Some pages of this thesis may have been removed for copyright restrictions.

If you have discovered material in Aston Research Explorer which is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please read our Takedown policy and contact the service immediately (openaccess@aston.ac.uk)

THE DESIGN AND IMPLEMENTATION OF A PORTABLE OPERATING SYSTEM FOR MICROCOMPUTERS

H. T. ESENDAL

Submitted for the Degree of
Doctor of Philosophy

University of Aston in Birmingham

April 1981

A B S T R A C T

THE DESIGN AND IMPLEMENTATION OF
A PORTABLE OPERATING SYSTEM
FOR MICROCOMPUTERS

by

Huseyin Tugrul Esendal

For the degree of Doctor of Philosophy

University of Aston in Birmingham

April 1981

Communication and portability are the two main problems facing the user. An operating system, called PORTOS, was developed to solve these problems for users on dedicated microcomputer systems.

Firstly, an interface language was defined, according to the anticipated requirements and behaviour of its potential users. Secondly, the PORTOS operating system was developed as a processor for this language.

The system is currently running on two minicomputers of highly different architectures. PORTOS achieves its portability through its high-level design, and implementation in CORAL66.

The interface language consists of a set of user commands and system responses. Although only a subset has been implemented, owing to time and manpower constraints, promising results were achieved regarding the usability of the language, and its portability.

KEYWORDS

Operating Systems
Portability
High-level Languages
Microcomputers

C O N T E N T S

Chapter	Page
1 INTRODUCTION	1
1.1 Why Microcomputers?	1
1.2 Problems Encountered by Users	2
1.2.1 Communication	2
1.2.2 Portability	2
1.3 The PORTOS Approach	4
1.3.1 Interface Language Design	5
1.3.2 Operating System Design	6
1.4 Presentation	6

PART I : DESIGN

2 DESIGN CONSIDERATIONS	7
2.1 The Starting Point	7
2.1.1 User Characteristics and Requirements	7
2.1.2 Applications To Be Supported	9
2.1.3 Target Hardware Configuration	10
2.2 PORTOS Virtual Computer	10
2.2.1 Single-language and Multi-language Systems	10
2.2.2 Files and File References	12
2.2.3 System Configuration	13
2.3 PORTOS Interface Language (PIL)	14
2.3.1 User Activities and PCL Commands	16
2.3.2 PCL Syntax	17
2.3.3 Command Names	19

2.3.4 System Responses 20

2.3.5 PRL Syntax 21

2.4 Overview of System Usage 22

2.4.1 System Startup 22

2.4.2 Communications Environment Hierarchy 23

2.5 PIL Processor 25

2.5.1 Compilers and Interpreters 25

2.5.2 Processor Facilities 26

2.5.3 Processor Portability 27

3 PORTOS INTERFACE LANGUAGE 30

3.1 PCL Commands and PRL Responses 30

3.2 File Handling Commands 30

3.2.1 CREATE Command 30

3.2.2 DELETE Command 32

3.2.3 RENAME Command 32

3.2.4 COPY Command 33

3.2.5 LOCK Command 33

3.2.6 UNLOCK Command 33

3.2.7 LIST Command 34

3.2.8 PRINT Command 34

3.3 Software Development Commands 35

3.3.1 EDIT Command 35

3.3.2 Language Processing Commands 37

3.3.3 LINK Command 38

3.3.4 RUN Command 39

3.3.5 DEBUG Command 39

Chapter　　　　　　　　　　　　　　　　　　　　　　　　Page

5 SYSTEM DESIGN ONTO PRIMITIVES 71

5.1 Memory Residency 71

5.2 System Tasks 72

5.3 Memory Organization 73

5.4 Task Management 74

5.5 Logical Portability of PORTOS 77

PART II : IMPLEMENTATION

6 IMPLEMENTATION CONSIDERATIONS 79

6.1 Choosing The Implementation Language 79

6.1.1 Systems Programming Languages 80

6.1.2 Modified Languages 81

6.1.3 Existing Applications Languages 82

6.1.4 CORAL As A Systems Programming Language 83

6.2 Host Computers 85

6.2.1 TI 990/10 Computer System 86

6.2.2 PRIME 300 Computer System 88

6.3 Implementation Subset 90

7 THE EARLY EXPERIMENTS 92

7.1 Foundations of PORTOS 92

7.2 PORTOS On The ICL-1904S 93

7.2.1 Implementation Methodology 94

7.2.2 Simulator Programs 95

7.2.3 PORTOS Routines 96

7.2.4 Validator Programs 98

7.3 Transfer To The TI 990/10 99

8 PORTOS IMPLEMENTATION AND PORTABILITY 101

8.1 PORTOS on the TI 990/10 101

8.2 Transfer of PORTOS to the PRIME 300 102

8.2.1 Source Code Modifications 103

8.2.1.1 Machine-dependent Segments 103

8.2.1.2 Machine-orientated Segments 104

8.2.1.3 Segment Parameterization 105

8.2.1.4 Compiler Requirements 106

8.2.2 Compilation and Disk Creation 107

8.3 A Comparison of the Two Implementations 110

8.4 Portability of the System 111

8.4.1 Relative and Absolute Portability 112

PART III : RESULTS

9 DISCUSSION AND CONCLUSIONS 114

9.1 Implementation Restrictions 114

9.1.1 Portability of the Primitives 115

9.1.2 Subset Orientation 117

9.2 Implementation on Atypical Configurations 118

9.3 Reflections on CORAL 119

9.4 Future Developments 120

Page

APPENDICES

Appendix A : PORTOS Command Language Syntax 122

Appendix B : PORTOS Response Language Syntax 123/

Appendix C : PCL Commands 124

Appendix D : CORAL66 Compilers 127

Appendix E : I) TI 990/10 Memory Organization 128

II) PRIME 300 Memory Organization 129

Appendix F : CORAL Keywords on Different Compilers 130

Appendix G : PORTOS Macro Definition Files 132

Appendix H : PIL Interface Source Listings 136

Appendix I : PORTOS Kernel Tasks Source Listings 161

Appendix J : PORTOS Kernel Routines Source Listings 173

Appendix K : PORTOS Hardware Interface Source Listings . . . 196

Appendix L : PORTOS Disk Build Program (PRIME 300) 212

Appendix M : PORTOS Supplementary Software (PRIME 300) . . . 215

Appendix N : PORTOS Supplementary Software (TI 990/10) . . . 217

REFERENCES 219

LIST OF FIGURES

Figure		Page
1.1	The PORTOS Development	4
2.1	PVC Files	12
2.2	Default Suffix Names	13
4.1	PORTOS Filing System	53
4.2	Directory Entry Format	54
4.3	Sector Format	57
4.4	PORTOS Channel Allocation For I-O	58
4.5	Error Codes in OPEN Primitive	59
4.6	PORTOS I-O Table Format	60
4.7	Free Partitions List	64
4.8	Flow of Control in PORTOS	66
5.1	PORTOS Tasks	72
5.2	PORTOS Memory Organization	74
6.1	Modified Languages for Systems Programming	82
6.2	TI 990/10 I-O Instructions	87
6.3	PRIME 300 I-O Instructions	89
8.1	PORTOS Disk Sector Allocation	108
8.2	Order of Residency on Disk	109
8.3	Implementation Figures	110

ACKNOWLEDGEMENTS

The author would like to express his gratitude to the following people:

Mr. I. H. Gould, B.Sc., M.Phil., FBCS, his supervisor, for his guidance, help, and patience throughout the duration of this research project.

Mr. K. J. Bowcock, B.Sc., FIMA, FBCS, Head of Department, Computer Centre, for his help in overcoming the managerial problems

Mr. W. R. Davy, M.A., M.Sc., LBCS, formerly of the Department of Electrical Engineering, for his help with the implementation of PORTOS on the Texas 990/10

Dr. M. J. Walker, M.Sc., Ph.D., for his help with the implementation of PORTOS on the PRIME 300

Mr. J. B. Lowe, M.Sc., FBCO, for his company, encouragement, and help when it was most needed

Mr. N. Toye, Laboratory Officer, for his help with the numerous problems that arose during the implementation on the PRIME 300 and his company in the laboratory

Mr. G. P. Gerrard, Operations and Services Manager, Loughborough University of Technology, Computer Centre, for his help with the practical problems of system compilation on the PRIME 300

CHAPTER 1 : INTRODUCTION

This thesis presents the design and implementation of a portable interactive operating system called PORTOS, developed for use on microprocessor-based single-user computer systems (henceforward referred to as microcomputers). The objectives of PORTOS are to facilitate user communication with the computer and to provide for software compatibility on different machines.

1.1) Why Microcomputers?

Microcomputers are cheap to acquire, maintain, and run. Unlike mainframe computers, they can be used when and where needed without being prohibited by costs. They are also robust and highly reliable. The environmental constraints that govern the operation of large computers have little or no effect on microcomputers.

This low-cost availability and versatility has brought considerable computing power to within reach of a large number of users, both organizations and individuals, for whom it was previously infeasible. There are, however, two problems facing these users: communication and software portability.

1.2) Problems Encountered by Users

1.2.1) Communication

Communication means the user's interaction with the computer for control functions that lie outside the domain of his programming language. It takes the form of a command-and-response dialogue. The user types in his command on the keyboard and waits for the response on the display unit, indicating either success or failure, so that he can decide on his next move.

This dialogue makes the user an active participant in the computing process, the outcome of which depends on the availability of a suitable interface language. Suitability is a measure of how well this language fulfils the user's needs and responds to his skills.

Many interface languages frequently display the characteristics of being an afterthought to operating system design [5, 79]. Their awkward syntax, unnatural constructs, and poor semantics render them needlessly cumbersome, unhelpful, and error-prone. As a result, these languages fail to reflect the user's "closeness" to the computer and establish the required rapport between the two.

1.2.2) Portability

Portability is the ease with which programs can be transferred from one computer system to another [86]. It is an attractive software characteristic because users who are committed to any one computer are getting fewer. They are often forced into relocation for personal

gains; or the rapid advances in microprocessor technology render their computers obsolete; or they can afford to upgrade their computing power at shorter intervals.

Whatever the reasons may be, economic considerations make it desirable to transfer those long-life programs in which time and effort have been invested. Problems arise if the new computer is from a different manufacturer, which, considering the wide choice of alternatives currently available, is very likely. It then becomes necessary to adapt these programs to their new environment.

Re-programming, which is a wasteful and discouraging task, is minimized by the use of machine-independent languages. They hide the underlying architecture behind compilers, and thus enable users to develop portable software that suffers the least disruption after the transfer. The portability achieved at program level, however, is offset by the problems encountered at user level.

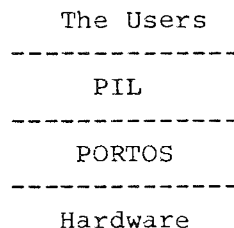
Moving to a different computer invariably means having to learn a new set of commands and conventions, because every manufacturer implements its own particular interface language. The differences manifest themselves in the actual command repertoire, syntax, capabilities, and response to users. The re-learning and familiarization necessitated by this lack of uniformity naturally results in a considerable waste of time and effort. Interface language incompatibility thus emerges as the main obstacle to portability.

1.3) The PORTOS Approach

PORTOS offers a solution to these problems, firstly, by providing an interface language that emphasizes usability, and secondly, by making this language available on different computers.

The PORTOS development consists of the four major stages shown in Figure 1.1, where the function of each subsequent stage is simply to implement the previous one.

Figure 1.1 : The PORTOS Development



The primary objective is to enable the user to express his intentions in his own terms. The starting point, therefore, is the specification of the personal and computational requirements of potential users.

PORTOS Interface Language, or PIL for short, defines a virtual computer to satisfy these requirements, and allows the user access to it. The constructs and facilities of the language determine the manner of communication and the user's capabilities on it [28]. This virtual computer is then implemented and supported by the PORTOS operating system, which is itself implemented on the host hardware in CORAL66.

1.3.1) Interface Language Design

Proposals to aid the development of user-orientated interface languages include the transformation of system design from an art form into an engineering discipline [6]; the imitation of person-to-person communication [56]; the flexibility of the user interface so that differential responses can be made to a variety of users [92]; and the parameterization of user-computer interaction [58].

The current lack of sufficient data in this field makes it premature to formulate a definite approach. A user-orientated language, by implication, must be based on the user's requirements. Language design, on the other hand, still remains very much a matter of experience and personal preferences.

A dialogue between the designers and users is therefore required for a successful design [48]. The main problem here is that, while feedback and iteration are crucial to the subsequent improvement of the language, the two parties are not necessarily in contact during the initial stages.

For this reason, the design of PIL is based on prediction and integration. The methodology is to predict the user's requirements, and then to integrate them with experience and knowledge in order to define a functionally and structurally suitable language. This approach is similar to that employed by the UNIQUE command language [64], in which a set of properties that an ideal language should have are defined and then implemented in user terms, within the bounds of practicality.

1.3.2) Operating System Design

PORTOS is designed as a processor for PIL [79]. Its facilities are orientated towards the requirements of the PIL virtual computer.

The design approach is a combination of different techniques [89]. Its general structure is top-down, starting with PIL and going down to the hardware. The kernel, which constitutes the bulk of the system, is designed in terms of modules and interfaces. Within each module, however, the approach is essentially bottom-up. The basic functions are designed first, to support the development of the higher levels.

PORTOS is made portable, firstly, by design, and secondly, by implementation in CORAL66. Processor portability in turn makes PIL portable, which then removes the user's need to learn new interface languages and modify programs; this being the second objective of PORTOS.

1.4) Presentation

The contents of the thesis are divided into three parts. Part I covers the design phase of the development while part II covers the implementation. Part III contains the discussion of the results and an appraisal of the approach.

PART I : DESIGN

CHAPTER 2 : DESIGN CONSIDERATIONS

2.1) The Starting Point

2.1.1) User Characteristics and Requirements

PORTOS users are envisaged to be non-computer specialists for whom the computer is simply a convenient tool with which to solve non-computing problems as quickly and painlessly as possible.

The focus of attention for these "casual" users is their particular problem and not the tools at hand. They need or prefer the computer solution but it still constitutes a small part of their work. They have a limited knowledge of computer science which they do not seek to enhance, because it is sufficient for this limited and subjective usage.

The typical PORTOS user, therefore, requires only the major control functions for his computing activities. Consequently, he has neither the reason nor the inclination to tackle the intricacies of a complex interface language. He seeks ease of learning and use, so as to minimize his involvement in all extraneous issues and concentrate on his problem.

The computer is judged by the user purely on the grounds of how well it serves his needs, based on three considerations. Firstly, the facilities at his disposal for problem solving; secondly, the means by which he can control these facilities; and thirdly, the manner and quality of computer response to his input.

The role played by this last consideration is generally over-shadowed by the obvious importance of the first two. It should not be neglected, however, because the user's attitute to computing is greatly influenced by it. Firstly, all error messages should be meaningful and self-explanatory. They should convey sufficient information for the user to remedy the situation easily and quickly. Messages that are difficult to understand or relate to the operation being performed reduce usability and alienate the user.

Secondly, response time is a major contributor to satisfactory interaction. The user should be provided with suitable monitoring messages when performing time-consuming operations such as compilation. These messages are necessary to dispel the feelings of annoyance and anxiety that arise when a computer shows no apparent signs of activity for more than a few seconds after a command has been issued. The aim is to keep the user occupied, as well as to impart relevant information, while his command is being processed.

The specific requirements of PORTOS users, and how they are satisfied, are discussed in the relevant sections below.

2.1.2) Applications To Be Supported

Microcomputers are highly suitable for a vast range of applications, which can be divided into three groups: network, process control, and stand-alone [27, 34, 71].

Network and process control applications are problem orientated and the computer is run in a dedicated environment for a specific purpose. Stand-alone applications, on the other hand, are user orientated. The computer is employed as a smaller version of the general-purpose mainframe computer for such functions as small business data processing, scientific problem solving, and hobby computing. It is these applications which are of interest to PORTOS users.

A stand-alone microcomputer is totally dedicated to the user's particular problem but only for as long as he requires it. The computer may be subsequently used with the same degree of dedication to solve a completely different problem. This orientation towards the user's personal convenience and advantage has given rise to the term "personal computer" to describe these microcomputers [45].

Personal computers are expected in the near future to replace the typewriter and the filing cabinet in offices, and to perform the necessary acquisition, storage, and presentation of information [24]. They are not suitable for large data base and information retrieval applications, however, owing to the limited capacity of their mass storage devices.

2.1.3) Target Hardware Configuration

Small physical size is a distinguishing characteristic of personal computers. A typical configuration consists of a CPU (a microprocessor with random-access memory), a Visual Display Unit (VDU) for user communication, a dual-drive diskette (or floppy disk) unit for mass storage, and a hardcopy printer [3]. This configuration is taken as the target.

2.2) PORTOS Virtual Computer

The next step is to find a computer that will satisfy the above requirements. Since there is no real computer that can achieve this, a virtual computer which is specifically designed for the purpose is needed. The PORTOS Virtual Computer (PVC) is such a system. It is a single-user, file-orientated interactive computer, the details of which are presented below.

2.2.1) Single-language and Multi-language Systems

The PVC is a multi-language computer system, like the Motorola EXORciser [54], Texas Intruments TX990 [88], and North Star Horizon [60]. On these systems, programming and operating system control functions are kept separate in their respective languages. Advantages are functional suitability, improved computer control through physical separation of logically separate functions, and independent development through modularity.

Those systems that combine programming and control functions in the same language are called single-language systems. Their control commands may be an integral part of the language repertoire, as in the Commodore PET [21], Apple [1], and Hewlett-Packard HP2000 [40] systems using BASIC. Alternatively, they may take the form of procedure calls, as in SOLO using Concurrent PASCAL [38] and OS6 using BCPL [83].

The single-language approach is particularly attractive to the non-computer scientist, because both his algorithm and system problems can be solved from within the same language. Theoretically speaking, any programming language can be extended to include the necessary control functions for use in similar environments [39, 50]. Wiederhold, for example, reports on such an implementation using PL/ACME and LISP [96].

The main disadvantage of single-language systems is that they confine their users to one language for all their needs, regardless of suitability. Furthermore, programming and system control activities differ in purpose, data types, and scope rules, which makes the separation of languages more meaningful. For a more comprehensive discussion on this subject, the reader is referred to the Proceedings of the IFIP Working Conference on Command Languages, 1975, North Holland, where the final verdict is in favour of the multi-language approach [39].

2.2.2) Files and File References

The PVC has an external and static filing system. All files are accessible at any time and they remain in existence until explicitly deleted by the user. The types of files recognized by the system are listed in Figure 2.1.

Figure 2.1 : PVC Files

Type of File	Contents
Character	ASCII Information
Object	Semi-compiled Binary (assembler/compiler output)
Binary	Executable Binary

Each file has a user-defined unique "file reference" by which it is identified. A complete file reference consists of a filename and a suffix. The use of the suffix is optional. It allows files with the same name but different characteristics or contents to exist in the system.

A filename can be up to six characters long. Any ASCII character except comma and period may appear in it, in any order. The suffix name, if used, is two characters long. The first character is always a period, which also separates it from the filename. The second character can be any ASCII character except comma.

If the suffix name is omitted in a file reference, then a default name is assumed depending on the operation being performed. These

default names, which are listed in Figure 2.2, are aimed at simplifying the management of files during software development. The user should refrain from using these suffix names outside their assigned domains in order to preserve consistency of usage and avoid clashes between file references and contents.

Figure 2.2 : Default Suffix Names

Suffix	Meaning
.A	Assembler Source
.B	Executable Binary
.L	Listing (compiler)
.M	Listing (assembler)
.O	Object Code
.S	Compiler Source

2.2.3) System Configuration

All information is contained in files, and all files are kept on diskette. PVC differentiates between diskettes as either system or user, similar to the Olivetti P6060 system [67]. Commands are interpreted by programs on system diskette to manipulate the information on user diskette.

Drives 0 and 1 of the diskette unit are dedicated to system and user diskettes respectively. Drive 0, which is called SYSTEM, is inaccessible to the user except through special commands, thus offering the system programs additional protection. Drive 1, called USER, which is also the default drive, is always directly accessible.

The alternative is for the system to share diskettes with the user. Sharing, however, leads to duplication of information. The system, or at least a minimum self-supporting subset of it, must exist on every diskette, thus leaving the user (or the system) a limited amount of space for programs and data.

Limited space means more diskettes in use during a session. This in turn means a higher rate of diskette handling by the user, reduced processing, and loss of usability. The objective behind separation is to minimize these undesirable characteristics.

2.3) PORTOS Interface Language (PIL)

The PVC is defined to the user by PIL. The definition includes establishing the interface through which the user can communicate with this computer and employ the facilities provided on it.

Communication involves a two-way flow of information. In order to define a suitable interface, therefore, PIL distinguishes between the input to and output from the computer, as command and response respectively. This distinction is reflected in its organization. PIL consists of two languages: (1) PORTOS Command Language (PCL) to handle the user input to the computer and (2) PORTOS Response Language (PRL) to handle the computer output to the user.

Simplicity and responsiveness are the key factors in their design. The simplicity of PCL is realized in terms of a low-level unambiguous syntax within a high-level command structure, with meaningful names and default settings. The responsiveness of PRL is characterized by quick and informative replies to the user in English as fully as possible, without becoming prohibitive in length. Their details are presented in Chapter 3.

It has been argued [56] that commands represent a very limited subset of human expressive powers (about 5%); that command languages are fundamentally alien to man; and that making them easier would not help the casual user. On the other hand, experience has shown [6] that a custom-built vocabulary of verb-like commands not only creates a conceptual framework that is easy to learn and use, but also aids processability and memory usage.

PIL employs a simple command structure, supplemented by a corresponding set of responses, to provide a friendly interface. Familiar words and characters are used consistently throughout the language, in order to maximize usability and compensate for user inexperience. It is, however, a single-purpose language, confined to system control function. This is unlike, for example, the KRONOS Control Language of CDC computers, which can also be used as a programming language owing to its computational powers [19].

2.3.1) User Activities and PCL Commands

The PVC being a user-dedicated computer on which there is no visible sharing of resources, removes the need for scheduling (or passive) commands. As a result, PCL is defined exclusively in terms of action (or active) commands.

Four types of user activity are identified by PCL. They are file handling, software development, system inquiries, and system operations. Each activity in turn consists of a number of events and each event is associated with a high-level command.

This high-level approach hides the operational details from the user and minimizes the number of commands in the language. Suitable default settings in turn minimize the number of parameters and special characters in the command string. The aim is to enable the user to perform even the most complex operations with only a few simple commands.

The study by Boies [9] into the behaviour of interactive users on the IBM TSS/360 reveals that, out of over 300 commands available to them, the majority know and use only a few, or they use them only in the simplest form. The same can be said of users on other similar systems.

The GEORGE3 command language, for example, also has a large number of commands, each one of which represents a basic system function [62, 66]. Any operation that is not defined by a built-in

command is performed by issuing a sequence of commands. It is a powerful but, for the majority of users, an unnecessarily detailed and essentially low-level language. Extensive use is therefore made of its macro facility to create a high-level repertoire and reduce the number of commands that users need to learn.

Absolute reduction in the number of commands in a language will, however, lead to other problems. JCL/360, for example, has only three commands: JOB, EXEC, and DD. This being the case, the necessary information is supplied as parameters, and this leads to a difficult syntax that is awkward to use [5]. So much so that a macro facility is again needed in order to simplify its use. PCL makes no attempt to combine different events in the same command.

2.3.2) PCL Syntax

PCL is a command-orientated language, the syntax of which is presented in Appendix A. Each command string begins with the command name, may optionally take one or more parameters, and is always terminated by carriage return.

There are no conditional or jump commands in the language and hence no label in the command string. Other language features such as block structuring and variables, which are considered desirable in control languages [14, 37, 62], are also excluded. These constructs are meaningful only within a macro (or command) definition facility, which PCL does not have for two reasons. Firstly, the user himself is expected to be in control interactively at all times to make the

decisions regarding the issuing of commands. Secondly, the small and simple user-orientated command repertoire of PCL eliminates the need for this facility.

The command name specifies the operation to be performed. The parameters, if any, define the entities involved in that operation. One or more spaces separate the command name from its parameters, and commas separate the parameters themselves from each other.

