INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
A program translator software solution for remote data acquisition

Greenberg, Robert Joseph, M.S.
The University of Arizona, 1987

Copyright ©1987 by Greenberg, Robert Joseph. All rights reserved.
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages
2. Colored illustrations, paper or print
3. Photographs with dark background
4. Illustrations are poor copy
5. Pages with black marks, not original copy
6. Print shows through as there is text on both sides of page
7. Indistinct, broken or small print on several pages
8. Print exceeds margin requirements
9. Tightly bound copy with print lost in spine
10. Computer printout pages with indistinct print
11. Page(s) lacking when material received, and not available from school or author.
12. Page(s) seem to be missing in numbering only as text follows.
13. Two pages numbered. Text follows.
14. Curling and wrinkled pages
15. Dissertation contains pages with print at a slant, filmed as received
16. Other

___________________________________________________________________________
___________________________________________________________________________

UMI
A PROGRAM TRANSLATOR SOFTWARE SOLUTION FOR REMOTE DATA ACQUISITION

by

Robert Joseph Greenberg

A Thesis Submitted to the Faculty of the SCHOOL OF RENEWABLE NATURAL RESOURCES
In Partial Fulfillment of the Requirements For the Degree of
MASTER OF SCIENCE WITH A MAJOR IN WATERSHED MANAGEMENT
In the Graduate College
THE UNIVERSITY OF ARIZONA

1987

Copyright 1987 Robert Joseph Greenberg
STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: [Signature]

APPROVAL BY THESIS COMMITTEE

This thesis has been approved on the date shown below:

Lloyd W. Gay
Thesis Director
Professor of Watershed Management

Edwin H. Carpenter
Associate Research Scientist in Sociology in Agricultural Economics

Martin M. Foose
Professor of Watershed Management

David A. King
Professor of Renewable Natural Resources

Dec. 7, 1987
Dec. 9, 1987
7 Dec 87
7 December 1987
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF ILLUSTRATIONS</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vii</td>
</tr>
</tbody>
</table>

## CHAPTER

1. INTRODUCTION. ................................. 1
   - The State of Remote Data Acquisition. ..... 2
   - Universal Type Data Acquisition Programs. 4
   - Program Languages .......................... 4
   - Methodology and Objectives. ................ 5

2. ANALOG MEASUREMENT. ....................... 8
   - Active Transducers. ........................ 8
   - Passive Transducers ........................ 9
   - Converting Analog Measurements to Digital Data. ................. 11

3. REAL-TIME TERMINOLOGY .................... 13

4. DEVELOPMENT TOOLS ......................... 17
   - Software Specifications .................... 17
   - System Program Language Attributes. .... 19
   - Application Program Language Attributes. .... 20
   - Hardware Specifications .................... 21
   - Portable Computer Selection ............... 21
   - Analog-to-Digital Conversion Instrument Selection ................................. 25
   - Mass-Storage Device Selection ............ 26
   - Printer Selection ......................... 26

5. THE ADAPT SYSTEM. ......................... 27
   - Setup File Headers ........................ 29
   - Setup File Body ............................ 31
# TABLE OF CONTENTS—Continued

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Categories</td>
<td>32</td>
</tr>
<tr>
<td>User Defined Variables</td>
<td>33</td>
</tr>
<tr>
<td>Channel Functions</td>
<td>33</td>
</tr>
<tr>
<td>Channel Commands</td>
<td>33</td>
</tr>
<tr>
<td>Time Functions</td>
<td>35</td>
</tr>
<tr>
<td>Mathematical Operators</td>
<td>35</td>
</tr>
<tr>
<td>Printer Specifications</td>
<td>35</td>
</tr>
<tr>
<td>Storage Specifications</td>
<td>39</td>
</tr>
<tr>
<td>Setup File Body Examples</td>
<td>40</td>
</tr>
<tr>
<td>Setup File Comments</td>
<td>42</td>
</tr>
<tr>
<td>Setup File Example</td>
<td>43</td>
</tr>
<tr>
<td>Application Programs</td>
<td>44</td>
</tr>
<tr>
<td>Translation Example</td>
<td>44</td>
</tr>
</tbody>
</table>

6. CONCLUSION                                    | 48   |

APPENDIX A: ADAPT USER'S GUIDE                  | 50   |
APPENDIX B: ADAPT COMPUTER PROGRAM LISTINGS.    | 57   |
APPENDIX C: SCREATE COMPUTER PROGRAM LISTING    | 62   |
APPENDIX D: SEDIT COMPUTER PROGRAM LISTING      | 66   |
APPENDIX E: STRANS1 COMPUTER PROGRAM LISTING    | 70   |
APPENDIX F: STRANS2 COMPUTER PROGRAM LISTING    | 77   |
APPENDIX G: PARSE COMPUTER PROGRAM LISTING      | 87   |
REFERENCES                                      | 92   |
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Representation of software solution</td>
<td>7</td>
</tr>
<tr>
<td>2.</td>
<td>Time-based components</td>
<td>14</td>
</tr>
<tr>
<td>3.</td>
<td>System and Application Programs</td>
<td>18</td>
</tr>
<tr>
<td>4.</td>
<td>Program language use</td>
<td>24</td>
</tr>
<tr>
<td>5.</td>
<td>Creating Setup Files</td>
<td>28</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Channel Functions Supported by ADAPT.</td>
<td>34</td>
</tr>
<tr>
<td>2.</td>
<td>Channel Commands Supported by ADAPT</td>
<td>34</td>
</tr>
<tr>
<td>3.</td>
<td>Mathematical Operators Supported by ADAPT</td>
<td>36</td>
</tr>
</tbody>
</table>
ABSTRACT

A software solution was developed for remote data acquisition applications, i.e., applications where line power is unavailable. The solution was developed in response to a lack of suitable software for environmental measurements using battery powered computers. The software solution is in the form of a program translator that creates programs dedicated to specific remote data acquisition applications from a dialect governed by specific rules.

A methodology is presented for defining real-time measurement applications based upon three time components: a scanning interval, an average period, and an averaging interval.

The software solution is termed ADAPT, an acronym for 'All-purpose Data Acquisition Program Translator'. ADAPT was written for a Hewlett-Packard hand-held computer, the HP-71, and a Hewlett-Packard data acquisition system, the HP-3421A. The methodology and algorithms may be applied to other computer and data acquisition systems.
CHAPTER 1

INTRODUCTION

Modern natural resource management practices rely upon the monitoring of environmental phenomena as a basis for decision making. The objective of this thesis is to provide a tool to assist the natural resource scientist in environmental data acquisition at remote environments. Remote Data Acquisition (RDA) is defined as the electronic collection and processing of analog measurements in an environment where line power is unavailable.

In studies of renewable natural resources, environmental data collection is often required in remote sites over extended periods of time. A variety of microclimate variables are commonly measured (wind speed, wind direction, air temperatures and humidity, solar radiation, precipitation, etc.). Data collection of this nature is often accomplished through the use of analog sensors. These devices are based on the properties of materials to provide a varying electrical signal (voltage, resistance, current) in response to environmental change.

In recent years, computer technology has provided great advancement to the measurement of analog signals.
Microprocessor technology applied to this field now permits intensive and complex measurements not previously possible. Modern, computer controlled, data acquisition systems convert analog signals to digital values that can be interpreted by a computer.

Measurements with computer controlled data acquisition systems range from the simple to the complex, i.e., from measurement with a single air temperature sensor to determine a daily mean, to the measurement of hydraulic pressure at many locations across an ocean floor to analyze fluctuations in the global sea-level.

The introduction of computers into the field of data acquisition has also imposed upon the natural resource scientist a requirement for computer science knowledge. While the advent of high-level computer programming languages has reduced the degree of computer familiarity required by the scientist (Malmstadt, 1981), presently available computer languages are undesirably complex for the specific application to remote data acquisition.

A higher-level language is needed if scientists are to exploit the potential of microcomputers without learning the many details of present programming languages.

**The State of Remote Data Acquisition**

Data acquisition systems are composed of three basic components: sensors, analog-to-digital (a-to-d)
convertors and microprocessor controllers. In recent years, the advancement of technology has been most pronounced in the latter. Although design of sensors and a-to-d convertors are becoming more refined, they have developed at a much slower rate than microprocessor technology. Presently, advances in the science of data acquisition are most promising within the realm of microprocessor and computer applications.

Advances in computer technology include improvements in hardware and software components. Until recently, a-to-d convertors and microprocessors required line power, which effectively limited their RDA applications. However, recent designs are becoming suitable for RDA applications. Inexpensive CMOS-based (complimentary metal oxide silicate) microcomputers are capable of intensive calculations with very low power requirements. Because these microcomputers are battery powered, they are capable of operating in remote environments where line power is not available. Future advancements in operating speed and data storage capabilities will be beneficial to RDA applications.

Software for RDA applications is currently available in two forms, 'universal-type' data acquisition programs and program languages. Each software method has its disadvantages.
Universal Type Data Acquisition Programs

Universal type data acquisition programs are presently available for large microcomputer controlled systems, but not for portable computers. Portable computers have proven valuable for remote data acquisition applications, but because of their recent emergence, suitable data acquisition software is not available.

Universal type data acquisition programs are typically large and complex in order to perform a wide variety of functions. This complexity reduces the amount of computer memory available for data storage and also reduces program operating speed. Portable computers typically have smaller memories and slower operating speeds than larger computers, which compound the disadvantages of such universal software.

Program Languages

It is beyond the scope of this paper to discuss the advantages of specific program languages for data acquisition applications. However, program languages that support flexible input-output and a variety of data structures are advantageous.

Program languages are becoming higher-level and require relatively less computer expertise than previously. Concepts of artificial intelligence have been applied to languages to reduce user effort. Recent versions of BASIC,
Pascal, and FORTRAN programming languages contain a wide-variety of intelligent functions and minimal structural constraints. Available languages are being customized for data acquisition applications.

Presently, the user may accomplish data acquisition with portable computers only by writing programs dedicated to specific tasks, because no universal data acquisition programs are available. Unfortunately, developing the necessary facility with a language in order to write data acquisition programs is difficult and time consuming, even with the advances in high-level languages mentioned above.

**Methodology and Objectives**

Clearly, available software for RDA is unsatisfactory. The primary objective of this study is to develop a software solution to this problem that facilitates a high-level conversation between the researcher and the computer. This objective is achieved through the following steps:

1) Synthesize background information necessary for development of the software solution. This includes classifying measurement types and establishing real-time terminology.

2) Select the tools needed for development of the
software solution. This includes specifying the hardware and software components used in the project.

3) Describe the software solution developed. The solution, a hybrid model combining the advantages of universal-type programs with those of programming languages, is a System Program that creates Application Programs dedicated to specific tasks (Figure 1). This System Program is entitled, All-purpose Data Acquisition Program Translator (ADAPT).
Figure 1. Representation of software solution.
CHAPTER 2

ANALOG MEASUREMENT

A transducer is defined as a device that converts information from a non-electrical domain to an electrical domain or from an electrical domain to a non-electrical domain (Malmstadt, 1981).

For Remote Data Acquisition applications, software should include support for the following type of measurement transducers (Stein, 1964):

1. Self-Generating or Active
2. Non-Self-Generating or Passive

**Active Transducers**

Active transducers produce an energy that is proportional to the state of their environment. A common type of active transducer is the thermocouple which is frequently used for temperature measurements. Thermocouple temperature measurement is based on the Seebeck phenomena, by which two wires, composed of dissimilar metals and joined at both ends, generate a thermoelectric voltage that is proportional to the temperature difference between the two junctions.
Temperature measurement using thermocouples is simple and inexpensive. The advantages of thermocouple measurement include wide temperature range, small size, high reliability, and ruggedness (Fritschen and Gay, 1979). The disadvantages of thermocouples include very low voltage output, a non-linear response, and the need for a temperature reference in order to determine absolute temperatures.

Radiometers, used for measuring solar intensity, are typically active transducers. They may detect radiation either thermally, photoelectrically, or photochemically), yet all common designs produce a voltage (Fritschen and Gay, 1979).

Current meters and anemometers that employ electromagnetic generators are active transducers as they output a voltage proportional to water or wind velocity. They may also employ mechanical or magnetic switches (as do tipping bucket raingages), photoelectric choppers or capacitance choppers to generate a frequency proportional to velocity (Fritschen and Gay, 1979). The frequency is a function of the absence or presence of a voltage or the change in sign of a voltage.

**Passive Transducers**

Passive transducers require the input of energy in order to produce an energy proportional to their
environment. Passive transducers include all impedance-based transducers such as resistors. Resistance transducers commonly used for temperature measurement include Thermistors and Resistance Temperature Detectors (RTD). Resistances in environmental sensors are commonly evaluated by inputting a known or constant current to the element, and then measuring the resulting voltage across the elements which is related to resistance by Ohm's Law. Thermistors, composed of semiconductor materials, exhibit extreme sensitivity and fast response. Self-heating errors in a thermistor can be quite large as they have a large temperature coefficient (Fritschen and Gay, 1979). Other disadvantages include an extremely non-linear response and a limited temperature range.

