE "MENU" OF COMMANDS.   TO DO THIS
SIMPLY TYPE THE  NAME OF THE CMS  FILE WHICH CONTAINS OR  WILL CONTAIN
YOUR  HP41C  SOURCE  CODE  INSTRUCTIONS  AFTER  THE  INVOKING  COMMAND
"HP41C".  AN EASY WAY TO DO THIS IS TO USE THE CMS "FLIST" FACILTITY.
FROM "FLIST" SIMPLY TYPE "PF19" IN THE COMMAND AREA.

 NOW, TO BEGIN:
-ENDINTRO
**********************************************************************
***                                                                 **
***  ESTABLISH A NEW HP41C PROGRAM SOURCE FILE.  INCLUDES TITLE     **
***  PROMPTING.                                                     **
***                                                                 **
**********************************************************************
-NEW
&BEGTYPE -ENDQQ
ENTER CMS FILENAME OF YOUR PROGRAM........(PF13 OR "STOP" RETURN TO CMS)
-ENDQQ
&SWITCH1 = ON
&READ ARGS
&GOTO -CHECK
-PROGNAM
```

```
&PROGNAME = &1
&PROGTYPE = HP41
&PROGMODE = A1
&SWITCH1 = OFF
STATE &PROGNAME &PROGTYPE &PROGMODE
&IF &RETCODE = 0 &GOTO -DISPLAY
CLRSCRN
&BEGTYPE -ENDINTRO2

ENTER THE LABEL YOU WISH TO HAVE PRINTED AT THE TOP OF THE BARCODE.
-ENDINTRO2
&READ ARGS
&STACK I &1 &2 &3 &4 &5 &6 &7 &8 &9 &10
&STACK LBL &PROGNAME
&STACK SET TABS 1 20 25 35 45 55 65
&STACK SET TRUNC 57
&STACK I
XEDIT &PROGNAME &PROGTYPE &PROGMODE
*************************************************************************
**                                                                     **
**                    COMMAND DISPLAY ROUTINE                          **
**                                                                     **
*************************************************************************
-DISPLAY
&IF /&SWITCH1 = /ON &GOTO -NEW

&IF /ASCII = /YES &GOTO -ENDDISP
CLRSCRN
&TYPE HP41C CROSS COMPILER ......... &PROGNAME ..........EDITION=17 SEP 81
&BEGTYPE -ENDDISP
SELECT DESIRED COMMAND FROM THE FOLLOWING:

PF-KEY   COMMAND  CODE   ACTION TAKEN BY PROGRAMMING  COMMAND SYSTEM

PF13      STOP      S    GETS YOU OUT OF THE HP41C CROSS COMPILER
PF14      HELP      H    SHORT EXPLANATION OF HOW TO USE THE CROSS COMPILER
PF15      ENTER     E    INTERACTIVE PROGRAM ENTRY (NO FILE CREATED)
PF16      BAR       B    SUBMIT JOB FOR PHYSICAL PRODUCTION OF BAR CODE
PF17      NEW       N    BEGIN WORK ON A NEW PROGRAM OR NAMED SUBROUTINE
PF18      DIREC     D    DIRECTORY OF COMMANDS
PF19      LIST      L    DISPLAY  NAMES   OF HP41C PROGRAMS ON DISK
PF20      OCOMP    C O   OFFLINE COMPILE AND AUTO GENERATION OF BAR CODE
PF21      PRINT     P    PRODUCE A HARDCOPY PRINTED LISTING OF THE PROGRAM
PF22       *        *    RESERVED FOR FUTURE USE BY HP41 EMMULATOR
PF23      COMP      C    COMPILE A SOURCE LISTING ON CMS DISK
PF24      XEDIT     X    EDIT THE PROGRAM USING THE CMS FULL-SCREEN EDITOR
          ERASE          ERASE THE SOURCE FILE,LISTING FILE AND TEXT FILE
          CMS            ALLOWS EXECUTION OF ANY VALID CMS COMMAND
```

```
        CP             ALLOWS EXECUTION OF ANY VALID CP COMMAND
-ENDDISP
&TYPE INPUT COMMAND:
&READ ARGS
CLRSCRN
*************************************************************************
**                                                                     **
**                   COMMAND CHECK ROUTINE                             **
**                                                                     **
*************************************************************************
-CHECK
&IF /CMS = /&1 &GOTO -CMSCMD
&IF /CP = /&1 &GOTO -CPCMD
&IF /PF13 = /&1 &GOTO -EXIT
&IF /PF14 = /&1 &GOTO -HELP
&IF /PF15 = /&1 &GOTO -ENTER
&IF /PF16 = /&1 &GOTO -SUBMIT
&IF /PF17 = /&1 &GOTO -NEW
&IF /PF18 = /&1 &GOTO -DISPLAY
&IF /PF19 = /&1 &GOTO -LISTFIL
&IF /PF20 = /&1 &GOTO -OCOMP
&IF /PF21 = /&1 &GOTO -TYPE
&IF /PF22 = /&1 &GOTO -NOTYET
&IF /PF23 = /&1 &GOTO -COMPILE
&IF /PF24 = /&1 &GOTO -XEDIT
&IF /STOP = /&1 &GOTO -EXIT
&IF /HELP = /&1 &GOTO -HELP
&IF /ENTER = /&1 &GOTO -ENTER
&IF /BAR = /&1 &GOTO -SUBMIT
&IF /NEW = /&1 &GOTO -NEW
&IF /DIREC = /&1 &GOTO -DISPLAY
&IF /LIST = /&1 &GOTO -LISTFIL
&IF /OCCMP = /&1 &GOTO -OCOMP
&IF /PRINT = /&1 &GOTO -TYPE
&IF /COMP = /&1 &GOTO -COMPILE
&IF /XEDIT = /&1 &GOTO -XEDIT
&IF /ERASE = /&1 &GOTO -ERASE
&IF /S = /&1 &GOTO -EXIT
&IF /H = /&1 &GOTO -HELP
&IF /E = /&1 &GOTO -ENTER
&IF /3 = /&1 &GOTC -SUBMIT
&IF /N = /&1 &GOTO -NEW
&IF /D = /&1 &GOTO -DISPLAY
&IF /P = /&1 &GOTO -TYPE
&IF /C = /&1 &IF /O = /&2 &GOTO -OCOMP
&IF /C = /&1 &GOTO -COMPILE
&IF /X = /&1 &GOTO -XEDIT
&IF /&1 = /FILEDEF &GOTO -INNER
```

```
&IF /&1 = /DATA &GOTO -INNER
&IF /&SWITCH1 = /ON &GOTO -PROGNAM
&TYPE ?? &1 &2 &3 &4 &5 UNRECOGNIZED
-ENDCHECK &GOTO -ENDDISP
*********************************************************
***                                                   **
***  PROCESS CMS OR CP COMMANDS PASSED TO THIS EXEC.  **
***                                                   **
***                                                   **
*********************************************************
-CMSCMD &ARG = &RANGE OF & 2 &N
&TRACE ON
&COMMAND &ARG
&TRACE OFF
&GOTO -ENDDISP
-CPCMD &ARG = &RANGE OF & 2 &N
&TRACE ON
CP &ARG
&TRACE OFF
&GOTO -ENDDISP
-XEDIT
&STACK SET TABS 1 20 25 35 45 55 65
&STACK SET TRUNC 57
XEDIT &PROGNAME &PROGTYPE &PROGMODE
&GOTO -DISPLAY
*********************************************************
**                                                    **
**    LIST HP41C PROGRAM FILES ON CMS DISK)           **
**                                                    **
*********************************************************
-LISTFIL
FLIST * HP41 A
&GOTO -DISPLAY
*********************************************************
**                                                    **
**                     COMPILE                        **
**                                                    **
*********************************************************
-COMPILE
FILEDEF 05 DISK &PROGNAME &PROGTYPE
FILEDEF 06 DISK &PROGNAME LISTING
FILEDEF 04 DISK &PROGNAME DATA
FILEDEF 02 DISK INSTR CODES &USERMODE
&TYPE CROSS-COMPILE BEGINS..
HPCROSS
BROWSE &PROGNAME LISTING
&GOTO -DISPLAY
*********************************************************
```

```
**                                                              **
**                   OFFLINE COMPILE                            **
**                                                              **
******************************************************************
-OCCMP
ERASE &PROGNAME DATA
FILEDEF 05 DISK &PROGNAME &PROGTYPE
FILEDEF 06 DISK &PROGNAME LISTING
FILEDEF 04 DISK &PROGNAME DATA
FILEDEF 02 DISK INSTR CODES &USERMODE
&TYPE CROSS-COMPILE BEGINS..
HPCROSS
PRINT &PROGNAME LISTING (UP
STATE &PROGNAME DATA
&IF &RETCODE = 0 &GOTO -SUBMIT
&TYPE  COMPILE OF &PROGNAME WAS NOT SUCCESSFUL.
&GOTO -ENDDISP
******************************************************************
***                                                             **
***   USING THE INTERACTIVE MODE, ENTER A NEW PROGRAM.          **
***                                                             **
******************************************************************
-ENTER
&BEGTYPE -ENDCAUTION
CAUTION.  USE OF THE INTERACTIVE ENTRY MODE REQUIRES THAT YOU PROPERLY
          CONTROL THE USE OF UPPER AND LOWER CASE.  ALSO, INTERACTIVE
          ENTRY DOES NOT CREATE A PERMANENT RECORD OF YOUR SOURCE CODE
          INPUT.  SHOULD A REVISION BE REQUIRED, YOU WOULD NEED TO
          RE-ENTER THE ENTIRE PROGRAM.  FOR THESE REASONS YOU MAY WISH
          TO XEDIT A SOURCE CODE FILE FIRST, AND THEN SUBMIT THIS FILE
          FOR CROSS-COMPILATION WITH THE "COMP" COMMAND.

          DO YOU WISH TO PROCEED WITH INTERACTIVE ENTRY? (Y/N)

INPUT RESPONSE:
-ENDCAUTION
&READ ARGS
&IF /&1 ¬= /Y  &GOTO -DISPLAY
CLRSCRN
&TYPE FIRST ENTER THE LABEL YOU WISH TO BE PRINTED AT THE TOP OF THE
&TYPE       BAR CODE.
&TYPE THEN  ENTER THE INSTRUCTIONS IN YOUR PROGRAM (IN UPPER CASE EXCEPT
&TYPE       FOR LOWER CASE ALPHABETIC LABELS WHICH ARE ALLOWED.)
&TYPE INPUT:
FILEDEF 04 DISK &PROGNAME DATA
FILEDEF 02 DISK INSTR CODES &USERMODE
HPCROSS
&GOTO -ENDDISP
```

```
************************************************************************
***                                                                  **
***    DISPLAY A MESSAGE THAT FUNCTION IS NOT AVAILABLE.             **
***                                                                  **
************************************************************************
-NOTYET
CLRSCRN
&BEGTYPE -ENDYET

THE FUNCTION YOU HAVE REQUESTED IS NOT YET AVAILABLE.  IF YOU HAVE ANY
IDEAS THAT SHOULD BE INCLUDED HERE, PLEASE CONTACT THE PROPONENT.

THANK YOU.
-ENDYET
&GOTO -ENDDISP
************************************************************************
**                                                                    **
**                    TYPE LISTING FILE                               **
**                                                                    **
************************************************************************
-TYPE
PRINT &PROGNAME LISTING (UP
&GOTO -DISPLAY
************************************************************************
**                                                                    **
**            ERASE SOURCE, LISTING AND TEXT FILES                    **
**                                                                    **
************************************************************************
-ERASE
ERASE &PROGNAME LISTING
ERASE &PROGNAME DATA
ERASE LOAD MAP
&TYPE WARNING     DO YOU WISH TO ERASE THE SOURCE CODE(YES/NO)?
&READ VARS &ANSW
&IF /&ANSW ¬= /YES &GOTO -DISPLAY
ERASE &PROGNAME &PROGTYPE &PROGMODE
&GOTO -DISPLAY
************************************************************************
**                                                                    **
**                 SUBMIT TO MVS FOR BATCH PROCESSING                 **
**                                                                    **
************************************************************************
-SUBMIT
-TRYSUBMIT
&PERM = SUBMIT
&BEGTYPE -ENDSUBM
ENTER JOB NAME AND USERID:
-ENDSUBM
```

```
&READ VARS &JN &USERID
&JN = &PIECE OF &JN 1 8
&USERID = &PIECE OF &USERID 1 4
&STACK I //&JN JOB (&USERID ,1011),'HP41CV BAR CODE',CLASS=A
&STACK C / ,1011)/,1011)/
&STACK I // EXEC FRTXCLGP
&STACK I //FORT.SYSIN DD *
&STACK GET VERSA FORTRAN &USERMODE
&STACK I //GO.PLOTPARM DD *
&STACK GET PLOTPARM JCL  &USERMODE
&STACK I //GO.SYSIN DD *
&STACK GET &PROGNAME DATA
&STACK I /*
&STACK FILE
XEDIT &PROGNAME JCL
EXEC SUBMIT &PROGNAME JCL
ERASE &PROGNAME JCL
&GOTO -DISPLAY
************************************************************************
**                                                                    **
**                     PRINT INSTRUCTIONS                             **
**                                                                    **
************************************************************************
-HELP
&BEGTYPE -ENDHELP

             HP41C CROSS COMPILER COMMAND PROCESSOR

YOU ARE CURRENTLY EXECUTING A CMS EXEC FILE THAT MAKES IT EASY TO INVOKE
THE HP41C CROSS COMPILER AND WRITE  PROGRAMS USING CMS AND THE IBM 3278
DISPLAY TERMINAL.  COMMON PROGRAMMING REQUIREMENTS SUCH AS EDITING CAN
BE ACCOMPLISHED IN THREE WAYS:

     --USING THE PROGRAMMED FUNCTION KEYS  (PF KEYS)
     --USING A SHORT COMMAND WORD
     --USING A ONE OR TWO LETTER MNEMONIC CODE

THE COMMAND ACTIONS AND THEIR ASSOCIATED PF KEYS AND CODES ARE ALL GIVEN
IN A DIRECTORY WHICH IS DISPLAYED WHEN THE COMMAND PROCESSOR IS WAITING
FOR YOUR INPUT.   MORE DETAILS ABOUT THE AVAILABLE COMMANDS FOLLOWS:



PF13   STOP   S    THIS COMMAND IS USED WHEN  YOU WISH TO STOP PROCESSING
```

			HP41C PROGRAMS AND RETURN TO CMS. IF YOU ARE EXECUTING A FUNCTION THAT WAS INVOKED FROM THE COMMAND MENU, IN MOST CASES PF13 WILL RETURN YOU TO THE MENU, AND BY PRESSING PF13 AGAIN YOU WILL RETURN TO CMS.
PF14	HELP	H	THIS COMMAND IS USED TO DISPLAY THE DETAILED EXPLANATION OF THE MENU COMMAND PROCESSOR AND ITS AVAILABLE COMMANDS. IF YOU HAVE QUESTIONS ABOUT THE PROCESS OF WRITING ACTUAL HP41C PROGRAMS YOU SHOULD CONSULT THE HP41 OWNER'S HANDBOOK.
PF15	ENTER	E	THIS COMMAND IS USED TO ENTER A PROGRAM USING THE CROSS-COMPILER IN AN INTERACTIVE MODE. THE ADVANTAGE OF THIS MODE IS THAT ANY SYNTACTICAL ERRORS IN THE HP41C PROGRAM ARE IMMEDIATELY IDENTIFIED BY THE CROSS-COMPILER AND AN ERROR MESSAGE IS SHCWN ON THE SCREEN. THE DISADVANTAGE IS THAT THE USER IS TOTALLY RESPONSIBLE FOR UPPER AND LOWER CASE BEING ENTERED PROPERLY.
PF16	BAR	B	THIS COMMAND IS USED ONCE THE HP41C PROGRAM IS WRITTEN AND COMPILED WITHOUT ERRORS. IT SUBMITS A JOB TO MVS BATCH FOR THE PHYSICAL PRODUCTION OF THE BAR CODE.
PF17	NEW	N	THIS COMMAND IS USED TO DIRECT THE ATTENTION OF THE COMMAND PROCESSOR TO A NEW HP41 PROGRAM SOURCE FILE. WHEN USED TO INITIATE NEW HP41C PROGRAMS, IT AUTOMATICALLY INSURES THAT A NEW FILE IS CREATED WITH FILETYPE "HP41" AND PROMPTS THE USER FOR THE PROGRAM TITLE WHICH IS THE MANDATORY FIRST LINE OF EVERY HP41C SOURCE CODE FILE.
PF18	DIREC	D	THIS COMMAND DISPLAYS THE FULL COMMAND MENU. IT HAS PRIMARY USE WHEN YOU FINISH AN OPERATION THAT FILLS THE SCREEN WITH TEXTUAL MATTER AND YOU RECEIVE ONLY THE PROMPT "INPUT COMMAND".
PF19	LIST	L	THIS COMMAND DISPLAYS "FLIST" FOR THOSE HP41C PROGRAMS THAT ARE ACTIVE ON YOUR A DISK. FROM THIS LIST, YOU CAN ERASE OLD PROGRAMS TO RELEASE DISK STORAGE, CHANGE THE NAME OF PROGRAMS, OR EXAMINE THE CONTENTS OF ANY PROGRAM.
PF20	CCOMP	C O	THIS COMMAND IS USED TO PRODUCE AN "OFFLINE" COMPILE. THE PROGRAM LISTING IS AUTOMATICALLY PRINTED IN HARD COPY ON THE HIGH SPEED PRINTER. IF THE COMPILE WAS WITHOUT ERROR THE BAR CODE IS AUTOMATICALLY PRODUCED.

```
PF21    PRINT   P     THIS COMMAND  PRINTS A COPY  OF THE "LISTING"  FILE ON
                      THE HIGH SPEED PRINTER.  IF YOU WISH TO HAVE A PRINTED
                      COPY OF  THE SOURCE CODE WITHOUT  THE CROSS-COMPILER'S
                      FEEDBACK,  IT IS BEST TO  SIMPLY PRINT THE  SOURCE CODE
                      CMS FILE BY ISSUING THE CMS PRINT COMMAND.

PF22    GO      G     THIS COMMAND  IS USED  TO INVOKE  THE HP41C  EMMULATOR
                      PROGRAM  WHICH ALLOWS  YOU TO  TEST  EXECUTION OF  THE
                      PROGRAM ON THE LARGE COMPUTER.  THE EMULATION PROGRAM
                      WILL EXECUTE  THE PROGRAM  EXACTLY AS  YOUR CALCULATOR
                      WOULD.  THIS COMMAND HAS NOT BEEN IMPLEMENTED AS  OF
                      17 SEP 81.


PF23    COMP    C     THIS COMMAND IS  USED TO INVOKE THE  CROSS COMPILER TO
                      TRANSLATE  AN HP41C  PROGRAM WRITTEN  ON  CMS DISK  IN
                      SOURCE  CODE FORM.   AFTER THE  COMPILE  THE USER  IS
                      AUTOMATICALLY PLACED  IN THE CMS  BROWSE MODE  FOR THE
                      OUTPUT "LISTING" FILE THAT RESULTED FROM THE COMPILE.

PF24    XEDIT   X     THIS COMMAND IS USED TO  INVOKE THE FULL-SCREEN EDITOR
                      TO MAKE MODIFICATIONS TO THE HP41C SOURCE CODE FILE.

IF YOU HAVE PROBLEMS USING THIS COMMAND PROCESSOR OR HAVE A SUGGESTION
FOR IMPROVEMENT, PLEASE CONTACT THE PROPONENT FOR THE HP41C SYSTEM.