All leading spaces are ignored and each comma may be followed by one or more spaces to improve readability. Any spaces embedded in the parameters themselves, however, are significant. The only other special character is the period (.), which acts as a separator between the filename and its suffix, if present.

The Futuredata 2300 control language [35], for example, represents each command by a single letter, and spaces are not allowed in the command string. This concatination of the command name with its parameters renders the command string awkward in appearance as well as in use. The gain in processability is negligible when compared with the loss in usability.

PCL physically separates the command name from its parameters for a more readable and hence a more usable command string. From the processability point of view, any character may be used as a separator and the outcome would be the same. Texas Instruments TX990 command language [83], for example, allows space and comma as interchangeable separators.

From the usability point of view, however, these characters in their respective roles as defined by PCL, are preferable for two reasons. Firstly, they are used in written English for the same purposes and would therefore be more natural and meaningful to the user. Secondly, the visible break in the command string as provided by space is essential for the required readability and usability.

2.3.3) Command Names

PCL uses meaningful and descriptive command names so that the user can associate each command logically with the operation being performed. This means, for example, the use of the name DELETE to delete a file rather than, say, DEL or DF. Meaningful names are also easier to remember than abbreviations and they minimize the chances of issuing wrong commands.

Command names are restricted to an upper limit of six characters. It is the author's experience that this allows sufficient scope for the creation of meaningful names without becoming unacceptable in length. Longer names reduce the usability of commands and do not necessarily aid clarity.

This restriction naturally poses a problem when a command name tends to be longer than six characters. In that case, if attempts to replace it with an equivalent but shorter name are unsuccessful, then an abbreviation which results in minimal loss of understandability is used.

2.3.4) System Responses

PRL has two constructs: message and prompt. A message informs the user of an event, condition, or status, and resumes execution. A prompt, on the other hand, informs the user that an input is expected from him and waits until it is received. The semantic content of the prompt dictates the type of input expected.

There are three kinds of messages: error, information, and monitoring. Error messages appear whenever a syntactic, semantic, or execution illegality is detected. These messages are usually short and do not exceed a single line. Information messages appear in response to an inquiry made by the user. These generally extend over a number of lines to aid clarity. Monitoring messages, which vary in length, acknowledge user commands and keep him informed of the processing activities.

In order to help the user to associate a particular response with his input, PRL messages contain, wherever applicable, the relevant operative words extracted from the input string. This feature is particularly helpful in, but not exclusive to, error messages.

A prompt is always issued from within a communications environment, which the user enters in order to perform a given operation. A communications environment is defined by a set of valid inputs, some data on which these inputs act, and a set of responses. Each such environment, therefore, is an interactive state which, once entered, expects further input from the user in order to complete its specified function.

The EDIT command, for example, is a request to enter the EDIT communications environment, requiring the user to issue editor commands. The COPY command, on the other hand, if issued with the required parameters in the same input string, may enter any number of states to perform its operation. It does not, however, create a communications environment because no further interaction with the user is necessary.

A prompt can be either a question or a communications environment indicator. The reply to a question prompt is a directive, the nature of which depends on the semantics of the question. The reply to a communications environment indicator is always a command that belongs to that particular environment.

Users often issue commands that are out of context. This is due to user forgetfulness or computer failure to indicate clearly the current environment [9]. Unable to remedy the former, PRL concentrates on the latter. Each communications environment that expects command input from the user has its own unique indicator, with which to prompt him and indicate the type of command expected.

2.3.5) PRL Syntax

PRL is a string-orientated language that follows closely the same rules as written English. Its syntax is presented in Appendix B.

The length of a PRL string is determined by its semantics. There

are no syntactic restrictions on it. The only exception to this rule is the communications environment indicator. It never exceeds a few characters in length, and the user has to consider its logical rather than semantic content before replying.

A prompt is always terminated by the ASCII character "bell". A message, on the other hand, is terminated by the PRL end-of-message character, which consists of one ASCII carriage return and two line feed characters. Long messages that extend over a number of lines are broken up by the PRL end-of-line character. It consists of one carriage return and one line feed character.

2.4) Overview of System Usage

2.4.1) System Startup

The startup (or bootstrap) operation consists of simply placing the system diskette in drive 0 and initiating the hardware bootstrap routine. The objective is to bring the system "alive". If all goes well, then PIL acknowledges the startup operation with the following message and prompt on the terminal:

```
PORTOS DEV=1.0 (4.5.79) NOW RUNNING
IF IN DOUBT, TYPE HELP

GO,
```

The message gives the development number and the implementation date of the current version of PORTOS for identification purposes. The prompt "GO," is the PIL communications environment indicator. The

commands presented in Chapter 3 are now available to the user. There is no logging in or any other mandatory initialization procedure required.

If an error is detected during the processing of a command, the user is next prompted with "ER," following the error message. This is to draw attention to the fact that the previous command has been ignored. Only successful completion is followed by the "GO," prompt. Any command that is not recognized, gives rise to the message

<command name> IS NOT A PORTOS COMMAND

Three characters have special control functions during input from the terminal. "Backspace" (CTRL and H) erases the most recent character in the input string. "Del" cancels the whole line, to which the system replies with the message "*CANCEL*" on the cancelled line, and the special prompt "?" on the next line. "Break" (CTRL and P) is the "panic button". The system response is to display "*BREAK*" on a new line, abandon the current activity, whatever it may be, and return the user to the PIL communications environment. These characters are effective regardless of the current communications environment.

2.4.2) Communications Environment Hierarchy

As soon as the system is started up, and whenever it is idle thereafter, the user enters the PIL communications environment. This is regarded as the top of the hierarchy. The user may, depending on

what commands are issued, either remain in this environment or enter any other that exists under it. Entry to and exit from a communications environment is achieved by issuing the appropriate command [82]. Exit from an environment is always to the "parent" above.

PIL commands create their own sub-communications environment only when further commands are required to perform the specified operation, such as file editing; or confirmation is required for a potentially destructive operation, such as diskette initialization; or user convenience dictates it owing to multiple-parameter input, such as date and time initialization.

The Texas Instruments DX 990/10 control language, for example, always prompts the user for every parameter, but simplifies the task by retaining the latest entry. That is to say, each command creates its own communications environment, even if only to accept a single parameter. The TSS/370 control language, on the other hand, has a partial prompting mechanism for any missing mandatory parameters.

PIL commands, aided by default settings, require the minimum number of parameters. The presence of mandatory parameters are inherent in the semantics of the command, which makes them easy to remember. This renders the creation of any additional communications environment unnecessary. The rigidity of the resulting mechanism is balanced by freedom from the overheads associated with the prompter program [65].

In order to ensure system integrity, the user should terminate

his interaction with the computer, that is to say, unload diskettes or switch power off, only when he is in the PIL communications environment and following a prompt. This is the "safe" state, when no diskette access operations are in progress that may corrupt the filing system.

2.5) PIL Processor

The function of a language processor is to bridge the gap between the virtual computer defined by the language and the host computer on which it is to exist. The processor of PIL is the PORTOS operating system.

2.5.1) Compilers and Interpreters

Interface language processors range from simple interpreters to big multi-pass compilers. For example, JCL/360 is compiled, whereas GEORGE3 is interpreted. It is also possible to compile a language into some intermediate code and then interpret this code by a resident runtime system [47, 55].

Portable control languages such as UNIQUE, GCL, and ABLE are all implemented by compilers [72]. Their machine-independent commands are translated into the equivalent commands of the host operating system. These compiler-generated commands are then executed by the system as if they were issued directly by the user himself.

Compilers offer several advantages. They can be made to generate commands that ensure the optimal use of the target machine and exploit its desirable features. Because compiler output is predictable, changes can be readily made to accommodate long-term variations in hardware availability. Furthermore, they enable character code conversions to be easily made in networks of dissimilar computers.

Their major disadvantage is the time overhead, which is acceptable in batch mode but not for interactive use. The alternatives are either to base the interface language on pre-compiled procedures or to make it directly interpretive.

PORTOS adopts the latter approach. Interpretive processing minimizes the computer response time and is ideally suited to the command-and-response type of processing for which PIL is designed.

2.5.2) Processor Facilities

PORTOS treats the real computer as the host, and not its operating system. It translates PCL commands into a form which it can understand and obey directly. In this context, the term "real" refers to the computer defined by the host assembly language. The fact that this computer is an abstraction of even simpler machine operations is ignored, because the assembly language is sufficiently close to the hardware for all control functions required by PORTOS.

This approach avoids the overheads, which would otherwise be incurred, of translating PCL commands into host system commands before

they can be obeyed, and of de-translating host replies into PRL constructs. It also protects PIL from the influence, for example on filenames, of any would-be host system.

In order to maintain this service, PORTOS provides facilities for user communication, the storage and access of information on diskette, loading and execution of programs, memory management, and hardware control. Their details are presented in Chapters 4 and 5.

2.5.3) Processor Portability

PIL is made portable by the availability of its processor on different computers. The aim is to simplify this availability so as to maximize the number of host computers. The portability under consideration here, therefore, is that of the PORTOS operating system in its capacity as the PIL processor.

To consider the portability of an operating system may appear in the first instance as a contradiction in requirements. An operating system is designed for low-level hardware control on behalf of the user, for which it requires the intimacy of access that only the host assembly language can provide. Portability, on the other hand, requires machine independence in order to have freedom from the binding characteristics of the underlying architecture.

Studies have shown that only about 5% of an operating system is actually machine dependent, and that the rest can be expressed in high-level algorithmic terms [17, 87]. Adhering to this approach,

PORTOS achieves its portability in three steps. The first is its separation into machine-dependent and independent parts during design. This is followed by the representation of its machine-independent parts in suitably machine-independent algorithms. The final step is the use of CORAL66 to implement these algorithms.

The concept of high-level operating systems without portability is not new. The Burroughs Corporation, for example, have been using Extended ALGOL and ESPOL to write their Master Control Program (MCP) operating systems on the B5000 and B6000 range of computers since 1961 [52]. These languages, however, are orientated towards the particular host architecture and hence the MCP is not portable. Extended ALGOL, for example, is 80% ALGOL60 and 20% extensions but in a typical program the extensions constitute about 50% of the code [12].

Other similar systems include the RMX-80 real-time operating system written in PL/M for the Intel 8080 microprocessor [13], the ZEUS time-sharing operating system written in PASS for the DEC PDP-11 computer [80], and the MCTS operating system written in MALUS for the CDC STAR computer [57].

The objective behind such high-level operating systems is increased readability and maintainability [99]. More time is spent reviewing software than actually writing it, because most systems are in a constant state of development during their lifetime in order to keep up with changing demands. High-level language constructs are logically and syntactically closer to natural languages, so that programs become easier to understand and modify.

Portability is a recently recognized objective. The arguments against it were that, in addition to the basic unsuitability of high-level languages to systems programming, the architectural differences between computers would render the operating system of one unreasonably inefficient on another [10, 31, 32]. It has been shown, however, by such operating systems as SOLO [38], MUSS [33], UNIX [59], OS6 [81], THOTH [16], and TRIPOS [74], that careful design and implementation will offer portability at an acceptable level of efficiency.

The two issues normally considered when assessing the efficiency of an operating system are: (1) the interface language, and (2) its implementation.

Interface language efficiency determines the usability of the computer at user level, based on the syntactic and semantic features of the language. It is measured in terms of the work invested or the number of commands issued by the user in order to achieve a given objective. Portable operating systems, by providing the same commands and facilities on different computers, eliminate such variations in efficiency; thus leaving the implementation as the only issue.

Implementation efficiency determines the ease with which the computer can process user commands. In other words, it is the efficiency of the operating system itself. The factors affecting it are program size and execution speed. Program size determines the amount of space occupied by the system, in memory as well as on disk,

which in turn determines what is available to the user. Execution speed, in this respect, is not the actual speed of the computer but the elapsed time between consecutive commands.

Implementation efficiency is, therefore, reflected in the user interface as storage utilization and response time. Their minimization is sought so as to increase the overall system usability.

PORTOS differs from the above-mentioned operating systems in two respects: firstly, it is meant for use on personal computers. Secondly, its design objectives are user-orientation and friendliness, to which portability is a major corollary.

The limited storage capacity and speed of microcomputers mean that care must be taken to ensure minimal resource consumption by the system. The PORTOS design is, therefore, orientated towards simplicity and ease of implementation; the objective being to minimize, firstly, the adverse effects on the user interface, and secondly, the fluctuations in efficiency between the various implementations.

The user would be affected if, for example, the full facilities of the interface language were not available due to storage limitations on the host computer; or the storage requirements of the operating system were to inhibit the transfer and development of large programs; or, to a lesser degree, the response time were to vary noticeably from computer to computer.

A portable system is always more wasteful of storage space and slower in execution than the hand-coded assembly version. This is due

to not only the inherent language and compiler inefficiencies but also the programming methodology dictated by portability requirements. The problem is resolved by the modularization and structuring of the operating system, as well as the choice of implementation language.

These factors, as they apply to PORTOS, are presented in detail in the subsequent chapters. Presented below, as an example, are the considerations taken into account when designing the facilities for information storage and security to meet the above constraints.

A major function of computers is to store information and allow the user access to it on request. The logical structuring of this information can take the form of a database (also called a databank) or individual files. PORTOS employs a conventional filing system because, as it was pointed out in Section 2.1.2, the storage limitations of microcomputers make databases unsuitable to implement.

A filing system can be organized in different ways. One approach that is particularly well-suited to random-access devices is the tree-structured filing system. In this organization, nodes represent the directories and branch lines represent the paths leading to each file or other (sub-) directories. The structure can be extended to any depth or level of complexity.

There are different ways in which such a filing system can be made available to the user. He may, for example, view the whole of the filing system as one unit and be allowed to access any file at any time by specifying the relevant directory-to-file path. This approach

is exemplified by the DX 10 operating system, where any number of sub-directories may be specified before the target file. Alternatively, he may be required to "attach" to a particular directory and be allowed to access only those files that exist under it. Naturally, those files that exist under a different (sub-) directory become accessible following a new attach operation. An example of this approach is the PRIMOS operating system.

These multi-level filing systems offer the user additional power in organizing or grouping his files. The resulting implementation, however, is space and/or time consuming, either or both of which may be critical on small computer systems. This is why PORTOS employs a single-level filing system, as mentioned earlier in Section 4.3.1, where a single directory contains all the files and sub-directories are not allowed. This approach is similar to that employed by the RT-11 operating system of DEC PDP-11 computers.

The hardware configuration for which PORTOS is intended was presented in Section 2.1.3. Each diskette is taken as an independent unit and represented by a self-contained directory. The user himself acts as the "master directory" in keeping track of what each directory (or diskette) contains. The diskette directory is implicitly defined during file access operations and are therefore hidden from the user. This approach was chosen because it is the simplest and the least resource-consuming form of the tree-structured filing systems. It facilitates not only the implementation but also the user interface.

Two vital issues related to information storage are privacy and security. Privacy is the protection of user files against unauthorized

access, while security is their protection against malicious or accidental damage.

PORTOS has no facilities for privacy because, on single-user systems, the user is the sole owner of the complete system and all its files, even if only conceptually. By the same token, there is assumed to be no danger of malicious destruction of information.

Protection against accidental damage or loss, however, is a major issue. The damage may result from a command inadvertantly issued by the user or some computer error, caused by either the hardware or the software. Moreover, diskettes, owing to their manufacturing characteristics and limitations, are particularly vulnerable to physical damage through their handling by the user.

Security is maintained by duplicating the information stored on the system. On large systems, this operation (called dumping) involves the copying of all those files that have been created or accessed since the last dump. It may be performed automatically by the operating system at regular intervals or it may be command-initiated by the operator. In either case, the user remains unaffected and therefore unaware of the dump operation taking place; except for, possibly, a message on his terminal to explain the increase in response time or the temporary inaccessibility of certain information.

Automatic dumping assumes the availablity of a peripheral device dedicated to this purpose, and also the running of a timer program to initiate the dumper in a multi-programming (or tasking) environment.

Neither assumption is applicable to the kind of personal computers for which PORTOS is meant. Firstly, a suitable dedicated peripheral is not available owing to economical reasons, and secondly, they do not have the capacity to support effective multi-programming.

A limited form of automatic dumping which is applicable to personal computers is the creation of a "backup" copy of all those files the contents of which are modified by the editor. This way, if anything goes wrong, a copy of the original file is available to the user. Since a separate device cannot be made available, however, this backup copy is kept on the same device, which creates two problems.

Firstly, these backup copies occupy valuable space on diskette; and deleting them to make room for other files, which is a natural tendency, defeats the whole purpose of making copies. Secondly, if the diskette directory is corrupted or the diskette itself is physically damaged, then the whole diskette becomes inaccessible and both the original file and its backup copy are lost.

PORTOS employs the command-driven approach, enabling the user to dump his files as and when he sees fit. The purpose-built MOVE command, the details of which are presented in the next chapter, can copy either individual files or the whole directory to another diskette. This alternative places the responsibility of securing his files on the user himself, but makes no additional demands on storage or the processing power of the computer.

As a result, PORTOS is a simple, efficienct system that is easy to implement and use.

CHAPTER 3 : PORTOS INTERFACE LANGUAGE

3.1) PCL Commands and PRL Responses

PCL commands are presented in their respective groups as defined in Section 2.2.1. This division is for managerial purposes only. It has no effect on how or when the user may issue them, or how they are handled by the system. All file reference parameters, unless otherwise stated, refer to files on the user diskette with no modifications to their suffix names.

PRL responses are presented as appropriate to each command. Those that apply to more than one command are mentioned only on their first occurrence. Their presence should be assumed from then on whenever applicable.

The language constructs are presented using the following conventions: all upper-case words, either in the command or output string, appear in full; command parameters contained in square brackets ([]) are optional; and parameters contained in sharp brackets (<>) are replaced by the appropriate PCL entries.

3.2) File Handling Commands

3.2.1) CREATE Command

CREATE <characterfile reference>[,E=<eof command>]

Enters the CREATE communications environment in order to create

the referenced file on diskette, and accept input from the terminal until the end-of-file command is issued.

The over-writing of files is not allowed. If the user wishes to re-create a file using an existing reference, he must first delete the previous version. Otherwise, the command is ignored with the message

<file reference> ALREADY EXISTS

This is a precaution against accidental destruction of files and also guarantees the uniqueness of each file reference.

The environment indicator is the ASCII character "backarrow" (_), following which any sequence of characters terminated by carriage return (including null) may be entered. The maximum number of characters per line is 80, which is the maximum width on most terminal devices.

The end-of-file command to terminate the CREATE environment is the string "OK" appearing as the first two characters in the input line. All trailing characters except carriage return are ignored. The user has the option to change this command, in case the same string appears as part of his input, simply by specifying

E=xx

as a parameter in the command line, where xx are any two ASCII characters.

3.2.2) DELETE Command

DELETE <file reference>

Deletes the referenced file from diskette if it is not protected against deletion. If the file does not exist, then the command is ignored with the message

<file reference> NOT FOUND

If the file does exist but it is protected against deletion, then the command is ignored with the message

<file reference> IS LOCKED

3.2.3) RENAME Command

RENAME <oldfile reference>,<newfile reference>

Changes the file reference from oldfile to newfile. The contents, characteristics, and attributes of the file remain unchanged after the operation. The command is ignored if newfile already exists.

3.2.4) COPY Command

COPY <sourcefile reference>,<targetfile reference>[,OUST]

Creates targetfile and copies the contents of sourcefile to it. Sourcefile remains unchanged after the operation. If targetfile already exists, then the parameter OUST comes into effect. If present, targetfile is over-written, providing it is not protected. Otherwise, the command is ignored.

3.2.5) LOCK Command

LOCK <file reference>

Puts a lock on the referenced file in order to protect it against deletion.

3.2.6) UNLOCK Command

UNLOCK <file reference>

Removes the lock on the referenced file.

3.2.7) LIST Command

LIST <characterfile reference>[,NUMBER]

Lists the contents of the referenced file on user terminal. The listing, which is always terminated by the string "*EOF*", is preceded by one and followed by two blank lines as delimiters. The optional qualifier NUMBER causes the display to be numbered from 1. If the file is non-ASCII, then the command is ignored with the message

<file reference> UNSUITABLE FOR THIS OPERATION

3.2.8) PRINT Command

PRINT <characterfile reference>[,<qualifiers list>]

Lists the contents of the referenced file on the printer. The listing is always terminated by the string "*EOF*" and form feed. On those printers where form feed is inapplicable, it is replaced by an appropriate number of line feeds.

The file reference parameter has two optional qualifiers: NUMBER, to number the listing, and TITLE, to include a heading. The numbering is subject to the same constraints as in the LIST command. The heading, which is repeated at the top of each page, contains the file reference, the date and time when it was produced, and the page number. The qualifiers may appear in the list in either order separated by commas.

3.3) Software Development Commands

3.3.1) EDIT Command

EDIT <characterfile reference>

Enters the EDIT communications environment in order to edit the referenced file. The response to the command is:

READY TO EDIT <file reference>

E?

where "E?" (for Edit) followed by one space is the environment indicator, following which the user may issue any edit command.

The exact nature of the EDIT communications environment is outside the scope of this current development. It is assumed that a suitable editor will be subsequently included in the system. No definition is therefore given of its commands, except those that affect PIL directly. The same also applies to the other commands in this group.

Editing is a major operation in file maintenance and software development. It has been shown that more than 75% of the commands issued during a typical session are editing commands [9]. They should have the same syntax as PIL commands, but in view of their extensive use, the names may be abbreviated to one or two characters.

There are two commands to take the user out of the EDIT environment. QUIT, to abandon it, and SAVE, to terminate it. QUIT does

not take any parameters and the editor responds with the message

EDIT ABANDONED

after which the user is returned to the PIL communications environment. The original file remains unaltered. The SAVE command has one optional parameter:

SAVE [<file reference>]

If the parameter is omitted, then the edited file is saved under the original reference. The previous version is over-written but the new file assumes its protection attributes. In other words, if the original file has a lock on it, the editor will over-write it with the new version but the saved file will remain protected.

If the user specifies a new reference and it already exists, then he is prompted with the question

REPLACE?

because those systems that allow the over-writing of files without confirmation are considered unsafe. If the answer is YES, the file is over-written and the previous version is lost. If it is NO, then the user is returned to the EDIT environment. Files are over-written in order to economize on diskette space. File generation numbers, whereby different versions of the same file can exist under the same reference, do not exist.

3.3.2) Language Processing Commands

<language name> <sourcefile reference>[,P=<options list>]

Any language processor may exist under PIL. The command name is taken from the language name either in full, such as CORAL and PASCAL, or as an abbreviation of it, such as FORT for FORTRAN and MASS for Macro Assembler.

The sourcefile reference is a mandatory parameter, containing the program to be assembled or compiled. Output from the processor is also sent to files, the references for which are taken from the source filename with the suffix name modified accordingly, as specified in Figure 2.2. Any existing files are automatically over-written. This approach was adopted because studies have shown that, in spite of the desirability of interactive compilation to improve computer usability [30], such facilities are seldom used [9].

In order to economize on diskette space, however, the processors display all errors on the VDU by default and produce source listings only on request. The behaviour of the processor is controlled by the options list parameter, which takes the form of a character string. These options are particular to each processor.

3.3.3) LINK Command

LINK <objectfile reference>

Enters the LINK communications environment in order to link (or consolidate) the referenced object program into an executable binary program. If the suffix name in file reference is omitted, then ".O" is assumed. The system responds to the command with

READY TO LINK <file reference>

L?

where "L?" (for Link) followed by one space is the environment indicator, following which the user may issue any link command. When all address references in the object module have been resolved, the system displays the message

LINK COMPLETED

The user should then save the linked program in a binary file on diskette with the SAVE command, which has the following format:

SAVE [<file reference>]

If the parameter is omitted, then the original file reference is taken with the suffix name ".B". Any existing version is over-written. If a different file reference is given and it already exists, then the confirmation procedure is the same as in EDIT command. The user may also QUIT the LINK environment.

3.3.4) RUN Command

RUN <binaryfile reference>

Loads into memory from diskette and executes the referenced binary program. If the suffix name in file reference is omitted, then ".B" is assumed. The user remains in the environment created by this program until an exit is made to the PIL environment, either normally on completion or abnormally on error.