RTD's are composed of any of a variety of metals that exhibit a change in resistance with a change in temperature. The standard material is platinum, but nickle and nickle iron alloys are also common. RTD's can provide accurate measurements that are more linear than those of thermocouples and thermistors. Their disadvantages include slow response and sensitivity to self-heating.

Passive transducers may also express a frequency by the change from an infinite to finite resistance, such as in the opening and closing of a circuit.

Nearly all environmental phenomena may be measured
electronically, given the capability of measuring voltage, resistance, and frequency, and the sensors to do so. Therefore, software developed for RDA should support all three types of measurement.

**Converting Analog Measurements to Digital Data**

An electronic data acquisition system must be able to convert analog measurements to digital values for computer interpretation. A wide variety of commercially available analog-to-digital conversion instruments exist.

The majority of available analog-to-digital (a-to-d) convertors that support computer communications require alternating current line power. This requirement makes them unsuitable for remote environments where line power is not available and the use of electrical generators is not feasible or desired. Battery powered a-to-d convertors typically have less accuracy and speed than their line-powered counterparts. Nevertheless, many are sufficient for various types of remote data acquisition work.

The accuracy of a measurement is the degree of agreement between the measured value and the true value (Malmstadt, 1981). Measurement accuracy is a function of the type, condition and calibration of the sensors and the accuracy of the a-to-d convertor. Thus accuracy must be evaluated with regard to the specific measurement
conditions. For example, measurement using thermocouples in natural environments requires the a-to-d convertor to have microvolt resolution if precise temperatures are desired. On the other hand, outputs from resistance-based circuits are often much larger than those from thermocouples and the resolution of the convertor need not be as great.

Accuracy is preserved once the measurements are converted to digital values. After conversion to digital values, accuracy must be maintained by the computer system through minimization of round-off error.
CHAPTER 3

REAL-TIME TERMINOLOGY

Software for specific remote data acquisition applications should be described by a standard methodology to simplify its understanding.

All RDA applications are primarily real-time based, i.e., their rate of execution is dependent upon an event external to the computer, such as a clock triggering. For the purpose of this project a series of time-based components were developed to encompass a large variety of applications common in environmental measurements. These components (Figure 2) are:

1. Scanning Interval (SI)
2. Averaging Period (AP)
3. Averaging Interval (AI)

The Scanning Interval is the length of time between two subsequent measurements, including any inactive period. All multiple-measurement applications have a SI.

\[ SI = \frac{\text{measurement period}}{\# \text{ of measurements}} \]

For example, if a thermocouple voltage was to be measured 48 times, at equal intervals, throughout a period of 24 hours then \( SI = .5 \) hours.
Averaging is commonly applied to RDA to improve measurement accuracy. A single measurement may be derived from the arithmetic average of a series of successive triggerings. The Averaging Period is the period in which measurements are summed to derive a single average value. Not all applications include an AP and a single SI must not contain more than one AP. Thus: \( \text{AP} \leq \text{SI} \)

Single tasking computers, those capable of performing only one task at a time, have difficulty when confronted with an application in which \( \text{AP} = \text{SI} \), as no processor time is available for calculating, printing, or storing the measurements.

The Averaging Interval is the interval in which single measurements are triggered within an AP.

\[ \text{AI} = \frac{\text{AP}}{\# \text{ of single measurements}} \]

For most applications, measurements within an AP are typically triggered at the maximum speed of the analog-to-digital conversion instrument. In these applications AI is a function of the conversion instrument. This is termed a machine AI.

The user may wish to choose an AI greater than that imposed by the hardware. In this case the analog-to-digital conversion instrument will pause between single measurements. An AI greater than a machine AI is termed a software AI. A software AI component may be
applied for a variety of reasons, including reducing the self-heating effect of resistance elements by providing a short cooling period between triggerings. Software AI's may also be used for taking single triggerings of a measurement that behaves cyclically and for which the period of its cycle is known.

Using multiple-channel analog-to-digital conversion instruments, a single triggering may actually consist of a sequence of triggerings on different channels, but only one trigger from each channel.

The great majority of RDA applications may be described using this uniform method for distinguishing time components.
CHAPTER 4
DEVELOPMENT TOOLS

This section discusses selection of tools used in the development of ADAPT and presents specifications for the software and hardware components used in the project.

**Software Specifications**

The software solution presented in this paper provides for the user a high-level conversation with the computer for RDA applications. The task of creating a more conversational computer language is significantly simplified when the language is dedicated to a specific application such as RDA.

The software solution is in the form of a program translator. A program translator translates one computer language into another. In this case, the software (ADAPT) will be translating from a higher-level dedicated language to an Application Program constructed of a standard program language. The language used to create ADAPT is referred to as the System Language and the language used by the programs created by the System is referred to as the Application Program language (Figure 3).
Figure 3. System and Application Programs.
The selection of program languages used to develop a System Program such as ADAPT is highly dependent upon the computer system chosen for the project. The following general discussion outlines the attributes required for project computer languages.

System Program Language Attributes

The following attributes are critical for the System language:

1. powerful string manipulations,
2. file transformations, and
3. available for project computer.

The language should be capable of interpreting a user created text file and constructing a companion text file. The companion file must be properly constructed such that it can be transformed into an operating Application Program.

Interpreting a string of text to a form more easily understood by the computer is termed parsing, the process of breaking down an input string into its most elementary parts (Seymour, 1984). Parsers rely heavily upon string manipulation functions.

The language should be capable of transforming a text file into an executable program. Some program languages include utilities, termed transformers, that convert a text file into a program language of its same
type. Transformers allow programs to write programs. A relatively limited selection of languages is available for a given portable computer. The System program language must be available for the portable computer system selected for the project.

Application Program Language Attributes

The following attributes are critical for the application program language:

1. commonly used by the scientific community,
2. readily edited or altered,
3. Flexible input/output,
4. Executes programs under alarm control,
5. Supports large floating-point arrays, and
6. Available for project computer.

To support advanced or extremely specific RDA applications, the Application Program should support editing and alteration by the user. It should be a program language commonly understood by the scientific community.

The primary task of the Application Program is to send and receive data to and from an analog-to-digital conversion instrument and the computer. In addition it should support the use of an external mass-storage device and printer. Flexible input and output is critical to a program that interacts with multiple external devices in a real-time environment.
The Application Program should be capable of executing from an off or low activity state by command of preset alarms. The alarms should be programmable and repeating to support consecutive SI's.

The Application Program should support large arrays of data in its internal memory to minimize or eliminate accessing a mass-storage device. The data arrays should support floating point numbers as the range of numbers encountered in data acquisition varies greatly.

The Application Program language must be able to run on the same portable computer as the System language. Because program language selection is heavily hardware dependent, the actual languages used in the project will be discussed in the following section.

Hardware Specifications

This section discusses hardware selected for the project, including the computer, a-to-d convertor, mass storage device, and printer.

Portable Computer Selection

An HP-71 handheld computer manufactured by the Hewlett Packard Company was chosen for the system. The computer represents state-of-the-art technology in CMOS microcomputers and was specifically designed to support applications similar to remote data acquisition (Wechsler,
In addition the computer is considered inexpensive and readily available. The manufacturer has in the past demonstrated expertise in designing computer hardware for scientific and engineering applications.

The predecessor of the HP-71, the HP-41 handheld computer has been proven durable for field use in hydrologic analysis (Greenberg, et al., 1983).

The HP-71 is 3.9" x 7.5" x 1" in size and weighs 12 ounces. It is supplied with 17.5 kilobytes of Random Access Memory (RAM). It is capable of accessing a maximum of 512 kilobytes of memory. Memory may be expanded by installing modules into four memory ports on the computer. The HP-71 has a CMOS microprocessor with a 20-bit internal architecture and a 4-bit data path. It was designed with emphasis on low-power consumption and high accuracy binary-coded-decimal math. The computer includes a 64 kilobyte operating system with an expanded form of the BASIC program language. The operating system is in the form of Read Only Memory, ROM, so RAM remains available for programs and data storage.

**HP-71 BASIC.** The BASIC program language is widely used in microcomputers within the scientific community. Recent versions support functions previously considered deficient in BASIC.

The BASIC language supplied with the HP-71 is an
interpretive BASIC that includes 240 keywords. It supports programmable alarms, file transformations, large arrays of data, and string manipulations (Wechsler, 1984). The HP-71 BASIC is suitable for much of the ADAPT operating system, and is considered an ideal language for the construction of the Application Programs.

**HP-71 FORTH/Assembler.** A FORTH operating language and an assembler are available for the HP-71 in a 32 kilobyte ROM module (Miller, 1984). FORTH is a compiled language that was developed at the University of Arizona in 1964 for telescope control at the Kitt Peak National Observatory. The primary advantage of FORTH over BASIC is execution speed.

A single program on the HP-71 may be composed of both FORTH and BASIC language. Segments of the program that were inefficient in BASIC were written in FORTH. The FORTH operating language also includes a machine code assembler that will ultimately permit time intensive calculations, such as polynomial solving, to be written in machine code. The System language will then become a unique combination of an interpreted BASIC, a compiled FORTH, and machine code created with the assembler (Figure 4).

**Interface Loop.** The HP-71 communicates to external devices via the proprietary Hewlett Packard - Interface
Figure 4. Program language use.
Loop, termed HP-IL (Katz, 1982). HP-IL is a low power, bit-serial interface designed for battery operated systems. The interface implementation on the HP-71 includes an independent 8-bit CMOS microprocessor and a 16 kilobyte ROM containing extensive input/output functions.

Devices in an HP-IL system are connected by two-wire cables leading from the output port of one device to the input port of the next, until all devices form a closed loop. Data is transmitted sequentially from device to device around the loop, until it returns to the source device where it is checked for errors. The maximum data rate of an HP-IL system is 20 kilobytes per second over a distance of 100 meters, although the HP-71 can transmit a maximum of 6 kilobytes per second. The more commonly used RS-232 interfaces consume more power, are slower, and are restricted to shorter transmission distances.

HP-IL systems have been tested at the University of Arizona for the past 2 years, for microclimate and surface energy budget measurements. The systems have proven to be quite adequate for these applications (Gay, 1984).

Analog-To-Digital Conversion Instrument Selection

An HP-3421A Data Acquisition System was chosen for a digital conversion instrument (Ressmeyer, 1983). The 3421A is a 30-channel battery powered, 5 1/2 digit, HP-IL unit. This is equivalent to a resolution of one microvolt
on the 200 millivolt range. No other battery powered a-to-d convertor on the market supports similar resolutions (Sampl, 1982).

The 3421A performs voltage, resistance, and frequency measurements. It has a medium speed, integrating voltmeter and scans at a rate of 2 channels per second at maximum resolution.

Mass-Storage Device Selection

An HP-9114A was selected as a mass-storage device suitable for RDA. The 9114A is a battery powered disk drive that stores a maximum of 710 kilobytes on a 3.5 inch microfloppy disk. The drive includes an HP-IL interface (Mims, 1984).

Printer Selection

An HP-2225B printer was chosen for hard-copy output for RDA application. The 2225B is an HP-IL battery powered ink-jet printer that uses 8.5 inch width paper. The printer speed is rated at 150 characters per second and it prints 200 pages on a single battery charge (Mims, 1984).
CHAPTER 5

THE ADAPT SYSTEM

ADAPT requires the user to communicate in a dialect governed by a specific set of rules. ADAPT examines the communication and creates an Application Program from the information. To create an Application Program, the user must first develop a Setup File. The Setup File, when complete, can be printed, and further edited. The final version of the Setup File is then transformed into an Application Program.

Setup Files Setup Files are common ASCII (American Standard Code for Information Interchange) text files. They contain information specific to a single dedicated RDA application. Setup Files may be created on the HP-71 or on a host computer and copied into the HP-71 (Figure 5).

The System queries the user for input into the Setup File (see Appendix A for specific user instructions). The input information is error-checked. If an error is detected, an English language warning message is generated.

The System supports the use of standard word processing and text editing software to edit and form Setup Files. This feature may be used by advanced users in place
Figure 5. Creating Setup Files.
of the query-response method. The System error-checks an entire Setup File after a text editing session and generates appropriate error messages. When the Setup File is in final form, the user commands the system to translate the file into the desired application program.

A Setup File is divided into three sections: Header, Body and Comments.

**Setup File Headers**

The Header includes information governing SI, AP, and AI as well as information controlling mass storage format, printer format, and powering down capabilities. The present time and date are also included in the header for reference purposes. The Header format is as follows:

- **Line 1**: ADAPT version, present time, present date
- **Line 2**: Powerdown specifier, Yes or No
- **Line 3**: SI (seconds), AP (seconds), AI (seconds)
- **Line 4**: Mass Storage Interval, Mass Storage File Name
- **Line 5**: Print Header Specifications: date, time, etc...

Line 1 is optional and used only for reference. Normally ADAPT inserts this line automatically.

Line 2 governs the powering down capabilities of the Application Program. The computer and some external devices are hardware capable of achieving an off or
low-power state when inactive. Powerdown is vital for conserving battery power for remote applications. A 'no powerdown' option is supported for applications where line power is available.