-ENDHELP
&GOTC -ENDDISP
*************************************************************************
**                                                                     **
**                      EXIT COMMAND PROCESSOR                         **
**                                                                     **
*************************************************************************
-EXIT
CP SPOOL PRINTER CLOSE NOCONT
CLRSCRN
&EXIT
```

```
C**********************************************************************
C**                                                                  **
C**   THIS IS THE MAIN ROUTINE FOR THE HP41C CROSS-COMPILER.         **
C**                                                                  **
C**   INPUT TO THIS PROGRAM IS A SUPERSET OF HP41C INSTRUCTIONS. THE **
C**   REGULAR INSTRUCTIONS ARE WELL DOCUMENTED IN THE HP41C OWNER'S  **
C**   HANDBOOK AND PROGRAMMING GUIDE (SEE ESPECIALLY  THE INDEX ON   **
C**   PAGE 271).  IN ADDITION TO THESE REGULAR INSTRUCTIONS, THE     **
C**   FOLLOWING COMMANDS ARE SUPPORTED:                              **
C**                                                                  **
C**      (       INDICATES A COMMENT CARD (NO INSTRUCTIONS GENERATED)**
C**      XROM    SUBROUTINE CALL TO A READ ONLY MEMORY               **
C**      LBL     KEY ASSIGNMENTS MAY BE MADE AS PART OF ALPHA LBL    **
C**                                                                  **
C**   OUTPUT OF THIS PROGRAM IS AN INTERMEDIATE FILE OF ZERO'S AND   **
C**   ONE'S REPRESENTING HP41C MACHINE INSTRUCTIONS.  THIS FILE WILL **
C**   NORMALLY BE PASSED TO A FORTRAN PROGRAM WHICH DRAWS HP41C BAR  **
C**   CODE ON A HIGH RESOLUTION PLOTTER.                             **
C**                                                                  **
C**                                                                  **
C**********************************************************************
      INTEGER ASGN$,EQ$,POS$,SEG$,PARS$,CON$,IN$,COMP$,FIND$,LCUT$
      INTEGER TRIM$
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      COMMON/TABLE/INST$,LINST,CODE,NINST
      INTEGER*2 INST$(6,111)
      INTEGER*4 LINST(111),CODE(111),NINST
      INTEGER*2 T$(80),TITLE$(133)
      INTEGER*4 LT,LTITLE,MTITLE
      INTEGER*4 FUNC$/'MAIN'/
      INTEGER*2 COMENT/'('/
      INTEGER*4 UNDER(15)/15*'____'/
      INTEGER*4 M(8000)/8000*-997
      INTEGER*4 M1/1/,B1/1/
      INTEGER*4 LINCNT/0/,PAGE/72/
      LOGICAL ERROR/.FALSE./
      NUMPGE=1
      LTITLE=80
      MTITLE=1
      IDIM=80
      MAX=2000
      NINST=111
      PDIGIT=.FALSE.
      PALPHA=.FALSE.
```

```
      DONE=.FALSE.
C
C
C
      READ(2,101) IPRT,PAGE
      READ(2,9101)(TITLE$(JJ),JJ=1,25)
9101  FORMAT(25A1)
101   FORMAT(12I5)
      DO 5 I=1,NINST
         READ(2,102)(INST$(J,I),J=1,6),CCDE(I),LINST(I)
102      FORMAT(6A1,4X,I5,10X,I5)
5     CONTINUE
C
C
C
C
      READ(5,103) (TITLE$(I),I=26,IDIM)
103   FORMAT(75A1)
C
C
C
C
C
      LENGTH=0
      DO 16 I=1,MAX
C
C            ( I=THE INSTR NUMBER, J=CHARACTER IN INSTR)
C
C
C
C     ATTEMPT TO READ A TEXT STRING.
C
            IF(IPRT.GE.20)WRITE(6,292) UNDER
292         FORMAT('1','NEXT INSTRUCTION:',15A4,//)
C
            IF(IN$(T$,LT,5)) 14,12,12
C
C
C     GO TO THE FOLLOWING INSTRUCTIONS IF A CHARACTER STRING FOUND
C
C
C
C
C     CHECK FOR A COMMENT CARD AND/OR PAGE EJECT
C
C          *** TWO "COMENT" CHARACTERS IN POSITION 1 AND 2 OF AN INPUT
C              LINE ARE CONSIDERED A MANDATORY PAGE EJECT PRAGMA.   ***
C
```

```
C
12     IF(T$(1).NE.COMENT)GOTO 13
          IF(MOD(LINCNT,PAGE).EQ.0.OR.(T$(2).EQ.COMENT.AND.LT.GE.2))
     1         CALL NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE,1)
          LINCNT=LINCNT+1
          WRITE(6,268) (T$(J),J=1,LT)
268       FORMAT('         ',110A1)
          IF(IPRT.GE.10)WRITE(6,263)
263       FORMAT(' FOUND COMMENT CARD.  NOTHING MORE DONE.')
          GOTO 16
C
C
C
C      IF NOT A COMMENT, INCREMENT THE INSTRUCTION COUNTER AND PRINT
C      THE INSTRUCTION.
C
13     LENGTH=LENGTH+1
          IF(MOD(LINCNT,PAGE).EQ.0)
     1         CALL NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE,1)
C
          LINCNT=LINCNT+1
          WRITE(6,269)LENGTH,(T$(J),J=1,LT)
269       FORMAT('         ',I4,' ',110A1)
C
C
C         TRIM OFF TRAILING COMMENTS
C
          IF(FIND$(COMENT,1,T$,LT,LOC)) 6000,9915,9914
9914      LT=LOC-1
          IF(TRIM$(T$,LT)) 6000,9916,9915
9916      CONTINUE
          GOTO 16
9915      CONTINUE
C
C
C
C      COMPILE THE TEXT INSTRUCTION.
C
       IF(COMP$(T$,LT,M,M1)) 15,16,20
15     ERROR=.TRUE.
 16    CONTINUE
C
C
C
C      GOTO THE FOLLOWING INSTRUCTIONS IF END OF FILE ENCOUNTERED
C
 14    WRITE(6,259)
259    FORMAT(' ********************END OF FILE********************')
```

```
C
C
C     GCTO THE FOLLOWING INSTRUCTIONS IF END COMPILATION
C
20       IF(ERROR) STOP
C
C
C     CALL THE BAR CODE GENERATOR.
C
30       MSAVE=M1
         B1=1
         IBAR=BSTR$(M,M1,ITOT,TITLE$)
         WRITE(6,301) ITOT
301      FORMAT(' END HP41C CROSS COMPILE',I5,' BYTES IN TOTAL PROGRAM')
C
C
C
C
      STOP
C
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      MAIN$=0
      RETURN
6002  WRITE(6,6003) FUNC$
6003  FORMAT(' *** FUNCTION CALL ERROR *** ',A4)
      MAIN$=0
      RETURN
      END
```

```
      INTEGER FUNCTION AIN$(INOPER,B)
C*************************************************************************
C**                                                                     **
C**   THIS FUNCTION CONVERTS A DECIMAL NUMBER <=256 INTO A BINARY       **
C**   NUMBER, WITH THE VALUES OF THE BINARY DIGITS STORED IN            **
C**   SUCESSIVE ELEMENTS OF AN 8 ELEMENT ARRAY OF INTEGERS.             **
C**                                                                     **
C**   THIS FUNCTION IS CALLED FROM BSTR$ IN THE HP-41CV COMPILER.       **
C**                                                                     **
C**                                                                     **
C**                                                                     **
C**                                                                     **
C**   THE RETURN VALUE OF THE FUNCTION AIN$ IS SET AS FOLLOWS:        **
C**         0  =  CONTINUE TO COMPILE                                   **
C**        -1  =  AN ERROR IN COMPILING THE INSTRUCTION.                **
C**                                                                     **
C**                                                                     **
C*************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*4 FUNC$/'AIN'/
      INTEGER*2 B(8)
      OPERND=INOPER
C
C
C
C     CHECK FOR VALID ENTRY  OPERAND
C
         IF((OPERND.GT.255).OR.(OPERND.LT.0)) GOTO 6000
C
C
C
C     CONVERT THE FIRST    BINARY DIGIT
C
         D1=OPERND-128
         IF(D1) 100,110,110
100      B(1)=0
         GOTO 120
110      B(1)=1
         OPERND=D1
C
C
C     CONVERT THE SECOND   BINARY DIGIT
C
120      D2=OPERND-64
         IF(D2) 200,210,210
200      B(2)=0
         GOTO 230
```

```
210       B(2)=1
          OPERND=D2
C
C
C     CONVERT THE THIRD      BINARY DIGIT
C
230       D3=OPERND-32
          IF(D3) 300,310,310
300       B(3)=0
          GOTO 340
310       B(3)=1
          OPERND=D3
C
C
C     CONVERT THE FOURTH     BINARY DIGIT
C
340       D4=OPERND-16
          IF(D4) 400,410,410
400       B(4)=0
          GOTO 450
410       B(4)=1
          OPERND=D4
C
C
C     CONVERT THE FIFTH      BINARY DIGIT
C
450       D5=OPERND-8
          IF(D5) 500,510,510
500       B(5)=0
          GOTO 560
510       B(5)=1
          OPERND=D5
C
C
C     CCNVERT THE SIXTH      BINARY DIGIT
C
560       D6=OPERND-4
          IF(D6) 600,610,610
600       B(6)=0
          GOTO 670
610       B(6)=1
          OPERND=D6
C
C
C     CCNVERT THE SEVENTH  BINARY DIGIT
C
670       D7=OPERND-2
          IF(D7) 700,710,710
```

```
700        B(7)=0
           GOTO 780
710        B(7)=1
           OPERND=D7
C
C
C       CCNVERT THE EIGHTH   BINARY DIGIT
C
780        D8=OPERND-1
           IF(D8) 800,810,6000
800        B(8)=0
           GOTO 1000
810        B(8)=1
C
C
C
C       WRITE OUT CONVERSION IF NECESSARY AND RETURN
C
1000    IF(IPRT.GE.20) WRITE(6,66) INOPER,(B(I),I=1,8)
66      FORMAT(' TRACE           AIN$    OPERAND=',I5,' BINARY=  ',8I1)
        AIN$=0
        RETURN
C
C
C
C
C       ERROR HANDLING SECTION FOLLOWS
C
6000    WRITE(6,6001) FUNC$
6001    FORMAT(' ***** CONVERSION ERROR   ****** ',A4)
        WRITE(6,6002) INOPER
6002    FCRMAT(' ERROR IN       AIN$    OPERAND=',I5)
        AIN$=-1
        RETURN
        END
```

```
      INTEGER FUNCTION ALPH$(A$,LA,M,M1)
C************************************************************************
C**                                                                    **
C**  THIS FUNCTION INTERPRETS ALPHABETIC CHARACTERS INTO HP41C KEY     **
C**  CODES.                                                            **
C**                                                                    **
C**   THE RETURN VALUE OF THE FUNCTION ALPH$ IS SET AS FOLLOWS:        **
C**         0  =  CONTINUE TO COMPILE                                  **
C**        -1  =  AN ERROR IN COMPILING THE INSTRUCTION.               **
C**                                                                    **
C**                                                                    **
C************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA
      INTEGER*2 A$(IDIM)
      INTEGER*2 BLNK/' '/,QUOTE/'"'/
      INTEGER*4 FUNC$/'ALPH'/
      INTEGER*4 M(1)
      INTEGER*4 FUNC$/'COMP'/
      INTEGER*2 C$(60)/' ','#','$','%','&','*','+',',','-','.','/','0',
     2                 '1','2','3','4','5','6','7','8','9',':',';','<',
     3                 '=','>','?','@','A','B','C','D','E','F','G','H',
     4                 'I','J','K','L','M','N','O','P','Q','R','S','T',
     5                 'U','V','W','X','Y','Z','|',' ',' ',' ',' ',' '/
      INTEGER*4 C2(60)/32,29,36,37,126,42,43,44,45,46,47,48,49,50,51,52,
     2                 53,54,55,56,57,58,59,60,61,62,63,13,65,66,67,68,
     3                 69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,
     4                 85,86,87,88,89,90,94,97,98,99,100,101/
      INTEGER*4 LC/60/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' ALPH$   :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
        DO 35 I=1,LA
            IF(A$(I).NE.QUOTE) GOTO 15
C           HAVE FOUND A QUOTE MARK--DISREGARD IF 1ST INSTR,ELSE QUIT
                IF(I.EQ.1) GOTO 35
                    GOTO 40
15          IZ=FIND$(A$(I),LA,C$,LC,LOC)
            IF(IZ.NE.0)GOTO 20
                WRITE(6,207)M1
207             FORMAT(' *****  INVALID CHARACTER    *****',5X,I5)
                A$(I)=BLNK
```

```
            IZ=1
20        M(M1)=C2(IZ)
          IF(IPRT.GE.10)WRITE(6,208)M1,C$(IZ),C2(IZ),M(M1)
208       FORMAT(' ALPH$',I5,' ALPHA CHARACTER          ',3X,A1,3X,I3,
     2                                                   T75,I3)
          M1=M1+1
35      CONTINUE
C
C
40    ALPH$=0
      RETURN
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA,LB,IDIM
6010  FORMAT(' LA=',I10,'     LB=',I10,'     IDIM=',I10)
      ALPH$=-1
      RETURN
      END
```

```
      INTEGER FUNCTION ALP1$(A$,LA,M,M1)
C***********************************************************************
C**                                                                   **
C**   STRING A$  BEGINS WITH A QUOTE AND HENCE IS AN ALPHA ENTRY      **
C**   INSTRUCTION.  THIS ROUTINE COMPILES SUCH INSTRUCTIONS.          **
C**                                                                   **
C**                                                                   **
C**   THE RETURN VALUE OF THE FUNCTION ALP1$ IS SET AS FOLLOWS:       **
C**        0  =  CONTINUE TO COMPILE                                  **
C**       -1  =  AN ERROR IN COMPILING THE INSTRUCTION.               **
C**                                                                   **
C**                                                                   **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      INTEGER*2 A$(IDIM)
C     INTEGER*2 BLNK/' '/
      INTEGER*2 QUOTE/'"'/,APPEND/'¬'/
      INTEGER*4 FUNC$/'ALP1'/
      INTEGER*4 M(1)
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' ALP1$   :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     STRIP OFF THE LEADING QUOTE
C
         IF(LCUT$(A$,LA,1)) 6015,6015,10
C
C
C
C     STRIP OFF THE TRAILING QUOTE,IF ANY
C
10       IF(A$(LA).EQ.QUOTE)LA=LA-1
C
C
C
C     SET THE LENGTH OF THE INSTRUCTION
C
         IF(LA.LT.15)GOTO 15
              WRITE(6,204)
204           FORMAT(' ***** ALPHA STRING TOO LONG ******')
              LA=15
```

```
15        IBYTE=LA+1                                                        ALP00490
          M(M1)=IBYTE                                                       ALP00500
          IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE                                ALP00510
215       FORMAT(' ALP1$',I5,' LENGTH OF THIS INSTR IS',I3)                 ALP00520
          M1=M1+1                                                           ALP00530
C                                                                           ALP00540
C                                                                           ALP00550
C                                                                           ALP00560
C     ENCODE THE TEXT LENGTH INSTRUCTICN                                    ALP00570
C                                                                           ALP00580
          M(M1)=240+LA                                                      ALP00590
          IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)                                ALP00600
211       FORMAT(' ALP1$',I5,' TEXT LENGTH INSTR',T75,I3)                   ALP00610
          M1=M1+1                                                           ALP00620
C                                                                           ALP00630
C                                                                           ALP00640
C                                                                           ALP00650
C     CHECK FOR ALPHA APPEND INSTRUCTION                                    ALP00660
C                                                                           ALP00670
          IF(A$(1).NE.APPEND) GOTO 50                                       ALP00680
C                                                                           ALP00690
C               HAVE IDENTIFIED AN ALPHA APPEND INSTRUCTION                 ALP00700
                M(M1)=127                                                   ALP00710
                IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)                          ALP00720
214             FORMAT(' ALP1$',I5,' ALPHA APPEND CHAR',T75,I3)             ALP00730
                M1=M1+1                                                     ALP00740
                IF(LCUT$(A$,LA,1)) 6015,6015,50                             ALP00750
C                                                                           ALP00760
C                                                                           ALP00770
C                                                                           ALP00780
C     ENCODE TEXTUAL STRING                                                 ALP00790
C                                                                           ALP00800
50        ALP1$=ALPH$(A$,LA,N,M1)                                           ALP00810
          RETURN                                                            ALP00820
C                                                                           ALP00830
C     ERROR HANDLING SECTION FOLLOWS                                        ALP00840
C                                                                           ALP00850
6000  WRITE(6,6001) FUNC$                                                   ALP00860
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)                            ALP00870
      WRITE(6,6010) LA,LB,IDIM                                              ALP00880
6010  FORMAT(' LA=',I10,'      LB=',I10,'      IDIM=',I10)                  ALP00890
      ALP1$=-1                                                              ALP00900
      RETURN                                                                ALP00910
6015  WRITE(6,6016)                                                         ALP00920
6016  FORMAT(' ***** INVALID OPERAND IN ALPHA ENTRY INSTR ****')            ALP00930
      ALP1$=-1                                                              ALP00940
      RETURN                                                                ALP00950
      END                                                                   ALP00960
```

```
      INTEGER FUNCTION ASGN$(A$,LA,B$,LB)
C*********************************************************************
C**                                                                 **
C**     THIS FUNCTION  IS A STRING ASSIGNMENT OPERATOR.  THE STRING **
C**     IN A$ IS COPIED INTO B$.   THE NULL STRING LA=0 IS A VALID  **
C**     STRING AND WILL BE COPIED CORRECTLY.                        **
C**                                                                 **
C**                                                                 **
C*********************************************************************
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM),B$(IDIM)
      INTEGER*4 FUNC$/'ASGN'/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' ASGN$    :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
      IF(LA.EQ.0) GOTO 20
10    DO 15 I=1,LA
         B$(I)=A$(I)
15    CONTINUE
20    LB=LA
      ASGN$=1
      RETURN
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA,LB,IDIM
6010  FORMAT(' LA=',I10,'     LB=',I10,'     IDIM=',I10)
      ASGN$=0
      RETURN
      END
```

```
      INTEGER FUNCTION BSTR$(M,M1,TOTAL,TITLE$)
C***********************************************************************
C**                                                                   **
C**   THIS FUNCTION TAKES AN ARRAY (M) OF MACHINE CODE (DECIMAL)      **
C**   INSTRUCTIONS AND CONVERTS THEM INTO AN ARRAY (W) OF BINARY      **
C**   INSTRUCTIONS.  IT ALSO COMPUTES THE BARCODE CHECKSUM, AND       **
C**   SEGMENTS THE ARRAY INTO BARCODE LINES.                          **
C**                                                                   **
C**                                                                   **
C**   THE RETURN VALUE OF THE FUNCTION BSTR$ IS SET AS FOLLOWS:       **
C**         0  =  CONTINUE TO COMPILE                                 **
C**        -1  =  AN ERROR IN COMPILING THE INSTRUCTION.              **
C**                                                                   **
C**                                                                   **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 W(133),TITLE$(133)
      INTEGER*2 ALPHA(133),BLNK/' '/,ZERO/'0'/,ONE/'1'/
      INTEGER*4 FUNC$/'BSTR'/
      INTEGER*4 M(1),M1,W1
      IF(IPRT.GE.10) WRITE(6,200)
200   FORMAT(' TRACE ',I3,' BSTR$   :  ',)
C
C
C
C     INITIALIZE COUNTERS
C
         CHECK=0
         TOTAL=0
         SEQNUM=0
         LEAD=0
         ROW=0
         P=1
         W1=27
C
C
C
C     WRITE THE TITLE TO THE BINARY CODE ARRAY
C
         WRITE(4,776)(TITLE$(JJ),JJ=26,IDIM)
776      FORMAT(80A1)
C
C
C
C     CHECK FOR END OF PROGRAM
C
320      IF(M(P).LE.-99) GOTO 530
```

```
C
C
C
C     EXTRACT NUMBER OF BYTES IN  INSTRUCTION
C
         IBYTE=M(P)
         NBYTE=IBYTE
         P=P+1
C
C
C
C     EXTRACT NEXT OPERAND OF THE INSTRUCTION
C
390      OPERND=M(P)
         P=P+1
C
C
C
C     CONVERT OPERAND    TO BINARY AND LOAD INTO ARRAY W
C
         CHECK=CHECK+OPERND
         IF(CHECK.GT.255) CHECK=CHECK-255
         IF(IPRT.GE.10) WRITE(6,555)ROW,OPERND,CHECK
555   FORMAT(' SEND TO AIN$ ROW:',I3,'  OPERAND:',I6,'  CHECKSUM=',I5)
         IF(AIN$(OPERND,W(W1)))  6000,420,420
C
C
C
C     IF SUCCESSFUL CONVERSION, DECREASE BYTES REMAINING
C     INCREMENT THE ROW COUNT, AND CHECK TO SEE IF END OF BARCODE ROW
C
420      IBYTE=IBYTE-1
         W1=W1+8
         ROW=ROW+1
         IF(ROW.EQ.13) GOTO 530
C
C
C
C     CHECK TO SEE IF INSTRUCTION HAS BEEN COMPLETELY ENCODED
C
480      IF(IBYTE.EQ.0) GOTO 320
         GOTO 390
C
C
C
C     PROCESS END OF BARCODE ROW, FIRST SAVE ENDING LOCATION IN TEMP
C     BARCODE ROW (THIS LOCATION WILL BE DIFFERENT DEPENDING ON
C     WHETHER YOU ENTER ROUTINE BY DETECTING END CF ROW CR BY END OF
```