3.3.5) DEBUG Command

DEBUG <binaryfile reference>

Enters the DEBUG communications environment in order to examine the executional behaviour of the referenced binary program. If the suffix name is omitted, then ".B" is assumed. The system responds to the command with

READY TO DEBUG <file reference>

D?

where "D?" (for Debug) followed by one space is the environment indicator, following which the user may issue any debug command. Exit from the DEBUG communications environment is made with the QUIT command.

3.4) Inquiry Commands

3.4.1) HELP Command

HELP

Enters the HELP communications environment in order to provide the user with information regarding the commands available in the system. The initial response to the command is to display the following on the terminal:

```
PORTOS OPERATING SYSTEM COMMAND MENU

    COMMAND TYPE      GROUP NUMBER
    ------------      ------------

    FILE HANDLING             1
    SOFTWARE DEVELOPMENT      2
    INQUIRY                   3
    SYSTEMS                   4
    ALL (NAMES ONLY)          5

WHICH GROUP DO YOU REQUIRE?
```

Depending on the user's reply, the information contained in Appendix C is partially displayed. Any reply outside the range 1 to 5 is ignored with the message

1 TO 5 ONLY PLEASE

and the prompt repeated. The partial command display is then followed by the prompt

DO YOU NEED HELP ON OTHER COMMANDS?

If the answer is YES, the initial menu is displayed again and the user is asked to select the group on which he needs help. If NO, he is returned to the PIL environment with the message

DONE

The partial display of the menu, and only after an explicit request from the user, offers several advantages. Firstly, the user is given help only if and when he needs it. Secondly, when he does need help, he is given the chance to concentrate on his particular area of difficulty without having to repeat what may be already familiar to him. Thirdly, the processing time of the command is minimized by restricting the amount of information displayed.

This particular HELP command provides the user with information on the usage of those commands that are available to him in the PIL communications environment. He may also need help on the commands available in other environments. To cater for this situation, PIL adopts the IBM System/370 TSO control language approach [65]. Each communications environment has its own HELP command.

3.4.2) FILES Command

FILES [<parameter list>]

Displays the contents of the user diskette in terms of files on the terminal as follows:

```
FILES ON DISK : <diskette name>
    <file reference 1> ... <file reference 6>
    .....
```

where "diskette name" is the name given by the user to that diskette during initialization. It is used for identification purposes only and is not crucial to the processing of the command. Six file references are displayed per line.

If the diskette is empty, say, after initialization, then the following message is displayed:

```
FILES ON DISK : <diskette name>
     **NONE**
```

The command may optionally take up to three parameters: SYSTEM, PRINT, and DETAIL, any or all of which may appear in the list in any order.

The parameter SYSTEM causes the contents of the system diskette (in drive 0) to be displayed. The parameter PRINT sends the display to the printer. The listing on the printer includes a heading by default, indicating the date and the time when it was produced.

The parameter DETAIL causes additional information on each file to be included in the display. The format is one file reference per line as follows:

FILES ON DISK : <diskette name>

REFERENCE	TYPE	PROTECT	SIZE

REFERENCE is the filename followed by its suffix, if present. TYPE can be ASCII, OBJECT or BINARY. PROTECT is either YES, if the file is protected against deletion, or NO, if it is not. SIZE gives the length of the file in number of sectors that it occupies on diskette.

3.4.3) SPACE Command

SPACE [<parameter list>]

Displays the available space on user diskette in terms of sectors. The message format is as follows:

FREE SPACE ON DISK : <diskette name>

<n> SECTORS OUT OF <m>

where "n" is the number of free sectors and "m" is the total number of sectors on the diskette. If n is equal to zero then the message "*DISK FULL*" is also displayed.

The command has two optional parameters, SYSTEM and PRINT. Either or both may appear in the parameter list in either order. The parameter SYSTEM gives the free space on the system diskette and PRINT sends the output to the printer.

3.4.4) TIME Command

TIME

Displays the date and the time of day on the user terminal. Assuming that the indicators have been initialized correctly (using the relevant PIL command), the message will have, for example, the following format:

16.JUL.79   13:22:46

The command may also be issued without first performing the initialization, in which case the time-of-day indicator will display the elapsed time. For example, the message

00.???.00   00.26.53

means the system has been running for 26 minutes and 53 seconds.

3.5) System Commands

3.5.1) INDISK (initialize disk) Command

INDISK

Enters the INDISK communications environment in order to initialize the diskette in drive 1 as an empty user diskette.

Initialization involves the writing of certain information to the diskette in pre-defined locations, invisible to the user but essential for housekeeping functions. It enables PORTOS to recognize and use new diskettes that contain either alien information or none at all.

The system responds to the command with the following prompt:

    NEW DISK IN DRIVE 1?

The user should place the diskette to be initialized in drive 1, if he has not already done so, and type YES. The next prompt is

    NEW DISK ID?

to which the user should reply by typing in the name he wishes to give that diskette (up to a maximum of six characters). The system then responds with the following message and prompt:

    READY TO INITIALIZE <diskette name>
    OK IF CONTENTS LOST?

If the answer is YES, then the diskette is initialized and the completion of the process is acknowledged with the message

    DONE

after which an exit is made to the PIL communications environment. If the answer is NO, either to this or the first prompt, then the process is abandoned with the message

INDISK ABANDONED

The user is prompted twice for confirmation simply as a safety measure, because this command will irreversibly destroy the original contents of the diskette in question.

3.5.2) INTIME (initialize time) Command

INTIME

Enters the INTIME communications environment in order to initialize the system date and time-of-day indicators. The user is prompted with the following questions:

```
INITIALIZE TIME
             DAY (1-31)? <n1>
MONTH (1ST 3 LETTERS)? <string>
 YEAR (LAST 2 DIGITS)? <n2>
           HOURS (0-23)? <n3>
         MINUTES (0-59)? <n4>
```

where "string" is an ASCII string and "n1" through "n4" are either one- or two-digit integer numbers as appropriate. Incorrect entries are ignored with the message

CANNOT BE <entry> TRY AGAIN

and the prompt repeated.

3.5.3) MOVE Command

MOVE <data>,<source drive>,<target drive>

Enters the MOVE communications environment in order to transfer data from the diskette in source drive to the diskette in target drive. The data being moved may be the contents of either a single file or the whole diskette.

If the data parameter is a file reference, then that file is moved. If a file with the same reference already exists on target diskette, then the user is prompted with

REPLACE?

for confirmation before the move takes place. If the answer is YES, then the existing file is over-written and an exit is made. If it is NO, then the process is abandoned with the message

MOVE ABANDONED

and the user is returned to the PIL environment.

This option of the command serves two important functions. It enables the user to install his own programs as commands and also allows him to modify the existing system programs.

Any binary file can be moved from USER diskette to SYSTEM diskette and installed as a command in its own right. After the

operation, the filename (without its suffix) becomes the command name, ready to be issued like any other PIL command. Any function or non-target peripheral can thus be incorporated into the system, simply by maintaining its handler program on SYSTEM diskette.

Modifications to existing system programs may be carried out in order to add a new facility, to improve efficiency, or to increase power. The relevant source program is first moved from SYSTEM to USER diskette so that it becomes accessible to the user. It may then be edited and compiled like a user program. The final step would be to install the resulting program back on the SYSTEM diskette. This approach naturally requires knowledge of the system and its implementation language.

The MOVE command also offers the ability to rename commands, into a foreign language if necessary, as befits the user's needs. The ability to augment or tailor PIL makes the system adaptable to changing needs and helps the user to make better use of it. On the other hand, the damage that can be done to language stability must also be borne in mind. The same command may vary in behaviour on two different computers because of modifications or users could end up with totally different command repertoires depending on their applications and choice of command names.

If the data parameter is the ASCII character asterisk (*), then the complete contents of source diskette are moved to target diskette. This option is necessary to make copies of diskettes. The system responds with following message and prompt:

MOVING DISK FROM <source> TO <target>
<target> CONTENTS WILL BE LOST, OK?

If the answer is NO, then the process is abandoned in the same manner as before. If it is YES, then the transfer takes place following the message

MOVE IN PROGRESS...

If the target is USER, then the process is terminated with the message

DONE

and exit is made to the PIL environment. If, however, target is SYSTEM, then, not knowing the final contents of the diskette, the command prompts the user with:

DONE
IS SYSTEM DISK IN DRIVE 0?

so that if some other diskette is in drive 0, the user can reload the relevant diskette. This prompt is necessary to ensure correct system configuration before continuing.

CHAPTER 4 : PORTOS OPERATING SYSTEM FACILITIES

4.1 General Structure

The PORTOS operating system is divided into three levels. The top level is the PIL interface, which consists of the PIL decoder and command handler programs. It is built on a common set of primitive operations, the semantics of which define PORTOS. The second level is the machine-independent kernel to support these primitive operations and manage resources. The bottom level is the hardware interface. It contains the machine-dependent peripheral drivers and interrupt handlers.

This structure is similar to that adopted by UNIX, which is said to be instrumental in its portability [59]. Although PORTOS achieves its portability differently from UNIX, it nevertheless benefits from this division. Firstly, it facilitates the functional classification and management of system operations. Secondly, it enables the separation of machine-dependent sections from the rest of the system.

4.2) PIL Interface

4.2.1) PIL Decoder

The function of the decoder is to establish and maintain communication with the user via PIL. It is, in effect, a translator between the PIL constructs and the rest of the system.

The decoder accepts one PCL command at a time from the user, and checks it for syntactic correctness. If no error is found, then it passes the command to the relevant command handler program for interpretation. This is done by setting up a parameter table in memory, searching the system diskette for the named command, and initiating its handler program. Initiation is achieved by loading the program into memory and transferring control to it.

When the active command handler program terminates its execution, the decoder resumes control. It picks up the status code from the program, translates it into the relevant PRL construct, and conveys it to the user. After this, the decoder is ready to accept the next command.

4.2.2.) Command Handler Programs

Each PCL command is interpreted by its corresponding command handler program. The interpretation is actually performed by breaking up the command into a series of primitive operations, that provide such services as file management, memory management, input-output, etc.

Command handler programs recognize each primitive operation by its name and the parameters by which to send or receive information. A request for service is made by passing this name to the kernel and setting up the necessary parameters. Error checking is performed before or after each request, as appropriate.

On exit from a command handler program, the control is returned to the PIL decoder via the relevant primitive operation. The function of the kernel now is to convey the necessary information to the decoder via its parameter table.

4.3) PORTOS Kernel and Primitive Operations

The kernel is made up of six managers, which are similar in concept to monitors [51]. Each manager is responsible for a particular resource, and is defined by one or more primitive operations (or primitives) to perform its function. A primitive itself may be defined in terms of simpler primitives.

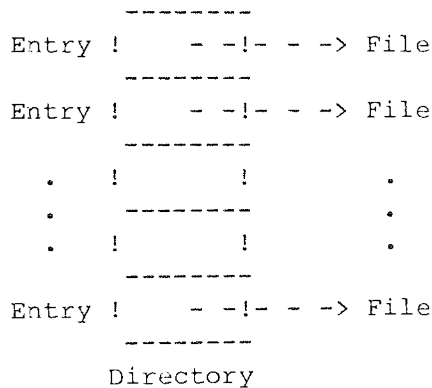
Managers, and primitives within them, are developed as independent modules. During operation, the primitives interact, not only with each other, but also with the other parts of the system, through their individual interfaces.

A manager may be considered as either high or low level, depending on its closeness either to the PIL computer or the hardware, without becoming machine-dependent. The file manager, for example, is high-level, whereas the memory manager is low-level. A high-level manager can request service from any other at a lower level, but not vice versa. Similarly, a primitive may be either high or low-level, depending on its closeness to PIL or the internal operation of the manager to which it belongs. These managers and their primitives are described below.

4.3.1) File Manager

The PORTOS filing system has a single-level tree structure as shown in Figure 4.1. At the root of the tree is the directory and the branches are the files. The directory contains one entry per file that exists under it. This structure is repeated on every diskette used by PORTOS so that each diskette is logically self-contained for file manipulation.

Figure 4.1 : PORTOS Filing System



Each directory entry contains the relevant information that is necessary to access that file, as shown in Figure 4.2. The file reference is the logical name by which the user refers to the file. Its attributes determine the operations that may be performed on it, while the sector number gives the location on diskette where the file is actually stored.

Figure 4.2 : Directory Entry Format

```
            -------------------
Word 0     ! Type   ! Protect !    Attributes
            -------------------
Word 1     !                  !
            -------------------
Word 2     !                  !    File Reference
            -------------------    (Filename + Suffix)
Word 3     !                  !    One character per byte
            -------------------    padded with spaces
Word 4     !                  !
            -------------------
Word 5     !                  !    Reserved
            -------------------
Word 6     !    Sector no.    !    Address of 1st sector
            -------------------    occupied by file
```

```
Attributes
----------
   Type      0 ASCII
             1 Object
             2 Binary

   Protect   0 Off    (unlocked)
             1 On     (locked)
```

The following primitives are provided by the manager for file manipulation:

COPY FILE(SOURCE, TARGET, KEY)

copies the contents of file in SOURCE to TARGET on user diskette. TARGET is created if it does not exist. If TARGET already exists, KEY determines whether to delete it (if KEY=1) or not (KEY=0).

CHANGE PROTECT(NAME, PROTECT KEY)

changes the protect attribute of the file NAME on user diskette in order to protect or unprotect it against deletion. The attribute is changed to "locked" if PROTECT KEY=1 and to "unlocked" if PROTECT KEY=0.

DELETE FILE(NAME,DEVICE)

deletes the file NAME on DEVICE (SYSTEM=0, USER=1) and removes its entry from the directory.

RENAME FILE(OLDNAME, NEWNAME)

changes the reference of file OLDNAME on user diskette to NEWNAME.

The existence of a file is made known to PORTOS by its directory entry. Any file access, therefore, whether for purposes of manipulation or data transfer, involves its directory entry. The following primitives are provided for the related house-keeping operations:

MAKE ENTRY(NAME, DEVICE, TYPE, PROTECT, SECTOR)

makes an entry for file NAME in the directory on DEVICE with attributes TYPE and PROTECT. The manager creates the entry and returns the address (SECTOR) of the sector that is allocated to the file.

FETCH ENTRY(NAME, DEVICE, SECTOR, TYPE, PROTECT)

searches the directory on DEVICE for file NAME. If it exists, its address on diskette and attributes are returned to the calling routine. If not, then -1 is returned in SECTOR.

KILL ENTRY(NAME, DEVICE, SECTOR)

removes the entry of file NAME from the directory on DEVICE. The manager then returns the address of the file on diskette to the calling routine. If the entry is not found, then SECTOR is set to -1.

File reference manipulation is an important feature of file management. The following primitives are provided for the relevant

suffix name manipulation:

WHAT SUFFIX(NAME)

returns the suffix name of file NAME as the single character following the period. If the file has no suffix name, then space (blank) is returned.

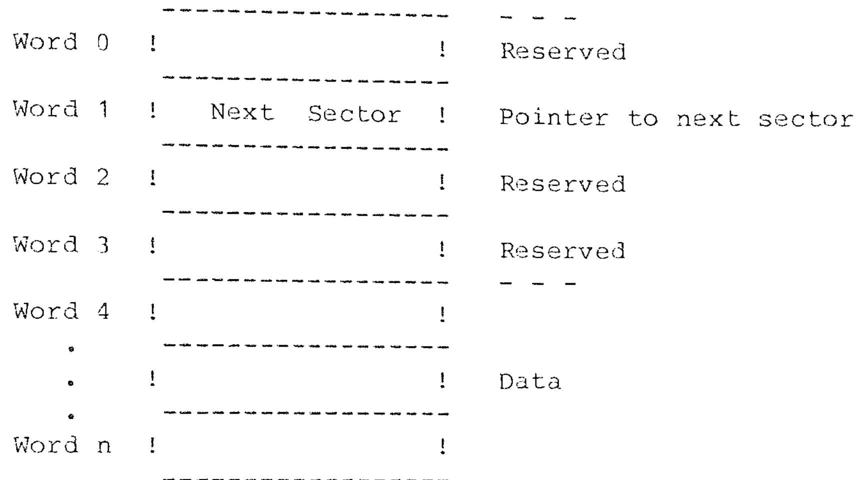
NEW SUFFIX(OLDNAME, NEWNAME, SUFFIX)

changes the reference of file OLDNAME to NEWNAME by replacing its suffix name with SUFFIX, which should contain a single character. If OLDNAME has no suffix name, then SUFFIX is simply appended to it with the period.

All PORTOS files are sequential. The directory entry always points to the first sector occupied by that file. If more than one sector is occupied, then they are chain-linked to each other by pointers. The last sector in the chain contains zeros in its pointer field.

The format of sectors on diskette is given in Figure 4.3. The first four words are the header, which PORTOS uses for its own house-keeping purposes. In the current version of the system, only the second word is used as a pointer. The other three words are reserved in anticipation of future developments and expanding requirements. The total number of words per sector is hardware-dependent.

Figure 4.3 : Sector Format



4.3.2) Input-Output Manager

The function of this manager is to handle the transfer of data to and from the peripherals. All transfers are achieved via channels. The following primitives are provided for this function:

READ RECORD(CHANNEL, BUFFER, N, RESULT)

reads N bytes into BUFFER from CHANNEL. RESULT is set to 1 if the end of file is read, otherwise it is 0.

WRITE RECORD(CHANNEL, BUFFER, N)

writes N bytes from BUFFER to CHANNEL.

READ CHAR(CHANNEL)

reads a single character from CHANNEL.

WRITE CHAR(CHANNEL, DATA)

writes a single character from DATA to CHANNEL.

Three devices are recognized by the manager: VDU, printer, and diskette. The allocation of channels to these devices is given in Figure 4.4. All channels are dedicated to their respective devices and cannot be used interchangeably.

Figure 4.4 : PORTOS Channel Allocation for I-O

Channel	Device	Function
0	VDU	Input-Output
1	Printer	Output
2 - 9	Diskette	Input or Output

Diskette input-output is a special case in that the transfer of data takes place actually to or from a particular file. It is necessary to associate the file in question with a given channel for the required function and then to break this link between the two after the completion of data transfer. These operations are performed by the following primitives respectively:

OPEN(CHANNEL, NAME, DEVICE, OPER, RESULT)

opens CHANNEL to file NAME on system (DEVICE=0) or user (DEVICE=1) diskette to read (OPER=1) or write (OPER=2) data. The error codes to which RESULT is set on return are given in Figure 4.5. Other errors cause the primitive to be aborted.

CLOSE(CHANNEL)

closes CHANNEL so as to terminate the data transfer to or from the related file.

Figure 4.5 : Error Codes in OPEN Primitive

Code	Meaning
0	OK
3	File not found (read)
4	File already exists (write)

Channels 0 and 1 do not need to be opened or closed for any data transfers. The manager maintains an input-output table for diskette access, the format of which is given in Figure 4.6, where logical file references are associated with physical diskette locations. The following primitives are provided to access this table:

SEEK IO ENTRY

returns the index number of the first free entry in the table. If the table is full, then -1 is returned.

LOOKUP(CHANNEL)

returns the index number of the entry for CHANNEL. If it is not found, then -1 is returned.

SHUTDOWN IO

closes all channels and clears the input-output table.

Table 4.6 : PORTOS I-O Table Format

Word	Contents	Function
0	CHANNEL	Channel number
1	SECTOR	Address of current sector
2	FUNCTION	1 Read, 2 Write
3	WORD	Address within io buffer
4	BYTE	0 Top byte, 1 Bottom byte
5	BUFFER	Address of io buffer in memory

The following primitives are provided to handle the data transfers involving the VDU:

GET RECORD(BUFFER, N)

reads N characters into BUFFER from VDU

PUT RECORD(BUFFER, N)

writes N characters from BUFFER to VDU

The GET RECORD primitive is provided specifically to implement the control characters defined in Section 2.4.4 for input from the CDU. The PUT RECORD simply complements it on output.

4.3.3) Diskette Manager

Before a file can sensibly exist under PORTOS, it must be allocated space on diskette and the system must be informed of its address. When the file is deleted, the space that it occupies is released for re-allocation in response to subsequent requests.

A sector is the smallest addressable unit on diskette for read-write operations and is, therefore, also the unit of allocation. The function of this manager is to monitor the availability of all free sectors on diskette and allocate or reclaim them as required. The following primitives are provided by the manager for this purpose:

GIVE SECTOR(DEVICE, SECTOR)

allocates a sector on DEVICE (0 for system and 1 for user diskettes). The manager returns the address of the allocated sector in SECTOR. If the request cannot be granted, i.e. the diskette is full, then -1 is returned in SECTOR.

RELEASE SECTOR(DEVICE, SECTOR)

releases the sector on DEVICE at address SECTOR.

Each diskette is viewed simply as a collection of individual sectors, numbered from zero to some hardware-dependent upper limit. The manager maintains a bit array called the Diskette Status Array, the length of which is equal to the total number of sectors on diskette. Each bit represents one sector and, depending on its position in the array, enables a particular sector to be individually addressed. A sector is available if its corresponding bit is set and it is unavailable if reset. Sectors are thus allocated and released, respectively, simply by resetting and setting their corresponding bit in the status array.

Each diskette, being logically self-contained, has its own status array to reflect its availability. This array is also allocated a sector and stored on the diskette to which it belongs. The array size, being a function of diskette size in terms of number of sectors, is determined during implementation.

4.3.4) Memory Manager

The manager uses the variable-length partitioned memory allocation scheme. It assumes a contiguous word-addressable store of a given size and allocates partitions from this area as needed.

The heart of the manager is its "free partitions list". When a request is made, the manager scans this list until it finds a partition that is equal to or larger than the demand. When it does, that partition is removed from the list and its base address is returned to the calling program. If the partition in question is larger than the demand, then it is split into two. The first part becomes the allocated area and the second part is returned to the list as a new free partition.

The manager has the following three primitives for this function:

ALLOCATE(SIZE, ADDR)

requests the allocation of SIZE words. The manager returns the start address of the allocated area in ADDR if the request is successful. If not, then -1 is returned.

FREE(SIZE, ADDR)

releases SIZE words starting at location ADDR.

INSERT(ADDR)

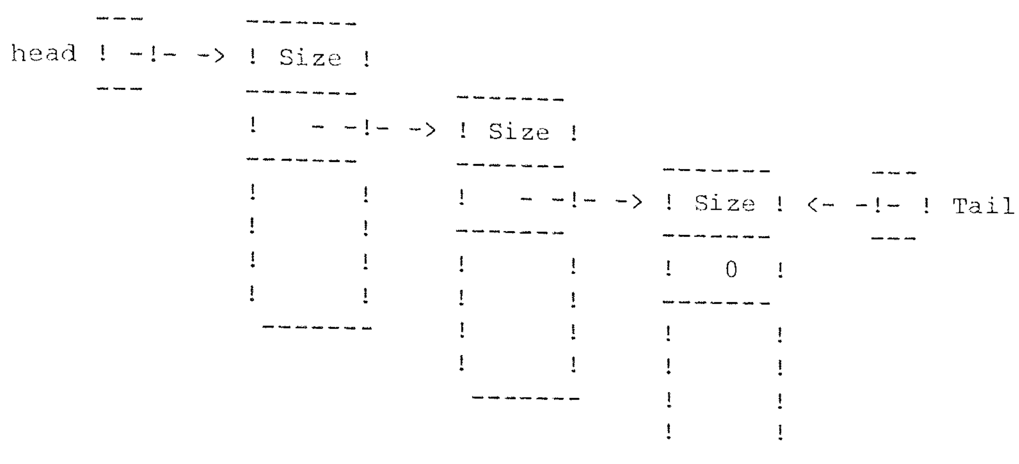
inserts the block starting at location ADDR into the free partitions list. The size of the block is given in the first word of the block.

The free partitions list is kept in ascending order of partition size at all times. Two global variables, HEAD and TAIL, point to the beginning and the end of the list respectively. Scanning is done from head to tail so that the first fit is also the best.

When a partition is freed it is returned to the list for subsequent re-allocation. The first task of the memory manager at this point is to check for adjacent free partitions, and if there are any, to merge them into one. Its next task is to scan the free partitions list in order to determine at which point to insert the new partition, because this part of the memory manager is also responsible for maintaining the list in ascending order. The actual insertion of either the original or the resulting new partition is a simple matter of adjusting pointers.

The manager maintains its free partitions list by creating self-contained partitions in memory and linking them together with pointers, as shown in Figure 4.7. The convention used is as follows: the first word of each partition gives its length and the second word is a pointer to the next free partition in the chain. The last partition in the list contains a zero in its second word.