Line 3 includes the time based parameters as discussed in chapter 3. If AP is not stated, it is interpreted as the period of one scan. If AI is not stated, a machine AI is used.

Line 4 delineates the mode of data storage and receives the name of the mass storage data file to be used by the Application Program. ADAPT accepts two modes of delineation, an interval based value or a percentage based value.

An interval based value i directs the Application Program to write to the mass-storage device after i SI's. For example if an i of 1 was used, the Application Program would store the data from a single SI on the mass storage device after each SI. If an i of 5 was used, the computer would accumulate data from 5 SI's before storing on the disk drive.

A percentage based value p, where 1 <= p <= 100, directs the Application Program to write to the mass-storage device after p percent of computer memory becomes filled. If p = 100, all available computer memory would become filled with data before it was stored on the
mass-storage device. Specifying \( p = 100 \) allows for minimum activity of the mass-storage device. The mass-storage device has the greatest power consumption of the peripherals used in this system. For long-term remote applications, it may become necessary to minimize access to the more power-consumptive devices to prolong battery life.

Line 4 also includes the name of the mass-storage data file to be used by the Application Program.

Line 5 is optional and includes information additional to the data to be included in the printout. Print options include present time, present date, the name of the mass-storage data file, the present position of the mass-storage data file, and status value that reflects the present operating status of the analog-to-digital conversion instrument.

The following is an example Setup File Header:

```
ADAPT version 1.0 84/11/22
No Powerdown
Mass Store: 10, Filename: SOLAR
Scan Interval: 360, Average Period: 30, Average Interval 5
Print: Date, Time, Record#
```

**Setup File Body**

The body contains explicit information on the number and type of measurement channels to be used. There are three categories of lines within the body of a Setup
File. Lines within the body of a Setup File are composed of user-defined variables, channel functions, channel commands, time functions, mathematical operators, printer specifications, and storage specifications.

Line Categories

The body of a Setup File contains three categories of lines: Prefix, Central, and Suffix.

Prefix lines are those containing commands that will be executed previous to the AP. Commands within a prefix line are not averaged and are only executed once during each SI. Prefix lines are optional. They are designated by a leading '<' (less than) symbol in the line.

Central lines are those lines containing commands that are executed and averaged during the AP. If an AI is specified, they are averaged at the rate designated by AI. Central lines are designated by a lack of a leading symbol.

Suffix lines are those lines containing commands that will be executed following the AP. Commands within a suffix line are not averaged and are only executed once during each SI. Suffix lines are optional. They are designated by a leading '>' (greater than) in each line.

The prefix, central, and suffix are subordinated delineations of SI. The Inactive period is the period when no measurements are being logged and the computer is awaiting the following SI.
User Defined Variables

The result of a measurement or calculation may be designated by defining a corresponding variable. All user defined variables must be enclosed in quotes. Examples of user defined variables are: "DC VOLTS", "X", "Solar Intensity (W/cm^2/s)" and "WETBULB one". Lines containing user defined variables require the presence of an = (equal) sign.

Channel Functions

Channel functions are commands that are understood by the analog-to-digital convertor. When channel functions are executed, the argument of the channel function is returned to the system. All channel functions are enclosed in apostrophes. Examples of channel functions understood by the HP-3421A Data Acquisition System are: 'DCV12' (measure DC volts on channel 12), and 'TEM26' (measure thermocouple temperature on channel 26). Channel functions must appear to the right of an = (equal) sign in a line. Table 1 contains a list of channel functions supported by ADAPT.

Channel Commands

Channel Commands are commands that are understood by the analog-to-digital convertor but do not return an argument. Channel commands are not enclosed by any type of
Examples of channel commands include OPN (open all channels), CLS8 (close channel 8) and RS (reset data acquisition system). Table 2 contains a list of channel commands supported by ADAPT.

### Table 1. Channel Functions Supported by ADAPT.

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCVxx</td>
<td>Measure DC volts on channel xx</td>
</tr>
<tr>
<td>ACVxx</td>
<td>Measure AC volts on channel xx</td>
</tr>
<tr>
<td>TWOxx</td>
<td>Measure two-wire resistance on channel xx</td>
</tr>
<tr>
<td>FWOxx</td>
<td>Measure four-wire resistance on channel xx and supply current on paired channel</td>
</tr>
<tr>
<td>TEMxx</td>
<td>Measure T-type Thermocouple measurements on channel xx</td>
</tr>
<tr>
<td>REFxx</td>
<td>Measure temperature of reference junction at channel xx</td>
</tr>
<tr>
<td>FRQxx</td>
<td>Measure frequency with 1 second gate time on channel xx</td>
</tr>
<tr>
<td>TOTxx</td>
<td>Totalizes events on channel xx</td>
</tr>
</tbody>
</table>

### Table 2. Channel Commands Supported By ADAPT.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLSxx</td>
<td>Close channel xx</td>
</tr>
<tr>
<td>CLPxx</td>
<td>Close channel xx and channel xx + 10, if xx greater than 19 then close channel xx and channel xx-20</td>
</tr>
<tr>
<td>OPN</td>
<td>Open channel xx, if xx is not specified, open all channels</td>
</tr>
<tr>
<td>RS</td>
<td>Reset data acquisition system</td>
</tr>
</tbody>
</table>
**Time Functions**

The system supports two time-based functions, time and date. Time is expressed in seconds of day and date is expressed as: YYDDD, where YY is the year of century (0-99) and DD is the day of year (1-366).

This format readily facilitates time-based arithmetic. However, on printer listing, time is expressed in standard hours-minutes-seconds format and date is expressed in year-month-day of month format.

**Mathematical Operators**

Mathematical operators include arithmetic, logarithmic, trigonometric, and relational operators. All mathematical operators supported by the HP-71's BASIC program language may be included in a Setup File. Parentheses are used to establish mathematical hierarchy. Table 3 lists mathematical operators supported by ADAPT.

**Printer Specifications**

An optional printout following each SI is available. A user-defined variable's value may be included in the printout by including the word PRINT (or P) and a specifier at the end of a Setup File line. The print specifier includes a value, i.d, that determines the number of integers printed left of the decimal point (i) and the number of decimal places printed to the right of the
Table 3. Mathematical Operators Supported by ADAPT.

<table>
<thead>
<tr>
<th>Arithmetic Operators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>description</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>DIV</td>
<td>Division, returns integer portion of quotient.</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>%</td>
<td>Percent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General Math</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>description</td>
</tr>
<tr>
<td>ABS</td>
<td>Returns absolute value of its argument</td>
</tr>
<tr>
<td>CEIL</td>
<td>Returns smallest integer greater than or equal to argument</td>
</tr>
<tr>
<td>CLASS</td>
<td>Returns value indicating class of argument</td>
</tr>
<tr>
<td>EXPONENT</td>
<td>Returns exponent of its normalized argument</td>
</tr>
<tr>
<td>FACT</td>
<td>Returns factorial of non-negative integer</td>
</tr>
<tr>
<td>FLOOR</td>
<td>Returns greatest integer less than or equal to argument</td>
</tr>
<tr>
<td>FP</td>
<td>Returns fractional part of numeric value</td>
</tr>
<tr>
<td>INT</td>
<td>Returns greatest integer less than or equal to argument</td>
</tr>
</tbody>
</table>
Table 3, Continued

<table>
<thead>
<tr>
<th>keyword</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>Returns integer part of argument</td>
</tr>
<tr>
<td>MAX</td>
<td>Returns larger of two values</td>
</tr>
<tr>
<td>MIN</td>
<td>Returns smaller of two values</td>
</tr>
<tr>
<td>MOD</td>
<td>Returns remainder of modulo reduction</td>
</tr>
<tr>
<td>RED</td>
<td>Returns remainder of argument reduction</td>
</tr>
<tr>
<td>RMD</td>
<td>Returns remainder of division</td>
</tr>
<tr>
<td>RND</td>
<td>Returns pseudo-random number</td>
</tr>
<tr>
<td>SGN</td>
<td>Returns -1, 0, 1 if argument is less than zero, equal to zero, or greater than zero, respectively.</td>
</tr>
<tr>
<td>SQR</td>
<td>Returns square root of argument</td>
</tr>
</tbody>
</table>

**Logical and Relational Operators**

<table>
<thead>
<tr>
<th>keyword</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Performs logical And of its operands</td>
</tr>
<tr>
<td>EXOR</td>
<td>Performs logical Exclusive Or of its operands</td>
</tr>
<tr>
<td>NOT</td>
<td>Performs logical Not of its operand</td>
</tr>
<tr>
<td>OR</td>
<td>Performs logical Or of its operands</td>
</tr>
<tr>
<td>=</td>
<td>Performs Equality test on its operands</td>
</tr>
<tr>
<td>#</td>
<td>Performs Inequality test on its operands</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Performs Less Than or Greater Than test on its operands</td>
</tr>
<tr>
<td>&lt;</td>
<td>Performs Less Than test on its operands</td>
</tr>
</tbody>
</table>
Table 3, Continued

<table>
<thead>
<tr>
<th>keyword</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=</td>
<td>Performs Less Than or Equal To test on its operands</td>
</tr>
<tr>
<td>&gt;</td>
<td>Performs Greater Than test on its operands</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Performs Greater Than or Equal To test on its operands</td>
</tr>
<tr>
<td>?</td>
<td>Performs Unordered Comparison test on its operands</td>
</tr>
</tbody>
</table>

**Logarithmic Operations**

<table>
<thead>
<tr>
<th>keyword</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP</td>
<td>Returns the number $2.718281828...$ raised to power given by argument</td>
</tr>
<tr>
<td>EXPM1</td>
<td>Returns value of $e^{\text{argument}} - 1$</td>
</tr>
<tr>
<td>LOG</td>
<td>Returns natural logarithm (base $e$) of argument</td>
</tr>
<tr>
<td>LOGP1</td>
<td>Returns $\text{LOG}(1 + \text{argument})$</td>
</tr>
<tr>
<td>LOG10</td>
<td>Returns logarithm (base $10$) of argument</td>
</tr>
</tbody>
</table>

**Trigonometric Operations**

<table>
<thead>
<tr>
<th>keyword</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS</td>
<td>Returns arccosine of its argument</td>
</tr>
<tr>
<td>ANGLE</td>
<td>Returns polar angle determined by $(X,Y)$ coordinate pair</td>
</tr>
<tr>
<td>ASIN</td>
<td>Returns arcsine of its argument</td>
</tr>
<tr>
<td>ATAN</td>
<td>Returns arctangent of its argument</td>
</tr>
<tr>
<td>COS</td>
<td>Returns cosine of its argument</td>
</tr>
</tbody>
</table>
Table 3, Continued

<table>
<thead>
<tr>
<th>keyword</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEG</td>
<td>Converts argument in radians to degrees</td>
</tr>
<tr>
<td>DEGREES</td>
<td>Sets unit of measure for expressing angles to degrees</td>
</tr>
<tr>
<td>RAD</td>
<td>Converts arguments expressed in degrees to radians</td>
</tr>
<tr>
<td>RADIANS</td>
<td>Sets unit of measure for expressing angles to radians</td>
</tr>
<tr>
<td>SIN</td>
<td>Returns sine of its argument</td>
</tr>
<tr>
<td>TAN</td>
<td>Returns tangent of its argument</td>
</tr>
</tbody>
</table>

decimal point (d). For example, if a value 74.2094 was to be printed with aspecifier of 3.2 the value would print as 74.21. Leading spaces are included.

Storage Specifications

If a value is to be stored, specifying STORE (or S) at the end of a Setup File line, will store the value as directed by the storage specifications line in the Setup File header.

Printer and storage specifications are only valid with lines containing an = (equal) sign. Printer and storage specifications are separated by a semicolon.
Setup File Body Examples

The following example measures the output voltages of two photovoltaic cells exposed to sunlight under a variable forest canopy. The voltages are averaged during the AP from channels 12 and 13 and assigned the used-defined variables "Photo Cell 1" and "Photo Cell 2". "Photo Cell 1" and "Photo Cell 2" are both stored but not printed. Previous to the AP, the present time is recorded as user-defined variable "Pretime". "Pretime" is stored but not printed. Following the AP, the analog-to-digital convertor is commanded to open all channels (OPN) to minimize voltage spike damage to the instrument and the sum of the two averaged voltages is assigned a user-defined variable "Both Volts". The summed value will be printed with a specifier of 3.4.

< "PreTime" = TIME ; STORE
"Photo Cell 1" = 'DCV12' ; STORE
"Photo Cell 2" = 'DCV13' ; STORE
> OPN ;
> "Both Volts" = "Photo Cell 1" + "Photo Cell 2" ; PRINT 3.4

In the next example, both the time and date are recorded, as well as the voltage of the battery supplying power to the system, preceding the AP. Four NiFe RTD's (\( R_0 = 600 \) ohms) are measured during the AP. Following the AP,
the averaged resistances are converted to Celsius degrees through a quadratic equation. Also following the AP, the HP-71 is directed to emit a tone (BEEP) for a duration of one second if the battery voltage has dropped below 6.0 volts (a relational operator is applied). The date, time, and averaged resistances are stored but not printed and the battery voltage and calculated temperatures are printed but not stored.