```
C      PROGRAM. THEN CHECK FOR CONTINUATION OF INSTRUCTION THAT MUST
C      CROSS BARCODE BOUNDARIES.
C
530       WP=W1
          IF(IBYTE.NE.0) GOTO 560
               TRAIL=0
               GOTO 580
C
C
C
C      CALCULATE NUMBER OF TRAILING BYTES IN BARCODE ROW
C
560       TRAIL=NBYTE-IBYTE
C
C
C
C      COMPUTE THIRD BYTE OF BARCODE ROW AND CONVERT TO BINARY
C
C      SINCE THE HP-41C INSTRUCTIONS ARE OF VARYING LENGTH, THEY WILL
C      MOST COMMONLY STRADDLE THE BORDER BETWEEN TWO ROWS OF BAR CODE.
C      THE THIRD BYTE OF A BAR-CODE ROW CONTAINS, IN THE 4 HIGH ORDER
C      BITS, THE NUMBER OF LEADING BYTES AND IN THE 4 LOW ORDER BITS
C      THE NUMBER OF TRAILING BYTES.
C
580       THIRD=(16*LEAD)+TRAIL
          W1=19
          CHECK=CHECK+THIRD
          IF(CHECK.GT.255) CHECK=CHECK-255
          IF(IPRT.GE.10) WRITE(6,555)ROW,THIRD,CHECK
          IF(AIN$(THIRD,W(W1)))  6000,1090,1090
C
C
C
C      COMPUTE SECOND BYTE OF BARCODE ROW AND CONVERT TO BINARY
C
C      THE SECOND BYTE IS SPLIT INTO TWO PARTS.  THE 4 HIGH ORDER BITS
C      CONTAIN THE PROGRAM TYPE (1=NONPRIVATE AND 2=PRIVATE), AND THE
C      4 LOW ORDER BITS CONTAIN THE SEQUENCE NUMBER, WHICH IS THE BAR-
C      CODE ROW NUMBER MINUS 1, MODULO 16.
C
1090      SECND=16+MOD(SEQNUM,16)
          SEQNUM=SEQNUM+1
          W1=11
          CHECK=CHECK+SECND
          IF(CHECK.GT.255) CHECK=CHECK-255
          IF(IPRT.GE.10) WRITE(6,555)ROW,SECND,CHECK
          IF(AIN$(SECND,W(W1)))  6000,1180,1180
C
```

```
C
C
C     COMPUTE FIRST  BYTE OF BARCODE RCW AND CONVERT TO BINARY
C
C     THE FIRST  BYTE CONTAINS THE CHECKSUM.  THIS BYTE IS A PARITY
C     CHECK IN THE FORM OF A RUNNING SUMMATION, MODULO 256 WITH A WRAP-
C     AROUND CARRY (0,1,2,...255,256,1,2,...).
C
C
1180      W1=3
          FIRST=CHECK
          IF(IPRT.GE.10) WRITE(6,555)ROW,FIRST,CHECK
          IF(AIN$(FIRST,W(W1)))  6000,1220,1220
C
C
C
C     ADD THE START AND STOP BITS AND ADD AN END CF ROW FLAG
C
1220      IF(IPRT.GE.20) WRITE(6,556)
556       FORMAT(' END OF BARCODE ROW**************************************')
          W(1)=0
          W(2)=0
          W(WP)=1
          ENDING=WP+1
          W(ENDING)=0
          ENDBIT=WP+2
          W(ENDBIT)=-99
C
C
C
C     TRANSFER THE COMPLETED BARCODE ROW EITHER DIRECTLY TO THE
C     PLOTTER, OR TO A AN ARRAY OF INTEGER*2 VARIABLES WHICH HOLD
C     ZERO'S OR ONE'S
C
C***********************************************************************
C
C       INSERT CALL TO VERSATEC HERE.
C
C***********************************************************************
C
C
          IF(IPRT.GE.20) WRITE(6,558) B1,ENDBIT
558       FORMAT(' TRANSFER TO BINARY ARRAY AT',I7,' NUMBER DIGITS',I5)
          DO 1350 I=1,ENDING
               IF(W(I).EQ.1) GOTO 559
                    ALPHA(I)=ZERO
                    GOTO 1350
559                 ALPHA(I)=ONE
```

```
1350       CONTINUE                                                         BST01930
           IF(ENDING.EQ.132) GOTO 1736                                      BST01940
           DO 1735 I=ENDBIT,132                                             BST01950
                ALPHA(I)=BLNK                                               BST01960
1735       CONTINUE                                                         BST01970
1736       WRITE(4,777) (ALPHA(I),I=1,132)                                  BST01980
777        FORMAT(66A1,/,5X,66A1)                                           BST01990
C                                                                           BST02000
C                                                                           BST02010
C                                                                           BST02020
C         IF REQUIRED, PRINT THE BARCODE RCW AS ZERO'S AND ONE'S ON PAPER   BST02030
C                                                                           BST02040
           IF(IPRT.GE.20)WRITE(6,201)(W(I),I=1,ENDING)                      BST02050
201        FORMAT(' ',132I1)                                                BST02060
C                                                                           BST02070
C                                                                           BST02080
C                                                                           BST02090
C                                                                           BST02100
C                                                                           BST02110
C        SET NUMBER OF LEADING BYTES FOR NEXT ROW AND RE-INITIALIZE         BST02120
C                                                                           BST02130
           LEAD=IBYTE                                                       BST02140
           TOTAL=TOTAL+ROW                                                  BST02150
           ROW=0                                                            BST02160
           W1=27                                                            BST02170
           IF(M(P)) 1400,480,480                                            BST02180
C                                                                           BST02190
C                                                                           BST02200
C                                                                           BST02210
C        SET FINAL VALUES AND RETURN                                        BST02220
C                                                                           BST02230
1400       BSTR$=0                                                          BST02240
           M1=1                                                             BST02250
           RETURN                                                           BST02260
C                                                                           BST02270
C                                                                           BST02280
C                                                                           BST02290
C        ERROR HANDLING SECTION FOLLOWS                                     BST02300
C                                                                           BST02310
6000     WRITE(6,6001) FUNC$                                                BST02320
6001     FORMAT(' ******  ERROR IN BARCODE PRODUCTION ****** ',A4)          BST02330
         PM1=P-1                                                            BST02340
         WRITE(6,6010) SEQNUM,PM1,M(PM1),CHECK                              BST02350
6010     FORMAT(' ROW=',I3,' M(',I4,')  OPERAND=',I3,' CHECKSUM=',I4)       BST02360
         BSTR$=-1                                                           BST02370
         RETURN                                                             BST02380
         END                                                                BST02390
```

```
      INTEGER FUNCTION COMP$(A$,LA,M,M1)
C**********************************************************************
C**                                                                  **
C**   THIS IS THE MASTER INSTRUCTION INTERPRETATION ROUTINE FOR THE  **
C**   HP41C COMPILER.  THIS ROUTINE IS USED BY BOTH THE BAR CODE     **
C**   GENERATOR AND THE CALCULATOR EMULATOR.                         **
C**                                                                  **
C**   INSTRUCTIONS ARE PASSED TO THIS ROUTINE ONE AT A TIME IN       **
C**   A TEXT STRING A$.  THE ARRAY M IS THE TOTAL ARRAY OF DECIMAL   **
C**   INTEGER KEY CODES (MACHINE INSTRUCTIONS), AND M1 IS THE        **
C**   POSITION WHERE  THE NEXT DECODED MACHINE INSTRUCTION WILL BE   **
C**   PLACED.  THUS, THE INPUT TO THIS ROUTINE IS A TEXTUAL HP41C    **
C**   INSTRUCTION AND THE OUTPUT IS ONE OR MORE DECIMAL KEY CODES    **
C**   PLACED APPROPRIATELY INTO ARRAY M.                             **
C**                                                                  **
C**   THE RETURN VALUE OF THE FUNCTION COMP$ IS SET AS FOLLOWS:      **
C**          1 =   END STATEMENT FOUND, END COMPILATION.             **
C**          0  =  CONTINUE TO COMPILE                               **
C**         -1  =  AN ERROR IN COMPILING THE INSTRUCTION.            **
C**                                                                  **
C**                                                                  **
C**                                                                  **
C**                                                                  **
C**********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      CCMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      INTEGER*2 A$(IDIM)
      INTEGER*2 BLNK/' '/
      INTEGER*2 T1$(40),T2$(40),SAV$(40),WORK$(80)
      INTEGER*2 SS1$(40),SS2$(40),SS3$(40),SS4$(40),SS5$(40)
      INTEGER*4 LT1,LT2,LSAV,LWORK
      INTEGER*4 LSS1,LSS2,LSS3,LSS4,LSS5
      INTEGER*4 M(1),M1
      INTEGER*4 FUNC$/'COMP'/
      INTEGER*2 END(3)/'E','N','D'/,
     1          RCL(3)/'R','C','L'/,
     3          QUOTE/'"'/,
C    4          APPEND/'¬'/,
     5          LBL(3)/'L','B','L'/,
     6          GTO(3)/'G','T','O'/,
     7          XEQ(3)/'X','E','Q'/,
     8          XRO(3)/'X','R','O'/,
     9          STO(3)/'S','T','O'/,P/'P'/,
     *          IND(3)/'I','N','D'/
```

```
C     2        END2(5)/'.','E','N','D','.'/,
C     2        AQUOTE(2)/'A','"'/,
C     2        MINUS/'-'/,
      INTEGER*4 MRCL(5)/144,32,'R','C','L'/,MSTO(5)/145,48,'S','T','O'/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' COMP$    :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C     SET FLAGS AND INITIALIZE COUNTERS
C
      INDIR=.FALSE.
C
C
C
C     CHECK FOR NULL STRING ENTRY INTO COMP$
C
5     IF(LA.NE.0) GOTO 10
            COMP$=0
            RETURN
C
C
C
C     CHECK FOR CATEGORY THREE SPECIAL INSTRUCTIONS
C
10    IF(EQ$(A$,LA,RCL,3,3)) 6000,15,11
11          COMP$=MEM$(A$,LA,M,M1,MRCL)
            PDIGIT=.FALSE.
            RETURN
C
15    IF(EQ$(A$,LA,STO,3,3)) 6000,20,16
C           MAKE A QUICK CHECK FOR THE STOP INSTRUCTION
16          IF(A$(4).EQ.P) GOTO 65
            COMP$=MEM$(A$,LA,M,M1,MSTO)
            PDIGIT=.FALSE.
            RETURN
C
20    IF(EQ$(A$,LA,LBL,3,3)) 6000,25,21
21          COMP$=LBL$(A$,LA,M,M1)
            PDIGIT=.FALSE.
            RETURN
C
25    IF(EQ$(A$,LA,GTO,3,3)) 6000,30,26
26          COMP$=GTO$(A$,LA,M,M1)
            PDIGIT=.FALSE.
            RETURN
C
30    IF(EQ$(A$,LA,XEQ,3,3)) 6000,35,31
```

```
31          COMP$=XEQ$(A$,LA,M,M1)
            PDIGIT=.FALSE.
            RETURN
C
35    IF(EQ$(A$,LA,XRO,3,3)) 6000,40,36
36          COMP$=XRO$(A$,LA,M,M1)
            PDIGIT=.FALSE.
            RETURN
C
40    IF(EQ$(A$,LA,END,3,3)) 6000,45,41
41          COMP$=END$(A$,LA,M,M1)
            PDIGIT=.FALSE.
            RETURN
C
C
C
C     CHECK FOR ALPHABETIC ENTRY INSTRUCTION.
C
45    IF(A$(1).NE.QUOTE) GOTO 50
46          COMP$=ALP1$(A$,LA,M,M1)
            PDIGIT=.FALSE.
            RETURN
C
C
C
C     CHECK FOR NUMERIC ENTRY INSTRUCTICN.
C
50    IF(NUMC$(A$,LA,IANSW))6000,55,51
51          COMP$=DIGT$(A$,LA,M,M1)
            PDIGIT=.TRUE.
            RETURN
C
C
C
C     CHECK FOR CATEGORY ONE INSTR(ONE BYTE) BY LOOKING FOR BLANK
C
55    P1=POS$(A$,LA,BLNK,1,1)
      IF(P1) 6000,65,70
C
C
C
C     NC BLANK IN STRING IMPLIES HAVE FCUND ONE BYTE INSTRUCTION
C
65       COMP$=IONE$(A$,LA,M,M1,1)
         PDIGIT=.FALSE.
         RETURN
C
C
```

```
C
C     BLANK IN STRING MEANS MULTI-WORD INSTRUCTION, NOW EXTRACT PREFIX
C
70       IF(PARS$(A$,LA,SS1$,LSS1)) 6000,65,75
C
C
C
C     CHECK FOR INDIRECT ADDRESSING
C
75    P6=POS$(A$,LA,IND,3,1)
      IF(P6) 6000,80,76
76            INDIR=.TRUE.
              IF(IPRT.GE.20)WRITE(6,235)
235           FORMAT(' DETECTED INDIRECT GOTO INSTRUCTION')
              IF(LCUT$(A$,LA,3)) 6000,6080,80
C
C
C
C     COMPILE THE PREFIX OF A MULTI-WORD INSTRUCTION
C
80       COMP$=IONE$(SS1$,LSS1,M,M1,2)
C
C
C
C     EXTRACT THE POSTFIX OF A MULTI-WORD INSTRUCTION
C
85       IF(PARS$(A$,LA,SS2$,LSS2)) 6000,90,6090
CC
C
C
C
C     COMPILE THE POSTFIX OF A MULTI-WORD INSTRUCTION
C
90       COMP$=MINO(COMP$,ITWO$(SS2$,LSS2,M,M1,INDIR))
         PDIGIT=.FALSE.
         RETURN
C
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      COMP$=-1
      RETURN
6080  WRITE(6,6081)
6081  FORMAT(' ***** ERROR                     *****')
      CCMP$=-1
```

```
      RETURN                                                            COM01930
 6090 WRITE(6,6091)                                                     COM01940
 6091 FORMAT(' ***** ERROR                          *****')             COM01950
      COMP$=-1                                                          COM01960
      RETURN                                                            COM01970
      END                                                               COM01980
```

```
      INTEGER FUNCTION CON$(A$,LA,B$,LB,C$,LC)
C***********************************************************************
C**                                                                   **
C**    STRING A$ AND STRING B$ ARE CONCATENATED AND PLACED IN C$.     **
C**                                                                   **
C**    IT IS FEASIBLE TO CON$(A$,LA,B$,LB,A$,LA)  OR                  **
C**                      CON$(A$,LA,B$,LB,B$,LB)                      **
C**    IN WHICH CASE THE APPROPRIATE STRING WILL BE REPLACED.         **
C**                                                                   **
C**                                                                   **
C**    THE NUMBER OF CHARACTERS IN THE RESULTING STRING C$ IS RETURNED**
C**    AS THE VALUE OF THE FUNCTION CON$, UNLESS THERE IS A LOSS OF   **
C**    CHARACTERS IN WHICH CASE THE NUMBER OF LOST CHARACTERS IS      **
C**    RETURNED AS A NEGATIVE NUMBER.                                 **
C**                                                                   **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM),B$(IDIM),C$(IDIM)
      INTEGER*4 FUNC$/'CON'/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
      IF(IPRT.GE.10) WRITE(6,201)LB,(B$(I),I=1,LB)
200   FORMAT(' TRACE ',I3,' CON$   :   ',110A1)
201   FORMAT('     AND',I3,' B$     :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
      IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C
C
C
      LOSS=0
C
C
C     DETERMINE LENGTH OF RESULT
C
      ILC=LA+LB
      IF(ILC.LE.IDIM) GOTO 20
         LOSS=ILC-IDIM
         ILB=LB-LOSS
         ILC=IDIM
         IF(IPRT.GE.05) WRITE(6,202) LOSS
202      FORMAT(' LOSS OF',I3,' CHARACTERS DURING CONCATENATION')
         GOTO 25
20    ILB=LB
25    ILA=LA
      INDEX=ILC
C
C
C
```

```
C      MCVE B$ INTO C$
C
       IF(ILB.LE.0) GOTO 40
       IND=ILB
       DC 35 I=1,ILB
          C$(INDEX)=B$(IND)
          IF(IPRT.GE.30)WRITE(6,207) IND,B$(IND),INDEX,C$(INDEX)
207       FORMAT(' MOVE B(',I3,')=',A1,'  IS NOW C(',I3,')=',A1)
          IND=IND-1
          INDEX=INDEX-1
35     CONTINUE
C
C
C
C      MCVE A$ INTO C$
C
40     IF(ILA.LE.0) GOTO 60
       IND=ILA
       DC 45 I=1,ILA
          C$(INDEX)=A$(IND)
          IF(IPRT.GE.30)WRITE(6,209) IND,A$(IND),INCEX,C$(INDEX)
209       FORMAT(' MOVE A(',I3,')=',A1,'  IS NOW C(',I3,')=',A1)
          IND=IND-1
          INDEX=INDEX-1
45     CONTINUE
C
C
C
C      SET LENGTH OF C$ AND ASSIGN VALUE OF CONS$ AND RETURN.
C
60     LC=ILC
65     IF(IPRT.GE.20) WRITE(6,203)LC,(C$(I),I=1,LC)
203    FORMAT(' CONCAT: LC=',I3,' "',110A1)
       IF(LOSS.NE.0) GOTO 70
          CON$=ILC
          GOTO 75
70     CON$=-LOSS
75     RETURN
C
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000   WRITE(6,6001) FUNC$
6001   FORMAT(' *** STRING LENGTH ERROR *** ',A4)
       CCN$=-1
       RETURN
       END
```

```
      INTEGER FUNCTION DIGT$(A$,LA,M,M1)
C*********************************************************************
C**                                                                 **
C**  THIS IS A FUNCTION THAT IS PART OF THE HP41C COMPILER.  IT IS  **
C**  CALLED WHEN A DIGIT ENTRY INSTRUCTION IS ENCOUNTERED.          **
C**                                                                 **
C**                                                                 **
C**   THE RETURN VALUE OF THE FUNCTION DIGT$ IS SET AS FOLLOWS:     **
C**         0  =  CONTINUE TO COMPILE                               **
C**        -1  =  AN ERROR IN COMPILING THE INSTRUCTION.            **
C**                                                                 **
C**                                                                 **
C*********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*2 A$(IDIM)
      INTEGER*2 PLUS/'+'/
      INTEGER*2 C$(13)/'0','1','2','3','4','5','6','7','8','9','.',
     2                 'E','-'/
      INTEGER*4 LC/13/
      INTEGER*4 FUNC$/'DIGT'/
      INTEGER*4 M(1)
      LOGICAL PDECIM,CHSFLG
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' DIGT$    :  ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     ADD A NULL INSTRUCTION BETWEEN ADJACENT DIGIT ENTRY INSTR.
C
      IF(.NOT.PDIGIT)GOTO 400
C        ADJACENT DIGIT ENTRY INSTRUCTION FOUND
         IBYTE=1
         M(M1)=IBYTE
         IF(IPRT.GE.20)WRITE(6,212)M1,IBYTE
212      FORMAT(' DIGT$',I5,' LENGTH OF THIS INSTR IS',I3)
         M1=M1+1
         M(M1)=0
         IF(IPRT.GE.20)WRITE(6,213)M1,M(M1)
213      FORMAT(' DIGT$',I5,' NULL INSTR FOR PRECEEDING DIGIT ENTRY',
     2                                            T75,I3)
         M1=M1+1
C
C
C
```

```
C
C       CHECK FOR DIGIT ENTRY INSTRUCTION PRECEEDED BY PLUS SIGN.
C
400     IF(A$(1).NE.PLUS)GOTO 450
C
C          NOTE THAT YOU GO AROUND THE FOLLOWING LINE  IF THE FIRST
C          DIGIT IS NOT A PLUS SIGN OR IF THE PLUS SIGN IS ALL ALONE.
C          A PLUS SIGN BY ITSELF INDICATES ADDITION NOT A DIGIT
C          ENTRY INSTRUCTION.  ADDITION IS COMPILED BY A TABLE LOOKUP.
C
               CALL LCUT$(A$,LA,1)
C
C
C
C       SET THE LENGTH OF THE INSTRUCTION
C
450        IBYTE=LA
           M(M1)=IBYTE
           IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215        FORMAT(' DIGT$',I5,' LENGTH OF THIS INSTR IS',I3)
           M1=M1+1
C
C
C
           DO 35 I=1,LA
15              IZ=FIND$(A$(I),LA,C$,LC,LOC)
                IF(IZ.NE.0)GOTO 20
                     WRITE(6,207)M1
207                  FORMAT(' ***** INVALID CHARACTER    *****',5X,I5)
                     DIGT$=-1
                     RETURN
20              M(M1)=IZ+15
                IF(IPRT.GE.10)WRITE(6,208)M1,C$(IZ),M(M1)
208             FORMAT(' DIGT$',I5,' DIGIT ENTRY INSTR     ',3X,A1,3X,
     2                                                      T75,I3)
                M1=M1+1
35         CONTINUE
C
C
40      DIGT$=0
        RETURN
C
C
C
C       ERROR HANDLING SECTION FOLLOWS
C
6000    WRITE(6,6001) FUNC$
6001    FORMAT(' *** STRING LENGTH ERROR *** ',A4)
```

```
      WRITE(6,6010) LA,LB,IDIM                                          DIG00970
 6010 FORMAT(' LA=',I10,'      LB=',I10,'      IDIM=',I10)              DIG00980
      DIGT$=-1                                                          DIG00990
      RETURN                                                            DIG01000
 6999 WRITE(6,602)M1                                                    DIG01010
  602 FORMAT(' ***** DIGIT ENTRY INSTR ERROR **',5X,I5)                 DIG01020
      DIGT$=-1                                                          DIG01030
      RETURN                                                            DIG01040
      END                                                               DIG01050
```