Figure 4.7 : Free Partitions List



This way the manager is relieved from the necessity of maintaining a separate list with pointers to blocks of memory. The partitions themselves make up the list in the form of a chain. The manager only needs to know where to find the first partition in order to be able to scan the list. The advantage of this scheme lies in the fact that when a particular partition is allocated, the memory locations containing the link information (however few) are also allocated as part of the partition, so that memory waste is totally eliminated.

When the system is started up, the whole of the available user memory is initialized as a single partition. In the course of the execution of PORTOS this original partition may be split up into any number of smaller ones, linked with pointers, depending on the requests made. The smallest partition that can be allocated is ten words. There is no upper limit to the number of partitions that may

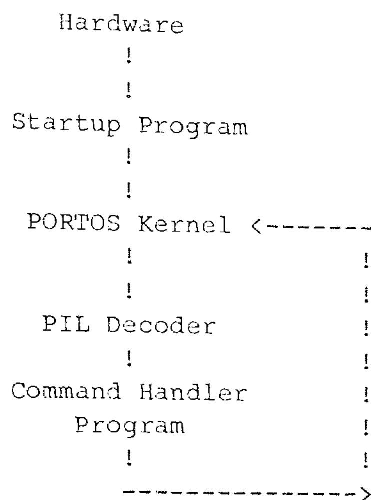
exist at any time because there is no need to impose such a restriction. The manager does not care how many partitions there are as long as there is a path to follow from the head of the list to the tail.

If the manager cannot satisfy a request for memory, then no action is taken except to inform the calling program of the outcome. Regarding the requests to release a partition, the validity of the parameters is assumed and the partition is returned to the list without any checks. The misuse of this primitive would naturally cause chaos as it is capable of corrupting the free partitions list if supplied with invalid parameters.

4.3.5) CPU Manager

The function of this manager is to share the CPU amongst the various system and user programs that would be running on it. PORTOS, however, is a single-user single-programming operating system and the sharing of the CPU is performed on a sequential basis. The flow of control is always from the active program, of which there is only one at any time, to other programs initiated by it. The overall flow of control during execution is given in Figure 4.8.

Figure 4.8 : Flow of Control in PORTOS



The following primitives are provided by the manager for transfer of control betwen programs:

EXECUTE(NAME, DEVICE, RESULT)

initiates the binary file NAME on DEVICE. RESULT is set to 3 if the file is not found and it is set to 10 if the file is not binary.

LOAD(DEVICE, SECTOR)

loads into memory the binary program at SECTOR on DEVICE and transfers control to it.

ERROR(NAME, CODE)

informs the user of error CODE involving the entity NAME and returns control to the kernel.

EXIT(STATUS)

returns control to the kernel with condition STATUS (OK if STATUS=0). It is the responsibility of this primitive now to initiate the PIL decoder.

4.3.6) Task Manager

The function of this manager is to coordinate the operation of the memory-resident primitives of PORTOS, known as tasks, which are responsible for the overall running of the system. The organization of PORTOS in memory during execution and the details of the task manager are presented in Chapter 5. The name is included here for completeness.

4.3.7) Miscellaneous Primitives

GET PARAM(ARRAY)

gets the next parameter from the input string and places it in ARRAY.

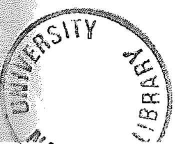
SET TIME(ARRAY)

initializes the system date and time indicators from ARRAY as shown below:

Array Word	Contents
1	Day
2	1st 2 letters of month
3	3rd letter of month and space
4	Year
5	Hours
6	Minutes
7	Seconds

FETCH TIME(ARRAY)

installs the system date and time indicators in ARRAY as shown above.


UNIVERSITY LIBRARY

4.4) Hardware Interface

The function of the hardware interface is to present to the upper levels a uniform picture of the underlying architecture, and keep PORTOS unaware of any changes. This is achieved by defining an "ideal" hardware machine, and interfacing that to PORTOS. This machine is described by a set of primitive operations, to drive its peripherals, handle interrupts, and perform all other low-level machine operations.

4.4.1) Peripheral Drivers

A peripheral driver is an assembly language program that actually operates or drives the relevant input-output device to perform the physical transfer of data on it. "Physical" means that the data is transferred without being interpreted in any way.

Each peripheral has its own driver program and each driver is designed to handle one unit of information at a time. For the terminal this is a single character, for the printer it is assumed to be a line, even if it is a character printer, and for the diskette it is a block, the size of which may vary from computer to computer. The particular details of each peripheral are hidden behind this interface within the actual driver programs.

PORTOS recognizes the following primitives to drive the VDU, the printer, and the diskette unit:

GET CHAR

reads a single character from the VDU keyboard and, if necessary, echo-prints it on the screen.

PUT CHAR(DATA)

prints the single character contained in DATA on the VDU screen.

PRINT LINE(BUFFER, N)

Prints N characters from BUFFER, or until CR is encountered, on printer.

DISKOP(OPER, DEVICE, SECTOR, ADDR)

reads (if OPER=1) or writes (if OPER=2) one block of information from or to the diskette in DEVICE (0 for system and 1 for user) into or from the memory location ADDR. The address on diskette is given by SECTOR.

4.4.2) Interrupt Handlers

Interrupts are generated internally by events such as hardware check, illegal instruction, and real-time clock, or externally by peripherals. The system has programs to handle these interrupts, so as to ensure correct system execution. It is their responsibility to service the relevant interrupt or request, and return to the interrupted program or exit to the operating system, as appropriate.

PORTOS provides the two primitives, INTERRUPT ON and INTERRUPT OFF, to enable and disable external interrupts, respectively. To control the real-time clock, the primitives CLOCK ON and CLOCK OFF are

provided.

4.4.3) Machine Operations

The following primitives are provided to perform the machine operations relevant to the execution of PORTOS.

RESET MACHINE

resets or initializes the hardware as appropriate, which may be done by either loading certain registers or executing priviledged instructions.

SUPERVISOR CALL

generates a software interrupt

LOAD PC(ADDR)

loads the program counter with the contents of ADDR.

Also included in this group is the bootstrap program. Its function is to reset the hardware, initialize the memory as expected by PORTOS, and transfer control to the kernel. The bootstrap program is classified as being hardware dependent, because it has to conform to the architectural requirements of the host computer.

CHAPTER 5 : SYSTEM DESIGN ONTO PRIMITIVES

This chapter presents the design of the mechanism by which the system facilities explained in Chapter 4 are employed by PORTOS during execution.

5.1) Memory Residency

PORTOS primitives are classified as either memory-resident tasks or diskette-resident system routines. Tasks are read into memory during the initial startup and remain there for the duration of the user's interaction with the computer. System routines are brought into memory when needed as part of command handler programs and they are removed afterwards.

Limited memory space makes this distinction necessary. Ideally, the whole of the operating system should be memory-resident for efficiency. The area occupied by the system, however, is lost to the user who needs it for software development. Total memory residency is therefore impractical, and sometimes impossible, except for very small operating systems or process-dedicated computers.

The important considerations are the choice of these tasks and their management during execution. The minimum requirement that the tasks have to satisfy is the ability to recover from any previous state and retain control of the computer. In addition, frequently-used

primitives may also be designed as tasks to provide fast service for system routines. Naturally, all interrupt handlers are also memory-resident. They are not recognized as tasks, however, because their operation is not controlled by PORTOS.

5.2) System Tasks

The PORTOS primitives designed as tasks are listed in Figure 5.1. They constitute the nucleus that is responsible for the initial startup and subsequent error recovery operations, as well as the interpretive execution of the PIL commands by the handler programs.

Figure 5.1 : PORTOS Tasks

Task Number	Task Name
1	ALLOCATE
2	FREE
3	INSERT
4	GIVE SECTOR
5	RELEASE SECTOR
6	EXIT
7	LOAD
8	DISKOP
9	GET RECORD
10	PUT RECORD
11	SHUTDOWN IO
12	FETCH TIME

The memory manager assumes control of the available memory space before all else in order to service the requests from the rest of the system. The bootstrap program, having installed the tasks in pre-defined memory locations, passes the control to the EXIT task. Its

function now is to determine the exit status, reset the system if necessary, and initiate the PIL decoder.

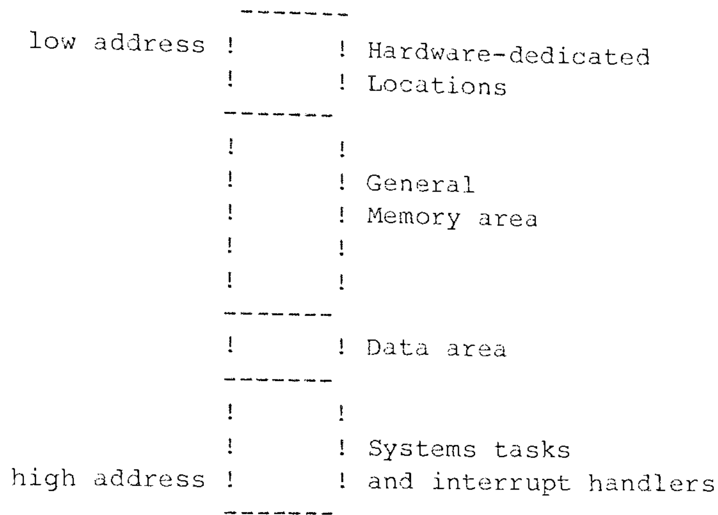
Command handler and user programs are all treated in exactly the same way by the tasks. They are loaded into memory, albeit from different diskettes, and given control. During progressing, programs may request service from any task any number of times. On completion, they always return control to the EXIT task, which then initiates the PIL decoder as before.

The SHUTDOWN IO task is part of the error recovery operation. All channels are closed on error exit to the operating system in order to ensure file integrity. The DISKOP task is needed by the CPU management tasks to access the diskette. It also services all subsequent diskette input-output requests. The diskette management tasks provide service for both the file and input-output managers, while the GET RECORD and PUT RECORD tasks serve all system routines wishing to communicate with the user.

5.3) Memory Organization

The sharing of the main memory during execution is shown in Figure 5.2. The hardware-dedicated locations are used by the computer for such functions as interrupt vectors, registers, etc. The size and details of this area are totally machine dependent. PORTOS occupies the top end of the memory for its data and code. The area in the middle is available to both user and system programs for general use.

Figure 5.2 : PORTOS Memory Organization



The data area of PORTOS is divided into dedicated and general-purpose locations. The dedicated locations are used by PORTOS for its tables, system variables, pointers, etc. The general-purpose locations are used for passing parameters between programs and tasks, as well as between the tasks themselves.

The tasks are placed in memory in the same order as shown in Figure 5.1. The order in which the interrupt handlers are installed is not important providing the interrupt vector locations are initialized accordingly. These functions are the responsibility of the bootstrap program.

5.4) Task Management

After they have been installed in memory, the tasks remain

dormant (or inactive) until they are "activated". Any task can be activated at any time. It will run to completion and return control to its "activator", which may be either a system program or another task. All activation and subsequent de-activation requests are handled by the task manager, which effectively functions as a switchboard between the tasks and their activators.

To activate a task means setting up the relevant parameters and passing the control to it, which is actually a two-part operation. Any program wishing to activate a task makes a request to the task manager using the identification number of that task (Figure 5.1). This is the "logical" first half. The task manager then determines the memory location of the task in question and passes the control to it. This is the "physical" second half.

The task manager is, therefore, the translator of logical requests into physical activations. It keeps track of which tasks have been activated and in what order, so that the flow of control is not lost. For this purpose, a table, called the "task vector table", is maintained in the system data area to monitor task entry points, return addresses, and the originator of every activated task.

A task may activate any number of other tasks but recursion or the re-activation of a previously activated task is not allowed. A task must complete its operation before it can be re-activated.

The transfer of control to the task manager itself is achieved through the software interrupt mechanism of the host computer. Task activation requests are converted by PORTOS into software interrupts.

The hardware response to software interrupts is to generate a "trap" and transfer control to a handler routine, which in this case happens to be the task manager.

Two levels of parameter passing are involved in task activation. Firstly, the manager has to know the identification number of the task to be activated; and secondly, the tasks themselves need to send or receive data as part of their service. This communication takes place via the system data area. The information that the task manager needs is in a dedicated location but the tasks themselves use the general-purpose locations for parameter passing.

There is only one "active" task in the system at any time. If more than one task has been activated in the course of an operation, then the most recent task is active and the others are temporarily dormant. The return of control from the active task is always to its "activator", unless an error is detected, in which case control is returned to the EXIT task to determine the cause.

The mechanism by which the active task returns the control to its activator is to request the activation of "task zero". Since no task with that identification number exists, it is reserved to designate "the activator of the current active task". The manager removes the active task from the chain, restores the state of the CPU to that of its activator, and resumes execution with the new active task.

5.5) Logical Portability of PORTOS

PORTOS is now ready to be implemented. Before proceeding with the implementation details, however, it would be appropriate to assess the logical portability of the system. The term "logical portability" means freedom from orientation towards any particular computer during design. Physical portability is achieved as a result of this logical portability if it is accompanied by the use of a high-level implementation language.

PORTOS is logically portable at this stage because its managers are built on machine-independent algorithms that can be implemented on different computers. The file and input-output managers, for example, are portable owing to the high-level nature of their functions. They are orientated towards the PIL virtual computer and have no contact with the hardware in any way.

The lower-level managers, on the other hand, are made portable through the recognition of the simplest and most common architectural characteristics so as to maximize mobility amongst the potential host computers, the abstraction of those hardware features that they control, and freedom from dependence on hardware support.

The memory manager, for example, assumes the memory space to be one contiguous sequentially-addressable block and initializes its free partitions list accordingly. Such a configuration is applicable to all computers and no hardware support is needed for its management. Also, the free partitions list can be initialized to reflect other memory organizations. Providing the pointers are set up correctly, the

manager would remain unaware of any such changes. This freedom, however limited it may be, naturally increases the flexibility and therefore the portability of the manager.

The diskette manager, on the other hand, by viewing diskettes solely in terms of sectors, hides from PORTOS the existence of hardware-dependent features such as recording surfaces, cylinders, and tracks. This is an abstraction to which any diskette can be made to conform, simply by initializing the status array accordingly.

The manager is thus freed from having to reflect any particular organization. Only during the actual diskette access operation is it necessary to acknowledge the addressing requirements of the host computer. The manager, therefore, owing to its logical addressing scheme, can be implemented on any hardware and confine the physical addressing of sectors to some low-level machine-dependent routine.

The CPU and task managers assume, as the only hardware support for their operations, the ability to access registers and control software interrupts. These simple requirements can be easily satisfied by all computers and hence present no portability problems.

PART II : IMPLEMENTATION

CHAPTER 6 : IMPLEMENTATION CONSIDERATIONS

6.1) Choosing The Implementation Language

The traditional approach to systems programming on microcomputers is to use the host assembly language. Assembly languages, being transparent, enable the creation of very compact and efficient programs. On the other hand, they are totally machine dependent. Software implementation on a different computer would mean the complete rewriting of the same programs, which is a wasteful and unnecessary exercise.

Machine-independent assembly languages, such as SICTRAN [25], with the syntactic structure of compiler languages but the facilities and operations of a simplified assembly language, have failed to gain acceptance. When portability is an objective, therefore, the use of a high-level language becomes mandatory. This may be

1) a systems programming language,
2) a modified applications language to suit systems programming, or
3) an existing applications language.

The two most important requirements are availability and suitability. Other considerations are the efficiency of the resulting object code, support for the language, ease of learning, and program

maintainability [17, 29, 87].

6.1.1) Systems Programming Languages

Languages that have been designed specifically for systems programming were naturally the first choice, because of their particular orientation to the function.

Languages like PL/360 [80] and STAB-1 [20], however, were excluded from consideration because of their dedication to a particular architecture (in their case, the IBM/360 and DEC PDP-11 respectively) and lack of portability. Languages like Bliss [100], designed with a particular architecture in mind but following a machine-independent philosophy, were also excluded. Although Bliss has been implemented on several computers, including an IBM/360 [77], to write compilers and operating systems, it is nevertheless considered a low-level and limited language [8, 99].

Taken into consideration were languages like PASCAL [22], BCPL [73], and their derivatives. PASCAL, for example, is considered by some to be the best language for systems programming [94]. Concurrent PASCAL [61], built for portability, has enabled SOLO to be implemented with less than 4% of it in assembly language, thus making the system highly portable [69]. BCPL, on the other hand, has been used successfully on the OS6 and TRIPOS operating systems, while its derivatives C and Eh have been used on UNIX and Thoth respectively.

However, none of these languages was available at the time, and it was decided to concentrate on a more accessible language.

6.1.2) Modified Languages

These are the languages that have been derived from an otherwise suitable "parent" applications language to meet the requirements of systems programming. Modifications to the parent language take the form of additions, deletions, and possibly minor syntactic changes.

Several modified languages were considered, because of their applicability, and are therefore discussed briefly below. Nevertheless, they also had to be rejected since they were unavailable.

Some of the languages in this group are listed in Figure 6.1. LRLTRAN, for example, has been used on the FLOE operating system for the CDC 7600 and the FROST operating system for the CDC 6600 computers, with very little machine coding in either case. PS440 has been used on the AEG TELEFUNKEN software, while CHILI has been used on the CHI operating system.

Figure 6.1 : Modified Languages for Systems Programming

Language	Parent
LRLTRAN [26]	FORTRAN
LITTLE [84]	"
EXT.FORTRAN [7]	"
EXT.ALGOL [76]	ALGOL60
ESPOL	"
DC-ALGOL	"
PS440 [78]	"
XPL [68]	PL/I
PL/S	"
SABRE PL/I	"
PL/C	"
ISPL [4]	"
CHILI [53]	"
SYSL [87]	"
ALGOL68-R [41]	ALGOL68

FORTRAN is a good parent language because of its widespread availability and support, the closeness of its constructs to machine language for an efficient implementation, and its ability for compilation directly into machine code [7, 84]. Block structured languages, on the other hand, are preferable as parent languages because of their superior syntax, powerful control structures, and varying data types. For further information on the derivatives of ALGOL60 and PL/I the reader is referred to Bergeron [8], Sammet [77], and Sammet [76].

6.1.3) Existing Applications Languages

In general, the managerial suitability of these languages is

overshadowed by their technical shortcomings. FORTRAN, for example, is the most widely used language, in spite of the incompatibilities that exist between implementations [70, 75]. It has been used by the NASA Electronics Centre, ERC, to write several operating systems for their HONEYWELL 516 and 832 computers [76, 80]. However, its lack of facilities for hardware access, bit and byte handling, strings, etc. make the language essentially unsuitable for systems programming. Library subroutines to overcome these deficiencies vary so much in vocabulary that portability suffers as a result [36].

PL/I, on the other hand, with its object code modularity, bit and pointer declarations, interrupt handling and memory management, is highly suited to systems programming [8, 76]. It has been used for the MULTICS operating system [23], as well as all the control software for the General Motors Company on their IBM 360/370 computers. PL/I is unsuitable for use on microcomputers, however, because of its large compilers, inefficient object code, and the assumption of an environment that causes implementation problems [42, 98].

CORAL66 (henceforward referred to as CORAL) is a notable exception. Designed for real-time minicomputer applications, its facilities are highly suitable for systems programming. Furthermore, this language optimally fulfilled the selection criteria, and was therefore chosen to implement PORTOS.

6.1.4) CORAL As A Systems Programming Language

CORAL is a standardized language, the official definition of

which is given in the "Blue Book" [97]. It is a machine-independent kernel language with extensions towards the host system. Its compiler is widely supported on numerous computers, including microprocessors, some of which are listed in Appendix D [2, 49]. Portable compilers have also come into existence [85]. CORAL is supported by several user and manufacturer groups to ensure continued availability.

Based on ALGOL60 with ideas from FORTRAN and JOVIAL, the familiar syntax and semantics of CORAL make it a very easy language to learn. Being a block-structured language, it lends itself to top-down design and structured programming. It is particularly well-suited to modularity, which aids software development and maintenance.

CORAL has facilities for memory access, address manipulation, bit and byte handling, in-line assembly coding, macros, and inter-segment communication. The built-in macro processor of CORAL provides for a limited but very useful form of extensibility [11, 15, 63]. It greatly enhances the portability of source programs because machine-dependencies can be coded as macro calls and then extended towards any hardware by appropriately altering their definitions [18, 90].

CORAL is a simple language with a small and efficient compiler. Studies of sample (non-scientific) programs have shown that the resulting object code from a CORAL compilation is, on average, about 25-30% longer than the equivalent assembly program. Webb, however, states that this figure may be as high as 40% [93].

In order to minimize the size of the language and its compiler, machine-dependent features such as input-output, file access, interrupts, memory management, etc. are not included in the language. CORAL thus requires a set of library procedures in order to run on the host system.

CORAL has been used as the implementation language for the MASCOT operating system on the Marconi Myriad computer [94].

6.2) Host Computers

Availability was a decisive factor in selecting not only the implementation language but also the host computers. Two requirements had to be satisfied for a successful implementation. The computer had to have the same configuration as the target specified in Section 2.1.3 and it had to support CORAL. This latter requirement arose because PORTOS does not have its own compiler yet and is, therefore, dependent on external availability.

The first requirement was easily satisfied by the microcomputer systems available at the time. The second requirement, however, presented problems. None of the computers supported CORAL and efforts to locate suitable cross-compilers were unsuccessful.

Faced with this situation, two minicomputer systems, a Texas 990/10 and a PRIME 300, were chosen as a compromise. They both resembled very closely the actual target configuration, were to become

supporters of CORAL, and were readily available. Given below are brief descriptions of these computers. For more detailed information the reader is referred to the relevant system manuals ("Model 990 Computer Assembly Programmer's Guide" ISBN 0-904047-17-2 for the Texas 990/10, and "PRIME 300 System Reference Manual" for the PRIME 300).

6.2.1) TI 990/10 Computer System

The host Texas Instruments TI 990/10 is a byte addressing computer system. It has a 16-bit word length and a memory capacity of 32K words. The system has a standard-ASCII user terminal, with several special-purpose keys, and a hard-copy system terminal. The mass storage device is one 1.5 million-word disk drive.

The memory organization of the TI 990/10 is given in Appendix E-I. The architecture has three hardware registers: the program counter (PC), the workspace pointer (WP), and the status register (ST). A workspace is any 16-word area of user memory. When the starting address of this block of memory is placed in the WP register, its contents become the workspace registers 0 through 15. These registers may be used as general-purpose arithmetic registers, address registers, or index registers.

The TI 990/10 computer uses vectored mode interrupt processing, with either 8 or 16 priority levels. Two memory words are reserved for each interrupt level. When an interrupt occurs, the contents of the first word are placed in the WP register and the contents of the second word are placed in the PC. The previous values of WP and PC are

saved in the Workspace Registers 13 and 14 of the new workspace respectively. This form of transfer of control is called a "context switch".

Data transfers to and from either terminal are achieved through the Communications Register Unit (CRU). The selection of different terminals is made on the basis of the address at which each terminal is attached to the CRU. This is called the "CRU Base Address", and it is contained in Workspace Register 12 of the current workspace. Having made the device selection, the programmer may then use any of the following instructions to perform the required function:

Figure 6.2 : TI 990/10 I-O Instructions

Instruction	Function
LDCR	load communications register (output)
STCR	store communications register (input)
SBO	set bit to 1
SBZ	set bit to 0
TB	test bit

The data transfers to and from the disk unit use Direct Memory Access (DMA) channels called the TILINE. The disk controller has a set of registers, located in high memory, which are initialized under program control with the values relevant to the intended transfer. The controller is then initiated to access and obey the contents of its registers. The memory access instructions are used to communicate with the disk controller.

The hardware bootstrap program is initiated from the front control panel of the computer. The input device for the bootstrap loader is the cassette unit, of which there are two mounted on the system terminal. When initiated, the microcode program reads an executable binary module from cassette unit CS1, places it in memory starting at location hexadecimal A0, and passes the control to it. The program that is read from cassette is entirely at the discretion of the programmer.

The assembler can produce either absolute or relocatable object code. The output from the consolidator (or LINK EDITOR) is relocatable binary in ASCII format, with the load information supplied by various tags within the module.