< "Battery" = 'DCV12' ; P 2.3
< "Date" = DATE ; S
< "Time" = TIME ; S
"RTD1" = 'FW02' ; S
"RTD2" = 'FW03' ; S
"RTD3" = 'FW04' ; S
"RTD4" = 'FW05' ; S
> "Tem1" = "RTD1"^2 * -1.564E-4 + "RTD1" * .35906 - 1.8219 ; P 3.3
> "Tem2" = "RTD2"^2 * -1.524E-4 + "RTD2" * .36802 - 1.8755 ; P 3.3
> "Tem3" = "RTD3"^2 * -1.509E-4 + "RTD3" * .36181 - 1.8017 ; P 3.3
> "Tem4" = "RTD4"^2 * -1.582E-4 + "RTD4" * .35708 - 1.8252 ; P 3.3
> If "Battery" < 6 then BEEP 1
The following example is identical to the preceding example except that resistances are converted to temperature during the AP and the temperature values are averaged. The averaged temperatures are printed and stored.

< "Battery" = 'DCV12' ; P 2.3
< "Time" = TIME ; S
"Tem1" = "FWO1"^2 * -1.564E-4 + "FWO1" * .35906 - 1.8219 ; P 3.3 ; S
"Tem2" = "FWO2"^2 * -1.524E-4 + "FWO2" * .36802 - 1.8755 ; P 3.3 ; S
"Tem3" = "FWO3"^2 * -1.509E-4 + "FWO3" * .36181 - 1.8017 ; P 3.3 ; S
"Tem4" = "FWO4"^2 * -1.582E-4 + "FWO4" * .35708 - 1.8252 ; P 3.3 ; S
> If "Battery" < 6 then BEEP 1

Although each resistance channel function ('FWOx') is mentioned twice throughout the central region of the Setup File, a measurement of that channel is made only once during each scan. The resulting value is used twice.

Setup File Comments

At the end of the Setup File, the user has the option of entering comments pertaining to the Setup File. The System ignores Setup File comments; they only exist as an aid to the user. Comments should include information on
the measurement site, types of sensors used, etc. Comment
lines are designated by a leading ! (exclamation) sign.

Setup File Example

The previous Setup File Body Example is included in
the following example of a complete setup file:

ADAPT version 1.0 84/11/04
No Powerdown
Mass Store: 100%, File Name: ALTARSOIL
SI: 300, AP: 60
Print: Date, Time, Record#
< "Battery" = 'DCV12'; P 2.3
< "Date" = DATE; S
< "Time" = TIME; S
"Tem1" = "FW01"^2 * -1.564E-4 + "FW01" * .35906 - 1.8219 ;
P 3.3 ; S
"Tem2" = "FW02"^2 * -1.524E-4 + "FW02" * .36802 - 1.8755 ;
P 3.3 ; S
"Tem3" = "FW03"^2 * -1.509E-4 + "FW03" * .36181 - 1.8017 ;
P 3.3 ; S
"Tem4" = "FW04"^2 * -1.582E-4 + "FW04" * .35708 - 1.8252 ;
P 3.3 ; S
> If "Battery" < 6 then BEEP 1
! Site: Altar Valley
! Elevation: 815
! Soil: Sandy Loam
Measures Soil Temperatures at 100 cm. depth at 4 locations within 60 sq. meter homogenous area during two week period November 3 - 16, 1984.

Application Programs

The System examines a completed Setup File and translates it into an Application Program. Application Programs are BASIC language programs that perform dedicated tasks as determined by the contents of the Setup File. When the System is commanded to translate, it prompts for the name of a previously created Setup File and for the name of the Application Program to be created. An Application Program name may differ from the name of its source Setup File. Setup Files may be edited and additional Application Programs may be created. In addition, the user experienced in BASIC language programming may edit and alter the Application Program.

Translation Example

The System translates the previous Setup File example into the following BASIC program.

10 ! SOILTEMP to ALTARTEM, 84/12/09
20 DESTROY ALL @ RESET HPIL @ RESET @ OPTION BASE 1 30 K5=0
      @ K4=1
40 D1=DEVADDR('HP34') @ M1=DEVADDR('HP91')
50 PRINTER IS :HP2 @ PRINT CHR$(15) @ PWIDTH INF
60 ASSIGN #1 TO ALTARSOIL:M1
70 K3=(MEM-500)/4.5
80 SHORT S1(K3)
90 REAL J1(5)
100 INPUT 'run period (s) ? ','inf';V1
110 DISP 'TIME= ';TIME$ @ WAIT 1
120 INPUT 'start at [hh:mm:ss] ? ';T$
130 IF T$=''' THEN 170
140 T1=VAL(T$)*3600 @ T$=T$[POS(T$,':')]+1 @
   T1=VAL(T$)*60+T1
150 T1=VAL(T$[4])+T1 @ ON TIMER #1,T1-TIME-.5 GOTO 160 @
BYE
160 OFF TIMER #1
170 IF V1 THEN ON TIMER #3,V1 GOTO 550
180 ON TIMER #1,300 GOTO 190
190 DISP 'logging... ';TIME$
200 C1=TIME
210 OUTPUT :D1 ;'DCV12'
220 ENTER :D1 ;W1
230 A1=W1
240 J1(1)=DATE
250 J1(2)=TIME
260 K2=0 @ J1(3)=0 @ J1(4)=0 @ J1(5)=0 @ A7=0
270 T1=TIME
280 OUTPUT :D1 ;'FW002-05'
290 ENTER :D1 ;W2,W3,W4,W5
300 K2=K2+1
310 J1(3)=W2^2*23.345+W2*1.1305+386.53+J1(3)
320 J1(4)=W3^2*23.847+W3*1.2372+387.01+J1(4)
340 A7=W5^2*24.201+W5*1.2104+386.8+A7
350 T3=TIME @ IF T3<T1 THEN T3=T3+86400
360 IF T3-T1<60 THEN GOTO 280
370 J1(3)=J1(3)/K2 @ J1(4)=J1(4)/K2 @ J1(5)=J1(5)/K2 @ A7=A7/K2
380 IF A8<6 THEN BEEP 1
390 IF K4+5>K3 THEN GOTO 420
400 FOR K1 = 1 TO 5 @ S1(K4)=J1(K1) @ K4=K4+1 @ NEXT K1 @ K5=K5+1
410 GOTO 430
420 PRINT #1;S1 @ K4=1 @ DESTROY S1 @ SHORT S1(K3) @ GOTO 390
430 C1=C1/3600 @ T9=FP(C1)*60 @ T9=INT(T9)/100+FP(T9)*.006+INT(C1)
440 T$=STR$(FP(T9)) @ T$=STR$(INT(T9))&"\":&T$[2,3]&"\":&T$[4,5]
450 PRINT TAB(19);"N";'* ';&T$&' ';&' ;DATE$&' ';&' record#';K5;' O'
460 PRINT USING '80\"\"'
470 IMAGE 16X,5D.2D,2X,3D.3D,2X,3D.3D,2X,3D.3D,2X,3D.3D,2X,
480 PRINT TAB(17);CHR$(27)"&D"
490 PRINT ' Battery TEM1 TEM2 TEM3 TEM4 '
500 PRINT CHR$(27)"@"
510 PRINT USING 470;A1,JI(3),JI(4),JI(5),A7,A8
520 PRINT USING '80"|''
530 PRINT
540 BYE

550 OFF TIMER #1 @ OFF TIMER #2 @ OFF TIMER #3
560 PRINT #1;S1 @ SECURE ALTARSOIL:M1 @ PWIDTH 80 @
BYE
CHAPTER 6

CONCLUSION

The ADAPT system is effective for RDA applications that lie within the constraints of the methodology described in this thesis.

However, it is inevitable that specific RDA tasks will exceed the capabilities of the ADAPT system. For example, applications where results are a function of information gathered from multiple SI's are not supported by ADAPT. In the development of ADAPT, applications such as these were considered uncommon. However, ADAPT can provide a partial solution for unusual applications by generating a BASIC language Application Program that can be upgraded by the user. Although ADAPT is presently specific to a single hardware system, the methodology and algorithms presented here may be applied to other hardware.

ADAPT is an effective tool for minimizing the degree of computer familiarity required by the Natural Resource Scientist. It not only presents solutions for RDA software applications but it also serves as a learning tool for development of BASIC language programs. The Application Programs illustrate various techniques and
algorithms for effective RDA software development.

ADAPT represents a first generation solution for development of RDA software. Development of higher-level versions will require extensive field testing and user interaction. Inevitable improvements in computer hardware, analog-to-digital convertors, and program languages should be supported by future versions of ADAPT.
APPENDIX A

ADAPT USER'S GUIDE

The following Hardware/Firmware is required to run ADAPT:

HP-71B Computer
HP-3421A Data Acquisition System
HP-9114A Flexible Disk Drive
HP-82441A FORTH/Assemble ROM
HP-82401A HP-IL Interface
HP-2225B Ink-Jet Printer

ADAPT optionally supports the HP92198A Video Interface.

ADAPT consists of the following utilities:
CREATE
EDIT
TRANSLATE
RUN
FORMAT
PRINT
All utilities are accessible via the main ADAPT program. CREATE is the first step for designing an application program. It prompts the user for information pertinent to the application. It error-checks and edits the inputs and constructs a Setup File.

EDIT is used to alter an existing Setup File. At the termination of the editing session, the Setup File is automatically error-checked.

TRANSLATE converts a Setup File to an Application Program. It requires no interaction with the user.

RUN removes ADAPT from the computer and executes an Application Program.

FORMAT creates a data file on a disk. The data file is customized for a specific Application Program.

PRINT lists a Setup File on the printer in an expanded format.

**Running ADAPT**

Ideally, as much free memory as possible should be made available before running ADAPT. The memory required to execute ADAPT is proportional to the complexity of the application. Any file not needed should be removed from the HP-71 memory.

The ADAPT disk must be inserted in the disk drive. To execute ADAPT for the first time, type: RUN ADAPT:MASSMEM (do not use the RUN key). If ADAPT has been
run previously or exists in the computers memory, type: RUN ADAPT.

ADAPT will prompt, "setup file name ? " . If the user wishes to EDIT or TRANSLATE an existing Setup File, type in the Setup File name and key ENDLINE. If the user wishes to CREATE a new Setup File, type in a name of a nonexistent file.

Creating a Setup File, CREATE

Upon entering a nonexistent file name, ADAPT loads the CREATE utility from the disk. CREATE then prompts; "application name ?" . Type in a name for the application program (up to 8 characters) and key ENDLINE. CREATE then begins to prompt for specific information to be loaded into the Setup File. The prompts are listed in table A.1

Prompt Explanation And Possible Responses

  powerdown Y/N ?  Powers down external devices when not active, power down maximizes battery life. Key Y (yes) or N (no).

  storage specifier ?  an i or p value as described in chapter 6. If a percentage i value is given, the value is followed by a % (percent) sign. If measurements are not to be stored, key ENDLINE without input.
file name ? (prompts only if a storage specifier was input) The name of the mass-storage data file on the disk in which measurements will be stored. (up to 10 characters)

SI (sec) ? Scanning Interval value in seconds.

AP (sec) ? Averaging Period in seconds. If no AP is to be used, key ENDLINE without input.

AI (sec) ? (prompts only if AP has been specified) Averaging Interval in seconds. If a machine AI is to be used, key ENDLINE without input.

Print Date Y/N ? Include present date on printer header. Key Y for yes and N for no.

Print Time Y/N ? Include time at beginning of SI on printer header. Key Y for yes and N for no.

Print Filename Y/N ? Include mass-storage filename on printer header. Key Y for yes and N for no.

Print Record# Y/N ? Include present position of disk (in records) on printer header. Key Y for yes and N for no.

Print Status Y/N ? Include status byte of data acquisition system on header. Key Y for yes and N for no.

fx. 1 ? Begin to input function 1. Include prefix and suffix specifiers where applicable.

print "xxx" ? (prompts only if previous function included an = (equal) sign. where xxx is the user-defined
variable to the left of the = (equal) sign in previous prompt. Key Y for yes and N for no.

**print code ?**  (prompts only if previous response was yes) print specifier; i.d, where i = number of digits left of decimal point and d = number of digits right of decimal point.

**store "xxx" Y/N ?**  (prompts only if fx. contains an = (equal) sign.) key Y to store on disk, key n to no store

**fx. 2 ?**  continues to prompt for functions, if all functions have been entered, key ENDLINE without input and continue to following step, otherwise, continue input line data in same sequence.

**! ?**  enter comments if any, if all comments have been entered, key ENDLINE without input and continue to next step, otherwise continue to enter comments.

A complete Setup File has now been created. Execution returns to the main ADAPT program.

**Printing a Setup File, Print**

After a Setup File has been created, ADAPT prompts "print ssss Y/N ? ", where ssss is the name of the previously created Setup File. Key Y to print the setup file, key N to continue.

**Translating a Setup File, TRANSLATE**

When ADAPT is executed, it prompts for "setup file
name ?. If the name of an existing Setup File is input, ADAPT will prompt "Translate or Edit ? " (note, this prompt also occurs immediately after creating a Setup File).