```
      INTEGER FUNCTION END$(A$,LA,M,M1)
C**********************************************************************
C**                                                                  **
C**   STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN END INSTRUCTION.   **
C**                                                                  **
C**                                                                  **
C**                                                                  **
C**   THE RETURN VALUE OF THE FUNCTION END$ IS SET AS FOLLOWS:      **
C**         0  =  CONTINUE TO COMPILE                                **
C**        -1  =  AN ERROR IN COMPILING THE INSTRUCTION.             **
C**                                                                  **
C**                                                                  **
C**********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM, IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
C
      COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
C
C     THIS SUBROUTINE PASSES THE NUMBER OF ELEMENTS IN M VIA COMMON /CNT
C
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      INTEGER*2 A$(IDIM)
C     INTEGER*2 BLNK/' '/
      INTEGER*4 FUNC$/'END'/
      INTEGER*4 M(1)
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' END$    :    ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C
C
C     END INSTRUCTION IS THREE BYTES LONG.  INDICATE LENGTH OF INSTR.
C
         M(M1)=3
         IF(IPRT.GE.20)WRITE(6,201)M1,IBYTE
201      FORMAT('   END$',I5,' LENGTH OF NEXT INSTR IS',I3)
         M1=M1+1
C
C
C
C     LOAD PREFIX CODE FOR END INSTRUCTION.  193 IS USED INSTEAD OF 192.
C     WHILE 192 IS THE HP STANDARD OP-CODE FOR "END", THE 193 IS USED TO
C     ENABLE 192 TO STAND FOR AN ALPHA LABEL INSTRUCTION.  THIS USAGE
C     IS STANDARD AMONG THE HP USERS GROUP PRACTICING SYNTHETIC PROGRAM-
```

```
C       MING.
C
           M(M1)=193
           IF(IPRT.GE.10)WRITE(6,210)M1,M(M1)
210        FORMAT('   END$',I5,'          END     INSTR',T75,I3)
           M1=M1+1
C
C
C
C       PROVIDE TWO NULL INSTRUCTIONS TO RESERVE SPACE FOR THE LINK
C       POINTERS.  ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
C       POINTERS WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN.  THIS
C       CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
C       BOUNDARIES WITHIN THE HP41CV MEMORY.  THE CHAIN OF LABELS IS
C       RECOMPILED BY THE WAND SOFTWARE, SO THE BYTES CONTAINING THE
C       CHAIN ARE SET TO ZERO BY THIS COMPILER.
C
           M(M1)=0
           IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211        FORMAT('   END$',I5,' TRAILNG NULL INSTR',T75,I3)
           M1=M1+1
           M(M1)=0
           IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
212        FORMAT('   END$',I5,' POINTER WILL BE RECOMPILED',T75,I3)
C
C
C
C       NOTE NUMBER OF ELEMENTS IN THE MACHINE CODE ARRAY AND SET END FLAG
C
           S2=M1
           M1=M1+1
C          DONE=.TRUE.
           WRITE(6,202)S2
202        FORMAT('1COMPILATION ENDED:',I5,' MACHINE CODES GENERATED')
           END$=1
           RETURN
C
C
C       ERROR HANDLING SECTION FOLLOWS
C
6000    WRITE(6,6001) FUNC$
6001    FORMAT(' *** STRING LENGTH ERROR *** ',A4)
        WRITE(6,6010) LA,LB,IDIM
6010    FORMAT(' LA=',I10,'      LB=',I10,'      IDIM=',I10)
        END$=-1
        RETURN
        END
```

```
      INTEGER FUNCTION EQ$(A$,LA,B$,LB,NUM)
C*********************************************************************
C**                                                                 **
C**      THIS FUNCTION  TESTS FOR STRING EQUALITY.  FOR THIS FUNCTION **
C**      THE RETURN VALUE IS CRUCIAL AS IT CONTAINS THE RESULTS OF THE **
C**      TEST FOR EQUALITY.                                         **
C**                                                                 **
C**      NUM DEFINES THE NUMBER OF CHARACTERS TO BE EXAMINED FOR    **
C**      EQUALITY, STARTING FROM THE LEFT MOST POSITION OF BOTH     **
C**      STRINGS.  THUS, THE STRINGS:                               **
C**                                                                 **
C**              A$='ABCDEFG' AND   B$='ABC'                        **
C**                                                                 **
C**      WILL BE "EQUAL" IF TESTED WITH  EQ$(A$,LA,B$,LB,LB)        **
C**      BUT "UNEQUAL" IF TESTED WITH    EQ$(A$,LA,B$,LB,LA)        **
C**                                                                 **
C**      TO TEST FOR ABSOLUTE EQUALITY, JUST ASIGN NUM TO BE SOME   **
C**      ARBITRARILY LARGE INTEGER,SAY 1000.  THE COMPARISON WILL   **
C**      TERMINATE APPROPRIATELY AT THE END OF THE SHORTEST STRING. **
C**                                                                 **
C**      IE.      EQ$(A$,LA,B$,LB,IDIM) WILL TEST ABSOLUTE EQUALITY **
C**                                                                 **
C**                                                                 **
C**      IT IS SUGGESTED THAT THIS ROUTINE BE USED IN AN ARITHMETIC **
C**      IF STATEMENT OF THE FORM:                                  **
C**                                                                 **
C**            IF(EQ$(A$,LA,B$,LB,LA)) 6002, 10, 20                 **
C**                                                                 **
C**         WHERE:    6002 IS AN ERROR HANDLING ROUTINE             **
C**                   10   IS THE ROUTINE WHEN STRINGS ARE NOT EQUAL **
C**                   20   IS THE ROUTINE WHEN STRINGS ARE EQUAL    **
C**                                                                 **
C*********************************************************************
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM),B$(IDIM)
      INTEGER*4 FUNC$/'EQ'/

      INTEGER*4 LA,LB,NUM
      IF(IPRT.GE.10) WRITE(6,200)LA,NUM,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' EQ$(',I3,'):',110A1)
      IF(IPRT.GE.10)WRITE(6,199)LB,(B$(I),I=1,LB)
199   FORMAT('   AND ',I3,'      :  ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
      IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C
C
C
      LENGTH=NUM
```

```
      IF((LENGTH.LE.LA).AND.(LENGTH.LE.LB))GOTO   10
         IF(LA.NE.LB) GOTO 5
            LENGTH=LA
            GOTO 10
C
C
C     STRINGS CAN NOT BE EQUAL BECAUSE HAVE BEEN ASKED TO EXAMINE
C     MORE CHARACTERS THAN SMALLEST STRING IN A COMPARISON OF UNEQUAL
C     STRINGS.
C
5     EQ$=0
      IF(IPRT.GE.20)WRITE(6,202)
      RETURN
C
C
C
C     EXAMINE CHARACTERS ONE-BY-ONE TO TEST FOR EQUALITY.
C
10    DO 15 I=1,LENGTH
         IF(IPRT.GE.30) WRITE(6,201) I,A$(I),I,B$(I)
201      FORMAT(' COMPARE A$(',I3,')=',A1,' WITH B$(',I3,')=',A1)
         IF(A$(I).EQ.B$(I)) GOTO 15
              EQ$=0
              IF(IPRT.GE.20)WRITE(6,202)I
202           FORMAT(' STRINGS FOUND UNEQUAL',I3,' POSITION')
              RETURN
15    CONTINUE
C
C
C     IF YOU GET BELOW HERE THE STRINGS WERE FOUND TO BE EQUAL
C
      EQ$=1
      IF(IPRT.GE.20)WRITE(6,203)
203   FORMAT(' STRINGS FOUND   EQUAL')
      RETURN
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FCRMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA,LB,IDIM
6010  FCRMAT(' LA=',I10,'     LB=',I10,'     IDIM=',I10)
      EQ$=-9999
      RETURN
      END
```

```
      INTEGER FUNCTION FIND$(A$,LA,B$,LB,LOC)
C************************************************************************
C**                                                                   **
C**   "FIND A$ IN B$."                                                **
C**                                                                   **
C**   STRING B$ IS SEARCHED FOR THE FIRST OCCURANCE OF A MATCH WITH   **
C**   CHARACTER A$.  A$ IS NOT ALLOWED TO BE MORE THAN ONE CHARACTER  **
C**   IN LENGTH.                                                      **
C**                                                                   **
C**   SINCE B$ IS MOST LIKELY A TABLE OF CHARACTERS, IT IS ALLOWED,   **
C**   AND MOST OFTEN IS OF A GREATER DIMENSION THAN IDIM, THE STANDARD **
C**   STRING DIMENSION.                                               **
C**                                                                   **
C**    THE VALUE OF THE FUNCTION FIND$ IS SET TO                      **
C**          LOC (LOCATION OF FIRST MATCH IN B$) IF MATCH FOUND       **
C**            0 NO MATCH IS FOUND                                    **
C**           -1 IF AN ERROR IS ENCOUNTERED.                          **
C**                                                                   **
C************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(1),B$(IDIM)
      INTEGER*2 OBJECT
      INTEGER*4 FUNC$/'FIND'/
      INTEGER*4 LOC,FIND$
      IF(IPRT.GE.10) WRITE(6,200)LA,A$(1)
200   FORMAT(' TRACE ',I3,' FIND$    :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
      IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C
C
C
      OBJECT=A$(1)
      INDEX=1
      DO 25 I=1,LB
         IF(IPRT.GE.40) WRITE(6,211) OBJECT,I,B$(I)
211      FORMAT(' COMPARE OBJECT=',A1,' WITH B$(',I3,')=',A1)
         IF(OBJECT.EQ.B$(INDEX))GOTO 30
         INDEX=INDEX+1
25    CONTINUE
C
C
C
C     NO MATCH FOUND
C
      LOC=0
      FIND$=0
      IF(IPRT.GE.20)WRITE(6,201)LOC
```

```
201    FORMAT(' NO SINGLE CHARACTER MATCH FOUND',I2)                    FIN00490
       RETURN                                                          FIN00500
C                                                                      FIN00510
C                                                                      FIN00520
C                                                                      FIN00530
C      HAVE FOUND A MATCH                                              FIN00540
C                                                                      FIN00550
30     LOC=INDEX                                                       FIN00560
       FIND$=INDEX                                                     FIN00570
       IF(IPRT.GE.30)WRITE(6,202)LOC                                   FIN00580
202    FORMAT(' HAVE FOUND SINGLE CHARACTER MATCH AT',I3)              FIN00590
       RETURN                                                          FIN00600
C                                                                      FIN00610
C                                                                      FIN00620
C                                                                      FIN00630
C      ERROR HANDLING SECTION FOLLOWS                                  FIN00640
C                                                                      FIN00650
6000   WRITE(6,6001) FUNC$                                             FIN00660
6001   FORMAT(' *** STRING LENGTH ERROR *** ',A4)                      FIN00670
       WRITE(6,6010) LA,LB,IDIM                                        FIN00680
6010   FORMAT(' LA=',I10,'     LB=',I10,'     IDIM=',I10)              FIN00690
       FIND$=-1                                                        FIN00700
       RETURN                                                          FIN00710
       END                                                             FIN00720
```

```
      INTEGER FUNCTION GTO$(A$,LA,M,M1)
C***********************************************************************
C**                                                                  **
C**   STRING A$  HAS BEEN IDENTIFIED TO CONTAIN A GTO INSTRUCTION.   **
C**                                                                  **
C**   THE RETURN VALUE OF THE FUNCTION GTO$ IS SET AS FOLLOWS:      **
C**        0  =  CONTINUE TO COMPILE                                 **
C**       -1  =  AN ERROR IN COMPILING THE INSTRUCTION.              **
C**                                                                  **
C**                                                                  **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      LCGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      INTEGER*2 A$(IDIM)
      INTEGER*2 BLNK/' '/
      INTEGER*2 QUOTE/'"'/
      INTEGER*2 IND(3)/'I','N','D'/
      INTEGER*2 LABEL(26)/'A','B','C','C','E','F','G','H','I','J',
     2                    'T','Z','Y','X','L',' ',' ',' ',' ',' ',
     3                    ' ',' ',' ',' ',' ',' '/
      INTEGER*4 FUNC$/'GTO'/
      INTEGER*4 M(1)
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' GTO$   :  ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     ESTABLISH DEFAULT PREFIX AND INSTRUCTION LENGTH VALUES
C     (THESE ARE THE VALUES FOR 3 BYTE LOCAL NUMERIC GOTO WITHOUT IND)
C
         IBYTE=3
         PREFIX=208
C
C
C
C     STRIP STRING OF "GTO" CHARACTERS.
C
         CALL LCUT$(A$,LA,3)
         IF(TRIM$(A$,LA)) 6015,6015,10
C
C
C
C     CHECK FOR ALPHANUMERIC VERSUS LOCAL LABELS
```

```
C                                                                       GT000490
10       IF(A$(1).EQ.QUOTE) GOTO 80                                     GT000500
C                                                                       GT000510
C                                                                       GT000520
C                                                                       GT000530
C     PROCESS LOCAL LABELS, FIRST CHECK FOR INDIRECT GTO INSTR          GT000540
C                                                                       GT000550
         P1=POS$(A$,LA,BLNK,1,1)                                        GT000560
         IF(P1) 6015,20,15                                              GT000570
C                                                                       GT000580
C                                                                       GT000590
C                                                                       GT000600
C     PROCESS GTO INDIRECT INSTRUCTION.                                 GT000610
C                                                                       GT000620
15       IF(EQ$(A$,LA,IND,3,3)) 6015,6020,16                            GT000630
16       CALL LCUT$(A$,LA,P1)                                           GT000640
         IF(IPRT.GE.20)WRITE(6,235)                                     GT000650
235      FORMAT(' DETECTED INDIRECT GOTO INSTRUCTION')                  GT000660
         INDIR=.TRUE.                                                   GT000670
         IBYTE=2                                                        GT000680
         PREFIX=174                                                     GT000690
C                                                                       GT000700
C                                                                       GT000710
C                                                                       GT000720
C     CHECK FOR NUMERIC OPERAND                                         GT000730
C                                                                       GT000740
20       IF(NUMC$(A$,LA,IANSW)) 6015,25,50                              GT000750
C                                                                       GT000760
C                                                                       GT000770
C                                                                       GT000780
C                                                                       GT000790
C                                                                       GT000800
C     OPERAND MUST BE REGISTER X,Y,Z,T CR L  OR A LOCAL ALPHA LABEL     GT000810
C                                                                       GT000820
25       DO 30 I=1,26                                                   GT000830
              INDEX=I                                                   GT000840
              IF(A$(1).EQ.LABEL(I)) GOTO 35                             GT000850
30       CONTINUE                                                       GT000860
C                                                                       GT000870
C        WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND         GT000880
         GOTO 6015                                                      GT000890
C                                                                       GT000900
C                                                                       GT000910
C                                                                       GT000920
C     HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPER-GT000930
C     AND.  THEN GOTO PROCESS A THREE BYTE INSTRUCTION.                 GT000940
C                                                                       GT000950
35       INDEX=INDEX+101                                                GT000960
```

```
      GOTO 75
C
C
C
C
C
C     OPERAND  MUST BE A NUMERIC LOCAL LABEL
C
50       IF(IVAL$(A$,LA,INDEX))6015,55,55
C
C
C
C     HAVE FOUND VALID NUMERIC LOCAL LABEL, CHECK FOR TWO BYTE INSTR
C
55       IF(INDEX.GT.14.OR.INDIR) GOTO 75
C
C
C
C     PROCESS A TWO BYTE INSTRUCTION, FIRST LOAD THE LENGTH OF INSTR
C
      IBYTE=2
      M(M1)=IBYTE
      IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
      M1=M1+1
C
C
C
C     HAVE FOUND VALID NUMERIC LOCAL LABEL <15, LOAD "GTO" INSTRUCTION
C
      M(M1)=177+INDEX
      IF(IPRT.GE.10)WRITE(6,213)M1,M(M1)
213   FORMAT('  GTO$',I5,' TWO BYTE GTO INSTR',T75,I3)
      M1=M1+1
C
C
C
C     LOAD NULL INSTR FOR TWO BYTE GTO INSTR  AND RETURN
C
      M(M1)=0
      IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)
      M1=M1+1
      GTO$=0
      RETURN
C
C
C
C      PROCESS   THE GOTO   INSTRUCTION  (OPERAND>14)
C
```

```
75        M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
210       FORMAT('  GTO$',I5,' LENGTH OF NEXT INSTR IS',I3)
          M1=M1+1
C
C
C
C      LOAD    THE     GTO INSTR  PREFIX
C
          M(M1)=PREFIX
          IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211       FORMAT('  GTO$',I5,' GTO PREFIX     INSTR',T75,I3)
          M1=M1+1
C
C
C
C      LOAD THREE BYTE GOTO INSTR NULL INSTR (POSITION HOLDER FOR POINTER
C
          IF(INDIR) GOTO 95
          M(M1)=0
          IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)
221       FORMAT('  GTO$',I5,' NULL  FOR  GTO INSTR',T75,I3)
          M1=M1+1
C
C
C
C      LOAD THE 2D OPERAND OF THE        GTO INSTR
C
C95       IF(INDIR) INDEX=INDEX+128
C         NOTE THAT FOR GTO IND THE HIGH ORDER BIT IS NOT SET
95        M(M1)=INDEX
          IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
212       FORMAT('  GTO$',I5,' GTO 2D OPERAND     ',T75,I3)
          M1=M1+1
          GTO$=0
          RETURN
C
C
C*************************************************************************
C*************************************************************************
C
C
C      PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE
C
80        K=0
          P2=POS$(A$,LA,QUOTE,1,2)
          IF(P2) 6015,120,85
C
```

```
C
C
C     CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS
C
85        LEFT=LA-P2
          IF(LEFT) 6015,100,6015
C
C
C     DELETE THE ENDING QUOTE BY TRUNCATING THE STRING ONE CHAR
C
100       LA=LA-1
C
C
C
C     SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL
C  (LINE 120 ACCOUNTS FOR BEGINNING QUOTE STILL ON STRING)
C
120       LENGTH=LA-1
C         FOR GTO H=2      FOR LBL H=4          FOR XEQ H=2
          H=2
          IBYTE=H+LENGTH
          M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
          M1=M1+1
C
C
C
C     HAVE FOUND VALID ALPHANUMERIC  LABEL,      LCAD "GTO" INSTRUCTION
C
          PREFIX=29
C         FOR GTO PREFIX=29    FOR LBL PREFIX=192    FOR XEQ PREFIX=30
          M(M1)=PREFIX
          IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
214       FORMAT('  GTO$',I5,' ALPHA    GTO INSTR',T75,I3)
          M1=M1+1
C
C
C
C
C
C     SET INDICATOR FOR NUMBER OF ALPHA CHARS IN LABEL
C
          U=240
C         FOR GTO U=240      FOR LBL   U=241        FOR XEQ U=240
          M(M1)=U+LENGTH
          IF(IPRT.GE.10)WRITE(6,216)M1,M(M1)
216       FORMAT('  GTO$',I5,' LENGTH CODE ALPH GTO',T75,I3)
          M1=M1+1
```

```
C
C
C
C
C
C      ADD ALPHABETIC CHARACTERS AND RETURN
C
140        LA=LENGTH+1
           GTO$=ALPH$(A$,LA,M,M1)
           RETURN
C
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000   WRITE(6,6001) FUNC$
6001   FORMAT(' *** STRING LENGTH ERROR *** ',A4)
       WRITE(6,6010) LA,LB,IDIM
6010   FORMAT(' LA=',I10,'     LB=',I10,'     IDIM=',I10)
       GTO$=-1
       RETURN
6015   WRITE(6,6016)
6016   FORMAT(' ***** INVALID SECOND OPERAND IN GTO INSTR ****')
       GTO$=-1
       RETURN
6020   WRITE(6,6021)
6021   FORMAT(' ***** FOUND THREE OPERANCS, EXPECTING IND *****')
       GTO$=-1
       RETURN
       END
```

```
      INTEGER FUNCTION IN$(A$,LA,IN)
C**********************************************************************
C**                                                                 **
C**   STRING A$ IS READ FROM UNIT IN.                               **
C**                                                                 **
C**   THE LENGTH OF A$ IS AUTOMATICALLY COMPUTED, NOT COUNTING ANY  **
C**   LEADING OR TRAILING BLANKS, WHICH ARE TRIMMED AWAY.           **
C**                                                                 **
C**   THE INPUT READER ASSUMES AN FIXED LENGTH INPUT RECORD OF 80   **
C**   CHARS.                                                        **
C**                                                                 **
C**********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM)
      INTEGER*2 BLNK/' '/
      INTEGER*2 CARD(80)
      INTEGER*4 FUNC$/'IN'/
      LOGICAL EOFILE(10)/10*.FALSE./
100   FORMAT(80A1)
C
C
C
C     CHECK FOR END OF FILE
C
      IF(.NOT.EOFILE(IN)) GOTO 5
         IN$=-1
         LA=0
         IF(IPRT.GE.20) WRITE(6,201) IN
201      FORMAT(' ATTEMPT TO READ AFTER END OF FILE ON UNIT ',I2)
         RETURN
C
C
C
C     READ THE ACTUAL CARD
C
5     READ (IN,100,END=999) (CARD(I),I=1,80)
      IF(IPRT.GE.20) WRITE(6,222)(CARD(I),I=1,78)
222   FORMAT(' ',78A1)
C
C
C
C     CHECK CARD FOR TRAILING BLANKS
C
58    IM=0
      DO 60 I=1,80
         INDEX=81-I
         IF(CARD(INDEX).NE.BLNK) GOTO 65
```