6.2.2) PRIME 300 Computer System

The host PRIME 300 is a word addressing computer system. It has a 16-bit word length and a memory capacity of 32K words. All terminals, user and system, are standard ASCII, with no special keys. The system had one hard-copy character printer. The mass storage devices were two 1.5 million-word disk drives, one fixed and one removable.

The memory organization of the PRIME 300 is given in Appendix E-II. The locations 0 to octal 37 make up the "High-Speed Register File", which contains such hardware registers as the accumulator, program counter, etc. The main memory is organized into groups of 512 words each, called "sectors", for paging. The CPU is capable of operating in two different addressing modes: sectored (S) or relative

(R), either of which can be selected under program control. Interrupt processing can be vectored or non-vectored, the selection of which can again be made under program control. Interrupts may be enabled or disabled individually or in groups.

The control of peripherals and the transfer of data are achieved using the following instructions:

Figure 6.3 : PRIME 300 I-O Instructions

Instruction	Function
INA	read character into A register
OTA	write character from A register
OCP	output control pulse
SKS	skip if satisfied

Each device has a unique "device code" (or address) for identification purposes. Selection is made depending on these values, which appear as the arguments of the I/O instructions. For example, the user terminal has a device code of octal 4 and the disk controller octal 21.

When initiated from the control panel, the hardware bootstrap program reads the contents of sector 0 on disk 0 into location octal 770, and begins execution from location octal 1000. The binary program which is read from the disk in this manner is entirely at the discretion of the programmer.

The assembler produces code to execute in R-mode. Any location in sector zero or within the range P-239 to P+256, relative to the program counter P, may be referenced directly. The locations outside these limits, however, can only be referenced indirectly via sector zero. The output of the consolidator is absolute binary in octal, with the load information given at the beginning of each module.

6.3) Implementation Subset

Time and manpower constraints have already placed restrictions on the specification of PIL. The details of its software development commands have had to be left largely undefined. The chosen host computers place further restrictions on PIL. The MOVE command, for example, would become redundant when there is one disk drive to support the system.

In other words, only a subset of PIL could be implemented meaningfully at this stage. The objective would be to demonstrate the usability of PIL, and give the user a "preview" of the overall language.

While it was possible to subset PIL at personal discretion, however, the subsetting of PORTOS could not be taken much further than orientating the system towards the single-disk configurations available at the time. It was found that the bulk of the system had to be implemented in order to support even a very limited subset of PIL commands. This is because different commands are supported by the same managers and primitives, and each command individually exercises the

various facilities of the kernel.

It was decided, therefore, to implement PORTOS adhering closely to the original specifications, and then create as many command handler programs as possible above it, given the prevailing time and manpower constraints. It was also decided to treat those parameters that would be superfluous for this implementation as dummy variables. The system would then suffer the least disruption when it was subsequently implemented on a target configuration.

CHAPTER 7 : THE EARLY EXPERIMENTS

7.1) Foundations of PORTOS

The first programming efforts were made on the University's ICL-1904S mainframe computer, under the control of GEORGE 3 operating system. This decision was taken because the only CORAL compiler available at the time was on this machine [44]. The objectives were, firstly, to become familiar with the CORAL language and its capabilities, and secondly, to learn the concept of systems programming and gain implementation experience.

It was also realized, however, that any program written on the ICL-1904S could be transferred to the host minicomputer and used as part of the actual implementation. This course of action would not only help to save time, but also provide valuable experience in software transportation. Consequently, the coding of PORTOS was initiated on the ICL-1904S and care was taken from the beginning to write portable programs.

Modularity was adopted as the basic implementation methodology, not least because modularized programs are easier to write, debug, and test. Firstly, it enabled the localization of operations and dependencies in source programs, in the knowledge that modifications would be necessary after their eventual transfer. Secondly, since not all the programs written on the ICL-1904S would be transferred, modularity enabled the separation of long- and short-term programs without disruptive side-effects. Thirdly, it enabled the long-term

programs to reflect the modular structure of PORTOS design.

In the following presentation, all programs written on the ICL-1904S as part of PORTOS will be called system routines (or simply routines).

7.2) PORTOS On The ICL-1904S

It was decided to start the implementation of PORTOS with the low-level managers and their primitives. The objective was to use them as the "building blocks" with which to construct and test the higher levels of the system. Confidence in low-level operations was essential in order to develop reliable software.

The diskette and memory managers were chosen to initiate the coding. These managers, while not machine-dependent themselves, nevertheless exercised first-hand control over the hardware. The first step, therefore, was to write a suite of CORAL programs to simulate the respective hardware features of the host computer, using the same abstraction as employed by each manager.

Following the implementation of the simulators, the relevant management routines were written to access them. Finally, a set of validation programs to test the operation of these routines were written, again in CORAL. On the ICL-1904S, PORTOS thus consisted of three sets of programs.

7.2.1) Implementation Methodology

The simulators were written as procedures. Any routine wishing to access a hardware feature had to call the relevant simulator, and pass to it the required parameters in the same order that was envisaged to exist on the host computer. These simulators were expected to be replaced subsequently by machine-dependent assembly programs to perform the same operations, but without changing their software interface to the rest of PORTOS.

The system routines themselves were also written as procedures. The objective was to call them in the same manner as they would be called in the actual implementation. Furthermore, developed as "black boxes", these routines would be unaware of the changes in the environment. In other words, the effect of their calling either a simulator or an actual assembly program would be the same. Similarly, it would become irrelevant whether it was a validator program or an actual system routine that required their services.

The validators, on the other hand, were written as master segments in order to consolidate the routines and simulators into executable programs. Their sole purpose, together with the simulators, was to ensure the correct execution of PORTOS routines.

Given below is a more detailed description of the programs that were developed, with an evaluation of their influence on the final system.

7.2.2) Simulator Programs

The first hardware feature to be simulated was a disk unit and its controller, using a one-dimensional integer array. The array was arbitrarily split up into groups of 32 elements each, to represent the sectors to or from which data could be transferred. For this experiment, a total of forty sectors were catered for (another arbitrary number).

The disk controller was simulated by a procedure to access this array. The calling routine had to specify the sector number, the operation required (read or write), and the memory address involved. The actual calculations to access the correct group of elements were confined to this procedure and hidden from the caller.

The main memory, taken to be a single contiguous block, was simulated using another one-dimensional integer array. Word-addressing was assumed as the access mechanism, with no provision for dedicated locations or hardware protection.

The array representing the disk was local to its simulator, because no other routine had or needed access to it. The memory array, on the other hand, was declared as a global variable in the common area because it was shared by different procedures. The memory manager needed access to it in order to maintain its free partitions list and the disk controller accessed it to perform its data transfers.

The user terminal was the only other hardware feature that was simulated. GEORGE 3 library procedures for input-output were used to

drive the terminal. The simulator only translated the PORTOS requests into the form required by these procedures and vice versa.

Interrupts were not included in the simulation exercise for two reasons. Firstly, the hardware already being simulated was sufficient to meet the implementation requirements at this stage. Secondly, before the point where interrupts would have entered the picture was reached, the compiler on the Texas 990/10 became available and the development was continued on this minicomputer. There was no need for simulation then because the actual interrupts themselves were used.

7.2.3) PORTOS Routines

The first routines to be written were the disk management routines, built around the Disk Status Array. CORAL does not have bit arrays. The Disk Status Array was therefore declared as an integer array, each element representing one track and the bits within each element representing the sectors on that track. Eight sectors were taken to constitute one track, which is a common configuration.

This representation of the disk, in terms of tracks and sectors, is closer to the actual configuration found on real computers. Nevertheless, it is still sufficiently abstract in itself so that the logical portability of the manager is not damaged. Given the limitations of CORAL, therefore, it was a convenient compromise to make between the abstraction adopted by PORTOS and the hardware.

Each sector was associated with a bit in its corresponding array

element from right to left. In other words, the rightmost bit represented sector 0, the next bit to the left represented sector 1, and so on. The index value of the element within the array indicated the track number. The conversion to and from the logical sector addressing employed by PORTOS was coded into the routines as appropriate.

Bit manipulation is a standard feature of CORAL. The availability of such operators as MASK (And), UNION (Inclusive or), and DIFFER (Not equivalent) meant that the routines could be written exclusively in Standard CORAL, thus offering high physical portability.

Next to be written were the memory management routines. Their coding demonstrated the power of another CORAL feature, the anonymous reference facility, without which the implementation would have required the use of assembly instructions. This construct offers the programmer the same power as the host assembly language to handle absolute addresses and their contents.

CORAL, like PORTOS, implements a word-addressable virtual computer. If, for example, ADDR is a pointer to a free partition in memory, then [ADDR] contains the size and [ADDR+1] contains its pointer to the next partition. The expression

```
ADDR := [ADDR+1]
```

is the link necessary to scan the free partitions list. As a result, the accessing of the memory array and its management presented no problems and the routines themselves were made highly portable in

Standard CORAL.

After the disk and memory management routines were completed, the development of the file manager was initiated. The routines to manipulate directory entries were written and tested, to perform such functions as the creation, retrieval, and deletion of file references.

Software development on the ICL-1904S was terminated at this stage, with the arrival of the CORAL compiler on the TI 990/10.

7.2.4) Validator Programs

Each routine that was to become a permanent part of PORTOS had a corresponding validator program. Their function was to test the routines for correct execution under all possible conditions.

The simulation exercises were run interactively, and it was the responsibility of the validator programs to maintain communication with the author. The input was in the form of commands relevant to the operation of the routine being tested. One command at a time was accepted. After checking the validity of each command and its arguments, if any, the appropriate routine was called to process this input. Upon completion, the results were evaluated and the author was informed of the outcome, either by the display of a message or printing of a table, as appropriate.

The input from the author, in effect, simulated the demands that

would be made on the particular primitive operation during normal execution. It was his task to issue the commands in a similarly unpredictable manner.

Input-output was achieved using the library procedures provided by the resident CORAL library. The terminal simulators written for this purpose were not used, because the development of the PORTOS input-output routines was postponed in favour of using the resident procedures. This approach was adopted in order to hasten the testing of the routines.

7.3) Transfer To The TI 990/10

When it was time to move to the TI 990/10, only the system routines were transferred. The simulators and the validators, being special-purpose programs that had no use outside their pre-defined environments, were abandoned. One exception to this rule was the validator of the file management routines, which was the most comprehensive of all the validators. It was transferred to the TI 990/10 as part of the system and developed further to become the PIL decoder.

Two factors crucial to the physical transferability of programs are compatibility of the transfer media and the character set of the data being transferred [91]. The routines developed on the ICL-1904S were transported to the TI 990/10 on paper-tape, which was the only compatible medium between the two computers. The data transferred were ASCII source listings. The character set of CORAL being well-suited to

paper-tape representation, and owing to the availability of the necessary handler software on both computers, the transfer was easily achieved.

The routines developed on the ICL-1904S have remained basically the same throughout their subsequent employment, first during the implementation of the rest of PORTOS and later during execution. Their success in coping with the demands made on them was taken as a favourable reflection on the development methodology.

CHAPTER 8 : PORTOS IMPLEMENTATION AND PORTABILITY

8.1) PORTOS on the TI 990/10

Following the transfer from the ICL-1904S, the development of PORTOS was completed, and the first implementation realised, on this computer. The first step was to replace the simulators with assembly language programs, to support the system on this particular architecture. In other words, the hardware interface on which PORTOS had to exist was defined.

The next step was to re-compile and implement the transferred routines. They were subsequently used to develop and implement the rest of PORTOS. Certain modifications had to be made in their source code before compilation, in order to conform to the requirements of the new compiler and its environment. These modifications did not affect the algorithm of the routines or their internal details.

Next to be implemented was the input-output manager. As communication with the computer was again necessary in order to debug and test the routines being developed, it was decided to use the PORTOS input-output routines themselves. It would have been wasteful and pointless to postpone their implementation any further, by using the compiler-supplied procedures.

The file manager was implemented next, followed by the PIL interface, using the routines already implemented. When PORTOS was

eventually ready for independent execution, the CPU and task managers were implemented, in close conjunction with the system format on disk and in memory. The implementation was completed by the actual creation of the system disk, from which PORTOS could then take over.

The current implementation of PORTOS on the TI 990/10 consists of 54 individually compileable segments. It was realized in approximately 2360 lines of source code and 200 assembly language instructions. On disk, PORTOS occupied 134 sectors. Its implementation details are presented in the following section, together with its transfer procedure.

8.2) Transfer of PORTOS to the PRIME 300

PORTOS was transferred from the TI 990/10 to the PRIME 300 in order to test and confirm its portability. The transfer medium was again paper-tape, on which the source listings of each individual segment (or program) was punched separately by the TI 990/10 handler software.

The PRIME 300, however, did not provide the necessary software for its paper-tape reader. A program had to be written for this purpose, in FORTRAN and PRIME assembly language. FORTRAN was used in order to take advantage of its library procedures for file access. The assembly language was used to drive the paper-tape reader.

This program, called READP, the source listing of which is presented in Appendix M, was designed to read one segment at a time

and store it in the file referenced by the implementor. It was not written intentionally for being portable, but it can be used on other computers if its file access and assembly instructions are suitably modified.

8.2.1) Source Code Modifications

The essence of the implementation procedure is to compile PORTOS on the host computer, and create its system support disk. The next step was, therefore, to make these segments conform to their new environment.

The implementation has four levels of machine-dependence inherent in its source code. They are, firstly, the machine-dependent segments that have to be re-written in the host assembly language; secondly, the machine-orientated segments that have to be suitably modified; thirdly, the "parameters" that describe the host computer to PORTOS; and lastly, the compiler requirements.

8.2.1.1) Machine-dependent Segments

The peripheral drivers, interrupt handlers, and all other assembly language routines belong to this group. Some implementors consider as portable those programs that are short or straightforward enough to be re-written easily and quickly [86]. The author prefers to call this "algorithm portability", and regards such programs as being non-portable. These hardware interface routines were, therefore,

re-written in the PRIME 300 assembly language. Their source listings are presented in Appendix K.

8.2.1.2) Machine-orientated Segments

Machine-orientated segments are those that are algorithmic in nature, and hence written in CORAL, but still reflect some characteristic of the host computer. The task manager and loader, for example, both contain typical examples of this kind of dependency. Their source code has to be modified to fit the particular environment.

The task manager, in response to the software interrupts, has to preserve the state of the interrupted program. To do so, depending on the hardware registers that need to be saved, it has to maintain tables of varying length and entries. Hence, the source code has to be altered to reflect the architecture. The system loader contains similar dependencies, because the conventions used by the current hosts of PORTOS are different. So much so, that the loader had to be re-written almost completely after the transfer.

Owing to modularity, however, the required modifications could be made easily and quickly, without the rest of PORTOS becoming aware of these changes.

8.2.1.3) Segment Parameterization

The parameterization of PORTOS segments is achieved using the macro facility of CORAL, thus localizing all such dependencies to "macro definition files". The implementation has four such files: PARAMS, CHARS, LINKS, and DBASE. They are presented in Appendix G. It is necessary to modify only the contents of these files and not individual segments.

The PARAMS file, for example, contains such parameters as the number of tracks on disk, the sector size, the boundaries of the user memory, etc. The CHARS file contains the ASCII control characters, such as carriage return, line feed, and del, that are used during input-output. Although the ASCII character set is standardized, conventions regarding the use of the parity bit differ. PORTOS being dependent on different host compilers, it is essential to make the system fully compatible with the host compiling system.

The LINKS file contains the definitions for task communication linkage mechanism. The coding of task activation requests is exactly the same in appearance as a procedure call, which it is, in effect. What to the programmer is a normal call, however, has to be converted into a software interrupt, with the correct execution parameters. The contents of the LINKS file provides this information to the compiler. The actual linkage definitions, being PORTOS-dependent but machine-independent, need not be modified. The software interrupt definition itself, however, which is also contained in this file, must be modified accordingly.

The D3ASE file contains the definition of the memory-resident data area of PORTOS, required by any routine wishing to access it. This definition, however, like the LINKS file, need not be modified after every transfer, except to relocate the data area, if necessary.

8.2.1.4) Compiler Requirements

Although CORAL is a standardized language, differences were found to exist between implementations. These differences can be divided into three groups: internal, external, and additions.

The internal differences affect keywords, strings, and communicator declarations. External differences influence the program format and compiler directives. Additions, although powerful, damaged source portability because support for those facilities is not guaranteed. Byte arrays and records, for example, which are examples of "BLANDFORD Extensions", were not available on the PRIME 300. A comparison of the CORAL keywords on different systems is presented in Appendix F.

Keywords were affected, firstly, by abbreviations, such as PROC for PROCEDURE, and secondly, by those additions that either reflect some characteristic of the machine, such as HEX, or enhance readability, such as GE instead of ">=". Strings were affected by three factors: internal representation, the syntax of control characters within strings, and the acceptability of the ASCII space as a printable character.

The macro definition files are introduced into PORTOS segments using the LIBRARY communicator declaration. This communicator accepts the relevant filename as parameter. Since the syntax of the filename had to conform to the requirements of the host operating system, all such declarations had to be suitably modified. The external differences influenced the format of source segments, and the manner in which compiler directives could be issued.

8.2.2) Compilation and Disk Creation

The next step is to compile and consolidate these modified PORTOS segments, the listings of which are presented in Appendices H, I, and J. The PIL interface and task segments are compiled as main programs (or master segments). All other segments are compiled as subroutines (or procedure segments), and consolidated with the appropriate main programs into executable binary modules.

The last step is the creation of the system disk. This process consists of formatting the disk as shown in Figure 8.1, and writing to it the binary modules previously created. A CORAL program called BUILD, which is presented in Appendix L, was written to perform this function.

Figure 8.1 : PORTOS Disk Sector Allocation

Sector	Name	Function
0	BOOTS	Bootstrap Program
1	STATUS	Disk Status Array
2	UDIR	User Disk Directory
3	SDIR	System Disk Directory
4	....	Start of General Storage

Although a single disk was used for system support, it was decided to have separate system and user directories for two reasons. Firstly, to preserve the logic of the PIL virtual computer, by maintaining the logical separation of these conceptually different entities. Secondly, to protect PORTOS from direct user accessibility, as it was originally intended. When the system is implemented on the target configuration, there will be one directory per diskette, and the general storage area will start from sector 3.

The algorithm for disk creation is as follows. Initially, the disk is assumed to be empty and the Disk Status Array initialized accordingly. The two directories are also initialized to nil. The modules are then read from the host system and written to disk, allocating sectors as needed. An entry is made in system directory for each module, so as to obtain a self-contained disk at the end of the process.

The order in which the modules were written to disk is given in Figure 8.2 below.

Figure 8.2 : Order of Residency on Disk

Module Number	Module Name	Function
B	Boots	Bootstrap Program
0	Task Manager	Task
1	Allocate	"
2	Free	"
3	Insert	"
4	Give Sector	"
5	Release Sector	"
6	Exit	"
7	Load	"
8	Diskop	"
9	Get Record	"
10	Put Record	"
11	Shutdown Io	"
12	Fetch Time	"
13	Handy	Interrupt Handler (Illegal instruction)
14	Timer	Interrupt Handler (Real-time clock)
15	Dcoder	PCL decoder
16	Xer	PRL decoder
17	Copy	Command Handler
18	Create	" "
19	Delete	" "
20	Files	" "
21	Help	" "
22	Intime	" "
23	List	" "
24	Lock	" "
25	Print	" "
26	Rename	" "
27	Space	" "
28	Time	" "
29	Unlock	" "
30	P$hlpf	ASCII file for HELP Command
31	P$fhan	" " "
32	P$sdev	" " "
33	P$inqr	" " "
34	P$syst	" " "

The tasks and interrupt handlers must be written to disk in the order shown, for the following reason. Each task has a unique identification number, by which the activating routines recognize it. The task manager, however, simply activates the tasks according to their order in its vector table, unaware of the individual number of

any one. The bootstrap program, which is responsible for setting up this table, is also unaware of their identification numbers. It installs the tasks one after the other as they appear on the disk and, assuming that they are in order, initializes the vector table accordingly. Hence, if their identification numbers do not agree with the order in which they are installed, unpredictable results would ensue during execution. The other modules of the system, however, may be in any order.

8.3) A Comparison of the Two Implementations

Quantitative details related to the existing implementations of PORTOS are presented in Figure 8.3 below.

Figure 8.3 : Implementation Figures

	TI 990/10	PRIME 300
Segments	54	54
CORAL Source	2360	2360
Assembly Code	200	940
Sectors on Disk	134	81
Memory Size (Code and data)	3400	2172

The resulting efficiency of the PRIME 300 implementation, both on disk and in memory, can be attributed to several factors. Firstly, the architectural characteristics of PRIME 300 and its machine instruction set provide for more compact binary programs, compared with the TI 990/10. Secondly, different code generation techniques used by compiler writers and differences of opinion regarding which features to implement more efficienctly may also affect memory usage [95, 98].

Disk usage is greatly influenced by the fact that PORTOS is dependent on the host system software for the creation of its binary modules, and the resulting differences in binary module format as produced by the host linkers. The TI 990/10 link program produces relocatable binary modules in ASCII representation, with tags to provide loading information. The PRIME 300, on the other hand, produces absolute memory image format which can be loaded without interpretation.

The PRIME 300 implementation was also faster in execution than the TI 990/10 implementation, due partly to its smaller program size and partly to its faster machine code. In the single-user environment of PORTOS, however, such differences are negligible and have no apparent effect on the overall performance.

8.4) Portability of the System

PORTOS has been transferred to and implemented on the PRIME 300 with much less effort than would have been necessary to re-write it completely. The resulting portability can be calculated from the following formula:

$$P = (1 - R) * 100 \%$$

where R is the ratio of the transfer effort to the re-writing effort. The greater the difference that exists between the two efforts, the

higher the portability.

The easiest way to calculate R is to relate it to the amount of assembly programming that was required. The total number of assembly instructions in the current implementation on the PRIME 300 is 8648. Of these, approximately 940 had to be hand-coded, and the rest were generated by the compiler.

Applying the above criterion to these figures, R is found to be 0.109, yielding a portability of 89.1%. However, a number of other factors must also be taken into consideration in order to obtain a more realistic figure. Any transfer effort is a function of the architectural differences between the two computers involved in the exercise, the familiarity of the implementor with the host hardware, the compatibility of the transfer media, the amount of additional software that has to be written, and the availability of the relevant up-to-date manuals and documents.

Their contribution to the portability effort was estimated to be about 10%, thus giving PORTOS an overall portability of 79.1%. A precise figure is difficult to evaluate because of the inherently abstract nature of some of these factors, as well as their interactions with each other.

8.4.1) Relative and Absolute Portability

The above figure is not attributed to the system as an absolute value. The factors that were taken into account in order to estimate P

were very much dependent on the interacting properties of the donor and the receiver computers. Obviously they would be affected, and therefore yield a different value, if the transfer were to take place between two more-compatible or less-compatible computers. Hence, the portability figure calculated above is the "relative portability" [69] of PORTOS, when transferred from a Texas 990/10 to a PRIME 300. The "absolute portability" of the system can be obtained, only after it has been transferred to a number of computers with different properties, using the following formula:

Absolute P = (P[1] + P[2] + ... +P[n])/n

where "n" is the number of transfers. P[1] is 79.1% for PORTOS.

CHAPTER 9 : DISCUSSION AND CONCLUSIONS

9.1) Implementation Restrictions

PIL has not been exposed to the criticisms of users yet, owing to the restrictions imposed on its specifications and the resulting implementation. A commercially viable and practically useful implementation is needed before its suitability and acceptability can be objectively assessed by the intended user population. Within the existing boundaries, however, the exercise has been successful in the sense that the same "friendly" interface has been made available with comparatively little effort on two different computers.

From the user's point of view, the only difference between the two implementations is the actual startup operation, which is dictated by the host hardware. On the TI 990/10, the bootstrap program must be contained on magnetic cassette, whereas on the PRIME 300 it is contained in sector 0 on disk.

This mandatory use of the cassette contradicts the PORTOS objective of being self-contained on disk, but on the other hand, it offers the implementor freedom from restrictions on program size. The TI 990/10 bootstrap program was written in CORAL. On the PRIME 300, the same algorithm had to be re-written in assembly language because, after the necessary modifications to it, the CORAL program was found to be too big to fit into one sector.

From the implementor's point of view, however, there are other more critical restrictions placed by the implementation on, firstly, the portability of the primitives, and secondly, the subset orientation.