If the user wishes to translate the Setup file into an Application Program, key T. ADAPT then prompts "application name ? ". Input a name (up to 8 characters) for the Application Program and key ENDLINE. Execution may require up to three minutes to translate a lengthy Setup File.

Editing a Setup File, EDIT

When ADAPT is executed, it prompts for "setup file name ? ". If the name of an existing Setup File is input, ADAPT will prompt "Translate or Edit ? ".

If the user wishes to edit the Setup File, key E. ADAPT the calls the text editor program within the FORTH/Assembler Module. For directions on using this text editor, see the FORTH/Assembler Module instruction module.

Upon ending an editing session (by keying E), ADAPT error- checks the entire Setup File. This may require up to two minutes to complete.

Formatting a Disk File, FORMAT

After a Setup File is translated, ADAPT prompts "format disk Y/N ?" . To format a disk, key Y. ADAPT will
then prompt "# of records ?". One record is required to store each SI's data. Make sure that the proper disk for storing the data is inserted in the disk drive. If ADAPT detects its own disk in the drive it will generate an error message. Input the number of records and key ENDLINE. ADAPT will format a data file of the proper dimensions for the Application Program previously translated.

If N was keyed program execution continues.

Running a Setup File, RUN

After the disk format prompt, ADAPT prompts "run nnnn Y/N ? ". where nnnn is the name of the Application Program. Key Y to run. When an application program is run, ADAPT and all of its utilities are cleared from memory and Application Program nnnn will be executed. If N was keyed, the Application Program will not be run and program execution ends.
APPENDIX B

ADAPT COMPUTER PROGRAM LISTINGS

10 ! ADAPT version 1.0, R.J.G. 10/30/84
50 A=DEVADDR('H16') @ SEND LISTEN A DDL 4 MTA DATA 0,0 @
    GOSUB 130
70 SEND UNL TALK A DDT 2 UNT UNL @ GOSUB 130 @ ENTER :A
    USING '#,8A';N$
100 IF N$[3,8]#'THESIS' THEN CALL ERROR(1) @ END
110 SEND LISTEN A DDL 7
120 GOTO 160
130 S=SPOLL(A) @ IF BIT(S,5) THEN 130
140 IF S THEN CALL ERROR(21) @ END
150 RETURN
160 OPTION BASE 1 @ DESTROY ALL @ SFLAG -1 @ PURGE TEST
170 ON ERROR GOTO 180 @ GOTO 190
180 OFF ERROR @ COPY ERRORF:A
190 K$=ADDR$('ERRORF') @ OFF ERROR
200 OFF ERROR @ RESET @ DIM K$[3],R$[96],R1$[96],F1$[12]
210 DIM L$[96],L9$[96],F9$[8],H$[8],G$[96]
220 GOTO 250
230 ASSIGN #3 TO HELP:A
240 IF DEVADDR('HP2225')<0 THEN GOTO 270

57
250 CALL YESNO('print display',K)
260 IF K THEN DISPLAY IS :PRINTER @ PRINT @ PRINT ELSE
   DISPLAY IS :DISPLAY @ SFLAG 2
270 IF F1$='ERRORF' THEN F1$='' 
280 IF F1$='' THEN 330 
290 FOR K=1 TO 99
300 R$=CAT$(K) @ IF R$='' THEN 330 
310 IF R$[1,LEN(F1$)]=F1$ AND R$[12,15]='TEXT' THEN GOTO 380
320 NEXT K
330 FOR K=1 TO 99
340 R$=CAT$(K) @ IF R$='' THEN 380 
350 IF R$[1,6]='ERRORF' THEN 370 
360 IF R$[12,15]='TEXT' THEN F1$=R$[1,8] 
370 NEXT K 
380 BEEP 999,.1 @ INPUT 'setup file name ? ',F1$;F1$ @
   F1$=UPRC$(F1$)
390 IF LEN(F1$)>8 THEN CALL ERROR(27) @ GOTO 380 
400 FORTHX " F1$" BASIC$ SYNTAXF' @ IF NOT FORTHI THEN
   CALL ERROR(28) @ GOTO 380 
410 DATA 'TEST','SEDIT','STRANS1','STRANS2','ADAPT',
   'SCREATE','FPANEL','ERRORF','HELPF'
420 RESTORE @ FOR K=1 TO 9 @ READ F2$ @ IF F1$=F2$ THEN
   CALL ERROR(31) @ GOTO 380 
430 NEXT K
440 ON ERROR GOTO 580

450 G$=ADD$(F1$) @ OFF ERROR

460 IF FILESZR(F1$)<6 THEN PURGE F1$ @ SFLAG 4 @ GOTO 580

470 BEEP 879,.1 @ DISP 'Translate or Edit ?'

480 K$=UP$(KEY$) @ IF K$=' ' THEN GOTO 480

490 K=POS('TE',K$) @ IF NOT K THEN 470

500 IF K=1 THEN GOTO 640

510 FORTHX '" SEDIT" FINDF' @ IF FORTHI THEN 560

520 SFLAG -1 @ IF MEM<2500 THEN PURGE SCREATE

530 IF MEM<2500 THEN PURGE STRANS2

540 IF MEM<2500 THEN PURGE STRANS1

550 IF MEM<2500 THEN CALL ERROR(34) @ END ELSE COPY SEDIT:A

560 CFLAG -1 @ CALL SEDIT(F1$) @ GOTO 470

570 GOTO 670

580 ON ERROR GOTO 590 @ GOTO 600

590 COPY SCREATE:A

600 CALL SCREATE(F1$)

610 OFF ERROR @ PURGE SCREATE

620 BEEP 500,.2 @ CALL YESNO('translate '&F1$,K)

630 IF NOT K THEN END

640 ON ERROR GOTO 650 @ RUN STRANS1

650 RUN STRANS1:A

660 'S': PURGE STRANS2

670 CALL YESNO('run '&F9$,K)

680 IF NOT K THEN END
690 SFLAG -1 @ PURGE STRANS @ PURGE SCREATE @ PURGE SEDIT @
RESET
700 DISP 'running ' &F9$ @ CHAIN F9$&':'' &STR$(A)
710 GOTO 880
720 !
730 SUB YESNO(Y$, K)
740 DISP Y$;' ? [Y/N]'@ GOTO 760
750 BEEP 1000,.2
760 BEEP 1700,.1
770 Y$=UPRC$(KEY$) @ IF Y$="" THEN 770 ELSE K=POS('NY?',Y$)
780 IF NOT K THEN 750 ELSE K=K-1
790 IF NOT K THEN DISP ' NO'@ GOTO 810
800 IF K=1 THEN DISP 'YES' ELSE CALL HELP
810 END SUB
820 SUB ERROR(E)
830 OFF ERROR
840 ASSIGN #5 TO ERRORF @ DIM E$[80]
850 SFLAG 7
860 RESTORE #5,E-1 @ BEEP 500,.1@ BEEP 400,.2
870 READ #5;E$ @ DISP E$ @ DESTROY ALL @ END SUB
880 END SUB
890 !
900 SUB PSETUP(F1$) @ EDTEXT (F1$), 'P N' @ END SUB
910 SUB TRIM(L$) @ CALL LTRIM(L$) @ CALL RTRIM(L$) @ END
SUB
61920 SUB LTRIM(L$) ! Remove left-hand spaces
930 IF L$[1,1]=" " THEN L$=L$[2] @ GOTO 930
940 END SUB
950 !
960 SUB RTRIM(G$)
970 K5=LEN(G$) @ IF G$[K5,K5]=" " THEN G$=G$[1,K5-1] @ GOTO 970
980 END SUB
APPENDIX C

SCREATE COMPUTER PROGRAM LISTING

30 SUB SCREATE(Fl$)
40 SFLAG -1 @ IF MEM<2500 THEN PURGE STRANS2
50 IF MEM<2500 THEN PURGE STRANS1
60 IF MEM<2500 THEN PURGE SEDIT
70 IF MEM<2500 THEN CALL ERROR(37) @ END
80 PURGE T @ CFLAG -1
90 DESTROY R2$,R5$ @ DIM R$[96],F3$[12],R1$[96],R5$[12]
100 FORTHX "PARSE" FINDF' @ IF NOT FORTHI THEN COPY
   PARSE:%16
110 IF NOT FLAG(4,0) THEN K=3 ELSE K=1
120 RESET @ DISP "recreating file "[K];Fl$;'...' @ WAIT .5
130 CREATE TEXT TEST,99 @ CREATE TEXT Fl$ @ CFLAG 12
140 ASSIGN #1 TO Fl$ @ SFLAG 5
150 R$=DATE$ @ R$='ADAPT version 1.0 '&R$ @ PRINT #1;R$
160 CALL YESNO('powerdown',K)
170 IF NOT K THEN R$='NO ' ELSE R$=''
180 PRINT #1;R$&'POWERDOWN LOOP' @ BEEP 1111,.1 @ INPUT
   'mass store interval?','1';F3$
190 CALL TRIM(F3$) @ R$='MASS STORE: '&F3$ @ F3$=F3$&'%'
200 IF NOT VAL(F3$[1,POS(F3$,'%')-1]) THEN 250
210 BEEP 999,.1 @ INPUT 'data file name ? ',';F3$ @ IF F3$='' THEN 210
220 IF LEN(F3$)>10 THEN CALL ERROR(30) @ GOTO 210
230 FORTHX '" F3$" BASIC$ SYNTAXF' @ IF NOT FORTHI THEN CALL ERROR(28) @ GOTO 210
240 R$=R$&', FILE NAME: '&UPRC$(F3$)
250 PRINT #1;R$
260 R$='SI' @ GOSUB 'SE' @ K1=Q
270 R$='AP' @ GOSUB 'SE' @ K2=Q
280 IF NOT K2 THEN 320
290 IF K2>=K1 THEN CALL ERROR(23) @ GOTO 260
300 R$='AI' @ GOSUB 'SE' @ K3=Q
310 K3=ABS(K3) @ IF K3>=K2 THEN CALL ERROR(25) @ GOTO 270
320 R$='SCAN INTERVAL:'&STR$(K1)&', AVERAGE PERIOD: '&STR$(K3)
330 IF K2 THEN R$=R$&', AVERAGE INTERVAL: '&STR$(K3)
340 PRINT #1;R$
350 DATA 'DATE','TIME','FILENAME','RECORD#','STATUS'
360 RESTORE 350 @ R1$='PRINT: '
370 FOR K1=1 TO 5
380 READ F2$ @ CALL YESNO('print '&F2$&',K)
390 IF K THEN R1$=R1$&F2$&','
400 NEXT K1
410 R1$=R1$[1,LEN(R1$)-1]
420 PRINT #1;R1$
430 K4=1 @ SFLAG 5 @ DESTROY V1$ @ DIM V1$[600] @ V1$=""
440 R1$=""
450 CFLAG 7 @ DISP 'fx. '&STR$(K4)&' ? '; @ BEEP 999,.2 @
   LINPUT '',R1$;R$
460 IF R$="" THEN GOTO 600
470 R5$="" @ CALL TRIM(R$)
480 R1$=R$ @ R1$=R1$[POS(R1$,"")+1] @
   R1$=""&R1$[1,POS(R1$,"")]] @ R5$=""
490 IF R1$=""" THEN GOTO 560
500 CALL YESNO('print '&R1$,K)
510 IF NOT K THEN 540
520 R5$=" ; P' @ BEEP 888,.1 @ DISP R1$;' digit code ? '; 
530 INPUT '';'';Q @ R5$=R5$&STR$(Q)&'
540 CALL YESNO('save '&R1$,K)
550 IF K THEN R5$=R5$&' ; S'
560 R$=R$&R5$ @ DISP 'checking...'
570 CALL PARSE(R$;V1$)
580 IF FLAG(7,0) THEN R1$=R$ @ GOTO 450
590 R$=R$&R5$ @ PRINT #1;R$ @ OFF IO @ DISP " OK" @
   K4=K4+1 @ RESTORE IO @ GOTO 440
600 BEEP 444,.1 @ DISP '! '; @ INPUT ''''';R$
610 IF R$="" THEN 630
620 PRINT #1;'' '&R$ @ GOTO 600
630 PRINT #1;'' @ DESTROY ALL @ PURGE TEST
640 GOTO 700
650 !
660 'SE': BEEP 700,.1 @ DISP R$;' sec ? '; @ INPUT '';R$
670 IF R$='' THEN Q=0 ELSE Q=ABS(VAL(R$))
680 RETURN
690 !
700 CALL YESNO('print '&F1$,K)
710 IF K THEN CALL PSETUP(F1$)
720 END SUB
APPENDIX D