```
      IM=IM+1                                                          IN$00490
60    CONTINUE                                                         IN$00500
65    IF(IM.EQ.0) GOTO 70                                              IN$00510
         IF(IPRT.GE.20) WRITE(6,207) IM                                IN$00520
207      FORMAT(' FOUND',I3,' TRAILING BLANKS IN INPUT STRING')       IN$00530
         IF(IM.NE.80) GOTO 70                                          IN$00540
              LA=0                                                     IN$00550
              IN$=0                                                    IN$00560
              IF(IPRT.GE.20) WRITE(6,208)                              IN$00570
208           FORMAT(' FOUND INPUT STRING IS ALL BLANKS')              IN$00580
              IF(IPRT.GE.20) WRITE(6,200)LA,(A$(I),I=1,LA)             IN$00590
              RETURN                                                   IN$00600
70       IEND=80-IM                                                    IN$00610
C                                                                      IN$00620
C                                                                      IN$00630
C                                                                      IN$00640
C     CHECK CARD FOR LEADING BLANKS                                    IN$00650
C                                                                      IN$00660
C                                                                      IN$00670
      IM=0                                                             IN$00680
      DO 10 I=1,IEND                                                   IN$00690
         IF(CARD(I).NE.BLNK) GOTO 15                                   IN$00700
         IM=IM+1                                                       IN$00710
10    CONTINUE                                                         IN$00720
15    IF(IM.EQ.0) GOTO 25                                              IN$00730
         IF(IPRT.GE.20) WRITE(6,211) IM                                IN$00740
211      FORMAT(' FOUND',I3,' LEADING BLANKS IN INPUT STRING')        IN$00750
25       IBEG=1+IM                                                     IN$00760
C                                                                      IN$00770
C                                                                      IN$00780
C                                                                      IN$00790
C     DETERMINE LENGTH OF INPUT STRING                                 IN$00800
C                                                                      IN$00810
      LA=IEND-IBEG+1                                                   IN$00820
      IF(LA.LE.IDIM) GOTO 30                                           IN$00830
           LOSS=LA-IDIM                                                IN$00840
           IF(IPRT.GE.10) WRITE(6,216) LOSS                            IN$00850
216        FORMAT(' STRING TOO LONG FOR MAX STRING LENGTH.  LOST',I3)  IN$00860
           LA=IDIM                                                     IN$00870
           IEND=IEND-LOSS                                              IN$00880
C                                                                      IN$00890
C                                                                      IN$00900
C                                                                      IN$00910
C     TRANSFER THE CARD CHARACTERS TO THE INPUT STRING.                IN$00920
C                                                                      IN$00930
30    INDEX=1                                                          IN$00940
      DC 85 I=IBEG,IEND                                                IN$00950
         A$(INDEX)=CARD(I)                                             IN$00960
```

```
      IF(IPRT.GE.30) WRITE(6,209) I,CARD(I),INDEX,A$(INDEX)
209   FORMAT(' MOVE CARD(',I3,')=',A1,'  IS NOW C(',I3,')=',A1)
      INDEX=INDEX+1
85    CONTINUE
C
C
C
C     CHECK FOR STRING ERROR  AND RETURN
C
      IF(IPRT.GE.20) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' IN$   :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
      IN$=LA
      RETURN
C
C
C
C     HANDLE END OF FILE CONDITION
C
999   ECFILE(IN)=.TRUE.
      IN$=-1
      LA=0
      IF(IPRT.GE.20) WRITE(6,215)
215   FCRMAT(' END OF FILE ENCOUNTERED')
      RETURN
C
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      IN$=-1
      RETURN
      END
```

```
      INTEGER FUNCTION IONE$(A$,LA,M,M1,IBYTE)
C*****************************************************************************
C**                                                                         **
C**   THIS FUNCTION IS THE TABLE DRIVEN INSTRUCTION LOOKUP.  IT IS          **
C**   USED TO TRANSLATE THE ONE BYTE INSTRUCTIONS IN THE HP41C              **
C**   COMPILER.                                                             **
C**                                                                         **
C**                                                                         **
C*****************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM, IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      COMMON/TABLE/INST$,LINST,CODE,NINST
      INTEGER*2 INST$(6,111)
      INTEGER*4 LINST(111),CODE(111),NINST
      INTEGER*2 A$(IDIM)
      INTEGER*4 FUNC$/'IONE'/
      INTEGER*4 M(1)
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' IONE$    :  ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C
      DO 50 I=1,NINST
         LENGTH=LINST(I)
         IF(LA.NE.LENGTH) GOTO 50
         DO 30 J=1,LENGTH
            IF(A$(J).NE.INST$(J,I))GOTO 50
30       CONTINUE
C
C
C        INSTRUCTION MATCHES SO CHECK TO SEE IF CORRECT NUMBER
C        OPERANDS.  INSTRUCTIONS 143 OR LESS MUST HAVE ONLY ONE
C        OPERAND.  INSTRUCTIONS 144 OR MORE MUST HAVE MORE THAN ONE.
C
         IF(IBYTE.EQ.1.AND.CODE(I).LE.143) GOTO 35
         IF(IBYTE.EQ.2.AND.CODE(I).GT.143) GOTO 35
         GOTO 6020
C
C
C
C        LOAD CORRECTLY MATCHING VALUES TO MACHINE CODE ARRAY
C
35       M(M1)=IBYTE
         IF(IPRT.GE.20)WRITE(6,212)M1,IBYTE
```

```
212       FORMAT(' IONE$',I5,' LENGTH OF NEXT INSTR IS',I3)
          M1=M1+1
          M(M1)=CODE(I)
          IF(IPRT.GE.10)WRITE(6,210)M1,(INST$(JJ,I),JJ=1,6),M(M1)
210       FORMAT(' IONE$',I5,' ',6A1,      ' INSTR',T75,I3)
          M1=M1+1
C         IF(LA.GT.LENGTH) GOTO 35
               LA=0
               IONE$=0
               RETURN
C
C
C
C      FCLLOWING COMMENT LINES HAVE BEEN RETAINED TO FACILITATE USE OF
C      PROGRAM UNDER RULE THAT THE LENGTH OF A$ MAY BE MORE THAN THE
C      LENGTH OF THE MATCH STRING.  CODE HAS BEEN TESTED AND PROVEN TO
C      SELECT FIRST SUBSTRING MATCH IN TABLE.
C
C35        LEFT=LA-LENGTH
C          LSTART=LENGTH+1
C          IF(SEG$(A$,LA,A$,LA,LSTART,LEFT).GT.0) GOTO 40
C                IONE$=0
C                RETURN
C40        IONE$=LA
C          RETURN
C
C
C
C      INSTRUCTION DOES NOT MATCH SO CHECK NEXT INSTRUCTION
C
50     CCNTINUE
C
C
C      NO MATCH FOUND
C
60     WRITE(6,215)
215    FORMAT(' ***** UNRECOGNIZABLE INSTRUCTION ****')
          M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,212)M1,IBYTE
          M1=M1+1
          M(M1)=0
          IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211       FORMAT(' IONE$',I5,' NULL         INSTR',T75,I3)
          M1=M1+1
C
       IONE$=-1
       RETURN
C
```

```
C       ERROR HANDLING SECTION FOLLOWS
C
6000    WRITE(6,6001) FUNC$
6001    FORMAT(' *** STRING LENGTH ERROR *** ',A4)
        WRITE(6,6010) LA,LB,IDIM
6010    FORMAT(' LA=',I10,'      LB=',I10,'      IDIM=',I10)
        ICNE$=-1
        RETURN
6020    WRITE(6,6021)
6021    FORMAT(' *** LENGTH OF INSTRUCTION DOES NOT MATCH NUMBER OPERNDS')
        IONE$=-1
        RETURN
        END
```

```
      INTEGER FUNCTION IRDR$(IDEV,VAL)
C***********************************************************************
C**                                                                  **
C**    THIS FUNCTION READS A FREE FORMAT VALUE AND CONVERTS IT TO    **
C**    AN INTEGER.                                                   **
C**                                                                  **
C**                                                                  **
C**                                                                  **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(80)
      INTEGER*2 BLNK/' '/,ZERO/'0'/,MINUS/'-'/
      DATA LL/256/
      INTEGER*4   RVAL,SIGN,IFN,FRAC
      INTEGER*4 IRDR$,VAL,IDEV
      IF(IPRT.GE.20)WRITE(6,200)IDEV
200   FORMAT(' TRACE IRDR$ ',I5)
C
C
C
C
      IF(IN$(A$,LA,IDEV))6000,12,12
C
C
12    SIGN=1
      IFN=0
   15 DO 1 II=1,LA
         IF(IPRT.GE.30) WRITE(6,213)A$(II)
213      FORMAT(' EXAMINING ',A1,' FOR INTEGER VALUE')
         IF(A$(II).EQ.MINUS)GO TO 10
         IF(A$(II).EQ.BLNK)GOTO 1
   20    ITEMP=A$(II)-ZERO
         ITEMP=ITEMP/LL
         IF((ITEMP.GT.9).OR.(ITEMP.LT.0))GOTO 6007
         IF(IPRT.GE.30) WRITE(6,216) ITEMP
216      FORMAT(' FOUND NUMERIC DIGIT',I2)
         IFN=IFN*10+ITEMP
         GO TO 1
   10    SIGN=-1
         IF(IPRT.GE.30)WRITE(6,214)
214      FORMAT(' FOUND MINUS SIGN')
    1    CONTINUE
217   FORMAT(' SUBROUTINE IRDR$ RETURNING VALUE ',I20)
      IF(IPRT.GE.30)WRITE(6,2225) IFN ,SIGN
2225  FORMAT(' SET FINAL SIGN OF',I20,' WITH',I20)

  660 IRDR$=SIGN*IFN
```

```
      VAL=IRDR$
      IF(IPRT.GE.20)WRITE(6,217) VAL
      RETURN
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001)
6001  FORMAT(' *** END OF FILE DETECTED *** ',A4)
      IRDR$=-1
      STOP
6007  WRITE(6,6008)(A$(I),I=1,LA)
6008  FORMAT(' *** ATTEMPT TO FIND INTG VALUE OF ALPHABETIC STRING:'/
     2      ' ',110A1)
      STOP
      END
```

```
      INTEGER FUNCTION ITWO$(A$,LA,M,M1,INDIR)
C*********************************************************************
C**                                                                 **
C**   STRING A$  IS A POSTFIX FOR A MULTI-WORD INSTRUCTICN.         **
C**                                                                 **
C**   THIS ROUTINE WILL EXAMINE THE POSTFIX AND RETURN A DECIMAL    **
C**   VALUE INTERPRETATION OF THE POSTFIX.                          **
C**                                                                 **
C**   INDIRECT INSTRUCTIONS WILL HAVE THE POSTFIX APPROPRIATELY     **
C**   SET WITH THE HIGH ORDER BIT ON, AS REQUIRED BY THE INDIR FLAG **
C**                                                                 **
C**                                                                 **
C**   THE RETURN VALUE OF THE FUNCTION ITWO$ IS SET AS FOLLOWS:     **
C**         0  =  CONTINUE TO COMPILE                               **
C**        -1  =  AN ERROR IN COMPILING THE INSTRUCTION.            **
C**                                                                 **
C**                                                                 **
C*********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM)
      INTEGER*2 BLNK/' '/
      INTEGER*2 LABEL(26)/'A','B','C','D','E','F','G','H','I','J',
     2                    'T','Z','Y','X','L',' ',' ',' ',' ',' ',
     3                    ' ',' ',' ',' ',' ',' '/
      INTEGER*4 FUNC$/'ITWO'/
      INTEGER*4 M(1)
      LOGICAL INDIR
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FCRMAT(' TRACE ',I3,' ITWO$   :  ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     CHECK FOR BLANK INDICATING A BOGUS THIRD OPERAND
C
         IF(TRIM$(A$,LA)) 6015,6030,10
10       IF(POS$(A$,LA,BLNK,1,1)) 6015,20,6020
C
C
C
C     CHECK FOR NUMERIC OPERAND
C
20       IF(NUMC$(A$,LA,IANSW)) 6015,25,50
C
C
C
C
```

```
C                                                                       ITW00490
C     OPERAND MUST BE REGISTER X,Y,Z,T OR L  OR A LOCAL ALPHA LABEL     ITW00500
C                                                                       ITW00510
25       DO 30 I=1,26                                                   ITW00520
              INDEX=I                                                   ITW00530
              IF(A$(1).EQ.LABEL(I)) GOTO 35                             ITW00540
30       CONTINUE                                                       ITW00550
C                                                                       ITW00560
C        WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND         ITW00570
         GOTO 6015                                                      ITW00580
C                                                                       ITW00590
C                                                                       ITW00600
C                                                                       ITW00610
C     HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPER-ITW00620
C     AND.  THEN GOTO PROCESS SECTION TO ACTUALLY LOAD BYTE.            ITW00630
C                                                                       ITW00640
35       INDEX=INDEX+101                                                ITW00650
         GOTO 75                                                        ITW00660
C                                                                       ITW00670
C                                                                       ITW00680
C                                                                       ITW00690
C                                                                       ITW00700
C                                                                       ITW00710
C     OPERAND  MUST BE A NUMERIC LOCAL LABEL                            ITW00720
C                                                                       ITW00730
50       IF(IVAL$(A$,LA,INDEX))6015,75,75                               ITW00740
C                                                                       ITW00750
C                                                                       ITW00760
C                                                                       ITW00770
C     HAVE FOUND VALID POSTFIX, CHECK FCR INDIRECT INSTRUCTION          ITW00780
C                                                                       ITW00790
75       IF(INDIR) INDEX=INDEX+128                                      ITW00800
C                                                                       ITW00810
C                                                                       ITW00820
C                                                                       ITW00830
C     LOAD THE SECOND OPERAND IN THE MACHINE CODE ARRAY                 ITW00840
C                                                                       ITW00850
95       M(M1)=INDEX                                                    ITW00860
         IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)                             ITW00870
212      FORMAT(' ITWO$',I5,'      2D OPERAND     ',T75,I3)             ITW00880
         M1=M1+1                                                        ITW00890
C                                                                       ITW00900
C                                                                       ITW00910
C                                                                       ITW00920
C     CLEAN-UP AND RETURN                                               ITW00930
C                                                                       ITW00940
      INDIR=.FALSE.                                                     ITW00950
      ITWO$=0                                                           ITW00960
```

```
      RETURN
C
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA,LB,IDIM
6010  FORMAT(' LA=',I10,'      LB=',I10,'      IDIM=',I10)
      ITWO$=-1
      RETURN
6015  WRITE(6,6016)
6016  FORMAT(' ***** INVALID SECOND OPERAND IN  INSTR ****')
      ITWO$=-1
      RETURN
6020  WRITE(6,6021)
6021  FORMAT(' ***** FOUND THREE OPERANDS, EXPECTING IND *****')
      ITWO$=-1
      RETURN
6030  WRITE(6,6031)
6031  FORMAT(' ****** FOUND SECOND OPERAND BLANK *******')
      ITWO$=-1
      RETURN
      END
```

```
      INTEGER FUNCTION IVAL$(A$,LA,VAL)
C***********************************************************************
C**                                                                   **
C**   CONVERTS A NUMERIC TEXT STRING TO INTEGER NUMERIC VALUE.        **
C**                                                                   **
C**                                                                   **
C**                                                                   **
C**                                                                   **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM)
      INTEGER*2 BLNK/' '/,ZERO/'0'/,MINUS/'-'/
      INTEGER*4 FUNC$/'VAL'/
      DATA LL/256/
      INTEGER*4   RVAL,SIGN,IFN,FRAC
      INTEGER*4 IVAL$,VAL
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' IVAL$    :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C
C
C
      SIGN=1
      IFN=0
   15 DO 1 II=1,LA
         IF(IPRT.GE.30) WRITE(6,213)A$(II)
213      FORMAT(' EXAMINING ',A1,' FOR INTEGER VALUE')
         IF(A$(II).EQ.MINUS)GO TO 10
         IF(A$(II).EQ.BLNK)GOTO 1
   20    ITEMP=A$(II)-ZERO
         ITEMP=ITEMP/LL
         IF((ITEMP.GT.9).OR.(ITEMP.LT.0))GOTO 6007
         IF(IPRT.GE.30) WRITE(6,216) ITEMP
216      FORMAT(' FOUND NUMERIC DIGIT',I2)
         IFN=IFN*10+ITEMP
         GO TO 1
   10    SIGN=-1
         IF(IPRT.GE.30)WRITE(6,214)
214      FORMAT(' FOUND MINUS SIGN')
    1    CONTINUE
217   FORMAT(' SUBROUTINE IVAL$ RETURNING VALUE ',I20)
      IF(IPRT.GE.30)WRITE(6,2225) IFN ,SIGN
2225  FORMAT(' SET FINAL SIGN OF',I20,' WITH',I20)
```

```
  660 IVAL$=SIGN*IFN
      VAL=IVAL$
      IF(IPRT.GE.20)WRITE(6,217) VAL
      RETURN
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      IVAL$=-1
      STOP
6007  WRITE(6,6008)(A$(I),I=1,LA)
6008  FORMAT(' *** ATTEMPT TO FIND REAL VALUE OF ALPHABETIC STRING:'/
     2       ' ',110A1)
      STOP
      END
```

```
      INTEGER FUNCTION LBL$(A$,LA,M,M1)
C**********************************************************************
C**                                                                  **
C**   STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN LBL INSTRUCTION.   **
C**                                                                  **
C**   THE RETURN VALUE OF THE FUNCTION LBL$  IS SET AS FOLLOWS:      **
C**        0  =  CONTINUE TO COMPILE                                 **
C**       -1  =  AN ERROR IN COMPILING THE INSTRUCTION.              **
C**                                                                  **
C**                                                                  **
C**********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      INTEGER*2 A$(IDIM)
      INTEGER*2 SS1$(20)
      INTEGER*4 LSS1
C     INTEGER*2 BLNK/' '/
      INTEGER*2 QUOTE/'"'/
      INTEGER*2 LABEL(26)/'A','B','C','D','E','F','G','H','I','J',
     2                    ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
     3                    ' ',' ',' ',' ',' ',' '/
      INTEGER*4 FUNC$/'LBL'/
      INTEGER*4 M(1)
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' LBL$    :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     STRIP STRING OF "LBL" CHARACTERS.
C
         CALL LCUT$(A$,LA,3)
         IF(TRIM$(A$,LA)) 6015,6015,10
C
C
C
C     CHECK FOR ALPHANUMERIC VERSUS LOCAL LABELS
C
10       IF(A$(1).EQ.QUOTE) GOTO 80
C
C
C
C     PROCESS LOCAL LABELS, FIRST CHECK FOR NUMERIC OR SINGLE CHAR ALPHA
C
```

```
20        IF(LA-2) 25,50,6015
C
C
C
C     LENGTH OPERAND IMPLIES SINGLE CHARACTER ALPHA LOCAL LABEL
C
25        DO 30 I=1,26
               INDEX=I
               IF(A$(1).EQ.LABEL(I)) GOTC 35
30        CONTINUE
C
C         WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND
          GOTO 6015
C
C
C
C     HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPER-
C     AND.  THEN GOTO PROCESS A TWO BYTE INSTRUCTION.
C
35        INDEX=INDEX+101
C
C
C
C     PROCESS A TWO BYTE INSTRUCTION
C
40        IBYTE=2
          M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
210       FORMAT('  LBL$',I5,' LENGTH OF NEXT INSTR IS',I3)
          M1=M1+1
C
C
C
C     LOAD TWO BYTE LBL INSTR (EITHER SINGLE CHAR ALPHA OR 2 DIGIT NUM)
C
          M(M1)=207
          IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)
211       FORMAT('  LBL$',I5,' TWO BYTE LBL INSTR',T75,I3)
          M1=M1+1
          M(M1)=INDEX
          IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)
212       FORMAT('  LBL$',I5,' LBL 2D OPERAND     ',T75,I3)
          M1=M1+1
          LBL$=0
          RETURN
C
C
C
```

```
C       LENGTH OF OPERAND IMPLIES MUST BE A NUMERIC LOCAL LABEL
C
50      IF(IVAL$(A$,LA,INDEX))6015,55,55
C
C
C
C       HAVE FOUND VALID NUMERIC LOCAL LABEL, CHECK FOR ONE BYTE INSTR
C
55         IF(INDEX.GT.14) GOTO 40
C
C
C
C       PROCESS A ONE BYTE INSTRUCTION, FIRST LOAD THE LENGTH OF INSTR
C
           IBYTE=1
           M(M1)=IBYTE
           IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
           M1=M1+1
C
C
C
C       HAVE FOUND VALID NUMERIC LOCAL LABEL <15, LOAD "LBL" INSTRUCTION
C
           M(M1)=1+INDEX
           IF(IPRT.GE.10)WRITE(6,213)M1,M(M1)
213        FORMAT('  LBL$',I5,' ONE BYTE LBL INSTR',T75,I3)
           M1=M1+1
           LBL$=0
           RETURN
C
C
C
C       PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE
C
80         K=0
           P2=POS$(A$,LA,QUOTE,1,2)
           IF(P2) 6015,120,85
C
C
C
C       LCOK FOR ANOTHER QUOTE
C
C
85         PL=P2+1
           P4=POS$(A$,LA,QUOTE,1,PL)
           IF(P4) 6015,95,90
C
C
```