9.1.1) Portability of the Primitives

After the system was transferred to the PRIME 300 and the implementation attempted, it was discovered that the existing primitives were not really suitable for sectored-addressing machines. Sectoring works as follows: all high-memory addresses and any location that is outside the address range P-239 to P+256, where P is the current value of the program counter, can only be referenced indirectly. Code produced by the host compilers and the assembler achieve this indirection via sector zero. Its memory locations octal 200 to 777 are dedicated to these cross-sector links and are used by all executing programs, where necessary.

When a program that cross-references a sector in this manner is loaded into memory, it initializes those locations in sector zero that it uses, destroying their previous contents. Re-loading automatically re-initializes the same locations. However, if a memory-resident program were to use any such location and its contents were destroyed by another program, then naturally the flow of execution would be lost and the outcome would be unpredictable.

The PORTOS tasks were faced with this problem of corruption. The host compiler had been provided with the option to produce code that

reduced these cross-sector references. That option was used during the compilation of the tasks, but the corrections were found to be still insufficient. Therefore, an additional level of indirection had to be hand-coded into the assembly code of the tasks so as to avoid cross-sector references altogether and make them totally self-contained.

Naturally, this is a short-term solution to the problem. There is no need to change the system design, but the tasks need to be re-written to take segmenting into account at the source level. Sectoring was not provided for during the original coding and implementation, because the author was not aware of the problems. The Texas 990/10 assembler produces code that caters for such indirections from within the same binary module, without referring to any external dedicated locations. The necessary alterations will be made as soon as time allows.

In the meantime, however, the successful outcome of this exercise has demonstrated the power and the value of consistent assembly code production by the compiler. This ability to work at the assembler level has also meant that segment-addressing machines were not excluded from the PORTOS domain, which would have been rather hampering. Nevertheless, it is a point against the immediate portability of the system, and therefore a long-term solution is essential.

9.1.2) Subset Orientation

The current implementation of PORTOS is orientated towards single-disk configurations with 8 sectors per track. Implementation on a different configuration would naturally require modifications to the source code. Owing to the modularity and parameterization of the code, however, these modifications would be slight and highly localized.

PORTOS is already parameterized to recognize different disk drives. In the current single-disk implementation, they are treated as dummies and are ignored. In the multi-disk implementation, they would take actual parameters and be used by the disk driver program, which would have to be re-written or modified as appropriate, to access the the relevant disk.

In order to implement PORTOS on a disk configuration that has other than 8 sectors per track, it would be necessary either to re-configure the disk controller or modify the disk manager and other related primitives.

The hardware configuration of the TI 990/10 disk unit, for example, is 203 cylinders per disk pack and 24 sectors per track. The disk controller, however, is capable of logically altering its tracks to accommodate any format, from 1 record per track to 24 records per track. A record is a logical block of data and may correspond to one or more physical sectors.

Before the disk could be built to support PORTOS, therefore, the disk controller had to be re-formatted to conform to this requirement.

A special program called FORMAT, which is presented in Appendix N, was written in CORAL to perform this operation. FORMAT must be run before the BUILD program, in order to prevent erroneous results during disk access.

The FORMAT program is totally machine-orientated to this particular kind of disk controller, with no portability in mind. Its transfer to other computers, although possible, would be pointless. On the PRIME 300, the reconfiguration of the disk controller was not required, as it already had 8 sectors per track.

9.2) Implementation on Atypical Configurations

The current implementations of PORTOS, supported by a single-disk unit, have proved that atypical configurations are not excluded from its domain, in spite of the original guidelines. It would be possible, therefore, to implement PORTOS on multi-disk unit configurations also. The disk drives would then be numbered from zero to some upper limit dependent on their numbers.

PIL has remained unaffected by support from a single disk. On the multi-disk configuration, it would become necessary to issue the disk name as a parameter, together with the relevant file reference. Drive 0 would again be SYSTEM and drive 1 the default drive USER1. The other drives would then be named USER2, USER3, etc. The syntax and logic of PIL, however, would still remain unaffected.

9.3) Reflections on CORAL66

The decision to use CORAL as the implementation language was not regretted at any time either during the development or the subsequent transportation. PORTOS owes its portability to the modular structure of its design, which, it must be admitted, was influenced by the facilities of CORAL for modular programming.

CORAL is a simple language that leaves the programmer to formulate the explicit specification of a number of operations for which other languages provide constructs, such as CASE and string handling. It is a temptation, therefore, to list the facilities that one would like to see incorporated into the language. On the other hand, PL/I has demonstrated the dangers of creating a language with too many constructs in it. Simplicity is one of the big advantages of CORAL, which the author feels should be preserved, and the temptation to propose extra facilities has therefore been resisted.

It is nevertheless felt that some of the existing features of the language could be enhanced in order to facilitate efficient programming. For example, systems programming is very much a string handling application, and strings are best stored one character per byte. It would, therefore, be very useful if the 'LITERAL' construct could take two characters in its argument. In other words, if an integer variable X contains the string "AB", the statements of the kind

```
'IF' X = 'LITERAL'(AB) 'THEN'... or

X := 'LITERAL'(XY)
```

would be helpful. Secondly, a loop construct of the form

'WHILE' condition 'DO'

without the 'FOR', would be a valuable enhancement. It would free the programmer from having to create dummy loop variables for those cases where exit from the loop is determined by an independent condition. Thirdly, the entries of a 'TABLE' declaration start from 0. It would be helpful if the beginning could be selected to be either 0 or 1. The use of tables was rejected in favour of arrays in many cases during coding, because of this limitation. Finally, byte arrays should become a part of the standard language.

9.4) Future Developments

The next stage is to be the expansion of the current PORTOS subset, to incorporate the missing software development facilities, and its implementation on the target configuration. Work is in progress to transfer the system to a DEC PDP-11/03 [43].

A P P E N D I C E S

Appendix A

PORTOS Command Language (PCL) Syntax

```
<command>::=<command string><command terminator>

<command string>::=<command name>!
                   <command name><command separator><parameter list>

<command name>::=any sequence of up to six characters except
                 space or comma or period

<parameter list>::=<parameter>!
                   <parameter><parameter separator><parameter list>

<parameter>::=any sequence of characters except
              space or comma

<command terminator>::=Carriage Return

<command separator>::=Space

<parameter separator>::=Comma
```

Appendix B

PORTOS Response Language (PRL) Syntax

<response>::=<prompt>!<message>

<prompt>::=<string><prompt terminator>

<message>::=<string><message terminator>

<string>::=any sequence of ASCII characters

<prompt terminator>::=Bell

<message terminator>::=Carriage Return Line Feed Line Feed

Appendix C

PIL Commands

I) File Handling Commands

Command	Parameters	Event
COPY	<source>,<target>[,OUST]	copy sourcefile contents to targetfile
CREATE	<file>[,E=<eof>]	create file on diskette
DELETE	<file>	delete file from diskette
LOCK	<file>	protect file against deletion
PRINT	<file>[,NUMBER[,TITLE]]	print contents of file on printer obeying options
RENAME	<old file>,<new file>	change filename from old to new
LIST	<file>	list file contents on VDU
UNLOCK	<file>	release file for deletion

II) Software Development Commands

Command	Parameters	Event
<language>	<source> [,P=<options>]	activate the relevant language compiler
DEBUG	<file>	debug binary program in file interactively from terminal
EDIT	<file>	edit character file
LINK	<file>	link object program in file into executable binary
RUN	<file>	load into memory and execute binary program in file

III) Inquiry Commands

Command	Parameters	Event
FILES	[SYSTEM[,PRINT[,DETAIL]]]	display contents of diskette directory obeying the options
HELP	None	display PIL command menu
SPACE	[SYSTEM[,PRINT]	display amount of diskette freepace in terms of sectors
TIME	None	display date and time of day on terminal

IV) System Commands

Command	Parameters	Event
INDISK	None	initialize user diskette for use by PIL
INTIME	None	set date and time of day
MOVE	<file>!*,<source>,<target>	move contents of either file or whole diskette (*) from source to target drive

Appendix D

CORAL66 Compilers Running

CTL Modular One (RSRE)	ICL 1900 (RMCS)
DEC PDP 9 (SDL)	7905 (ICL)
PDP 10 (SDL)	System 4 (Aberdeen Med.S.)
PDP 11 (CAP)	Intel 8080A (GEC)
PDP 15 (SDL)	Konsberg KS500 (SDL)
System 10 (SDL)	Marconi 12/12 (SDL)
System 20 (SDL)	CHLL (Marconi)
Ferranti ARGUS 400 (RSRE)	LOCUS16 (Marconi)
ARGUS 500 (RSRE)	NORD 10 (SDL)
ARGUS 700 (Ferranti)	Plessey System 250 (CAP)
FMI 600B (Ferranti)	XL4 (RSRE)
FMI 600D (Ferranti)	PRIME 200 (SDL)
GEC 2050 (GEC)	300 (SDL)
4070 (GEC)	400 (SDL)
4080 (GEC)	Rank Xerox 560 (Logica)
4082 (GEC)	SIGMA 6 (Logica)
920ATC (CAP)	SIGMA 7 (Logica)
920C (CAP)	SIGMA 9 (Logica)
Elliott 900 (GEC)	Texas 990/10 (SDL)
Myriad I (RSRE)	Honeywell 316 (SDL)
Myriad II (RSRE)	516 (SDL)
Myriad III (RSRE)	716 (SDL)
IBM 360/370 (CSS)	

CORAL66 Compilers Under Development

CIL System 90 (Albatros Ltd)	Data General NOVA (Lancaster U)
Ferranti FMI600E (Ferranti)	UNIVAC 1108 (National Eng.Lab)
Interdata 5/16 (Interdata)	Varian V70 (SDL)
6/16 (Interdata)	MODCOMP II (Leasco)
7/16 (Interdata)	IV (Leasco)
8/16 (Interdata)	Smiths SDC 16 (Smiths Ltd)
7/32 (Interdata)	SDC 18/1 (Smiths Ltd)
8/32 (Interdata)	

Reference

1) CORAL66 Bulletin NCC 1976 Issue 6

Appendix E

I) Memory Organization of the Texas 990

Hex.Address	Contents
0000 . . . 003C	Level 0 to 15 Interrupt Vectors
0040 . . . 007C	XOP 0 to 15 Transfer Vectors
0080 . . . F7FE	General Memory Area
F800 . . . FBFF	TILINE (DMA)
FC00 . . . FFFA	Programmer Panel & Loader
FFFC . . . FFFE	Restart Transfer Vector

II) Memory Organization of the PRIME 300

Octal Address	Contents
000000	Index Register X
01	A Register
02	B Register
03	Stack Pointer
04	Floating Point High
05	Floating Point Low
06	Visible Shift Counter
07	Program Counter
10	Page Map Address Register
11	Microcode Scratch Location
12	Effective Address Save
13	Microcode Scratch Location
14	Y Register Save for Control Panel & DMA
15	M Register Save for Control Panel & DMA
16	Microcode Scratch Location
17	Microcode Scratch Location
20 ... 37	DMA Range Start Address Pairs
40 ... 57	Reserved for DMC Channel Pairs
60	Power Fail Interrupt & Timer
61	Real-Time Clock Increment
62	Restricted Execution Violation
63	Standard Interrupt
64	Page Fault: Addressed Page Not in Memory
65	Supervisor Call Trap
66	Unimplemented Instruction Interrupt
67	Memory Data Parity Error
70	Machine Check: CPU Detected Error
71	Missing Module: No Memory at Location
72	Illegal Instruction Interrupt
73	Page Write Violation
74	Floating Point Exception
75	Procedure Stack Underflow
76 ... 100	Debugging Scratch Area
101 ... 177	Interrupt Vectors
200 ... 777	General Cross Sector Links
1000	Start of General Memory Area

Appendix F

A Comparison of CORAL Keywords on Different Compilers

Keyword	Official	ICL1900	Texas990	PRIME300	PDP-11
ABSOLUTE	*		*	*	*
AND	*	*	*	*	*
ANON		*			
ANSWER	*	*	*	*	*
ARRAY	*	*	*	*	*
BEGIN	*	*	*	*	*
BIT	*	*	*	*	*
BITS	*	*	*	*	*
BYTE		*	*		*
CODE	*	*	*	*	*
COMMENT	*	*	*	*	*
COMMON	*	*	*	*	*
CORAL			*	*	
DEFINE	*	*	*	*	*
DELETE	*	*	*	*	*
DIFFER	*	*	*	*	*
DO	*	*	*	*	*
DUMPON		*			
ELSE	*	*	*	*	*
END	*	*	*	*	*
ENTRY		*			
EQ			*	*	
ERRORLIST		*			
EXTERNAL	*		*	*	*
FIELD		*			
FINISH	*	*	*	*	*
FIXED	*	*	*	*	*
FLOATING	*	*	*	*	*
FOR	*	*	*	*	*
GE			*	*	
GOTO	*	*	*	*	*
GT			*	*	
HEX			*	*	
IF	*	*	*	*	*
INFINITY					*
INTEGER	*	*	*	*	*
IS					*
LABEL	*	*	*	*	*
LE			*	*	
LIBRARY	*	*	*	*	*
LIST		*			
LITERAL	*	*	*	*	*
LOCATION	*	*	*	*	*

Appendix F cont.

Keyword	Official	ICL1900	Texas990	PRIME300	PDP-11
LONG				*	
LOWER		*			
LT			*	*	
MASK	*	*	*	*	*
MOD					*
NE			*	*	
NOLIST		*			
OCTAL	*	*	*	*	*
OF		*			
OR	*	*	*	*	*
OVERFLOW		*			
OVERLAY	*	*	*	*	*
PRESET	*	*	*	*	*
PROCEDURE	*	*	*	*	*
PROGRAM		*	*	*	
RECORD		*			
RECURSIVE	*	*		*	*
SEGMENT		*	*	*	
SEMICOMPILED		*			
SENDTO		*			
SLA				*	
SLC		*			
SLL		*		*	
SPECIAL		*			
SRA		*		*	
SRC				*	
SRL		*		*	
STEP	*	*	*	*	*
SWITCH	*	*	*	*	*
TABLE	*	*	*	*	*
THEN	*	*	*	*	*
TRACE		*			
UNION	*	*	*	*	*
UNSIGNED	*	*	*	*	*
UNTIL	*	*	*	*	*
UPPER		*			
VALUE	*	*	*	*	*
WHILE	*	*	*	*	*
WITH	*	*	*	*	*
WORK		*			

Appendix G

PORTOS Macro Definition Files

I) Macro Definition File DBASE

```
'COMMENT'   FILE: DBASE
       PORTOS MEMORY-RESIDENT DATABASE;
'ABSOLUTE' ('INTEGER' HEAD  /'OCTAL'(30000); (WORD ADDRESSES)
            'INTEGER' TAIL  /'OCTAL'(30001);
            'INTEGER' ACTIVE/'OCTAL'(30002);
            'INTEGER' TNUM  /'OCTAL'(30003);
            'INTEGER''ARRAY' VECTOR/'OCTAL'(30004) [1:12]; ( TASK   )
            'INTEGER''ARRAY' RETURN/'OCTAL'(30020) [1:12]; ( VECTOR )
            'INTEGER''ARRAY' FATHER/'OCTAL'(30034) [1:12]; ( TABLE  )
            'INTEGER' PAR1  /'OCTAL'(30050);  (PARAMETERS)
            'INTEGER' PAR2  /'OCTAL'(30051);
            'INTEGER' PAR3  /'OCTAL'(30052);
            'INTEGER' PAR4  /'OCTAL'(30053);
            'INTEGER' PAR5  /'OCTAL'(30054);
            'INTEGER' PAR6  /'OCTAL'(30055);
            'INTEGER' PAR7  /'OCTAL'(30056);
            'INTEGER' PAR8  /'OCTAL'(30057);
            'INTEGER' TEL1  /'OCTAL'(30060);  (TABLE ELEMENTS)
            'INTEGER' TEL2  /'OCTAL'(30061);
            'INTEGER' TEL3  /'OCTAL'(30062);
            'INTEGER' TEL4  /'OCTAL'(30063);
            'INTEGER''ARRAY' DATE/'OCTAL'(30104) [1:4]; (DATE INDICATOR)
            'INTEGER' HOURS /'OCTAL'(30110);  (TIME OF DAY)
            'INTEGER' MINS  /'OCTAL'(30111);
            'INTEGER' SECS  /'OCTAL'(30112)   );
          'DEFINE' TASK AREA "'OCTAL'(30113)"; (THE BEGINNING OF)
'ABSOLUTE' ('INTEGER' PPTR  /'OCTAL'(37777); (PARAM.POINTER)
              'TABLE' IO TABLE /'OCTAL'(40000) [6, 5]
               [ CHANNEL  'INTEGER' 0;
                 SECTOR   'INTEGER' 1;
                 FUNCTION 'INTEGER' 2;
                 WORD     'INTEGER' 3;
                 BYTE     'INTEGER' 4;
                 BUFFER   'INTEGER' 5 ]);
```

II) Macro Definition File PARAMS

```
'COMMENT'     FILE: PARAMS
     PORTOS EXECUTION PARAMETERS FOR THE PRIME 300;
     'DEFINE' SBS         "448"; (SECTOR BLOCK SIZE IN WORDS)
     'DEFINE' SBSM1       "447"; (SECTOR SIZE - 1)
     'DEFINE' SDISK       "0";   (SYSTEM DRIVE)
     'DEFINE' UDISK       "1";   (USER DRIVE)
     'DEFINE' TOP TRACK  "405"; (TOTAL TRACKS - 1)
     'DEFINE' MAXMOD       "22"; (TOTAL NO.OF MODULES IN SYSTEM)
     'DEFINE' UMSA "'HEX'(200)"; (USER MEMORY WORD START ADDRESS)
     'DEFINE' UMSIZE "'HEX'(2FC0)"; (USER MEMORY SIZE IN WORDS)
     'DEFINE' STR DCODER "'HEX'(C4C3),'HEX'(CFC4),'HEX'(C5D2),'HEX'(AEC2)";
     'DEFINE' STR XER    "'HEX'(D8C5),'HEX'(D2AE),'HEX'(C2A0),'HEX'(A0A0)";
     'DEFINE' RESET MEMORY
               "HEAD := UMSA; TAIL := HEAD;
                [HEAD] := UMSIZE; [HEAD+1] := 0  ";
```

III) Macro Definition File CHARS

```
'COMMENT'     FILE: CHARS
      PORTOS CONTROL CHARACTERS FOR THE PRIME 300;
      'DEFINE' CR      "'OCTAL'(215)";    (CARRIAGE RETURN)
      'DEFINE' LF      "'OCTAL'(212)";    (LINE FEED)
      'DEFINE' BS      "'OCTAL'(210)";    (BACK SPACE)
      'DEFINE' SP      "'OCTAL'(240)";    (SPACE)
      'DEFINE' KILL    "'OCTAL'(210)";    (BACK SPACE)
      'DEFINE' DEL     "'OCTAL'(377)";    (RUB OUT)
      'DEFINE' BEL     "'OCTAL'(207)";    (BELL)
      'DEFINE' ITT     "'OCTAL'(337)";    (INVITATION TO TYPE)
      'DEFINE' ERL     "'OCTAL'(277)";    (ITT AFTER ERASE LINE)
      'DEFINE' EOF     "'OCTAL'(223)";    (END OF FILE)
      'DEFINE' BREAK   "'OCTAL'(001)";
      'DEFINE' BLANK   "'OCTAL'(120240)"; (SP PER BYTE)
      'DEFINE' TOP BYTE(P) "'BITS'[8,8]P";
      'DEFINE' BOT BYTE(P) "'BITS'[8,0]P";
```

IV) Macro Definition File LINKS

```
'COMMENT'     FILE: LINKS
              PORTOS TASK COMMUNICATION LINKAGE (PRIME 300);
     'DEFINE' SUPERVISOR CALL
                "'CODE''BEGIN'
                  ´    SVC      / SV CALL     ;´
                     'END'  ";
     'DEFINE' ACTIVATE(TXZ)
                "TNUM:=TXZ;
                 SUPERVISOR CALL  ";
     'DEFINE' ALLOCATE(AXZ,BXZ)
                "PAR1:=AXZ;
                 ACTIVATE(1);
                 BXZ:=PAR2  ";
     'DEFINE' FREE(AXZ,BXZ)
                "PAR1:=AXZ;
                 PAR2:=BXZ;
                 ACTIVATE(2)  ";
     'DEFINE' GIVE SECTOR(AXZ,BXZ)
                "ACTIVATE(4);
                 BXZ:=PAR1  ";
     'DEFINE' RELEASE SECTOR(AXZ,BXZ)
                "PAR1:=BXZ;
                 ACTIVATE(5)  ";
     'DEFINE' EXIT(AXZ)
                "'BEGIN'
                 PAR8:=AXZ;
                 ACTIVATE(6)
                 'END'  ";
     'DEFINE' LOAD(AXZ,BXZ)
                "PAR1:=BXZ;
                 ACTIVATE(7)  ";
     'DEFINE' DISKOP(AXZ,BXZ,CXZ,DXZ)
                "PAR1:=AXZ;
                 PAR2:=CXZ;
                 PAR3:=DXZ;
                 ACTIVATE(8)  ";
     'DEFINE' GET RECORD(AXZ,BXZ)
                "PAR1:='LOCATION'(AXZ[1]);
                 PAR2:=BXZ;
                 ACTIVATE(9)  ";
     'DEFINE' PUT RECORD(AXZ,BXZ)
                "PAR1:='LOCATION'(AXZ[1]);
                 PAR2:=BXZ;
                 ACTIVATE(10)  ";
     'DEFINE' SHUTDOWN IO
                "ACTIVATE(11)  ";
     'DEFINE' FETCH TIME(AXZ)
                "PAR1:='LOCATION'(AXZ[1]);
                 ACTIVATE(12)  ";
```

PIL Interface Source Listings

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL'('PROCEDURE' WRITE CHAR(VI,VI);
10             'PROCEDURE' READ RECORD(VI,IA,VI,LI);
11             'PROCEDURE' EXECUTE(IA,VI,LI);
12             'PROCEDURE' ERROR(IA,VI);
13             'PROCEDURE' NEW SUFFIX(IA,IA,VI);
14             'PROCEDURE' DISPLAY(VI,VI)  );
15  'DELETE' VI; 'DELETE' IA; 'DELETE' LI;
16  'SEGMENT' PCL DECODER
17  'BEGIN'
18       'INTEGER' RESULT, LOCK, I, ENTRY, ELEM, C, PTR;
19       'INTEGER' DOTS, KEY, NPAR;
20       'INTEGER''ARRAY' B[1:80];
21       'INTEGER''ARRAY' PNAME[1:5];
22       'INTEGER''ARRAY' CNAME[1:4]:=BLANK,BLANK,BLANK,BLANK;
23
24  'INTEGER''PROCEDURE' SAVE COMMAND;
25  'BEGIN'
26       'INTEGER' J, S, KEY, LOCK;
27
28       J:=1; S:=0; LOCK:=0;
29       'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
30           'BEGIN'
31           'IF' B[PTR] = SP
32              'THEN''ANSWER' 0
33              'ELSE''IF' B[PTR] = CR
34                       'THEN''ANSWER' 1
35              'ELSE''BEGIN'
36                     'IF' J>3
37                        'THEN' ERROR(CNAME,8);
38                     'IF' S=0
39                        'THEN''BEGIN'
40                              TOP BYTE(CNAME[J]):=B[PTR];
41                              S:=1
42                              'END'
43                        'ELSE''BEGIN'
44                              BOT BYTE(CNAME[J]):=B[PTR];
45                              S:=0; J:=J+1
46                              'END'
47                     'END';
48           PTR:=PTR+1
49           'END';
50       'ANSWER' 2
```