SEDIT COMPUTER PROGRAM LISTING

30 SUB SEDIT(F1$)
40 SFLAG -1 @ PURGE TEST @ CREATE TEXT TEST,99 @ PURGE T @
   IF MEM<2500 THEN PURGE STRANS2
50 IF MEM<2500 THEN PURGE STRANS1
60 IF MEM<2500 THEN PURGE SCREATE
70 IF MEM<2500 THEN CALL ERROR(37) @ END
80 CFLAG -1
90 FORTHX "" PARSE" FINDF' @ IF NOT FORTHI THEN COPY
   PARSE:@16
100 ASSIGN #4 TO F1$
110 X=3 @ DIM G$[40] @ BEEP 999,.1 @ INPUT 'search string ?
   '','';G$
120 IF G$=" THEN 150 ELSE G=SEARCH(G$, 1,1, 999,4) @ IF NOT
   G THEN DISP 'NOT FOUND' @ GOTO 110
130 G$=STR$(INT(G)+1)&' T' @ GOTO 150
140 X=0 @ G$=STR$(G)&' T'
150 ASSIGN #4 TO * @ DISP 'reeediting '[X];F1$;''...
160 EDTEXT (F1$),(G$) @ DISP 'checking '&F1$&''...
170 DIM R$[96],R1$[96]
180 ASSIGN #4 TO F1$ @ RESTORE #4,1 @ READ #4;R$

66
190 IF NOT POS(UPRC$(R$),'POWERDOWN') THEN CALL ERROR(32) @
G=2 @ GOTO 140
200 G=3 @ READ #4;R$ @ R$=UPRC$(R$)
210 IF NOT POS(R$,'MASS STORE:') THEN CALL ERROR(33) @ GOTO 140
220 H=POS(R$,':') @ K=POS(R$,'&') @ IF K THEN
I=VAL(R$[H+1,K-1]) ELSE I=VAL(R$[I+1])
230 IF K AND I>100 THEN CALL ERROR(36) @ CALL ERROR(33) @ GOTO 140
240 IF NOT I THEN 310
250 J=POS(R$,'FILE NAME:') @ IF NOT J THEN CALL ERROR(33) @ GOTO 140
260 I=POS(R$,'':,J)+1
270 IF R$[I]='' THEN CALL ERROR(33) @ GOTO 140
280 IF R$[I,I]=' ' THEN I=I+1 @ GOTO 280
290 N$=R$[I] @ CALL RTRIM(N$) @ IF LEN(N$)>10 THEN CALL
ERROR(30) @ CALL ERROR(33) @ GOTO 140
300 FORTHX "N$" BASIC$ SYNTAXF' @ IF NOT FORTHI THEN CALL
ERROR(28) @ CALL ERROR(33) @ GOTO 1
310 G=4 @ READ #4;R$ @ R$=UPRC$(R$)
320 IF NOT POS(R$,'SCAN INTERVAL:') THEN CALL ERROR(34) @ GOTO 140
330 Pl=POS(R$,':')+1 @ IF R$[Pl+Pl]='' THEN CALL ERROR(34) @
GOTO 140
340 ON ERROR GOTO 360
350 T1=VAL(R$[P1]) @ OFF ERROR @ GOTO 370
360 OFF ERROR @ CALL ERROR(34) @ GOTO 140
370 I=POS(R$,'AVERAGE PERIOD:') @ IF NOT I THEN 440 ELSE ON ERROR GOTO 390
380 T2=VAL(R$[I+15]) @ OFF ERROR @ GOTO 400
390 OFF ERROR @ CALL ERROR(34) @ GOTO 140
400 IF T2>T1 THEN CALL ERROR(23) @ CALL ERROR(34) @ GOTO 140
410 I=POS(R$,'AVERAGE INTERVAL:') @ IF NOT I THEN 440 ELSE ON ERROR GOTO 430
420 ON ERROR GOTO 390 @ T3=VAL(R$[I+17])
430 OFF ERROR @ IF T3>T2 THEN CALL ERROR(25) @ CALL ERROR(34) @ GOTO 140
440 G=5 @ READ #4;R$ @ R$=UPRCS$(R$)&','
442 IF NOT POS(R$,'PRINT:') THEN CALL ERROR(35) @ GOTO 150
450 I=POS(R$,':')+1
460 IF R$[I]='' THEN GOTO 500
470 IF R$[I,I]=' ' OR R$[I,I]=',' THEN I=I+1 @ GOTO 460
480 J=POS(R$,'','I) @ IF POS('DATETIMENAME|RECORD|STATUS',R$[I,J-1]) THEN I=J @ GOTO 460
490 CALL ERROR(35) @ GOTO 140
500 C=FILESZR(F1$) @ G=5 @ DIM V1$[600] @ V1$="" @ RESET
510 CFLAG 7 @ G=G+1 @ IF G=C THEN RESET @ GOTO 550
520 READ #4;R$ @ CALL LTRIM(R$) @ IF R$[1,1]="!' THEN SFLAG
69

0 @ GOTO 510

530 IF FLAG(0) THEN CALL ERROR(17) @ GOTO 140 ELSE CALL PARSE(R$,VI$)

540 IF FLAG(7) THEN GOTO 140 ELSE GOTO 510

550 END SUB
APPENDIX E

STRANS1 COMPUTER PROGRAM LISTING

10 DEF FNS$(G$,H$)=G$[POS(G$,H$)+1]
20 DEF FNT$(G$,H$)=G$[1,POS(G$,H$)-1]
30 OPTION BASE 1 @ STD @ RESET
40 DIM F3$[12],F8$[10]
50 SHORT A(3)
60 R$=PEEK$ ("2F958",2) @ P=HTD(R$[2]&R$[1,1]) @ IF P=0 THEN  
   P=80
70 IF P<71 THEN PWIDTH 40 @ P=40 @ GOTO 110
80 IF P<80 THEN PWIDTH 71 @ P=71 @ GOTO 110
90 IF P<142 THEN PWIDTH 80 @ P=80 @ GOTO 110
100 PWIDTH 142 @ P=142
110 IF DEVADDR('HP2')<0 THEN PRINTER IS :DISPLAY
120 BEEP 1500,.1 @ INPUT 'application name ? ','';F9$
130 IF F9$='' THEN 120
140 IF F9$=F1$ THEN CALL ERROR(24)
150 PRINT @ GOSUB 'LINE' @ R1$='SETUP FILE NAME' @ F2$=F1$
   @ F3$=' ' @ GOSUB 'SP'
160 R1$='APPLICATION PROGRAM NAME' @ F2$=F9$ @ GOSUB 'SP'
170 R1$='PRESENT TIME' @ F2$=TIME$ @ GOSUB 'SP' @  
   R1$='PRESENT DATE:' @ F2$=DATE$ @ GOSUB 'SP'

70
180 OFF IO @ DISP "translating..." @ RESTORE IO
190 SFLAG -1 @ IF F1$#F9$ THEN PURGE F9$ @ CFLAG -1 @
CREATE TEXT F9$
200 ASSIGN #1 TO F1$ @ ASSIGN #2 TO F9$ @ PRINT #2;'9999!'
210 L=1 @ L$=DATE$ @ L$='!'&F1$&' to '&F9$&', '&L$ @ GOSUB
'WRITE'
220 L$='DESTROYALL@RESETHPIL@RESET@OPTIONBASE1' @ RESTORE
#1,1 @ READ #1;R$
230 IF NOT POS(UPRC$(R$),'NO') THEN L$=L$&'@ SFLAG-1'
240 L=2 @ GOSUB 'WRITE'
250 READ #1,2;R1$ @ R$=R1$ @ R$=FNS$(R$,':') @ IF
POS(R$,'%) THEN 280
260 K2=VAL(R$) @ IF NOT K2 THEN SFLAG 10
270 IF K2=1 THEN SFLAG 7
280 SFLAG 6 @ L$='K5=O0'
290 IF NOT FLAG(7) AND NOT FLAG(10) THEN L$=L$&'K4=1' @
GOSUB 'WRITE'
300 READ #1,4;R1$ @ R1$=FNS$(R1$,'%') @ CALL 'LTRIM'(R1$)
310 IF POS(UPRC$(R1$),'TIME') THEN SFLAG 5
320 IF R1$='' THEN SFLAG 11
330 L$="D1=DEVADDR('HP34')" @ IF NOT FLAG(10) THEN
L$=L$&"@M1=DEVADDR('HP91')"
340 L=4 @ GOSUB 'WRITE'
350 IF NOT FLAG(11) THEN L$='PRINTERIS:HP2@PRINT' @ GOSUB
'WRITE'
360 IF FLAG(10) THEN 390 ELSE READ #1,2;R$
370 R$=FNS$(R$,':') @ R$=FNS$(R$,':') @ CALL 'TRIM'(R$)
380 F8$=R$ @ L$='ASSIGN#1T0'&R$':'&M1' @ L=10 @ GOSUB 'WRITE'
390 DESTROY V1$,V2$,V3$,V4$,V5$,V6$,V7$
400 RESTORE #1,5 @ DIM V1$[600] @ V1$='' '
410 K2=0 @ K3=0 @ K4=0 @ Q1=5 @ FOR Q2=1 TO 3 @ A(Q2)=99 @ NEXT Q2
420 READ #1,Q1;R$ @ IF POS(R$,'!') THEN 520
430 Q2=2 @ IF POS(R$[1,5],'<') THEN Q2=1
440 IF POS(R$[1,5],'>') THEN Q2=3
450 K=POS(R$,'''')+1 @ I=A(Q2)
460 I=FP(I)+MIN(Q1,INT(I)) @ I=INT(I)+MAX(FP(I),Q1/1000) @ A(Q2)=I
470 IF K=1 THEN Q1=Q1+1 @ GOTO 420
480 R$=R$[K] @ K1=POS(R$,'''') @ K2=MAX(K2,K1-1) @ IF POS(V1$,R$[1,K1]) THEN 500
490 K3=K3+1 @ V1$=V1$&R$[1,K1]
500 R$=R$[K1+1] @ GOTO 450
510 NEXT Q1
520 Q1=65 @ DIM V2$(K3)[K2],V3$(K3)[6]
530 FOR K=1 TO K3 @ K1=POS(V1$[1],'''') @ V2$(K)=V1$[1,K1-1] @ V1$=V1$[K1+1]
540 IF NOT MOD(K+1,10) THEN Q1=Q1+1
V3$(K)$ = CHR$(Q1) \& STR$(\text{MOD}(K, 9))$ @ NEXT $K$

DESTROY V1$ @ SHORT V8(K3) @ Q3 = 0 @ Q4 = 1 @ SFLAG 8 @ SFLAG 9

FOR $K$ = 1 TO K3

Q2 = SEARCH(V2$(\text{RES})\&1, 5, 9990, 1) @ READ #1, Q2; R$ @ IF

INT(FP(Q2) \* 1000) > POS(R$, ';'') THEN 630

V8(K) = INF @ IF NOT POS(R$, ';'') THEN GOTO 630

R$ = FNSS(R$, ';'') @ Q2 = POS(R$, 'P') @ IF Q2 THEN

V8(K) = VAL(R$[Q2 + 1]) @ CFLAG 9

IF NOT POS(R$, 'S') THEN 630

CFLAG 8 @ V8(K) = 0 - V8(K) @ Q3 = Q3 + 1 @

V3$(K)$ = 'J1(' & STR$(Q4)$ & ')' @ Q4 = Q4 + 1

NEXT $K$

DIM V4$[250] @ V4$ = '' @ RESTORE #1, 5

READ #1; R$ @ IF POS(R$, '!') THEN 720

F2$ = CHR$(178) @ IF POS(R$[1,5], '<') THEN F2$ = CHR$(177)

IF POS(R$[1,5], '>') THEN F2$ = CHR$(179)

K = POS(R$, ''') + 1 @ IF K = 1 THEN 650

R$ = R$[K] @ K1 = POS(R$, ''') @ F3$ = R$[1, K1 - 1] @ IF K1 = 5 THEN F3$ = F3$[1, 3] & '0' & F3$[4]

IF POS(V4$, F3$, K4) THEN 710 ELSE K4 = K4 + 1 + K @

V4$ = V4$ & F2$ & F3$ & '''