```
C
C      FCUND ANOTHER (THIRD OR MORE) QUOTE
C
90        P2=P4
          GOTO 85
C
C
C
C      CHECK AFTER LAST QUOTE FOR KEY ASSIGNMENT CODE
C
95        LEFT=LA-P2
          IF(LEFT) 6015,100,105
C
100       LA=LA-1
          GOTO 120
C
105       PSTART=P2+1
          CALL SEG$(A$,LA,SS1$,LSS1,PSTART,LEFT)
          K=IVAL$(SS1$,LSS1,NO)
          LA=LA-LEFT-1
C
C
C
C      SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL
C
120       LENGTH=LA-1
          IBYTE=4+LENGTH
          M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
          M1=M1+1
C
C
C
C      HAVE FOUND VALID ALPHANUMERIC  LABEL,     LOAD "LBL" INSTRUCTION
C
          M(M1)=192
          IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
214       FORMAT('  LBL$',I5,' ALPHA    LBL INSTR',T75,I3)
          M1=M1+1
C
C
C
C      PROVIDE ONE NULL INSTRUCTION  TO RESERVE SPACE FOR THE LINK
C      PCINTER.  ALL ALPHANUMERIC LABEL AND END INSTRUCTIONS CONTAIN
C      POINTERS WHICH LINK THEM ALTOGETHER INTO A LABEL CHAIN.  THIS
C      CHAIN IS USED TO IDENTIFY THE POSITION OF LABELS AND PROGRAM
C      BOUNDARIES WITHIN THE HP41CV MEMORY.  THE CHAIN OF LABELS IS
C      RECOMPILED BY THE WAND SOFTWARE, SO THE BYTES CONTAINING THE
```

```
C      CHAIN ARE SET TO ZERO BY THIS COMPILER.
C
          M(M1)=0
          IF(IPRT.GE.10)WRITE(6,215)M1,M(M1)
215       FORMAT('  LBL$',I5,' TRAILNG NULL INSTR',T75,I3)
          M1=M1+1
C
C
C
C      SET INDICATOR FOR NUMBER OF ALPHA CHARS IN LABEL
C
          M(M1)=241+LENGTH
          IF(IPRT.GE.10)WRITE(6,216)M1,M(M1)
216       FORMAT('  LBL$',I5,' LENGTH CODE ALPH LBL',T75,I3)
          M1=M1+1
C
C
C
C      ENCODE KEY ASSIGNMENT
C
          IF(K.NE.0) GOTO 130
C               SINCE K=0 IMPLIES NULL KEY ASIGNMENT
                M(M1)=0
                IF(IPRT.GE.10)WRITE(6,217)M1,M(M1)
217             FORMAT('  LBL$',I5,' NULL KEY ASSIGNMENT ',T75,I3)
                M1=M1+1
                GOTO 140
C
C               SINCE K¬=0 IMPLIES A KEY ASSIGNMENT TO BE MADE
130             K1=0
                IF(K.GT.0) GOTO 135
                K1=8
                K=IABS(K)
135             A1=K/10
                B1=MOD(K,10)
                K=16*(B1-1)+A1+K1
                M(M1)=K
                IF(IPRT.GE.10)WRITE(6,218)M1,M(M1)
218             FORMAT('  LBL$',I5,' LBL  KEY ASSIGNMENT ',T75,I3)
                M1=M1+1
C
C
C
C      ADD ALPHABETIC CHARACTERS AND RETURN
C
140       LA=LENGTH+1
          LBL$=ALPH$(A$,LA,M,M1)
          RETURN
```

```
C                                                                       LBL02410
C                                                                       LBL02420
C     ERROR HANDLING SECTION FOLLOWS                                    LBL02430
C                                                                       LBL02440
6000  WRITE(6,6001) FUNC$                                               LBL02450
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)                        LBL02460
      WRITE(6,6010) LA,LB,IDIM                                          LBL02470
6010  FORMAT(' LA=',I10,'     LB=',I10,'     IDIM=',I10)                LBL02480
      LBL$=-1                                                           LBL02490
      RETURN                                                            LBL02500
6015  WRITE(6,6016)                                                     LBL02510
6016  FCRMAT(' ***** INVALID SECOND OPERAND IN LBL INSTR ****')         LBL02520
      LBL$=-1                                                           LBL02530
      RETURN                                                            LBL02540
      END                                                               LBL02550
```

```
      INTEGER FUNCTION MEM$(A$,LA,M,M1,WHICH)
C*****************************************************************************
C**                                                                         **
C**   STRING A$ CONTAINS AN MEMORY INSTRUCTION, EITHER AN STO OR A          **
C**   RCL                                                                   **
C**                                                                         **
C**   THE RETURN VALUE OF THE FUNCTION MEM$ IS SET AS FOLLOWS:              **
C**        0  =  CONTINUE TO COMPILE                                        **
C**       -1  =  AN ERROR IN COMPILING THE INSTRUCTION.                     **
C**                                                                         **
C**                                                                         **
C*****************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM, IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      CCMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      INTEGER*2 A$(IDIM)
C     INTEGER*2 BLNK/' '/
      INTEGER*2 IND(3)/'I','N','D'/
      INTEGER*2 C$( 5)/'T','Z','Y','X','L'/
      INTEGER*4 LC/ 5/
      INTEGER*4 WHICH(5)
      INTEGER*4 FUNC$/'MEM'/
      INTEGER*4 M(1)
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' MEM$   :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     STRIP STRING OF "MEM" CHARACTERS.
C
         IF(LCUT$(A$,LA,3)) 6015,6015,7
7        IF(TRIM$(A$,LA)) 6015,6015,10
C
C
C
C     ESTABLISH MOST LIKELY INSTR LENGTH AND PREFIX
C
10       IBYTE=2
         PREFIX=WHICH(1)
         H=WHICH(2)
C
C
C     CHECK FOR INDIRECT ADDRESS
C
```

```
C
20          P6=POS$(A$,LA,IND,3,1)
            IF(P6) 6000,40,21
21                 INDIR=.TRUE.
                   IF(LCUT$(A$,LA,3)) 6000,6015,22
22                 IF(TRIM$(A$,LA)) 6015,6015,40
C
C
C
C       CHECK FOR NUMERIC SECOND OPERAND
C
40          IF(NUMC$(A$,LA,IANSW)) 6015,45,60
C
C
C
C       PROCESS NON-NUMERIC SECOND OPERAND
C
45                 IF(LA.GT.1) GOTO 6015
                   IZ=FIND$(A$(1),LA,C$,LC,LCC)
                   IF(IZ.NE.0)GOTO 46
                          WRITE(6,207)A$(1)
207                       FORMAT(' ***** INVALID CHARACTER     *****',5X,A1)
                          MEM$=-1
                          RETURN
46                 POSTFX=IZ+111
                   GOTO 100
C
C
C
C       PROCESS NUMERIC POSTFIX OPERANDS
C
60          POSTFX=IVAL$(A$,LA,VAL)
C
C
C
C       CHECK FOR ONE BYTE VERSUS TWO BYTE NUMERIC OPERANDS
C
            IF((POSTFX.GT.15).CR.(INDIR)) GOTO 75
C
C
C
C       PROCESS ONE BYTE NUMERIC SECOND OPRANDS
C
                   IBYTE=1
                   PREFIX=POSTFX+H
                   GOTO 100
C
C
```

```
C
C      PROCESS TWO BYTE NUMERIC OPERANDS
C
75            CONTINUE
C
C
C      SET THE LENGTH OF THE INSTRUCTION
C
100       M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215       FORMAT('  MEM$',I5,' LENGTH OF THIS INSTR IS',I3)
          M1=M1+1
C
C
C
C      ENCODE THE PREFIX OF THIS INSTRUCTION
C
          M(M1)=PREFIX
          IF(IPRT.GE.10)WRITE(6,211)M1,(WHICH(I),I=3,5),M(M1)
211       FORMAT('  MEM$',I5,' ',3A1,'      INSTR',T75,I3)
          M1=M1+1
C
C
C
C      ENCODE THE POSTFIX OF THIS INSTRUCTION
C
          IF(IBYTE.EQ.1) GOTO 125
          IF(INDIR)POSTFX=POSTFX+128
          M(M1)=POSTFX
          IF(IPRT.GE.10)WRITE(6,221)M1,(WHICH(I),I=3,5),M(M1)
221       FORMAT('  MEM$',I5,' ',3A1,' INSTR POSTFIX',T75,I3)
          M1=M1+1
C
C
C

C
C
C
125    MEM$=0
       RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000   WRITE(6,6001) FUNC$
6001   FORMAT(' *** STRING LENGTH ERROR *** ',A4)
       WRITE(6,6010) LA,LB,IDIM
6010   FORMAT(' LA=',I10,'     LB=',I10,'      IDIM=',I10)
```

```
      MEM$=-1                                                           MEM01450
      RETURN                                                            MEM01460
 6015 WRITE(6,6016)                                                     MEM01470
 6016 FCRMAT(' ***** INVALID SECOND OPERAND IN MEM INSTR ****')          MEM01480
      MEM$=-1                                                           MEM01490
      RETURN                                                            MEM01500
      END                                                               MEM01510
```

```
      INTEGER FUNCTION NEWPG$(LINCNT,NUMPGE,TITLE$,LTITLE,MTITLE)
C*************************************************************************
C**                                                                     **
C**     THIS SUBROUTINE IS USED TO PLACE MAKE THE OUTPUT LISTING        **
C**     AS ATTRACTIVE AS POSSIBLE.  IT PERFORMS ALL THE HOUSEKEEPING    **
C**     FUNCTIONS ASSOCIATED WITH GOING TO A NEW PAGE.                  **
C**                                                                     **
C**                                                                     **
C**     IT SHOULD BE CALLED BY THE USE OF A STATEMENT SUCH AS           **
C**                                                                     **
C**          IF(MOD(LINCNT,PAGE).EQ.0)                                  **
C**        1      CALL NEWPG$(LINCNT,PAGENUMBER,TITLE$,LTITLE,MTITLE)   **
C**                                                                     **
C**                                                                     **
C*************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 TITLE$(LTITLE,MTITLE)
C     INTEGER*2 BLNK/' '/
C     INTEGER*2 CARD(80)
C     INTEGER*4 FUNC$/'NEWPG'/
297   FORMAT(//,' ')
299   FCRMAT('1',////)
200   FORMAT('          ',116A1)
201   FCRMAT(' ',132A1)
C
C
C
C     PRINT THE OUTPUT PAGE HEADING
C
      WRITE(6,299)
      IF(MTITLE.LE.0) GOTO 75
      IF(LTITLE.GT.116)LTITLE=116
      WRITE(6,200)(TITLE$(JJ,1),JJ=1,LTITLE)
      IF(MTITLE.EQ.1) GOTO 75
      DC 50 II = 2,MTITLE
         WRITE(6,201)(TITLE$(JJ,II),JJ=1,LTITLE)
50    CONTINUE
C
C
C
C     UPDATE THE PAGE COUNTER AND RESET THE LINE COUNTER
C
75    NUMPGE=NUMPGE+1
      LINCNT=MTITLE
      WRITE(6,297)
C
C
```

```
C      EXIT
C
       NEWPG$=NUMPGE
       RETURN
C
C
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000   WRITE(6,6001) FUNC$
6001   FORMAT(' *** PAGE OUTPUT    ERROR *** ',A4)
       NEWPG$=-1
       RETURN
       END
```

```
      INTEGER FUNCTION NUMC$(A$,LA,IANSW)
C*************************************************************************
C**                                                                     **
C**   THIS FUNCTION CHECKS TO SEE IF A STRING IS ALL NUMERIC.           **
C**                                                                     **
C**   NUMERIC IS DEFINED TO MEAN ALL DIGITS,"+",'-','.','E',OR BLNK     **
C**                                                                     **
C**   THE RETURNED VALUE OF THE INSTRUCTION IS 0 IF NOT NUMERIC         **
C**                                            1 IF NUMERIC             **
C**                                           -1 IF ERROR ENCOUNTERED   **
C**                                                                     **
C**                                                                     **
C**                                                                     **
C*************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM)
      INTEGER*4 FUNC$/'NUMC'/
      INTEGER*2 DIGIT(15)/'0','1','2','3','4','5','6','7','8','9',
     2                    ' ','.','-','+','E'/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' NUMC$   :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
      DC 10 I=1,LA
         IF(LA.NE.1) GOTO 3
C                NOT ALLOWED TO EXAMINE LAST 5 CHARS IF LA IS ONE.
                 LOOK=10
                 GOTC 4
3        LOOK=15
4        DO 5 J=1,LOOK
                 IF(A$(I).EQ.DIGIT(J)) GOTO 10
5        CONTINUE
C
C                HAVE FOUND A NON-NUMERIC CHARACTER
7                IANSW=0
                 NUMC$=0
                 IF(IPRT.GE.20) WRITE(6,201)
201              FORMAT(' STRING DETERMINED TO BE NON-NUMERIC')
                 RETURN
C
C       CHARACTER FOUND TO BE NUMERIC,TAKE NEXT CHARACTER
10      CONTINUE
C
C
C     IF YOU GET TO HERE THE STRING MUST BE ALL NUMERIC.
```

```
      IANSW=1
      NUMC$=1
      IF(IPRT.GE.20) WRITE(6,202)
202   FORMAT(' STRING FOUND TO BE NUMERIC')
      RETURN
C
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA,LB,IDIM
6010  FORMAT(' LA=',I10,'      LB=',I10,'      IDIM=',I10)
      NUMC$=-1
      RETURN
      END
```

```
      INTEGER FUNCTION PARS$(A$,LA,B$,LB)
C*************************************************************************
C**                                                                     **
C**     STRING A$ IS SEARCHED FOR THE OCCURANCE OF THE 1ST NON-LEADING **
C**     BLANK.  THEN A$ IS SPLIT INTO TWO SUBSTRINGS, THE LEADING       **
C**     TOKEN (FIRST WORD) IS PLACED IN B$ AND THE REMAINDER IS PLACED **
C**     IN A$.                                                          **
C**                                                                     **
C**     THE NUMBER OF NON-BLANK CHARACTERS REMAINING IN A$ AFTER THE    **
C**     REMOVAL OF THE FIRST WORD IS RETURNED AS THE FUNCTION VALUE     **
C**     POS$.                                                           **
C**                                                                     **
C**     AN ATTEMPT TO PARSE A NULL STRING WILL RESULT IN A SPECIAL      **
C**     CHARACTER BEING PLACED IN THE 1ST POSITION OF A$.  A SUBSEQUENT**
C**     ATTEMPT TO REPARSE THIS STRING WILL RESULT IN A FATAL ERROR.   **
C**     THIS FEATURE IS INTENDED TO PREVENT UNCONTROLLED LOCPING OF THE**
C**     PARSE FUNCTION ON A NULL STRING.  THE SPECIAL CHARACTER USED    **
C**     IS KNOWN ONLY TO THIS ROUTINE AND MAY BE USED AS A REGULAR      **
C**     CHARACTER FOR IN ANY STRING OF LENGTH GREATER THAN ZERO.       **
C**                                                                     **
C*************************************************************************
      IMPLICIT INTEGER(A-Z)
      CCMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM),B$(IDIM)
      INTEGER*2 BLNK/' '/
      INTEGER*2 HALT/' '/
      INTEGER*4 FUNC$/'PARS'/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' PARS$    :  ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     CHECK TO SEE IF INPUT STRING IS NULL
C
      IF(LA.NE.0) GOTO 5
         IF(IPRT.GE.20) WRITE(6,204)
204      FORMAT(' ATTEMPTED TO PARSE A NULL STRING.')
         IF(A$(1).EQ.HALT) GOTO 6005
         A$(1)=HALT
         PARS$=0
         LB=0
         RETURN
C
C
C
C     TRIM THE INPUT STRING OF LEADING BLANKS
C
```

```
5       IM=0
        DO 10 I=1,LA
           IF(A$(I).NE.BLNK) GOTO 15
           IM=IM+1
10      CONTINUE
15      IF(IM.EQ.0) GOTO 25
           IF(IPRT.GE.20) WRITE(6,201) IM
201        FORMAT(' FOUND',I3,' LEADING BLANKS IN INPUT STRING')
           IF(IM.LT.LA) GOTO 20
                LA=0
                LB=0
                PARS$=0
                IF(IPRT.GE.20) WRITE(6,202)
202             FORMAT(' FOUND INPUT STRING IS ALL BLANKS')
                RETURN
20         LEFT=LA-IM
           DO 23 I=1,LEFT
                A$(I)=A$(I+IM)
23         CONTINUE
           LA=LEFT
C
C
C
C       LOCATE THE FIRST NON-LEADING BLANK IN A$  (THEREBY DETERMINE LB)
C
25      LB=0
        DO 30 I=1,LA
           IF(A$(I).EQ.BLNK) GOTO 35
           LB=LB+1
30      CONTINUE
C
C
C       CONSTRUCT TOKEN
C
35      DO 45 I=1,LB
           B$(I)=A$(I)
45      CONTINUE
        IF(IPRT.GE.20) WRITE(6,205) LB,(B$(I),I=1,LB)
205     FORMAT(' PARS$ FOUND TOKEN',I3,' "',60A1)
C
C
C       REMOVE TOKEN FROM FRONT OF INPUT STRING
C
        LEFT=LA-LB-1
        IF(LEFT.GT.0) GOTO 50
           LA=0
           GOTO 75
50      DO 55 I=1,LEFT
```

```
      A$(I)=A$(I+LB+1)
55    CONTINUE
      LA=LEFT
C
C
C     CHECK $A FOR TRAILING BLANKS
C
      IF(LA.EQ.0) GOTO 75
58    IM=0
      DO 60 I=1,LA
         INDEX=LA-I+1
         IF(A$(INDEX).NE.BLNK) GOTO 65
         IM=IM+1
60    CONTINUE
65    IF(IM.EQ.0) GOTO 75
         IF(IPRT.GE.20) WRITE(6,207) IM
207      FORMAT(' FOUND',I3,' TRAILING BLANKS IN INPUT STRING')
         IF(IM.LT.LA) GOTO 70
              LA=0
              PARS$=0
              IF(IPRT.GE.20) WRITE(6,208)
208           FORMAT(' FOUND REMAINING STRING IS ALL BLANKS')
              RETURN
70       LA=LA-IM
75    PARS$=LA
      IF(IPRT.GE.20.AND.LA.EQ.0) WRITE(6,209)
209   FCRMAT(' REMAINING STRING AFTER PARSE FUNCTION IS NULL')
      RETURN
C
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      PARS$=-1
      RETURN
6005  WRITE(6,6006)
6006  FORMAT(' *** FATAL ERROR: ATTEMPTED TO PARSE A NULL STRING TWICE')
      STOP
      END
```

```
      INTEGER FUNCTION POS$(A$,LA,B$,LB,LSTART)
C*********************************************************************
C**                                                                 **
C**     THE STRING A$ IS SEARCHED FOR THE OCCURANCE OF THE SUBSTRING **
C**     B$, AND THE POSITION OF THE 1ST CHARACTER OF A$ WHERE B$ IS  **
C**     A MATCH IS ASSIGNED TO POS$.  THE SEARCH BEGINS AT LSTART    **
C**     IN A$.  A ZERO IS RETURNED IF NO MATCH IS FOUND.  A NEGATIVE **
C**     NUMBER IS RETURNED UPON AN ERROR.                            **
C*********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM),B$(IDIM)
      INTEGER*4 FUNC$/'POS'/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' POS$    :    ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
      IF(LB.GT.IDIM.OR.LB.LT.0) GOTO 6000
C
C
C
      IF(LB.LE.(LA-LSTART+1))GOTO 10
         POS$=0
         IF(IPRT.GE.20) WRITE(6,201)
201      FORMAT(' FIRST STRING TOO SHORT TO CONTAIN SECOND STRING')
         RETURN
C
C
10    LEFT=LA-LB+1
      DO 25 I=LSTART,LEFT
         DO 20 J=1,LB
                JN=I+J-1
                IF(IPRT.GE.20)WRITE(6,202)JN,A$(JN),J,B$(J)
202             FORMAT(' COMPARE A$(',I3,')=',A1,' WITH B$(',I3,')=',A1)
                IF(A$(JN).NE.B$(J))GOTO 25
20       CONTINUE
      POS$=I
      IF(IPRT.GE.20)WRITE(6,203) I
203   FORMAT(' AT POSITION',I3,' SECOND STRING FOUND IN FIRST')
      RETURN
C
C
C
25    CONTINUE
C
C     ROUTINE WILL FALL THROUGH TO FOLLOWING IF NO MATCH FOUND.
C
      POS$=0
      IF(IPRT.GE.20)WRITE(6,204)
```