```
51  'END';
52
53  'INTEGER''PROCEDURE' SAVE PARAMETER;
54  'BEGIN'
55        'INTEGER' J, OUT, S, KEY, LIMIT;
56
57  'PROCEDURE' MOVE PAR;
58  'BEGIN'
59        'IF' TOP BYTE(PNAME[1])='LITERAL'(.) 'OR'
60             PTR > LIMIT 'OR' DOTS > 1
61           'THEN' ERROR(PNAME,11);
62        'FOR' I:=0,I+1 'WHILE' I<=4 'DO'
63            [ENTRY+ELEM+I]:=PNAME[I+1];
64        LIMIT:=80;
65        ELEM:=ELEM+5
66  'END';
67
68        NPAR:=NPAR+1;
69        'IF' NPAR > 4 'THEN''ANSWER' 1;
70        DOTS:=0; OUT:=0; LOCK:=0; S:=0; LIMIT:=80;
71        ENTRY:='LOCATION'(TEL1); J:=1;
72        'FOR' I:=1,I+1 'WHILE' I<=5 'DO' PNAME[I]:=BLANK;
73        'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
74            'BEGIN'
75            'IF' B[PTR] = 'LITERAL'(.)
76               'THEN''BEGIN'
77                     LIMIT:=PTR+2;
78                     DOTS:=DOTS+1
79                     'END';
80            'IF' B[PTR] = CR
81               'THEN''BEGIN'
82                     MOVE PAR;
83                     'ANSWER' 1
84                     'END'
85               'ELSE''IF' B[PTR] = 'LITERAL'(,)
86                     'THEN''BEGIN'
87                           MOVE PAR;
88                           'ANSWER' 0
89                           'END'
90               'ELSE''BEGIN'
91                     'IF' J>4
92                        'THEN''BEGIN'
93                              'IF' BOTBYTE(PNAME[1])<>'LITERAL'(=)
94                                 'THEN' ERROR(PNAME,8)
95                              'END';
96                     'IF' S = 0
97                        'THEN''BEGIN'
98                              TOP BYTE(PNAME[J]):=B[PTR];
99                              S:=1
100                             'END'
101                       'ELSE''BEGIN'
102                             BOT BYTE(PNAME[J]):=B[PTR];
103                             S:=0; J:=J+1
104                             'END'
105                    'END';
106          PTR:=PTR+1
```

```
107              'END';
108          'ANSWER' 2
109  'END';
110
111          LOCK:=0; NPAR:=0;
112          'FOR' I:=0,I+1 'WHILE' I<=4 'DO'
113              CHANNEL[I]:=-1; (CLEAR IO TABLE)
114          ENTRY:='LOCATION'(TEL1);
115          PPTR:=ENTRY;
116          'FOR' KEY:=0 'WHILE' LOCK = 0 'DO'
117              'BEGIN'
118              'FOR' I:=0,I+1 'WHILE' I<=19 'DO'
119                  [ENTRY+I]:=BLANK;
120              'IF' PAR7 = 0 'OR' PAR7 = 111
121                 'THEN' DISPLAY(0,"GO, ")
122                 'ELSE' DISPLAY(0,"ER, "); PAR7:=0;
123              READ RECORD(0,B,80,I); WRITE CHAR(0,LF);
124              C:=0; ELEM:=0; RESULT:=0;
125              'IF' B[1] <> CR
126                 'THEN''BEGIN'
127                        PTR:=1;
128                        'FOR' I:=0 'WHILE' KEY=0 'DO'
129                            'BEGIN'
130                            'IF' RESULT = 1
131                               'THEN''BEGIN'
132                                      NEW SUFFIX(CNAME,CNAME,'LITERAL'(B));
133                                      EXECUTE(CNAME,SDISK,RESULT);
134                                      NEW SUFFIX(CNAME,CNAME,0);
135                                      'IF' RESULT=3
136                                         'THEN' RESULT:=1;
137                                      ERROR(CNAME,RESULT)
138                                      'END'
139                               'ELSE''IF' RESULT=2
140                                         'THEN' KEY:=1
141                               'ELSE''IF' B[PTR] = SP
142                                         'THEN' KEY:=0
143                               'ELSE''BEGIN'
144                                        'IF' C = 0
145                                           'THEN''BEGIN'
146                                                  RESULT:=SAVE COMMAND;
147                                                  C:=1
148                                                  'END'
149                                         'ELSE' RESULT:=SAVE PARAMETER
150                                       'END';
151                             PTR:=PTR+1
152                             'END'
153                        'END'
154              'END'
155  'END'
156  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'EXTERNAL'('PROCEDURE' DISPLAY('VALUE''INTEGER','VALUE''INTEGER');
 7             'PROCEDURE' WRITE CHAR('VALUE''INTEGER','VALUE''INTEGER')  );
 8  'SEGMENT' PRL ERROR MESSAGES
 9  'BEGIN'
10        'INTEGER' EC; (ERROR CODE)
11
12  'PROCEDURE' PUT NAME;
13  'BEGIN'
14        'INTEGER' I, ENTRY;
15
16        ENTRY := 'LOCATION'(TEL1);
17        'FOR' I:=0,I+1 'WHILE' I<=3 'DO'
18            'BEGIN'
19            'IF' [ENTRY+I] = BLANK
20               'THEN' I := 5
21               'ELSE''BEGIN'
22                     WRITE CHAR(0,TOP BYTE([ENTRY+I]));
23                     WRITE CHAR(0,BOT BYTE([ENTRY+I]))
24                     'END'
25            'END'
26  'END';
27
28        EC := PAR8;  (FETCH ERROR CODE)
29        PAR7 := EC;
30        'IF' EC = 1
31           'THEN''BEGIN'
32                 PUT NAME;
33                 DISPLAY(0," IS NOT A PORTOS COMMAND")
34                 'END'
35           'ELSE''IF' EC = 2  'THEN' DISPLAY(0,"PARAMETER MISSING")
36           'ELSE''IF' EC = 3
37                  'THEN''BEGIN'
38                        PUT NAME;
39                        DISPLAY(0," NOT FOUND")
40                        'END'
41           'ELSE''IF' EC = 4
42                  'THEN''BEGIN'
43                        PUT NAME;
44                        DISPLAY(0," ALREADY EXISTS")
45                        'END'
46           'ELSE''IF' EC = 5  'THEN' DISPLAY(0,"DISK FULL")
47           'ELSE''IF' EC = 6  'THEN' DISPLAY(0,"PROGRAM TOO BIG")
48           'ELSE''IF' EC = 7  'THEN' DISPLAY(0,"LOAD ERROR")
49           'ELSE''IF' EC = 8
50                  'THEN''BEGIN'
51                        DISPLAY(0,"NAME "); PUT NAME;
52                        DISPLAY(0," TOO LONG")
53                        'END'
54           'ELSE''IF' EC = 9
55                  'THEN''BEGIN'
56                        PUT NAME;
```

```
57                         DISPLAY(0,"IS LOCKED")
58                         'END'
59          'ELSE''IF' EC = 10
60                   'THEN''BEGIN'
61                         PUT NAME;
62                         DISPLAY(0," UNSUITABLE FOR THIS OPERATION")
63                         'END'
64          'ELSE''IF' EC = 11
65                   'THEN''BEGIN'
66                         DISPLAY(0,"BAD SYNTAX : ");
67                         PUT NAME
68                         'END'
69          'ELSE''IF' EC = 12
70                   'THEN' DISPLAY(0,"DISK PROTECTED")
71          'ELSE''IF' EC = 13
72                   'THEN' DISPLAY(0,"DISK IO ERROR")
73          'ELSE''IF' EC = 111
74                   'THEN''BEGIN'
75                         DISPLAY(0,"!CL!PORTOS DEV=1.0 (4.5.79) NOW RUNNI
76                         DISPLAY(0,"!CL!IF IN DOUBT, TYPE HELP")
77                         'END'
78          'ELSE''IF' EC >= 20 'AND' EC <=30
79                   'THEN''BEGIN'
80                         EC:=EC-20;
81                         'IF' EC = 2
82                            'THEN''BEGIN'
83                                 DISPLAY(0,"CHANNEL "); PUT NAME;
84                                 DISPLAY(0," ALREADY OPEN")
85                                 'END'
86                            'ELSE''IF' EC = 3
87                                      'THEN''BEGIN'
88                                            DISPLAY(0,"CHANNEL ");
89                                            PUT NAME;
90                                            DISPLAY(0," NOT OPEN")
91                                            'END'
92                            'ELSE''IF' EC = 4
93                                      'THEN' DISPLAY(0,"IO TABLE FULL")
94                            'ELSE''IF' EC = 5
95                                      'THEN''BEGIN'
96                                            DISPLAY(0,"ILLEGAL IO ON");
97                                            PUT NAME
98                                            'END'
99                         'END'
100         'ELSE' DISPLAY(0,"BAD TRANSMISSION");
101       DISPLAY(0,"!CLL!");
102       EXIT(0)
103 'END'
104 'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' VI "'VALUE''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' GET PARAM(IA);
10              'PROCEDURE' COPY FILE(IA,IA,VI)  );
11  'DELETE' IA; 'DELETE' VI;
12  'SEGMENT' COPY COMMAND HANDLER
13  'BEGIN'
14        'INTEGER''ARRAY' FROM[1:5];
15        'INTEGER''ARRAY' TO[1:5];
16        'INTEGER''ARRAY' QUALIFIER[1:5];
17        'INTEGER' OUST:=0;
18
19        GET PARAM(FROM);
20        GET PARAM(TO);
21        'IF' FROM[1]=BLANK 'OR' TO[1]=BLANK
22           'THEN' EXIT(2); (PARAMETER MISSING)
23        GET PARAM(QUALIFIER);
24        'IF' QUALIFIER[1]='OCTAL'(147725)
25           'THEN' OUST:=1;
26        COPY FILE(FROM, TO, OUST);
27        EXIT(0)
28  'END'
29  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' WRITE CHAR(VI,VI);
10              'PROCEDURE' GET PARAM(IA);
11              'PROCEDURE' ERROR(IA,VI);
12              'PROCEDURE' OPEN(VI,IA,VI,VI,LI);
13              'PROCEDURE' CLOSE(VI);
14              'PROCEDURE' READ RECORD(VI,IA,VI,LI);
15              'PROCEDURE' WRITE RECORD(VI,IA,VI)  );
16  'DELETE' IA; 'DELETE' VI; 'DELETE' LI;
17  'SEGMENT' CREATE COMMAND HANDLER
18  'BEGIN'
19      'INTEGER' LOCK, KEY, RESULT;
20      'INTEGER''ARRAY' B[1:80];
21      'INTEGER''ARRAY' FNAME[1:5];
22      'INTEGER''ARRAY' QUALIFIER[1:5];
23      'INTEGER''ARRAY' EOF COM [1:2] := 'LITERAL'(O),'LITERAL'(K);
24      'INTEGER' NEW EOF := 'OCTAL'(142675);  (NEW EOF COMMAND IND. E=)
25
26      GET PARAM(FNAME);
27      'IF' FNAME[1]=BLANK 'THEN' EXIT(2);
28      GET PARAM(QUALIFIER);
29      'IF' QUALIFIER[1] = NEW EOF
30        'THEN''BEGIN'
31             EOF COM[1] := TOP BYTE(QUALIFIER[2]); (GET NEW EOF COMMAND
32             EOF COM[2] := BOT BYTE(QUALIFIER[2])
33             'END';
34      OPEN(3,FNAME,UDISK,2,RESULT);
35      'IF' RESULT > 0 'THEN' ERROR(FNAME,RESULT);
36      KEY := 0; WRITE CHAR(0,LF);
37      'FOR' LOCK:=0 'WHILE' KEY=0 'DO'
38          'BEGIN'
39          WRITE CHAR(0,ITT); (PROMPT FOR INPUT)
40          READ RECORD(0,B,80,RESULT); WRITE CHAR(0,LF);
41          'IF' B[1]=EOF COM[1] 'AND' B[2]=EOF COM[2]
42             'THEN''BEGIN'
43                   CLOSE(3);
44                   WRITE CHAR(0,LF); KEY := 1
45                   'END'
46             'ELSE' WRITE RECORD(3, B, 80)
47          'END';
48      EXIT(0)
49  'END'
50  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' VI "'VALUE''INTEGER'";
 9  'DEFINE' LI "'LOCATION''INTEGER'";
10  'EXTERNAL' ('PROCEDURE' DELETE FILE(IA,VI);
11              'PROCEDURE' GET PARAM(IA)   );
12  'DELETE' IA; 'DELETE' LI; 'DELETE' VI;
13  'SEGMENT' DELETE COMMAND HANDLER
14  'BEGIN'
15        'INTEGER''ARRAY' FNAME[1:5]; (FILENAME)
16
17        GET PARAM(FNAME);
18        'IF' FNAME[1] = BLANK
19           'THEN' EXIT(2);  (PARAMETER MISSING)
20        DELETE FILE(FNAME,UDISK);
21        EXIT(0)
22  'END'
23  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' VI "'VALUE''INTEGER'";
 8  'EXTERNAL' ('PROCEDURE' DISPLAY(VI,VI);
 9              'PROCEDURE' WRITEN(VI,VI,VI,VI);
10              'INTEGER''PROCEDURE' LENGTH(VI,VI);
11              'PROCEDURE' GET PARAM('INTEGER''ARRAY');
12              'PROCEDURE' WRITE CHAR(VI,VI)  );
13  'DELETE' VI;
14  'SEGMENT' FILES COMMAND HANDLER
15  'BEGIN'
16        'INTEGER' ADDR, ENTRY, I, IND, KEY, COUNT, SIZE;
17        'INTEGER''ARRAY' ID[1:3];
18        'INTEGER''ARRAY' CALENDER[1:7];
19        'INTEGER''ARRAY' QUALIFIER[1:5];
20        'INTEGER' CN, DETAIL, DEVICE:=0,0,1;
21        'INTEGER' PR STR:='OCTAL'(150322);
22        'INTEGER' DE STR:='OCTAL'(142305);
23        'INTEGER' SY STR:='OCTAL'(151731);
24
25        'FOR' I:=1,2,3 'DO'
26            'BEGIN'
27            GET PARAM(QUALIFIER);
28            'IF' QUALIFIER[1]=PR STR
29               'THEN' CN:=1
30               'ELSE''IF' QUALIFIER[1]=DE STR
31                         'THEN' DETAIL:=1
32               'ELSE''IF' QUALIFIER[1]=SY STR
33                         'THEN' DEVICE:=0
34            'END';
35        ALLOCATE(SBS, ADDR);
36        DISKOP(1,DEVICE,0,ADDR);
37        ID[1]:=[ADDR]; ID[2]:=[ADDR+1]; ID[3]:=[ADDR+2]; (FETCH ID)
38        'IF' DEVICE = 0
39           'THEN''BEGIN'
40                 DISKOP(1,UDISK,3,ADDR)
41                 'END'
42           'ELSE''BEGIN'
43                 DISKOP(1,SDISK,2,ADDR)
44                 'END'; (READ DIRECTORY)
45        DISPLAY(CN,"!CL!FILES ON DISK : ");
46        'FOR' I:=1,I+1 'WHILE' I<=3 'DO'
47            'BEGIN'
48            WRITE CHAR(CN,TOP BYTE(ID[I]));
49            WRITE CHAR(CN,BOT BYTE(ID[I]))
50            'END';
51        'IF' CN=1
52           'THEN''BEGIN'
53                 FETCH TIME(CALENDER);
54                 DISPLAY(1,"       ON ");
55                 WRITEN(1,CALENDER[1],2,0); WRITE CHAR(1,'LITERAL'(.));
56                 WRITE CHAR(1,TOP BYTE(CALENDER[2]));
```

```
 57              WRITE CHAR(1,BOT BYTE(CALENDER[2]));
 58              WRITE CHAR(1,TOP BYTE(CALENDER[3]));
 59              WRITE CHAR(1,'LITERAL'(.));
 60              WRITEN(1,CALENDER[4],2,0);
 61              DISPLAY(1,"   AT ");
 62              WRITEN(1,CALENDER[5],2,0); WRITE CHAR(1,'LITERAL'(:));
 63              WRITEN(1,CALENDER[6],2,0); WRITE CHAR(1,'LITERAL'(:));
 64              WRITEN(1,CALENDER[7],2,0)
 65              'END';
 66        DISPLAY(CN,"!CLL!   ");
 67        'IF' DETAIL=1
 68           'THEN''BEGIN'
 69              DISPLAY(CN,"REFERENCE   TYPE   PROTECT  SIZE");
 70              DISPLAY(CN,"!CL!   ");
 71              DISPLAY(CN,"---------  ------  -------  ----");
 72              DISPLAY(CN,"!CLL!   ")
 73              'END';
 74        IND := 0; KEY := 0; COUNT := 0;
 75        'FOR' ENTRY:=ADDR+4,ENTRY+7 'WHILE' KEY = 0 'DO'
 76           'BEGIN'
 77           'IF' [ENTRY] = 'LITERAL'(*)
 78              'THEN' KEY := 1
 79              'ELSE''IF' [ENTRY] <> 'LITERAL'(#)
 80                    'THEN''BEGIN'
 81                          IND := 1;
 82                          'FOR' I:=1 'STEP' 1 'UNTIL' 4 'DO'
 83                             'BEGIN'
 84                             WRITE CHAR(CN,TOPBYTE([ENTRY+I]));
 85                             WRITE CHAR(CN,BOTBYTE([ENTRY+I]))
 86                             'END';
 87                          'IF' DETAIL=0
 88                             'THEN''BEGIN'
 89                                   DISPLAY(CN,"   ");
 90                                   COUNT:=COUNT+1;
 91                                   'IF' COUNT >= 6
 92                                      'THEN''BEGIN'
 93                                            DISPLAY(CN,"!CLS3!");
 94                                            COUNT := 0
 95                                            'END'
 96                                   'END'
 97                             'ELSE''BEGIN'
 98                                   'IF' TOP BYTE([ENTRY])=0
 99                                      'THEN' DISPLAY(CN,"   ASCII ")
100                                      'ELSE' DISPLAY(CN,"   BINARY");
101                                   'IF' BOT BYTE([ENTRY])=0
102                                      'THEN' DISPLAY(CN,"    NO   ")
103                                      'ELSE' DISPLAY(CN,"    YES  ");
104                                   SIZE:=LENGTH(DEVICE,[ENTRY+6]);
105                                   WRITEN(CN,SIZE,5,1);
106                                   DISPLAY(CN,"!CLS3!")
107                                   'END'
108                          'END'
109           'END';
110        'IF' IND=0 'THEN' DISPLAY(CN,"   **NONE**");
111        DISPLAY(CN,"!CLL!");
112        FREE(SBS, ADDR);
```

Appendix H cont.

```
113         EXIT(0)
114   'END'
115   'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' OPEN(VI,IA,VI,VI,LI);
10              'PROCEDURE' READ RECORD(VI,IA,VI,LI);
11              'PROCEDURE' WRITE RECORD(VI,IA,VI);
12              'PROCEDURE' CLOSE(VI);
13              'PROCEDURE' DISPLAY(VI,VI);
14              'PROCEDURE' WRITE CHAR(VI,VI)  );
15  'DELETE' VI; 'DELETE' IA; 'DELETE' LI;
16  'SEGMENT' HELP COMMAND HANDLER
17  'BEGIN'
18        'INTEGER''ARRAY' B[1:80];
19        'INTEGER''ARRAY' HLPF [1:4] := 'OCTAL'(150244),'OCTAL'(144314),
20                                       'OCTAL'(150306),'OCTAL'(127323);
21        'INTEGER''ARRAY' FHAN [1:4] := 'OCTAL'(150244),'OCTAL'(143310),
22                                       'OCTAL'(140716),'OCTAL'(127323);
23        'INTEGER''ARRAY' SDEV [1:4] := 'OCTAL'(150244),'OCTAL'(151704),
24                                       'OCTAL'(142726),'OCTAL'(127323);
25        'INTEGER''ARRAY' INQR [1:4] := 'OCTAL'(150244),'OCTAL'(144716),
26                                       'OCTAL'(150722),'OCTAL'(127323);
27        'INTEGER''ARRAY' SYST [1:4] := 'OCTAL'(150244),'OCTAL'(151731),
28                                       'OCTAL'(151724),'OCTAL'(127323);
29        'INTEGER' KEY, LOCK, ERR;
30        'INTEGER' ONE, TWO, THREE, FOUR, FIVE :=  'LITERAL'(1), 'LITERAL'(
31                                       'LITERAL'(3),'LITERAL'(4),'LITERAL'(
32
33  'PROCEDURE' LIST HELPFILE('INTEGER''ARRAY' FNAME);
34  'BEGIN'
35        'INTEGER' I;
36
37        OPEN(3, FNAME, SDISK, 1, ERR);
38        'IF' ERR <> 0 'THEN' EXIT(ERR);
39        'FOR' I:=0 'WHILE' ERR=0 'DO'
40            'BEGIN'
41            READ RECORD(3, B, 80, ERR);
42            'IF' ERR=0
43               'THEN''BEGIN'
44                     WRITE RECORD(0,B,80)
45                     'END'
46            'END';
47        CLOSE(3)
48  'END';
49
50        'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
51            'BEGIN'
52            LIST HELPFILE(HLPF);
53            KEY:=0;
54            'FOR' LOCK:=0 'WHILE' KEY=0 'DO'
55               'BEGIN'
56               DISPLAY(0,"!CL!      WHICH GROUP DO YOU REQUIRE? ");
```

```
57              READ RECORD(0,B,80,ERR); WRITE CHAR(0,LF);
58              'IF' B[1] >= ONE 'AND' B[1] <= FIVE
59                 'THEN' KEY:=1
60                 'ELSE' DISPLAY(0,"1 TO 5 ONLY PLEASE!CL!");
61              'END';
62          'IF' B[1] = ONE
63             'THEN' LIST HELPFILE(FHAN)
64             'ELSE''IF' B[1] = TWO
65                    'THEN' LIST HELPFILE(SDEV)
66             'ELSE''IF' B[1] = THREE
67                    'THEN' LIST HELPFILE(INQR)
68             'ELSE''IF' B[1] = FOUR
69                    'THEN' LIST HELPFILE(SYST)
70             'ELSE''BEGIN'
71                   DISPLAY(0,"!CLLS8!PORTOS COMMAND REPERTOIRE!CLL!");
72                   DISPLAY(0,"!S3!COPY    CREATE  DEBUG   DELETE  EDIT!C
73                   DISPLAY(0,"!S3!FILES   HELP    INDISK  INTIME  LIST!C
74                   DISPLAY(0,"!S3!LINK    LOCK    MOVE    PRINT   RENAME.
75                   DISPLAY(0,"!S3!RUN     SPACE   TIME    UNLOCK!CLL!")
76                   'END';
77          KEY := 0;
78          'FOR' LOCK:=0 'WHILE' KEY=0 'DO'
79              'BEGIN'
80              DISPLAY(0,"DO YOU NEED HELP ON OTHER COMMANDS? ");
81              READ RECORD(0,B,80,ERR); WRITE CHAR(0,LF);
82              'IF' B[1]='LITERAL'(N) 'AND' B[2]='LITERAL'(O)
83                 'THEN''BEGIN'
84                       DISPLAY(0,"!CL!DONE!CLL!");
85                       EXIT(0)
86                       'END';
87              'IF' B[1]='LITERAL'(Y) 'AND' B[2]='LITERAL'(E)
88                 'THEN' KEY:=1
89                 'ELSE' DISPLAY(0,"YES OR NO PLEASE!CLL!")
90                'END'
91          'END';
92        EXIT(0)
93  'END'
94  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 7  'DEFINE' LI "'LOCATION''INTEGER'";
 8  'EXTERNAL' ('PROCEDURE' DISPLAY(VI,VI);
 9              'PROCEDURE' SET TIME(IA);
10              'PROCEDURE' WRITEN(VI,VI,VI,VI);
11              'PROCEDURE' READ RECORD(VI,IA,VI,LI);
12              'INTEGER''PROCEDURE' READN(VI);
13              'PROCEDURE' WRITE CHAR(VI,VI)  );
14  'DELETE' VI; 'DELETE' IA; 'DELETE' LI;
15  'SEGMENT' INTIME COMMAND HANDLER
16  'BEGIN'
17        'INTEGER' I, LOCK, KEY;
18        'INTEGER''ARRAY' CALENDAR[1:7];
19        'INTEGER''ARRAY' B [1:10];
20
21  'INTEGER''PROCEDURE' WHICH MONTH;
22  'BEGIN'
23        'INTEGER' I;
24        'INTEGER''ARRAY' M [1:12] :=
25                   'OCTAL'(145301),'OCTAL'(143305),'OCTAL'(146701),
26                   'OCTAL'(140720),'OCTAL'(146701),'OCTAL'(145325),
27                   'OCTAL'(145325),'OCTAL'(140725),'OCTAL'(151705),
28                   'OCTAL'(147703),'OCTAL'(147317),'OCTAL'(142305);
29
30        'INTEGER''ARRAY' N [1:12] :=
31                   'OCTAL'(147240),'OCTAL'(141240),'OCTAL'(151240),
32                   'OCTAL'(151240),'OCTAL'(154640),'OCTAL'(147240),
33                   'OCTAL'(146240),'OCTAL'(143640),'OCTAL'(150240),
34                   'OCTAL'(152240),'OCTAL'(153240),'OCTAL'(141640);
35
36        'FOR' I:=1,I+1 'WHILE' I <=12 'DO'
37            'BEGIN'
38            'IF' CALENDAR[2] = M[I] 'AND' CALENDAR[3] = N[I]
39               'THEN''ANSWER' 0
40            'END';
41        'ANSWER' 1
42  'END';
43
44  'PROCEDURE' BAD INPUT('VALUE''INTEGER' OBJECT);
45  'BEGIN'
46        DISPLAY(0,"CANNOT BE ");
47        WRITEN(0,OBJECT,2,0);
48        DISPLAY(0," TRY AGAIN!CLL!")
49  'END';
50
51        LOCK:=0;
52        DISPLAY(0,"!CL!INITIALIZE TIME!CL!");
53        'FOR' KEY:=0  'WHILE' LOCK=0 'DO'
54            'BEGIN'
55            DISPLAY(0,"!S11!DAY (1-31)? "); KEY:=READN(0);
56            'IF' KEY>0 'AND' KEY<=31
```