GOTO 680

DIM V5$(K4)[6], V6$(K4)[2] @ Q1 = 86 @ FOR K = 1 TO K4

K1 = POS(V4$, '''') @ V5$(K) = UPRC$(V4$[1, K1 - 1]) @
V$=V$(K)
740 IF LEN(V$(K))=5 THEN V$(K)=V$(K)[1,4]'0'&V$(K)
750 IF NOT MOD(K-1,10) THEN Q1=Q1+1
760 V$(K)=CHR$(Q1)&STR$(MOD(K,10)) @ NEXT K
770 DESTROY V$ @ IF FLAG(10) THEN 910
780 L$='SHORT' @ L=50
790 R1$='INTERNAL DATA STORAGE SIZE'
800 READ #1,2;R$ @ R$=FNS$(R$,':') @ IF POS(R$,'%') THEN
810 CALL LTRIM(R$)
820 L$=L$&SI(K3)' @ GOSUB 'WRITE'
830 F2$='(MEMORY-500)/'
840 L$='K3=((MEM-500)/4.5' @ IF VAL(R$)=100 THEN L$=L$&')' @ GOTO 860
850 L$=L$&'*.'&R$&')'
860 L=45 @ F2$=F2$&STR$(INT(Q3*4.5/(VAL(R$)/100)))
870 GOTO 890
880 K=VAL(R$) @ K8=Q3*K @ L=56 @ L$=L$&SI('&STR$(K8)&')' @
890 IF LEN(L$)#5 THEN GOSUB 'WRITE'
900 F3$='intervals' @ GOSUB 'SP'
910 READ #1,3;R$ @ R$=FNS$(R$,':') @ T1=VAL(R$) @
920 IF FLAG(10) THEN 940
930 R1$="DISK DUMP INTERVAL" @ F2$=STR$(K*T1) @ GOSUB 'SP'
940 R1$='SCANNING INTERVAL' @ F2$=STR$(T1) @ GOSUB 'SP'
950 T2=VAL(R$) @ IF T2 AND POS(R$,:') THEN R$=FNS$(R$,:') @ T3=VAL(R$)
960 IF T2 THEN R1$='AVERAGING PERIOD' @ F2$=STR$(T2) @ GOSUB 'SP'
970 IF T3 THEN R1$='AVERAGING INTERVAL' @ F2$=STR$(T3) @ GOSUB 'SP'
980 L=60 @ IF Q3 THEN L$='REALJ1('''&STR$(Q3)&''')' @ GOSUB 'WRITE'
990 L$="INPUT 'run period (s) ? ',inf';V1" @ L=80 @ GOSUB 'WRITE'
1000 IF NOT FLAG(10) THEN R1$='DISK RECORD SIZE:' @ F2$=STR$(Q3*8) @ F3$='bytes' @ GOSUB 'SP'
1010 L=9000 @ L$='OFFTIMER#1@OFFTIMER#2@OFFTIMER#3' @ GOSUB 'WRITE'
1020 L$='' @ IF NOT FLAG(7) AND NOT FLAG(10) THEN L$='PRINT #1;S1@SECURE'&F8$&':M1@'
1030 IF NOT FLAG(9) THEN L$=L$&'PWIDTH('&STR$(P)&')@BYE' @ GOSUB 'WRITE'
1040 L=84 @ L$="'TIME=';TIME$@WAIT1" @ GOSUB 'WRITE'
1050 L$="INPUT'start at [hh:mm:ss] ? ',T$" @ GOSUB 'WRITE'
1060 GOSUB 'LINE' @ PRINT @ PRINT @ PRINT CHR$(27)&'&d0' @ CHAIN STRANS2:@16
1070 !
1080  'WRIT': ! Remove right-most chr. and WRITE
1090  L$=L$[1,LEN(L$)-1]
1100  'WRITE': IF L$='' THEN RETURN
1110  IF L=L1 THEN L=L+2
1120  BEEP 200,.01 @ FOR Q=FILESZR(F9$)-1 TO 0 STEP -1
1130  READ #2,Q;G$ @ IF VAL(G$)<L THEN 1150  
1140  NEXT Q
1150  INSERT #2,Q+1;STR$(L+10000)[2]&L$ @ L1=L @ RETURN
1160  !
1170  'LINE': FOR V=1 TO P/2 @ PRINT '||'; @ NEXT V @ PRINT @ RETURN
1180  PRINT '==' @ NEXT V @ PRINT CHR$(27)&'='&CHR$(13) @ FOR V=1 TO P/2-1
1190  PRINT '==' @ NEXT V @ ENDLINE @ PWIDTH P @ PRINT @ RETURN
1200  FOR V=1 TO 8
1210  PRINT CHR$(27)&'*B160w' @ PRINT USING "255'"'
1220  NEXT V @ PWIDTH P @ ENDLINE @ PRINT @ RETURN
1230  'SP': PWIDTH INF
1240  PRINT CHR$(252)&'&R1$&
1250  IF F3$="" THEN 1270
1260  FOR V=LEN(F2$) TO INT(P/4) @ PRINT '.'; @ NEXT V @ PRINT F3$;
1270  PRINT TAB(P+8);CHR$(252) @ PWIDTH P @ RETURN
APPENDIX F

STRANS2 COMPUTER PROGRAM LISTING

30 DEF FNS$(G$,H$)=G$[POS(G$,H$)+1]
40 DEF FNT$(G$,H$)=G$[1,POS(G$,H$)-1]
50 L$="T1=VAL(T$)*3600@T$=T$[POS(T$,'':')+l]
   @T1=VAL(T$)*60+T1" @ GOSUB 'WRITE'
60 L=90 @ L$="T1=VAL(T$[4])+T1@ONTIMER#1,T1-TIME-.5
   GOTO100@BYE" @ GOSUB 'WRITE'
70 L=100 @ L$="OFFTIMER#1@IFV1THENONTIMER#3,V1GOTO9000" @
   GOSUB 'WRITE'
80 L$='ONTIMER#1,'&STR$(T1)&'GOTO1000' @ L=200 @ GOSUB
   'WRITE'
90 L=1000 @ L$='logging... ';TIME$" @ GOSUB 'WRITE'
100 IF FLAG(5,0) THEN L$='C1=TIME' @ GOSUB 'WRITE'
110 K1=1 @ GOSUB 'LIST'
120 K1=1 @ GOSUB 'EXCHANGE'
130 L=2000
140 IF NOT T2 THEN 270 ELSE L$='K2=0@' @ Q4=999 @ T1=0
150 FOR K1=INT(A(2)) TO FP(A(2))*1000
160 READ #1,RES;R$
170 IF NOT POS(R$,'='') THEN 230
180 R$=FNS$(R$,'''') @ R$=R$[1,POS(R$,'''')-1] @ K=1
77
190 IF V2$(K)#R$ THEN K=K+1 @ GOTO 190
200 Q4=MIN(Q4,K) @ T1=MAX(T1,K) @ L$=L$&V3$(K)"'=0@' @ IF LEN(L$)<60 THEN 230
210 GOSUB 'WRIT'
220 L$="'
230 NEXT K1
240 GOSUB 'WRIT'
250 L=2050 @ IF NOT T2 THEN 270 ELSE L$='T1=TIME' @ GOSUB 'WRITE'
260 IF T3 THEN L=1900 @ L$='ONTIMER#2,'&STR$(T3)&'GOTO2200' @ GOSUB 'WRITE'
270 L=2200 @ K1=2 @ GOSUB 'LIST'
280 IF T2 THEN L$='K2=K2+1' @ GOSUB 'WRITE'
290 K1=2 @ GOSUB 'EXCHANGE'
300 IF NOT T2 AND K2<6 THEN 400 ELSE
   L$='T3=TIME@IFT3<T1THEN0=0+86400' @ GOSUB 'WRITE'
310 L$='IFT3-T1<"&STR$(T2)"THEN'
320 IF T3 THEN L$=L$&'BYEELSEOFFTIMER#2' ELSE L$=L$&'GOTO 2200'
330 GOSUB 'WRITE'
340 L$="'
350 FOR K=Q4 TO T1
360 L$=L$&V3$(RES)"'="&V3$(RES)"'/K2@'
370 IF LEN(L$)>55 THEN GOSUB 'WRIT' @ L$="'
380 NEXT K
390 GOSUB 'WRIT'
400 L=3000
410 K1=3 @ GOSUB 'LIST' @ K1=3 @ GOSUB 'EXCHANGE'
420 IF FLAG(10) THEN 490 ELSE L=4000
430 IF FLAG(7) THEN L$='PRINT#1;J1@K5=K5+1' @ GOSUB 'WRITE'
   @ GOTO 490
440 IF FLAG(6,0) THEN R1$='K3' ELSE R1$=STR$(K8)
450 L$='IFK4+''&STR$(Q3)&''>&R1$&'THEN GOTO 4100' @ GOSUB 'WRITE'
460 L$='FOR K1=1 TO ''&STR$(Q3)&'' @ S1(K4)=J1(K1)@K4=K4+1
   @ NEXT K1@K5=K5+1' @ GOSUB 'WRITE'
470 L$='GOTO 5000' @ GOSUB 'WRITE'
480 L=4100 @ L$='PRINT#1;S1@K4=1DESTROY S1@SHORTS1(''&R1$&'')
   @ GOTO 4000' @ GOSUB 'WRITE'
490 IF FLAG(11) THEN 1110
500 Q2=0 @ L=5000 @ K=0 @ K$='* '
510 L$='''&K$&''' @ READ #1,4;R$ @ R$=FNS$(R$,':') @ IF
   R$=''' THEN 620
520 IF NOT POS(R$,'TIME') THEN 560
530 L$="C1=C1/3600@T9=FP(C1)*60@T9=INT(T9)/100+FP(T9)
   *.006+INT(C1)'' @ GOSUB 'WRITE'
540 L$="T$=STR$(FP(T9))@T$=STR$(INT(T9))&':'&T$[2,3]
   &':'&T$[4,5]'' @ GOSUB 'WRITE'
550 L$="''&K$&''&T$&' ''
560 IF POS(R$,'DATE') THEN L$=L$&''&''&K$&'';''&DATE$&' '''"
570 IF POS(R$,'FILENA') THEN L$=L$&"'"&K$&"file: "
   &F8$&""
580 IF POS(R$,'RECORD') THEN L$=L$&"'"&K$&"record#';
   K5; '"
590 L$=L$[1,LEN(L$)-1]&K$[1,1] @
   L$=""&CHR$(14)&"''&L$&CHR$(15)&""
600 L$="PRINT TAB("&STR$(INT((P+9-LEN(L$))/2))&")";&L$ @
   GOSUB 'WRITE'
610 GOSUB 'PLINE'
620 FOR K=1 TO K3 @ IF ABS(V8(K))#INF THEN
   Q2=Q2+ABS(INT(V8(K)))+2
630 IF FP(V8(K)) THEN Q2=Q2+ABS(FP(V8(K))*10)+1
640 NEXT K @ IF NOT Q2 THEN 1110
650 Q1=CEIL(Q2/P) @ DESTROY V9 @ INTEGER V9(Q1) @ K1=1 @
   Q=0
660 FOR K=1 TO Q1
670 Q2=ABS(V8(K1)) @ IF Q2=INF THEN 700
680 Q=Q+INT(Q2)+2 @ IF FP(Q2) THEN Q=Q+1+FP(Q2)*10
690 IF Q<P THEN V9(K)=V9(K)+1 ELSE Q=0 @ GOTO 710
700 K1=K1+1 @ IF K1>K3 THEN 720 ELSE 670
710 NEXT K
720 Q2=0
730 K1=1 @ FOR K=1 TO Q1
740 Q3=0 @ L$=' ' @ FOR K2=1 TO V9(K)
750 Q2=ABS(V8(K1)) @ K1=K1+1 @ IF Q2=INF THEN 750
760 IF INT(Q2) THEN L$=L$&STR$(INT(Q2))&'D' @ Q3=Q3+INT(Q2)
770 IF FP(Q2) THEN L=FP(Q2) @ L$=L$&STR$(L)&'D' @
     Q3=Q3+L*10+1
780 Q3=Q3+2
790 L$=L$&',2X,' 
800 NEXT K2
810 L=K+5499 @ L$=L$[1,LEN(L$)-4]
820 K2=POS(L$,'1') @ IF NOT K THEN 850
830 G$=L$[K2+1,K2+1] @ IF G$#'D' OR G$#',' THEN 850
840 L$=L$[1,K2-1]&L$[K2+1] @ GOTO 820
850 L$='IMAGE'&STR$(INT((P-Q3)/2))&'X,'&L$ @ GOSUB 'WRITE'
860 NEXT K
870 L=6000 @ K2=0 @ FOR K=1 TO Q1
880 R$='PRINT USING'&STR$(K+5499)&';' @ L$="" @ L=L+10
890 Q3=0 @ FOR K1=1 TO V9(K)
900 K2=K2+1 @ IF K2>K3 THEN 1010 ELSE Q2=ABS(V8(K2))
910 IF Q2=INF THEN 900 ELSE Q4=1
920 IF INT(Q2) THEN Q4=Q4+INT(Q2)
930 IF FP(Q2) THEN Q4=Q4+FP(Q2)*10+1
940 G$=V2$(K2) @ Q3=Q3+Q4+1
950 IF LEN(G$)>Q4-1 THEN 970
960 G$=' '&G$ @ GOTO 950
970 L$=L$&G$[1,Q4-1]&' '
980 IF LEN(L$)>65 AND K1#V9(K) THEN L$="PRINT'"&L$&'";" @
    GOSUB 'WRITE' @ L$=""
990  R$=R$&V3$(K2)&','
1000  NEXT  K1
1010  Q3=INT((P-Q3)/2)
1020  L$="PRINT'"&L$&"''" @ R1$=L$
1030  L=L-3 @ L$="PRINT TAB("&STR$(Q3+1)&");CHR$(27)&'&dD';"
1040  GOSUB 'WRITE' @ L=L+5 @ L$=R1$ @ GOSUB 'WRITE'
1050  L$="PRINT CHR$(27)&'&d@';" @ GOSUB 'WRITE' @ L$=R$ @
      GOSUB 'WRIT'
1060  NEXT  K
1070  GOSUB 'PLINE' @ L$='PRINT' @ GOSUB 'WRITE' @ GOTO 1100
1080  Q3=INT((P-Q3)/2) @ IF  Q3>1  THEN
      L$="TAB("&STR$(Q3)&");''&L$ ELSE L$='''&L$
1090  L$='PRINT'&L$ @ GOSUB 'WRITE' @ L$=' ' @ RETURN
1100  L$='BYE' @ L=7000 @ GOSUB 'WRITE'
1110  BEEP 1200,.1 @ DISP 'copying to disk...' @ ASSIGN #2
       TO *.
1120  ON ERROR GOTO 1130 @ RENAME F9$&':%16' TO F9$&'LS' @
       GOTO 1140
1130  OFF ERROR @ IF ERRN=255022 THEN 1140 ELSE PURGE
       F9$&'LS:%16' @ GOTO 1120
1140  DESTROY V1$ @ TRANSFORM F9$ INTO BASIC :%16
1150  PURGE F9$ @ BEEP 1990,.1 @ DISP 'translation done...'
       @ BEEP 1300,.1
1160  RESET @ RUN ADAPT,'S'
1170  !
1180 'PLINE':
1190 L$="PRINTUSING"&STR$(P)&""&CHR$(252)&"" & "
1200 'WRIT': ! Remove right-most chr. and WRITE
1210 L$=L$[1,LEN(L$)-1]
1220 'WRITE': IF L$=' ' THEN RETURN
1230 IF L=L1 THEN L=L+2
1240 BEEP 200,.01 @ FOR Q=FILESZR(F9$)-1 TO 0 STEP -1
1250 READ #2,Q;G$ @ IF VAL(G$)<L THEN 1270
1260 NEXT Q
1270 INSERT #2,Q+1;STR$(L+10000)[2]&L$ @ L1=L @ RETURN
1280 !
1290 'EXCHANGE': FOR K2=INT(A(K1)) TO FP(A(K1))*1000
1300 READ #1,RES;R$
1310 IF POS(R$,'=') THEN 1390 ELSE Q1=POS(R$,'<')
1320 IF NOT Q1 THEN Q1=POS(R$,'>') @ R$=R$[Q1+1] @ CALL TRIM(R$)
1330 DATA 'WAIT','CALL','DEGREES','RADIANS','BEEP'
1340 RESTORE 1330
1350 FOR Q2=1 TO 5
1360 READ F3$ @ IF POS(UPRC$(R$),F3$) THEN SFLAG 3 @ GOTO 1390
1370 NEXT Q2
1380 R$="OUTPUTD1;"&R$&"" @ GOTO 1530
1390 IF POS(R$,';') THEN R$=FNT$(R$,';')
1400 IF K1=1 THEN R$=FNS$(R$,'<')
1410 IF K1=3 THEN R$=FNS$(R$,'>')
1420 Q1=POS(R$,'"') @ IF NOT Q1 THEN 1460 ELSE
    Q2=POS(R$,'"',Q1+1)
1430 R1$=R$[Q1+1,Q2-1] @ K=1
1440 IF V2$(K)#R1$ THEN K=K+1 @ GOTO 1440
1450 R$=R$[1,Q1-1]&V3$(K)&R$[Q2+1,LEN(R$)] @ GOTO 1420
1460 Q1=POS(R$,'"') @ IF NOT Q1 THEN 1510 ELSE
    Q2=POS(R$,'"',Q1+1)
1470 R1$=R$[Q1+1,Q2-1] @ K=1 @ IF LEN(R1$)=4 THEN
    R1$=R1$[1,3]"0'&R1$[4]
1480 L$=V5$(K)[2,6]
1490 IF L$#UPRC$(R1$) THEN K=K+1 @ GOTO 1480
1500 R$=R$[1,Q1-1]&V6$(K)&R$[Q2+1,LEN(R$)] @ GOTO 1460
1510 IF K1#2 OR NOT T2 OR FLAG(3,0) THEN 1530 ELSE
    Q1=POS(R$,'=') @ R1$=R$[1,Q1-1]
1520 R$=R1$="('&R$[Q1+1]+'&')"&R1$
1530 L$=R$ @ GOSUB 'WRITE'
1540 NEXT K2
1550 RETURN
1560 !
1570 'LIST:
1580 K$=CHR$(K1+176) @ K=1
1590 IF V5$(K)[1,1]#K$ THEN K=K+1 ELSE 1610
1600 IF K>K4 THEN RETURN ELSE 1590
1610 R$="OUTPUTD1;" @ F3$="ENTERD1;"
1620  K2=VAL(V5$(K)[5])  @  LS=R$&V5$(K)[2,6]&','  @
       RL$=F3$&V6$(K)&'  @  K=K+1 @ Q=1
1630  IF  K>K4 THEN 1710
1640  KL=VAL(V5$(K)[5])  @  IF  POS(L$,'REF') THEN 1700
1650  IF  V5$(K)[1,1]#K$ THEN 1710
1660  KL=VAL(V5$(K)[5])  @  IF  NOT  POS(L$,V5$(K)[2,4]) THEN
                   1700
1670  IF  KL-1=K2 THEN  LS=LS[1,LEN(L$)-Q]&'-'  @  Q=4 ELSE  Q=1
1680  LS=LS&V5$(K)[5]&','
1690  K2=KL @ RL$=RL$&V6$(K)&','  @  IF  K=K4 THEN 1710 ELSE
                   K=K+1 @ GOTO 1640
1700  GOSUB 1720 @ GOSUB 'WRIT' @ GOTO 1620
1710  GOSUB 1720 @ GOTO 'WRIT'
1720  LS=LS[1,LEN(L$)-1]&""'
1730  GOSUB 'WRITE'
1740  LS=RL$ @ RETURN
1750  END SUB
1760  !
1770  SUB  TRIM(L$) @ CALL  LTRIM(L$) @ CALL  RTRIM(L$) @ END
       SUB
1780  SUB  LTRIM(L$) ! Remove left-hand spaces
1790  IF  LS[1,1]='' THEN  LS=LS[2] @ GOTO 1790
1800  END SUB
1810  !
1820  SUB  RTRIM(G$)
1830  K5=LEN(G$)  @ IF G$[K5,K5]='' THEN G$=G$[1,K5-1]  @
       GOTO 1830
1840  END SUB
APPENDIX G