```
204    FORMAT(' FIRST STRING SEARCHED AND SECOND STRING NOT FOUND')
       RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000   WRITE(6,6001) FUNC$
6001   FORMAT(' *** STRING LENGTH ERROR *** ',A4)
       POS$=-1
       RETURN
       END
```

```
      INTEGER FUNCTION RCUT$(A$,LA,NUM)
C*************************************************************************
C**                                                                     **
C**   STRING A$ HAS NUM CHARACTERS REMOVED FROM THE RIGHT.              **
C**                                                                     **
C**   THE VALUE OF THE FUNCTION RCUT$ IS SET TO                         **
C**          LA IF LA IS GREATER THAN 0                                 **
C**           0 IF THE NULL STRING IS LEFT AFTER THE REMOVAL            **
C**          -1 IF AN ERROR IS ENCOUNTERED.                             **
C**                                                                     **
C**                                                                     **
C**                                                                     **
C*************************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM)
      INTEGER*4 FUNC$/'RCUT'/
      INTEGER*4 NUM
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' RCUT$    :    ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
      LEFT=LA-NUM
      IF((LEFT).GT.0) GOTO 20
         IF(IPRT.GE.10)WRITE(6,202)
202      FORMAT(' STRING REDUCED TO NULL STRING BY RCUT$')
         LA=0
         RCUT$=0
         RETURN
C
C
C
20    CONTINUE
      LA=LEFT
      IF(IPRT.GE.20)WRITE(6,201)LA,(A$(I),I=1,LA)
201   FORMAT(' STRING NOW',I4,' "',110A1)
      RCUT$=LA
      RETURN
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      RCUT$=-1
      RETURN
      END
```

```
      INTEGER FUNCTION SEG$(A$,LA,B$,LB,LSTART,LOUT)
C***********************************************************************
C**                                                                   **
C**      A SUBSTRING IS EXTRACTED FROM A$ AND ASSIGNED TO B$.  THE    **
C**      SUBSTRING IN A$ BEGINS AT POSITION LSTART AND IS LOUT CHAR-  **
C**      ACTERS LONG.                                                 **
C**                                                                   **
C**      IF EITHER A$ IS THE NULL STRING                              **
C**             OR A$ DOES NOT HAVE ENOUGH CHARACTERS                 **
C**           THEN B$ IS RIGHT PADDED WITHBLANKS SO IT HAS LOUT CHAR  **
C**                                                                   **
C**      IF LOUT=0, THEN B$ BECOMES THE NULL STRING                   **
C**                                                                   **
C**      THE ACTUAL NUMBER OF NON-BLANK CHARACTERS OBTAINED FROM A$   **
C**      IS RETURNED AS THE VALUE OF THE FUNCTION SEG$.               **
C**                                                                   **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM),B$(IDIM)
      INTEGER*2 BLNK/' '/
      INTEGER*4 FUNC$/'SEG'/
      LOGICAL FIRST
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' SEG$    :    ',110A1)
      FIRST=.TRUE.
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
      IF(LSTART.GT.IDIM.OR.LSTART.LT.0) GOTO 6000
      IF(LOUT.GT.IDIM.OR.LOUT.LT.0) GOTO 6000
C
C
C
      IF(LOUT.NE.0) GOTO 5
         LB=LOUT
         IF(IPRT.GE.20) WRITE(6,202)
202      FORMAT(' NULL STRING ASSIGNED BY SEG$')
         SEG$=0
         RETURN
C
C
C
5     IF(LSTART.LE.LA) GOTO 15
         DO 10 I=1,LOUT
              B$(I)=BLNK
10       CONTINUE
         LB=LOUT
         IF(IPRT.GE.20) WRITE(6,201) LSTART,LOUT
201      FORMAT(' START POINT (',I3,') GREATER THAN LENGTH FIRST STRING'
```

```
    2          /,'     NEW STRING HAS',I3,' BLANKS')
          SEG$=0
          RETURN
C
C
C
15    DO 35 I=1,LOUT
          IM=I
          IF(I.GT.(LA-LSTART+1))GOTO 25
               B$(I)=A$(LSTART+I-1)
               GOTO 35
25        B$(I)=BLNK
          IF(.NOT.FIRST) GOTO 35
               FIRST=.FALSE.
               SEG$=IM-1
35    CONTINUE
      IF(FIRST) SEG$=IM
      IF(IPRT.GE.20) WRITE(6,203) SEG$
203   FORMAT(' SEG$ OBTAINED ',I3,' CHARACTERS FROM FIRST STRING')
      LB=LOUT
      RETURN
C
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      SEG$=-1
      RETURN
      END
```

```
      INTEGER FUNCTION TRIM$(A$,LA)
C*********************************************************************
C**                                                                 **
C**   THIS FUNCTION STRIPS A STRING OF LEADING AND TRAILING BLANKS. **
C**                                                                 **
C**   THE RETURN VALUE OF THE FUNCTION IS SET TO THE LENGTH OF THE  **
C**   STRING REMAINING AFTER THE BLANKS ARE REMOVED.                **
C**                                                                 **
C*********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM)
      INTEGER*2 BLNK/' '/
      INTEGER*4 FUNC$/'TRIM'/
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' TRIM$    :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C
C     CHECK A$ FOR TRAILING BLANKS
C
58    IM=0
      DC 60 I=1,LA
         INDEX=LA-I+1
         IF(A$(INDEX).NE.BLNK) GOTO 65
         IM=IM+1
60    CONTINUE
65    IF(IM.EQ.0) GOTO 70
         IF(IPRT.GE.20) WRITE(6,207) IM
207      FORMAT(' FOUND',I3,' TRAILING BLANKS IN  STRING')
         IF(IM.NE.LA) GOTO 70
              LA=0
              TRIM$=0
              IF(IPRT.GE.20) WRITE(6,208)
208           FORMAT(' FOUND STRING IS ALL BLANKS')
              RETURN
70       IEND=LA-IM
C
C
C
C     CHECK A$ FOR LEADING BLANKS
C
C
      IM=0
      DO 10 I=1,IEND
         IF(A$(I).NE.BLNK) GOTO 15
```

```
          IM=IM+1
10     CONTINUE
15     IF(IM.EQ.0) GOTO 25
          IF(IPRT.GE.20) WRITE(6,211) IM
211       FORMAT(' FOUND',I3,' LEADING BLANKS IN STRING')
25        IBEG=1+IM
C
C
C
C      DETERMINE LENGTH OF INPUT STRING
C
       LA=IEND-IBEG+1
       IF(LA.LE.IDIM) GOTO 30
            LOSS=LA-IDIM
            IF(IPRT.GE.10) WRITE(6,216) LOSS
216         FORMAT(' STRING TOO LONG FOR MAX STRING LENGTH.  LOST',I3)
            LA=IDIM
            IEND=IEND-LOSS
C
C
C
C      TRANSFER THE A$ CHARACTERS TO THE INPUT STRING.
C
30     INDEX=1
       DO 85 I=IBEG,IEND
          A$(INDEX)=A$(I)
          IF(IPRT.GE.30) WRITE(6,209) I,A$(I),INDEX,A$(INDEX)
209       FORMAT(' MOVE A$(',I3,')=',A1,'  IS NOW A$(',I3,')=',A1)
          INDEX=INDEX+1
85     CONTINUE
C
C
C
C      CHECK FOR STRING ERROR  AND RETURN
C
       IF(IPRT.GE.10) WRITE(6,222)LA,(A$(I),I=1,LA)
222    FORMAT(' TRIM$ ',I3,' AFTER :  ',11)A1)
       IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
       TRIM$=LA
       RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000   WRITE(6,6001) FUNC$
6001   FORMAT(' *** STRING LENGTH ERROR *** ',A4)
       WRITE(6,6010) LA,LB,IDIM
```

```
6010  FCRMAT(' LA=',I10,'       LB=',I10,'       IDIM=',I10)              TRI00970
      TRIM$=-1                                                           TRI00980
      RETURN                                                             TRI00990
      END                                                                TRI01000
```

```
      INTEGER FUNCTION XEQ$(A$,LA,M,M1)
C***********************************************************************
C**                                                                   **
C**   STRING A$  HAS BEEN IDENTIFIED TO CONTAIN AN XEQ INSTRUCTION.   **
C**                                                                   **
C**                                                                   **
C**                                                                   **
C**   THE RETURN VALUE OF THE FUNCTION XEQ$ IS SET AS FOLLOWS:        **
C**        0  =  CONTINUE TO COMPILE                                  **
C**       -1  =  AN ERROR IN COMPILING THE INSTRUCTION.               **
C**                                                                   **
C**                                                                   **
C***********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      COMMON/FLAGS/DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      COMMON/CNTR/P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      LOGICAL DONE,PDIGIT,PALPHA,DIGIT,ALPHA,INDIR,FLAG1,FLAG2
      INTEGER*4 P1,P2,P3,P4,P5,P6,P7,P8,P9,S1,S2
      INTEGER*2 A$(IDIM)
C     INTEGER*2 BLNK/' '/
      INTEGER*2 QUOTE/'"'/
      INTEGER*2 IND(3)/'I','N','D'/
      INTEGER*2 LABEL(26)/'A','B','C','D','E','F','G','H','I','J',
     2                    'T','Z','Y','X','L',' ',' ',' ',' ',' ',
     3                    ' ',' ',' ',' ',' ',' '/
      INTEGER*4 FUNC$/'XEQ'/
      INTEGER*4 M(1)
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' XEQ$   :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
C     ESTABLISH DEFAULT PREFIX AND INSTRUCTION LENGTH VALUES
C     (THESE ARE THE VALUES FOR 3 BYTE LOCAL NUMERIC XEQ  WITHOUT IND)
C
         IBYTE=3
         PREFIX=224
C
C
C
C     STRIP STRING OF "XEQ" CHARACTERS.
C
         CALL LCUT$(A$,LA,3)
         IF(TRIM$(A$,LA)) 6015,6015,10
C
C
```

```
C
C     CHECK FOR ALPHANUMERIC VERSUS LOCAL LABELS
C
10       IF(A$(1).EQ.QUOTE) GOTO 80
C
C
C
C     PROCESS LOCAL LABELS, FIRST CHECK FOR INDIRECT XEQ INSTR
C
         P6=POS$(A$,LA,IND,3,1)
         IF(P6) 6015,20,16
C
C
C
C     PROCESS XEQ INDIRECT INSTRUCTION.
C
16       P1=P6+3
         CALL LCUT$(A$,LA,P1)
         IF(IPRT.GE.20)WRITE(6,235)
235      FORMAT(' DETECTED INDIRECT  XEQ INSTRUCTION')
         INDIR=.TRUE.
         IBYTE=2
         PREFIX=174
C
C
C
C     CHECK FOR NUMERIC OPERAND
C
20       IF(NUMC$(A$,LA,IANSW)) 6015,25,50
C
C
C
C
C
C     OPERAND MUST BE REGISTER X,Y,Z,T CR L  OR A LOCAL ALPHA LABEL
C
25       DO 30 I=1,26
              INDEX=I
              IF(A$(1).EQ.LABEL(I)) GOTO 35
30       CONTINUE
C
C        WILL FALL THROUGH TO THIS CODE IF NO VALID LABEL FOUND
         GOTO 6015
C
C
C
C     HAVE FOUND A MATCH IN LOCAL LABEL TABLE, SET VALUE OF SECOND OPER-
C     AND.  THEN GOTO PROCESS A THREE BYTE INSTRUCTION.
```

```
C                                                                             XEQ00970
35        INDEX=INDEX+101                                                     XEQ00980
          GOTO 75                                                             XEQ00990
C                                                                             XEQ01000
C                                                                             XEQ01010
C                                                                             XEQ01020
C                                                                             XEQ01030
C                                                                             XEQ01040
C     OPERAND  MUST BE A NUMERIC LOCAL LABEL                                  XEQ01050
C                                                                             XEQ01060
50        IF(IVAL$(A$,LA,INDEX))6015,75,75                                    XEQ01070
C                                                                             XEQ01080
C                                                                             XEQ01090
C                                                                             XEQ01100
C      PROCESS   THE GOTO   INSTRUCTION  (OPERAND>14)                         XEQ01110
C                                                                             XEQ01120
75        M(M1)=IBYTE                                                         XEQ01130
          IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE                                  XEQ01140
210       FORMAT('  XEQ$',I5,' LENGTH OF NEXT INSTR IS',I3)                   XEQ01150
          M1=M1+1                                                             XEQ01160
C                                                                             XEQ01170
C                                                                             XEQ01180
C                                                                             XEQ01190
C     LOAD    THE     XEQ INSTR  PREFIX                                       XEQ01200
C                                                                             XEQ01210
          M(M1)=PREFIX                                                        XEQ01220
          IF(IPRT.GE.10)WRITE(6,211)M1,M(M1)                                  XEQ01230
211       FORMAT('  XEQ$',I5,' XEQ PREFIX     INSTR',T75,I3)                  XEQ01240
          M1=M1+1                                                             XEQ01250
C                                                                             XEQ01260
C                                                                             XEQ01270
C                                                                             XEQ01280
C     LOAD THREE BYTE GOTO INSTR NULL INSTR (POSITION HOLDER FOR POINTERXEQ01290
C                                                                             XEQ01300
          IF(INDIR) GOTO 95                                                   XEQ01310
          M(M1)=0                                                             XEQ01320
          IF(IPRT.GE.10)WRITE(6,221)M1,M(M1)                                  XEQ01330
221       FORMAT('  XEQ$',I5,' NULL  FOR  XEQ INSTR',T75,I3)                  XEQ01340
          M1=M1+1                                                             XEQ01350
C                                                                             XEQ01360
C                                                                             XEQ01370
C                                                                             XEQ01380
C     LOAD THE 2D OPERAND OF THE        XEQ INSTR                             XEQ01390
C                                                                             XEQ01400
95        IF(INDIR) INDEX=INDEX+128                                           XEQ01410
C         NOTE THAT FOR XEQ IND THE HIGH ORDER BIT IS SET                     XEQ01420
          M(M1)=INDEX                                                         XEQ01430
          IF(IPRT.GE.10)WRITE(6,212)M1,M(M1)                                  XEQ01440
```

```
212       FORMAT('  XEQ$',I5,' XEQ 2D OPERAND     ',T75,I3)
          M1=M1+1
          XEQ$=0
          RETURN
C
C
C***********************************************************************
C***********************************************************************
C
C
C     PROCESS ALPHANUMERIC LABEL, FIRST LOOK FOR SECOND QUOTE
C
80        K=0
          P2=POS$(A$,LA,QUOTE,1,2)
          IF(P2) 6015,120,85
C
C
C
C     CHECK AFTER LAST QUOTE FOR BOGUS CHARACTERS
C
85        LEFT=LA-P2
          IF(LEFT) 6015,100,6015
C
C
C     DELETE THE ENDING QUOTE BY TRUNCATING THE STRING ONE CHAR
C
100       LA=LA-1
C
C
C
C     SET INSTRUCTION LENGTH FOR ALPHABETIC GLOBAL LABEL
C  (LINE 120 ACCOUNTS FOR BEGINNING QUOTE STILL ON STRING)
C
120       LENGTH=LA-1
C         FOR GTO H=2     FOR LBL H=4         FOR XEQ H=2
          H=2
          IBYTE=H+LENGTH
          M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,210)M1,IBYTE
          M1=M1+1
C
C
C
C     HAVE FOUND VALID ALPHANUMERIC  LABEL,     LCAD "XEQ" INSTRUCTION
C
          PREFIX=30
C         FOR GTO PREFIX=29    FOR LBL PREFIX=192    FOR XEQ PREFIX=30
          M(M1)=PREFIX
```

```
      IF(IPRT.GE.10)WRITE(6,214)M1,M(M1)
214   FORMAT('  XEQ$',I5,' ALPHA     XEQ INSTR',T75,I3)
      M1=M1+1
C
C
C
C
C
C     SET INDICATOR FOR NUMBER OF ALPHA CHARS IN LABEL
C
      U=240
C     FOR GTO U=240      FOR LBL    U=241        FOR XEQ U=240
      M(M1)=U+LENGTH
      IF(IPRT.GE.10)WRITE(6,216)M1,M(M1)
216   FORMAT('  XEQ$',I5,' LENGTH CODE ALPH XEQ',T75,I3)
      M1=M1+1
C
C
C
C
C
C     ADD ALPHABETIC CHARACTERS AND RETURN
C
140   LA=LENGTH+1
      XEQ$=ALPH$(A$,LA,M,M1)
      RETURN
C
C
C     ERROR HANDLING SECTION FOLLOWS
C
6000  WRITE(6,6001) FUNC$
6001  FORMAT(' *** STRING LENGTH ERROR *** ',A4)
      WRITE(6,6010) LA,LB,IDIM
6010  FORMAT(' LA=',I10,'     LB=',I10,'     IDIM=',I10)
      XEQ$=-1
      RETURN
6015  WRITE(6,6016)
6016  FORMAT(' ***** INVALID SECOND OPERAND IN XEQ INSTR ****')
      XEQ$=-1
      RETURN
6020  WRITE(6,6021)
6021  FORMAT(' ***** FOUND THREE OPERANDS, EXPECTING IND *****')
      XEQ$=-1
      RETURN
      END
```

```
      INTEGER FUNCTION XRO$(A$,LA,M,M1)
C**********************************************************************
C**                                                                  **
C**   STRING A$ HAS BEEN IDENTIFIED TO CONTAIN AN XROM INSTRUCTION.  **
C**                                                                  **
C**   THE XROM INSTRUCTIONS ARE SUBROUTINE CALLS TO HP SUPPLIED      **
C**   ROM ENCODED SUBROUTINES.  THE SECOND AND THIRD OPERANDS MUST   **
C**   BE NUMERIC.                                                    **
C**                                                                  **
C**                                                                  **
C**                                                                  **
C**                                                                  **
C**   THE RETURN VALUE OF THE FUNCTION XRO$ IS SET AS FOLLOWS:       **
C**         0  =  CONTINUE TO COMPILE                                **
C**        -1  =  AN ERROR IN COMPILING THE INSTRUCTION.             **
C**                                                                  **
C**                                                                  **
C**********************************************************************
      IMPLICIT INTEGER(A-Z)
      COMMON/TEXT/IDIM,IPRT
      INTEGER*2 A$(IDIM)
      INTEGER*2 COMMA/','/,SS1$(40)
      INTEGER*4 FUNC$/'XRO'/
      INTEGER*4 M(1)
      REAL*4 RFRAC,RDIF
      IF(IPRT.GE.10) WRITE(6,200)LA,(A$(I),I=1,LA)
200   FORMAT(' TRACE ',I3,' XRO$    :   ',110A1)
      IF(LA.GT.IDIM.OR.LA.LT.0) GOTO 6000
C
C
C
         IF(LCUT$(A$,LA,4)) 6015,6020,7
7        IF(TRIM$(A$,LA))    6015,6020,10
10       P1=POS$(A$,LA,COMMA,1,1)
         P2=P1-1
         IF(SEG$(A$,LA,SS1$,LSS1,1,P2))6015,6020,12
12       IF(LCUT$(A$,LA,P1)) 6015,6020,15
15       IF(IVAL$(SS1$,LSS1,ROM)) 6030,20,20
20       IF(IVAL$(A$,LA,PGM)) 6030,25,25
25       IFRAC=ROM/4
         IFIRST=160+IFRAC
         RFRAC=FLOAT(ROM)/4.0
         RDIF=RFRAC-FLOAT(IFRAC)
         RDIF=256.0*RDIF
         IDIF=INT(RDIF)
         ISECND=PGM+IDIF
C
C
```

```
C      SET THE LENGTH OF THE INSTRUCTION
C
100       IBYTE=2
          M(M1)=IBYTE
          IF(IPRT.GE.20)WRITE(6,215)M1,IBYTE
215       FORMAT('  XRO$',I5,' LENGTH OF THIS INSTR IS',I3)
          M1=M1+1
C
C
C
C      ENCODE THE PREFIX OF THIS INSTRUCTION
C
          M(M1)=IFIRST
          IF(IPRT.GE.10)WRITE(6,211)M1,IFIRST,ROM,M(M1)
211       FORMAT('  XRO$',I5,' PREFIX= ',I3,' ROM= ',I3,T75,I3)
          M1=M1+1
C
C
C
C      ENCODE THE POSTFIX OF THIS INSTRUCTION
C
          M(M1)=ISECND
          IF(IPRT.GE.10)WRITE(6,221)M1,ISECND,PGM,M(M1)
221       FORMAT('  XRO$',I5,' POSTFIX=',I3,' PGM= ',I3,T75,I3)
          M1=M1+1
C
C
C
C
C
C
125    XRO$=0
       RETURN
C
C      ERROR HANDLING SECTION FOLLOWS
C
6000   WRITE(6,6001) FUNC$
6001   FORMAT(' *** STRING LENGTH ERROR *** ',A4)
       WRITE(6,6010) LA,IDIM
6010   FORMAT(' LA=',I10,'                         IDIM=',I10)
       XRO$=-1
       RETURN
6015   WRITE(6,6016)
6016   FCRMAT(' ***** INVALID ROM NUMBER IN XRO INSTR ****')
       XRO$=-1
       RETURN
6020   WRITE(6,6021)
```