PARSE COMPUTER PROGRAM LISTING

30 SUB PARSE(R$,V1$)
40 DIM Z0$[2],R5$[28],R1$[96],K1$[8],K2$[8],R2$[28]
50 Z0$=PEEK$("256E9",2) @ K1=1 @ K3=0 @ CFLAG 3 @ K7=0
60 K=POS(R$,'=',K7+1) @ IF NOT K THEN 90 ELSE K7=K
70 K2=POS(R$,','",K1) @ IF K2 AND K2<K THEN K3=K3+1 @
  K1=K2+1 @ GOTO 70
80 IF MOD(K3,2) THEN 60 ELSE SFLAG 3
90 IF NOT FLAG(3) THEN 110
100 K=POS(R$,';') @ IF K THEN R5$=R$[K] @ R$=R$[1,K-1]
110 DATA "","","","'",',',','],''
120 RESTORE 110 @ CFLAG 13
130 FOR K=1 TO 3
140 READ K1$,K2$
150 K3=1 @ K2=0 @ K1=0
160 K5=POS(R$,K1$,K3) @ IF K5>0 THEN K1=K1+1 @ K3=K5+1 ELSE
  SFLAG 8
170 K5=POS(R$,K2$,K3) @ IF K5>0 THEN K2=K2+1 @ K3=K5+1 ELSE
  SFLAG 9
180 IF NOT FLAG(9,0) OR NOT FLAG(8,0) THEN 160
190 IF K1#K2 THEN 220

87
200 NEXT K
210 GOTO 230
220 DISP MSG$(47017);';'K1$;K2$ @ BEEP 300,1 @ SFLAG 7 @
      GOTO 890
230 CALL LTRIM(R$)
240 R2$=R$[1,1]
250 K1=POS('<>!',R2$) @ IF NOT K1 AND NOT FLAG(5) AND NOT
      FLAG(6) THEN 330
260 IF K1=1 AND FLAG(5) THEN 330
270 IF K1=2 AND FLAG(6) THEN 330
280 IF K1=1 AND FLAG(5) THEN GOTO 330
290 IF K1=2 AND NOT FLAG(5) AND NOT FLAG(6) THEN SFLAG 6 @
      GOTO 330
300 IF K1=3 AND FLAG(5) THEN GOSUB 'R' @ CALL ERROR(10) @
      GOTO 890
310 IF K1=2 AND FLAG(5) THEN GOSUB 'R' @ CALL ERROR(7) @
      GOTO 890
320 IF K1<2 AND FLAG(6) THEN GOSUB 'R' @ CALL ERROR(9) @
      GOTO 890
330 IF NOT K1 THEN CFLAG 5
340 IF K2=2 THEN SFLAG 6 @ CFLAG 5
350 IF I THEN 460
360 R$=UPRC$(R$)
370 DATA "BEEP ","DEGREES","RADIANS","CALL ","WAIT ","IF 
380 RESTORE 370
89

390 FOR K=1 TO 6
400 READ R2$ IF POS(R$,R2$) THEN SFLAG 13 @ GOTO 460
410 NEXT K @ IF FLAG(3) THEN 460
420 RL$=';'&R$ @ K1=1
430 FOR K=1 TO 99 @ K2=POS(Rl$,";",K1) @ IF NOT K2 THEN 890
   ELSE K1=K2+1
440 IF NOT POS('CLPCLSOPNTOTWRT',RL$[K1,K1+2]) THEN GOSUB 'R' @ CALL ERROR(26) @ GOTO 890
450 NEXT K
460 ASSIGN #4 TO TEST @ RL$=' '&R$&' '
470 K2=POS(Rl$ "") @ IF NOT K2 THEN 580
480 IF K2<1 THEN GOSUB 'R' @ CALL ERROR(5) @ GOTO 890
490 K3=POS(R1$,""",K2+1)
500 IF K3-K2>6 THEN GOSUB 'R' @ CALL ERROR(26) @ GOTO 890
510 R2$=RL$[K2+1,K2+5]
520 IF POS('ACVBITDCVFRQFWOREDREFTEMTWO',UPRC$(R2$[1,3])) THEN 540
530 GOSUB 'R' @ CALL ERROR(26) @ GOTO 890
540 OUTPUT :83 ;R2$&';OPN'
550 IF BIT(SPOLL('%83'),5) THEN GOSUB 'R' @ CALL ERROR(26) @ GOTO 890
560 IF VAL(R1$[K3-2,K3])>30 THEN GOSUB 'R' @ CALL ERROR(19) @ RETURN
570 RL$=RL$[1,K2-1]& ' '&RL$[K3+1] @ GOTO 470
580 K2=POS(R1$,"") @ IF NOT K2 THEN 660 ELSE
K3 = POS(R1$, '"', K2+2)

590 IF K2<POS(R1$, '=') THEN CFLAG 12 ELSE SFLAG 12
600 R2$=""&R1$[K2+1, K3-1]&"

610 IF FLAG(12) THEN 630 ELSE IF POS(V1$, R2$) THEN CALL ERROR(15) @ GOTO 890
620 GOTO 640
630 IF NOT POS(V1$, R2$) THEN GOSUB 'RM' @ CALL ERROR(16) @ GOTO 890
640 IF NOT FLAG(12) THEN V1$=V1$&R2$[2]
650 R1$=R1$[1, K2-1]&' B '&R1$[K3+1] @ GOTO 580
660 R2$=R1$[1, 3] @ K2=POS(R2$, '<') @ K2=MAX(POS(R2$, '>'), K2)
670 IF K2 THEN R1$=R1$[K2+1]
680 K2=1 @ R2$=R5$
690 GOTO 780
700 IF NOT POS(R2$, 'P') THEN 750
710 IF POS( PRINT;', R2$[K2,K2]) THEN K2=K2+1 @ GOTO 710
720 Q1=VAL(R2$[K2]) @ IF NOT Q1 THEN GOSUB 'RM' @ CALL ERROR(6) @ GOTO 890
730 IF FP(Q1*10) THEN GOSUB 'RM' @ CALL ERROR(11) @ GOTO 890
740 GOTO 690
750 IF NOT POS(R2$, ';') THEN 780
760 IF NOT POS(R2$[K2], 'S') THEN GOSUB 'RM' @ CALL ERROR(18) @ GOTO 890
770 K2=K2+1 @ GOTO 690
780 R1$='0001'&R1$ @ RESTORE #4,0 @ PRINT #4;R1$ @ ASSIGN #4 TO *
790 ON ERROR GOTO 820
800 SFLAG -1 @ TRANSFORM TEST INTO BASIC T @ OFF ERROR
810 PURGE T @ CFLAG -1 @ GOTO 890
820 OFF ERROR @ PURGE T @ GOSUB 'RM' @ CFLAG -1 @ CALL
     ERROR(20) @ GOTO 890
830 CALL YESNO('print '&F1$,K)
840 IF K THEN CALL PSETUP(F1$) @ GOTO 890
850 'RM': IF NOT FLAG(3) THEN 880
860 FOR K=LEN(V1$)-1 TO 1 STEP -1 @ IF V1$[K,K]="" THEN
     V1$=V1$[1,K] @ GOTO 880
870 NEXT K
880 'R': POKE '2F6E9',Z0$ @ RETURN
890 END SUB
REFERENCES


Katz, Robert, 1982. The Hewlett-Packard Interface -- HPIL. Byte. 7:76-92