```
6021  FORMAT(' ***** INVALID PROGRAM NUMBER IN XRO INSTR *****')         XR000970
      XRO$=-1                                                            XR000980
      RETURN                                                             XR000990
6030  WRITE(6,6031)                                                      XR001000
6031  FORMAT(' ****** NUMERIC CONVERSION ERROR IN XRO INSTR ****')       XR001010
      XRO$=-1                                                            XR001020
      RETURN                                                             XR001030
      END                                                                XR001040
```

```
C***********************************************************************
C**                                                                   **
C**   THIS IS THE PROGRAM WHICH INTERFACES THE HP41CV CROSS COMPILER **
C**   TO THE VERSATEC HIGH RESOLUTION PLOTTER.  IF THE CROSS          **
C**   COMPILER IS USED ON A DIFFERENT PLOTTER, ONLY THIS ROUTINE      **
C**   NEEDS TO BE CHANGED.                                            **
C**                                                                   **
C***********************************************************************
      REAL*4  HEIGHT,UNIT,DOUBLE,CSIZE,CSPACE,TSIZE,TSPACE
      REAL*4  PLONG,PWIDE,HFACTR,ZERO,TMAR,SMAR
      INTEGER*2 IN(133),BLNK/' '/,ZER/'0'/,ONE/'1'/,TITLE(80)
      INTEGER*4 PERPGE,IROW
C
C
C
C     INITIALIZE VARIABLES AND POSITION PEN AT ORIGIN
C
C      ONLY TWO VARIABLES (NIBS AND HEIGHT) SHOULD BE CHANGED BY
C      THE USER.  OTHER VARIABLES ARE COMPUTED BASED ON THESE
C      VARIABLES.
C
C      NIBS IS THE WIDTH OF THE PLOTTER LINE  (INTEGER 1 TO 4)
C      SETTING NIBS AFFECTS THE WIDTH OF THE BAR CODE ROW.  A
C      SETTING OF 4 WILL GIVE BARS OF 5 TO 6.5 INCHES IN WIDTH.
C
            NIBS=4
C
C
C
C      HEIGHT IS THE BASIC HEIGHT OF THE BAR CODE ROW.  BARS MAY BE
C      MADE IN ALMOST ANY HEIGHT FROM 0.2 TO 0.5 INCHES.  RECOMMEND
C      A HEIGHT OF 0.40 INCHES, WHICH WILL GIVE 12 BARS PER PAGE.
C
C      HEWLETT-PACKARDS BARS ARE .33 INCHES HIGH
C
            HEIGHT=0.40
C
C
C      UNIT IS THE BASIC UNIT WIDTH -- IT IS THE WIDTH OF THE
C             SPACE BETWEEN BARS AND THE WIDTH OF A ZERO BAR.
            UNIT=0.005*FLOAT(NIBS)
C      COUBLE IS TWICE THE WIDTH OF A UNIT BAR -- --IT IS THE WIDTH
C             OF A ONE BAR.
            DOUBLE=0.010*FLOAT(NIBS)
C      CSIZE IS THE HEIGHT OF THE BAR CODE ROW NUMBER
            CSIZE=0.15*HEIGHT
C      CSPACE IS THE HEIGHT OF THE SPACE BETWEEN THE BAR CODE ROW
C             NUMBER AND THE ACTUAL BAR CODE ROW ITSELF.
```

```
            CSPACE=0.08*HEIGHT
C     TSIZE IS THE HEIGHT OF THE TITLE CHARACTERS.
            TSIZE=0.22*HEIGHT
C     TSPACE IS THE HEIGHT OF THE SPACE BETWEEN THE TITLE AND THE
C           FIRST BARCODE ROW NUMBER.
            TSPACE=CSIZE+(1.5*CSPACE)
C     HFACTR IS THE RELATIVE HEIGHT OF THE SUM OF THE BAR HEIGHT
C           PLUS THE SPACE BETWEEN THE BARS, INCLUDING LABELS
            HFACTR=1.7*HEIGHT
C     PLONG IS THE HEIGHT OF EACH PAGE OF BARCODE
            PLONG=11.0
C     PWIDE IS THE WIDTH OF EACH PAGE CF BARCODE
            PWIDE=8.5
C     TMAR IS THE HEIGHT OF THE TOP MARGIN OF THE PAGE
            TMAR=1.0
C     ESPACE IS THE HEIGHT OF THE MARGINS REPRESENTS HOW
C           FAR THE PAPER IS ADVANCED BETWEEN PAGES
            ESPACE=TMAR+1.5
C     SMAR IS THE HEIGHT OF THE LEFT SIDE MARGIN ON THE PAGE
            SMAR=1.5
C     ZERO IS A LEFT MARGIN OFFSET USED TO MOVE THE PHYSICAL
C           PLOT AWAY FROM THE LEFT MARGIN OF THE PLOTTER
            ZERO=3.0
C     PERPGE IS THE NUMBER OF BARCODE ROWS THAT WILL BE DRAWN PER PAGE
             PERPGE=IFIX((PLONG-ESPACE)/HFACTR)
            CALL PLOTS(0,0,0)
            CALL PLOT(0.0,0.0,3)
            IROW=0
            IPRT=0
            X=TMAR
            Y=ZERO
C
C
C
C     READ THE TITLE OF THE BARCODE PROGRAM FROM THE INPUT FILE
C
         READ(5,103)(TITLE(I),I=1,80)
103      FORMAT(80A1)
C
C
C
C     READ THE ROW OF ZERO'S AND ONE'S FROM THE INPUT FILE
C
10       READ(5,101,END=3000)(IN(I),I=1,132)
101      FORMAT(66A1,/,5X,66A1)
         IF(IPRT.GE.20) WRITE(6,201) (IN(I),I=1,132)
201      FORMAT(' ',132A1)
C
```

```
C
C      WRITE THE TITLE ON THE PLOT
C
20        IF(MOD(IROW,PERPGE).NE.0)GOTO 30
          X=X+ESPACE
          CALL  NEWPEN(2)
          XM1=X-TMAR
          XM2=X+PLONG-TMAR
          YM1=Y-SMAR
          YM2=Y+PWIDE-SMAR
          CALL  PLOT(XM1,YM1,3)
          CALL  PLOT(XM2,YM1,2)
          CALL  PLOT(XM2,YM2,2)
          CALL  PLOT(XM1,YM2,2)
          CALL  PLOT(XM1,YM1,2)
          CALL  PLOT(X,Y,3)
          CALL  SYMBOL(X,Y,TSIZE,TITLE,90.0,72)
          CALL  NEWPEN(NIBS)
          X=X+TSPACE
          Y=ZERO
          CALL  PLOT(X,Y,3)
C
C
C
C      LABEL THE BAR CODE ROW
C
30        ROWNUM=RCWNUM+1.0
          IROW=IROW+1
          CALL  NEWPEN(2)
          CALL  NUMBER(X,Y,CSIZE,ROWNUM,90.0,-1)
          CALL  NEWPEN(NIBS)
          X=X+CSPACE
          CALL  PLOT(X,Y,3)
C
C
C      CCNVERT THE ZERO'S AND ONE'S INTO BARS OF CORRECT WIDTH
C
       DO 1000  I=1,132
C
C
C
C      CHECK FOR ZERO OR ONE BIT
50        IF(IN(I).EQ.ZER) GOTO 100
          IF(IN(I).EQ.ONE) GOTO 200
          IF(IN(I).EQ.BLNK) GOTO 2000
C
C
C
```

```
C      DRAW A ZERO BAR OF UNIT WIDTH                                    VER01450
C                                                                       VER01460
100           X=X+HEIGHT                                                VER01470
              CALL PLOT(X,Y,2)                                          VER01480
              Y=Y+DOUBLE                                                VER01490
              X=X-HEIGHT                                                VER01500
              CALL PLOT(X,Y,3)                                          VER01510
              GOTO 1000                                                 VER01520
C                                                                       VER01530
C                                                                       VER01540
C                                                                       VER01550
C                                                                       VER01560
C                                                                       VER01570
C                                                                       VER01580
C      DRAW A ONE BAR OF DOUBLE UNIT WIDTH                              VER01590
C                                                                       VER01600
200           X=X+HEIGHT                                                VER01610
              CALL PLOT(X,Y,2)                                          VER01620
              Y=Y+UNIT                                                  VER01630
              X=X-HEIGHT                                                VER01640
              CALL PLOT(X,Y,3)                                          VER01650
              X=X+HEIGHT                                                VER01660
              CALL PLOT(X,Y,2)                                          VER01670
              Y=Y+DOUBLE                                                VER01680
              X=X-HEIGHT                                                VER01690
              CALL PLOT(X,Y,3)                                          VER01700
              GOTO 1000                                                 VER01710
C                                                                       VER01720
C                                                                       VER01730
C                                                                       VER01740
1000      CONTINUE                                                      VER01750
C                                                                       VER01760
C                                                                       VER01770
C                                                                       VER01780
C      GOTO NEXT ROW                                                    VER01790
C                                                                       VER01800
2000      X=X+HFACTR                                                    VER01810
          Y=ZERO                                                        VER01820
          CALL PLOT(X,Y,3)                                              VER01830
          GOTO 10                                                       VER01840
C                                                                       VER01850
C                                                                       VER01860
C                                                                       VER01870
C      FINISH UP THE PLOT:  PROGRAM IS CCMPLETE                         VER01880
C                                                                       VER01890
3000      CALL PLOT(0.,0.,+999)                                         VER01900
          STOP                                                          VER01910
          END                                                           VER01920
```

THE FOLLOWING IS THE DATA SET READ BY THE CROSS COMPILER.

```
0     62
HP41C SOURCE CODE:
+                 64     7      1     1     1
-                 65     8      6     1     1
*                 66     6      5     1     1
/                 67     9      7     1     1
X<Y?              68    97     92     4     1
X>Y?              69   101     95     4     1
X<=Y?             70    94     90     5     1
Σ+                71     3      2     2     1
Σ-                72     4      3     2     1
HMS+              73    50     48     4     1
HMS-              74    51     49     4     1
MOD               75    60     58     3     1
%                 76     1      8     1     1
%CH               77     2      9     3     1
P-R               78    63     61     3     1
R-P               79    68     67     3     1
LN                80    56     54     2     1
X|2               81   102     94     3     1
SQRT              82    80     78     4     1
Y|X               83   103    101     3     1
CHS               84    25     23     3     1
E|X               85    39     35     3     1
LOG               86    58     56     3     1
10|X              87    11    103     4     1
E|X-1             88    40     38     5     1
SIN               89    79     77     3     1
COS               90    32     30     3     1
TAN               91    87     85     3     1
ASIN              92    19     17     4     1
ACOS              93    13     11     4     1
ATAN              94    21     19     4     1
DEC               95    34     32     3     1
1/X               96    10    102     3     1
ABS               97    12     10     3     1
FACT              98    41     39     4     1
X#0?              99    90     98     4     1
X>0?             100   100     96     4     1
LN1+X            101    57     55     5     1
X<0?             102    92     93     4     1
X=0?             103    98    100     4     1
INT              104    53     51     3     1
FRC              105    45     43     3     1
D-R              106    33     31     3     1
R-D              107    67     66     3     1
HMS              108    49     47     3     1
HR               109    52     50     2     1
RND              110    72     71     3     1
OCT              111    61     59     3     1
CLΣ              112    26     24     3     1
X<>Y             113    96     89     4     1
PI               114    64     62     2     1
CLST             115    30     28     4     1
R|               116    74     65     2     1
RDN              117    71     70     3     1
LASTX            118    55     53     5     1
```

CLX	119	31	29	3	1
X=Y?	120	99	99	4	1
X#Y?	121	91	97	4	1
SIGN	122	78	76	4	1
X<=0?	123	93	91	5	1
MEAN	124	59	57	4	1
SDEV	125	76	74	4	1
AVIEW	126	22	20	5	1
CLD	127	28	26	3	1
DEG	128	35	33	3	1
RAD	129	69	68	3	1
GRAD	130	48	46	4	1
ENTER↑	131	38	37	6	1
STOP	132	86	84	4	1
RTN	133	73	72	3	1
BEEP	134	23	21	4	1
CLA	135	27	25	3	1
ASHF	136	18	16	4	1
PSE	137	66	64	3	1
CLRG	138	29	27	4	1
AOFF	139	15	13	4	1
AON	140	16	14	3	1
OFF	141	62	60	3	1
PROMPT	142	65	63	6	1
ADV	143	14	12	3	1
RCL	144	70	69	3	2
STO	145	85	83	3	2
ST+	146	82	79	3	2
ST-	147	83	81	3	2
ST*	148	81	80	3	2
ST/	149	84	82	3	2
ISG	150	54	52	3	2
DSE	151	36	34	3	2
VIEW	152	89	87	4	2
ΣREG	153	5	4	4	2
ASTO	154	20	18	4	2
ARCL	155	17	15	4	2
FIX	156	44	42	3	2
SCI	157	75	73	3	2
ENG	158	37	36	3	2
TONE	159	88	86	4	2
XROM	160			4	2
XROM	161			4	2
XROM	162			4	2
XROM	163			4	2
XROM	164			4	2
XROM	165			4	2
XROM	166			4	2
XROM	167			4	2
SF	168	77	75	2	2
CF	169	24	22	2	2
FS?C	170	47	45	4	2
FC?C	171	43	41	4	2
FS?	172	46	44	3	2
FC?	173	42	40	3	2
X<>	206	95	88	3	2

LIST OF REFERENCES

1. Wickes, W. C., _Synthetic Programming on the HP41C_, Larkin Publications, P.O. Box 987, College Park, Maryland 20740, 1980.

2. Hewlett-Packard Corporation, _The HP41C/41CV Alphanumeric Full Performance Programmable Calculator Owner's Handbook and Programming Guide_, 1980.

3. Dahl, O. J., Dijkstra, E. W., and Hoare, C. A. R., _Structured Programming_, Academic Press, 1972.

4. Hamming, R. W., _The Art of Programming the TI-59_, instructional monograph used at the Naval Postgraduate School, April 1980.

5. Weir, M. D., _Calculator Clout_, Prentice Hall, 1981.

6. Knuth, D.E., _The Art of Computer Programming_, Volume 3, Addison-Wesley, 1973.

7. Mendenhall, W., Scheaffer, R. L., and Wackerly, D. D., _Mathematical Statistics with Applications_, Duxbury Press, 1981.

8. Hillier, F. S. and Lieberman, G. J., _Introduction to Operations Research_, Holden-Day, 1980.

9. U. S. Army Human Engineering Laboratory, Technical Memorandum 1-76, _A Compiler for Pocket Calculators_, by D. B. Blazie, January 1976.

10 Carvalho, C. J., "BASIC Barcode Program," _PPC Calculator Journal_, v. 8, n. 2, March-April 1981.

11. McNeal, T., "Generating Bar Code in the Hewlett-Packard Format," _BYTE_, v. 6, n. 1, January 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	1
4.	Associate Professor S. H. Parry, Code 55Py Department of Operations Research Naval Postgraduate School Monterey, California 93940	5
5.	Associate Professor R. H. Shudde, Code 55Su Department of Operations Research Naval Postgraduate School Monterey, California 93940	5
6.	Office of the Commanding General U. S. Readiness Command ATTN: General Donn A. Starry MacDill AFB, Florida 33621	1
7.	Office of the Commanding General U. S. Army Training and Doctrine Command ATTN: General Glenn Otis Fort Monroe, Virginia 23651	1
8.	Chief TRADOC Research Element Monterey Naval Postgraduate School Monterey, California 93940	1
9.	Headquarters U. S. Army Training and Doctrine Command ATTN: Director, Studies and Analysis Directorate Mr. S. Goldberg Fort Monroe, Virginia 23651	1
10.	Headquarters U. S. Army Training and Doctrine Command ATTN: ATCG-T (BG Morelli) Fort Monroe, Virginia 23651	1

11. Headquarters 1
U. S. Army Training and Doctrine Command
ATTN: Director, Maneuver Directorate Combat Developments (LTC John VanZant)
Fort Monroe, Virginia 23651

12. Headquarters 1
U. S. Army Training and Doctrine Command
ATTN: ATTG-AE (LTC Smith)
Fort Monroe, Virginia 23651

13. Mr. Walter Hollis 1
Deputy Under Secretary of the Army
(Operations Research)
Department of the Army
Washington, D. C. 20310

14. LTG Howard Stone 1
Commanding General
U.S. Army Combined Arms Center
Fort Leavenworth, Kansas 66027

15. Director 1
Combined Arms Combat Development Activity
ATTN: ATZL-CAC-A (Mr. Lee Pleger)
Fort Leavenworth, Kansas 66027

16. Director, BSSD 1
Combined Arms Combat Development Activity
ATTN: ATZLCA-DS
Fort Leavenworth, Kansas 66027

17. Director 1
Combat Analysis Office
ATTN: Mr. Kent Pickett
Fort Leavenworth, Kansas 66027

18. Commandant 1
U. S. Army Command and General Staff College
ATTN: Education Advisor
Room 123, Bell Hall
Fort Leavenworth, Kansas 66027

18. Commandant 2
U. S. Army Command and General Staff College
ATTN: ATZL-SWB (MAJ Witschonke)
Fort Leavenworth, Kansas 66027

19. Dr. Wilbur Payne, Director 1
U. S. Army TRADOC Systems Analysis Activity
White Sands Missile Range, New Mexico 88002

20. Headquarters, Department of the Army 1
Office of the Deputy Chief of Staff
for Operations and Plans
ATTN: DAMO-2D
Washington, D.C. 20310

21. Commander 1
U. S. Army Concepts Analysis Agency
ATTN: MOCA-WG (LTC Earl Darden)
8120 Woodmont Avenue
Bethesda, Maryland 20014

22. Director 1
U. S. Army Material Systems Analysis Activity
ATTN: DRXSY-CM (Mr. Bill Niemeyer)
Aberdeen Proving Grounds, Maryland 21005

23. Director 1
U. S. Army Night Vision and Electro-Optical Lab
ATTN: DEL-NV-VI (Mr. Frank Shields)
Fort Belvoir, Virginia 22060

24. Director 1
U. S. Army TRADOC Systems Analysis Activity
ATTN: Mr Ray Heath
White Sands Missile Range, New Mexico 88002

25. Director 1
Combat Developments Studies Division
ATTN: MAJ W. Scott Wallace
U. S. Army Armor Agency
Fort Knox, Kentucky 40121

26. Commandant 1
U. S. Army Field Artillery School
ATTN: ATSF-MBT (CPT Steve Starner)
Fort Sill, Oklahoma 73503

27. Director 1
Combat Developments
U. S. Army Aviation Agency
Fort Rucker, Alabama 36362

28. Director 1
Combat Developments
U. S. Army Infantry School
Fort Benning, Georgia 31905

29. Director 1
Armored Combat Vehicle Technology Program
ATTN: COL Fleming
U. S. Army Armor Center
Fort Knox, Kentucky 40121

30. Director 1
Combat Developments
ATTN: MAJ William D. Meiers
U. S. Army Air Defense Agency
Fort Bliss, Texas 79905

31. Commander 1
U. S. Army Logistics Center
ATTN: ATCL-OS (Mr. Cammeron/CPT Schuessler)
Fort Lee, Virginia 23801

32. Director 1
U. S. Army Material Systems Analysis Agency
ATTN: DRXSY-AA (Mr. Tom Coyle)
Aberdeen Proving Grounds, Maryland 21005

33. Deputy Chief of Staff for Combat Developments 1
U. S. Army Training and Doctrine Command
ATTN: (MG Dick Boyle)
Fort Monroe, Virginia 23651

34. Deputy Commanding General 1
Combined Arms Combat Development Activity
ATTN: ATZL-CA-DC
Fort Leavenworth, Kansas 66027

35. Commandant 1
U. S. Army Signal School
Fort Gordon, Georgia 30905

36. Associate Professor Arthur L. Schoenstadt, Code 53Zh 1
Department of Mathematics
Naval Postgraduate School
Monterey, California 93940

37. Associate Professor James K. Hartman, Code 55Hh 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

38. Professor James G. Taylor, Code 55Tw 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

39. Professor Gary K. Poock, Code 55Pk 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

40. Professor Donald R. Barr, Code 55Bn 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

41. Mr. Henry Horn, Editor 1
Hewlett-Packard Keynotes
Hewlett-Packard Corporation
1000 N. E. Circle Blvd.
Corvallis, Oregon 97330

42. Mr. Richard Nelson, Editor 1
PPC Journal
2541 W. Camden Place
Santa Ana, California 92704

43. Headquarters, Department of the Army
    Office of the Deputy Chief of Staff for Personnel
    ATTN: DAPE-MPE-SS (Captain James Richmann)
    Washington, D. C. 20310 2