```
57              'THEN''BEGIN'
58                    CALENDAR[1]:=KEY;
59                    LOCK:=1
60                    'END'
61              'ELSE' BAD INPUT(CALENDAR[1])
62            'END';
63        LOCK:=0;
64        'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
65            'BEGIN'
66            DISPLAY(0,"MONTH (1ST 3 LETTERS)? ");
67            READ RECORD(0,B,10,KEY); WRITE CHAR(0,LF);
68            TOP BYTE(CALENDAR[2]):=B[1]; BOT BYTE(CALENDAR[2]):=B[2];
69            TOP BYTE(CALENDAR[3]):=B[3]; BOT BYTE(CALENDAR[3]):=SP;
70            KEY := WHICH MONTH;
71            'IF' KEY = 0
72               'THEN' LOCK := 1
73               'ELSE''BEGIN'
74                     DISPLAY(0,"CANNOT BE ");
75                     'FOR' I:=2,3 'DO'
76                         'BEGIN'
77                         WRITE CHAR(0,TOP BYTE(CALENDAR[I]));
78                         WRITE CHAR(0,BOT BYTE(CALENDAR[I]))
79                         'END';
80                     DISPLAY(0," TRY AGAIN!CLL!")
81                     'END'
82            'END';
83        LOCK:=0;
84        'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
85            'BEGIN'
86            DISPLAY(0,"!S1!YEAR (LAST 2 DIGITS)? "); KEY:=READN(0);
87            'IF' KEY >= 80 'AND' KEY < 100
88               'THEN''BEGIN'
89                     CALENDAR[4]:=KEY;
90                     LOCK:=1
91                     'END'
92               'ELSE' BAD INPUT(CALENDAR[4])
93            'END';
94        LOCK:=0;
95        'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
96            'BEGIN'
97            DISPLAY(0,"!S9!HOURS (1-23)? "); KEY:=READN(0);
98            'IF' KEY>=0 'AND' KEY<24
99               'THEN''BEGIN'
100                    CALENDAR[5]:=KEY; LOCK:=1
101                    'END'
102              'ELSE' BAD INPUT(KEY)
103           'END';
104       LOCK:=0;
105       'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
106           'BEGIN'
107           DISPLAY(0,"!S7!MINUTES (0-59)? "); KEY:=READN(0);
108           'IF' KEY>=0 'AND' KEY<60
109              'THEN''BEGIN'
110                    CALENDAR[6]:=KEY; LOCK:=1
111                    'END'
112              'ELSE' BAD INPUT(KEY)
```

```
113             'END';
114         CALENDAR[7]:=0;
115         SET TIME(CALENDAR);
116         DISPLAY(0,"!CLL!");
117         EXIT(0)
118   'END'
119   'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 4  'DEFINE' LI "'LOCATION''INTEGER'";
 5  'EXTERNAL' ('PROCEDURE' DISPLAY(VI,VI);
 6              'PROCEDURE' GET PARAM(IA);
 7              'PROCEDURE' ERROR(IA,VI);
 8              'PROCEDURE' WRITEN(VI,VI,VI,VI);
 9              'PROCEDURE' WRITE RECORD(VI,IA,VI);
10              'PROCEDURE' OPEN(VI,IA,VI,VI,LI);
11              'PROCEDURE' CLOSE(VI);
12              'PROCEDURE' READ RECORD(VI,IA,VI,LI)  );
13  'DELETE' IA; 'DELETE' VI; 'DELETE' LI;
14  'LIBRARY' (DBASE);
15  'LIBRARY' (CHARS);
16  'LIBRARY' (LINKS);
17  'LIBRARY' (PARAMS);
18  'SEGMENT' LIST COMMAND HANDLER
19  'BEGIN'
20       'INTEGER' LOCK, KEY, RESULT, LNUM, NUMBER := 0,0,0,0,0;
21       'INTEGER' NU STR := 'OCTAL'(147325);
22       'INTEGER''ARRAY' B[1:80];
23       'INTEGER''ARRAY' FNAME[1:5];
24       'INTEGER''ARRAY' QUALIFIER[1:5];
25
26       GET PARAM(FNAME);
27       'IF' FNAME[1]=BLANK 'THEN' EXIT(2);
28       GET PARAM(QUALIFIER);
29       'IF' QUALIFIER[1] = NU STR 'THEN' NUMBER:=1;
30       OPEN(3,FNAME,UDISK,1,RESULT);
31       'IF' RESULT <> 0 'THEN' ERROR(FNAME,RESULT);
32       DISPLAY(0,"!CL!");
33       'FOR' LOCK:=0 'WHILE' KEY=0 'DO'
34            'BEGIN'
35            READ RECORD(3, B, 80, RESULT);
36            'IF' RESULT = 0
37                'THEN''BEGIN'
38                      'IF' NUMBER = 1
39                          'THEN''BEGIN'
40                                LNUM:=LNUM+1;
41                                WRITEN(0,LNUM,5,1);
42                                DISPLAY(0," ")
43                                'END';
44                      WRITE RECORD(0,B,80)
45                      'END'
46                'ELSE''BEGIN'
47                      KEY:=1
48                      'END'
49            'END';
50       CLOSE(3);
51       DISPLAY(0,"*EOF*!CLL!");
52       EXIT(0)
53  'END'
54  'FINISH'
```

Appendix H cont.

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' VI "'VALUE''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' CHANGE PROTECT(IA,VI);
10               'PROCEDURE' GET PARAM(IA)  );
11  'DELETE' IA; 'DELETE' VI;
12  'SEGMENT' LOCK COMMAND HANDLER
13  'BEGIN'
14         'INTEGER''ARRAY' FNAME[1:5];
15
16         GET PARAM(FNAME);
17         'IF' FNAME[1]=BLANK
18            'THEN' EXIT (2);    (PARAMETER MISSING)
19         CHANGE PROTECT(FNAME, 1); (LOCK FILE)
20         EXIT(0)
21  'END'
22  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 4  'DEFINE' LI "'LOCATION''INTEGER'";
 5  'EXTERNAL' ('PROCEDURE' DISPLAY(VI,VI);
 6              'PROCEDURE' GET PARAM(IA);
 7              'PROCEDURE' ERROR(IA,VI);
 8              'PROCEDURE' WRITEN(VI,VI,VI,VI);
 9              'PROCEDURE' OPEN(VI,IA,VI,VI,LI);
10              'PROCEDURE' CLOSE(VI);
11              'PROCEDURE' READ RECORD(VI,IA,VI,LI);
12              'PROCEDURE' WRITE RECORD(VI,IA,VI);
13              'PROCEDURE' WRITE CHAR(VI,VI)   );
14  'DELETE' IA; 'DELETE' VI; 'DELETE' LI;
15  'LIBRARY' (D3ASE);
16  'LIBRARY' (CHARS);
17  'LIBRARY' (LINKS);
18  'LIBRARY' (PARAMS);
19  'SEGMENT' PRINT COMMAND HANDLER
20  'BEGIN'
21         'INTEGER' KEY, I, NLINE, RESULT;
22         'INTEGER''ARRAY' B[1:80];
23         'INTEGER''ARRAY' FNAME[1:5];
24         'INTEGER''ARRAY' QUALIFIER[1:5];
25         'INTEGER''ARRAY' CALENDER[1:7];
26         'INTEGER' NUMBER, TITLE := 0, 0;
27         'INTEGER' NU STR:='OCTAL'(147325);
28         'INTEGER' TI STR:='OCTAL'(152311);
29
30         GET PARAM(FNAME);
31        'IF' FNAME[1]=BLANK 'THEN' EXIT(2);
32         OPEN(3, FNAME, UDISK, 1, RESULT);
33         'IF' RESULT > 0 'THEN' ERROR(FNAME,RESULT);
34         'FOR' I:=1,2 'DO'
35              'BEGIN'
36              GET PARAM(QUALIFIER);
37              'IF' QUALIFIER[1]=NU STR
38                 'THEN' NUMBER:=1
39                 'ELSE''IF' QUALIFIER[1]=TI STR
40                             'THEN' TITLE:=1
41              'END';
42          'IF' TITLE=1
43             'THEN''BEGIN'
44                   DISPLAY(1,"LISTING OF FILE : ");
45                   'FOR' I:=1,I+1 'WHILE' I<=4 'DO'
46                        'BEGIN'
47                        WRITE CHAR(1,TOP BYTE(FNAME[I]));
48                        WRITE CHAR(1,BOT BYTE(FNAME[I]))
49                        'END';
50                   FETCH TIME(CALENDER);
51                   DISPLAY(1,"           ON ");
52                   WRITEN(1,CALENDER[1],2,0); WRITE CHAR(1,'LITERAL'(.));
53                   WRITE CHAR(1,TOP BYTE(CALENDER[2]));
54                   WRITE CHAR(1,BOT BYTE(CALENDER[2]));
55                   WRITE CHAR(1,TOP BYTE(CALENDER[3]));
56                   WRITE CHAR(1,'LITERAL'(.));
```

```
57                    WRITEN(1,CALENDER[4],2,0);
58                    DISPLAY(1,"    AT ");
59                    WRITEN(1,CALENDER[5],2,0); WRITE CHAR(1,'LITERAL'(:));
60                    WRITEN(1,CALENDER[6],2,0); WRITE CHAR(1,'LITERAL'(:));
61                    WRITEN(1,CALENDER[7],2,0); DISPLAY(1,"!CLL!")
62                    'END';
63          KEY := 0;
64          'FOR' NLINE:=1,NLINE+1 'WHILE' KEY=0 'DO'
65              'BEGIN'
66              READ RECORD(3, B, 80, RESULT);
67              'IF' RESULT = 1
68                 'THEN''BEGIN'
69                       DISPLAY(1,"*EOF*!CLL!");
70                       KEY := 1
71                       'END'
72                 'ELSE''BEGIN'
73                       'IF' NUMBER=1
74                          'THEN''BEGIN'
75                                WRITEN(1,NLINE,5,1);
76                                DISPLAY(1," ")
77                                'END';
78                       WRITE RECORD(1,B,80)
79                      'END'
80              'END';
81          CLOSE(3);
82          EXIT(0)
83    'END'
84    'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'EXTERNAL' ('PROCEDURE' RENAME FILE(IA,IA);
 9              'PROCEDURE' GET PARAM(IA)  );
10  'DELETE' IA;
11  'SEGMENT' RENAME COMMAND HANDLER
12  'BEGIN'
13         'INTEGER''ARRAY' FROM[1:5], TO[1:5];
14
15         GET PARAM(FROM);
16         GET PARAM(TO);
17        'IF' FROM[1]=BLANK 'OR' TO[1]=BLANK
18           'THEN' EXIT(2);  (PARAM.MISSING)
19         RENAME FILE(FROM, TO);
20         EXIT(0)
21  'END'
22  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (PARAMS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (LINKS);
 7  'DEFINE' VI "'VALUE''INTEGER'";
 8  'EXTERNAL' ('PROCEDURE' DISPLAY(VI,VI);
 9              'PROCEDURE' GET PARAM('INTEGER''ARRAY');
10              'PROCEDURE' WRITE CHAR(VI,VI);
11              'PROCEDURE' WRITEN(VI,VI,VI,VI)  );
12  'DELETE' VI;
13  'SEGMENT' SPACE COMMAND HANDLER
14  'BEGIN'
15          'INTEGER' ADDR, I, J, X, SECTOR COUNT;
16          'INTEGER''ARRAY' ID [1:3];
17          'INTEGER''ARRAY' SECTORS [0:7] := 1,2,4,8,16,32,64,128;
18          'INTEGER' CN, DEVICE:=0,1;
19          'INTEGER''ARRAY' QUALIFIER[1:5];
20          'INTEGER' PR STR:='OCTAL'(150322);
21          'INTEGER' SY STR:='OCTAL'(151731);
22
23          'FOR' I:=1,2 'DO'
24              'BEGIN'
25              GET PARAM(QUALIFIER);
26              'IF' QUALIFIER[1]=PR STR
27                  'THEN' CN:=1
28                  'ELSE''IF' QUALIFIER[1]=SY STR
29                             'THEN' DEVICE:=0
30              'END';
31          ALLOCATE(SBS, ADDR);
32          DISKOP(1,DEVICE,0,ADDR);
33          ID[1]:=[ADDR]; ID[2]:=[ADDR+1]; ID[3]:=[ADDR+2]; (FETCH ID)
34          DISKOP(1, DEVICE, 1, ADDR); (READ DISC STATUS)
35          SECTOR COUNT := 0;
36          'FOR' I:=ADDR+4,I+1 'WHILE' I <= ADDR+4+TOP TRACK 'DO'
37              'BEGIN'
38              'IF' [I] > 0
39                  'THEN''BEGIN'
40                        'FOR' J:=0 'STEP' 1 'UNTIL' 7 'DO'
41                             'BEGIN'
42                             X := [I] 'MASK' SECTORS[J];
43                             'IF' X = SECTORS[J]
44                                 'THEN' SECTOR COUNT := SECTOR COUNT + 1
45                             'END'
46                        'END'
47              'END';
48          FREE(SBS, ADDR);
49          DISPLAY(CN,"!CL!FREE SPACE ON DISK : ");
50          'FOR' I:=1,I+1 'WHILE' I<=3 'DO'
51              'BEGIN'
52              WRITE CHAR(CN,TOP BYTE(ID[I]));
53              WRITE CHAR(CN,BOT BYTE(ID[I]))
54              'END'; DISPLAY(CN,"!CL!    ");
55          WRITEN(CN,SECTOR COUNT,5,1);
56          DISPLAY(CN," SECTORS OUT OF ");
```

```
57         WRITEN(CN, (TOP TRACK+1)*8, 5, 1 );
58         'IF' SECTOR COUNT=0 'THEN' DISPLAY(CN,"  *DISC FULL*");
59         DISPLAY(CN,"!CLL!");
60         EXIT(0)
61   'END'
62   'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'DEFINE' VI "'VALUE''INTEGER'";
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'EXTERNAL' ('PROCEDURE' DISPLAY(VI,VI);
 9             'PROCEDURE' WRITE CHAR(VI,VI);
10             'PROCEDURE' WRITEN(VI,VI,VI,VI)  );
11  'SEGMENT' TIME COMMAND HANDLER
12  'BEGIN'
13       'INTEGER' DOT := 'LITERAL'(.);
14       'INTEGER' COLON := 'LITERAL'(:);
15       'INTEGER''ARRAY' CALENDER [1:7];
16
17       FETCH TIME(CALENDER);
18
19       DISPLAY(0,"!CL!    ");
20       WRITEN(0,CALENDER[1],2,0);              (DAY)
21       WRITE CHAR(0,DOT);
22
23       WRITE CHAR(0,TOP BYTE(CALENDER[2])); (MONTH)
24       WRITE CHAR(0,BOT BYTE(CALENDER[2]));
25       WRITE CHAR(0,TOP BYTE(CALENDER[3]));
26       WRITE CHAR(0,DOT);
27
28       WRITEN(0,CALENDER[4],2,0);              (YEAR)
29       DISPLAY(0,"   ");
30
31       WRITEN(0,CALENDER[5],2,0);              (HOURS)
32       WRITE CHAR(0,COLON);
33
34       WRITEN(0,CALENDER[6],2,0);              (MINUTES)
35       WRITE CHAR(0, COLON);
36
37       WRITEN(0,CALENDER[7],2,0);              (SECONDS)
38       DISPLAY(0,"!CLL!");
39       EXIT(0)
40  'END'
41  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' PIL INTERFACE
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' VI "'VALUE''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' CHANGE PROTECT(IA,VI);
10              'PROCEDURE' GET PARAM(IA)   );
11  'DELETE' IA; 'DELETE' VI;
12  'SEGMENT' UNLOCK COMMAND HANDLER
13  'BEGIN'
14         'INTEGER''ARRAY' FNAME[1:5];
15
16         GET PARAM(FNAME);
17         'IF' FNAME[1]=BLANK
18            'THEN' EXIT(2);     (PARAMETER MISSING)
19         CHANGE PROTECT(FNAME, 0); (UNLOCK FILE)
20         EXIT(0)
21  'END'
22  'FINISH'
```

Appendix I

PORTOS Kernel Tasks Source Listings

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'SEGMENT' TASK MANAGER
 5  'BEGIN'
 6        'INTEGER' DAD, INDEX, R;
 7
 8        INDEX := TNUM;
 9        'IF' INDEX > 0
10           'THEN''BEGIN'
11                 'COMMENT' ACTIVATE TASK "TNUM";
12                 FATHER [INDEX] := ACTIVE; (SAVE ACTIVE TASK)
13                 ACTIVE := INDEX;
14                 R := [['OCTAL'(65)]];     (SAVE  PC)
15                 RETURN [ACTIVE] := R;
16                 R := VECTOR [ACTIVE];     (ENTRY POINT OF TASK "TNUM")
17                 'CODE''BEGIN'
18                   ´    JMP    ´R´,*     ;´
19                        'END'
20                 'END'
21           'ELSE''BEGIN'
22                 'COMMENT' RE-ACTICATE THE ORIGINATOR
23                           OF THE ACTIVE TASK;
24                 DAD := FATHER [ACTIVE];
25                 FATHER [ACTIVE] := 0;
26                 R := RETURN [ACTIVE];
27                 ACTIVE := DAD;    (RESTORE CPU STATE)
28                 'CODE''BEGIN'
29                   ´    JMP    ´R´,*    ;´
30                        'END'
31                 'END'
32
33  'END'
34  'FINISH'
```

Appendix I cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'SEGMENT' ALLOCATE PRIMITIVE
 6  'BEGIN'
 7        'INTEGER' S; (SCANNER)
 8        'INTEGER' P; (PREVIOUS)
 9
10        'COMMENT'   **  ALLOCATE MEMORY  **;
11
12        'DEFINE' DEMAND "PAR1";
13        'DEFINE' BASE   "PAR2";
14        'DEFINE' PARAM  "PAR4";
15
16        'IF' HEAD = 0 'OR' DEMAND > [TAIL] 'OR' DEMAND < 10
17           'THEN' BASE:=-1 'COMMENT' REQUEST DENIED;
18           'ELSE''BEGIN'
19                 P := HEAD;
20                 'FOR' S := HEAD, [S+1] 'WHILE' P <>0 'DO'
21                     'BEGIN'
22                     'IF' DEMAND <= [S]
23                        'THEN''BEGIN'
24                              BASE := S;
25                              'IF' P = HEAD
26                                 'THEN' HEAD := [S+1]
27                                  'ELSE' [P+1]:=[S+1];
28                              'IF' HEAD=0 'THEN' TAIL := 0;
29                              'IF' DEMAND < [S]
30                                 'THEN''BEGIN'
31                                       [S+DEMAND]:=[S]-DEMAND;
32                                       S := S + DEMAND;
33                                       PARAM := S;
34                                       ACTIVATE(3)
35                                       'END';
36                               P := 0
37                               'END'
38                          'ELSE'
39                               P := S
40                     'END'
41                 'END';
42         ACTIVATE(0)
43  'END'
44  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'SEGMENT' FREE PRIMITIVE
 6  'BEGIN'
 7         'INTEGER' I, S, P, TOP, UL;
 8
 9         'DEFINE' AMOUNT "PAR1";
10         'DEFINE' ADDR   "PAR2";
11         'DEFINE' PARAM  "PAR4";
12
13         I := 0;
14         'IF' HEAD = 0
15            'THEN''BEGIN'
16                  HEAD := ADDR; TAIL := ADDR;
17                  [HEAD]:= AMOUNT; [HEAD+1] := 0
18                  'END'
19            'ELSE''BEGIN'
20                  TOP := ADDR + AMOUNT;
21                  'FOR' S := HEAD, [S+1] 'WHILE' S <> 0 'DO'
22                      'BEGIN'
23                      UL := S + [S];
24                      'IF' ADDR = UL 'OR' TOP = S
25                         'THEN''BEGIN'
26                                I := 1;
27                               AMOUNT := AMOUNT + [S];
28                               'IF' HEAD = TAIL
29                                  'THEN''BEGIN'
30                                        HEAD := 0;
31                                        TAIL := 0
32                                        'END'
33                                  'ELSE''IF' S = HEAD
34                                           'THEN' HEAD := [S+1]
35                                           'ELSE''BEGIN'
36                                                 [P+1]:=[S+1];
37                                                 'IF' [P+1] = 0
38                                                    'THEN' TAIL := P
39                                                 'END';
40                               'IF' ADDR = UL 'THEN' ADDR := S
41                               'END';
42                      'IF' I = 0 'THEN' P := S
43                                 'ELSE' I := 0
44                      'END';
45                  [ADDR] := AMOUNT;
46                  PARAM := ADDR;
47                  ACTIVATE(3);
48                  'END';
49         ACTIVATE(0)
50  'END'
51  'FINISH'
```

Appendix I cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'SEGMENT' INSERT PRIMITIVE
 6  'BEGIN'
 7        'COMMENT'   **  INSERT MEMORY BLOCK IN LIST  **;
 8
 9        'COMMENT' THE TASK ASSUMES THAT THE LENGTH OF
10                  PARTITION TO BE INSERTED INTO THE FREE
11                  LIST IS GIVEN IN THE FIRST WORD;
12
13        'INTEGER' S; (SCANNER : POINTING TO CURRENT PARTITION)
14        'INTEGER' P; (PREVIOUS : POINTING TO PREVIOUS PARTITION)
15
16        'DEFINE' ADDR "PAR4";
17
18        'IF' HEAD = 0
19           'THEN''BEGIN'
20                 HEAD := ADDR;
21                 [ADDR+1] := 0;
22                 TAIL := ADDR
23                 'END'
24           'ELSE''IF' [ADDR] <= [HEAD]
25                  'THEN''BEGIN'
26                        [ADDR+1] := HEAD;
27                        HEAD := ADDR
28                        'END'
29                  'ELSE''IF' [ADDR] >= [TAIL]
30                         'THEN''BEGIN'
31                               [TAIL+1] := ADDR;
32                               [ADDR+1] := 0;
33                               TAIL := ADDR
34                               'END'
35                         'ELSE''BEGIN'
36                               S := HEAD;
37                               P := HEAD;
38                               'FOR' S:=[S+1] 'WHILE' S<>0 'DO'
39                                  'BEGIN'
40                                  'IF' [ADDR]<=[S]
41                                     'THEN''BEGIN'
42                                           [P+1] := ADDR;
43                                           [ADDR+1] := S
44                                           'END'
45                                     'ELSE'
46                                           P := S
47                                  'END'
48                               'END';
49
50        ACTIVATE(0)
51
52  'END'
53  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (PARAMS);
 5  'LIBRARY' (CHARS);
 6  'SEGMENT' GIVE SECTOR PRIMITIVE
 7  'BEGIN'
 8        'INTEGER' ADDR, I, TN, SN;
 9        'INTEGER''ARRAY' SECTORS[0:7] := 1, 2, 4, 8, 16, 32, 64, 128;
10        'LIBRARY' (LINKS);
11
12  'INTEGER''PROCEDURE' FIND FREE SECTOR('VALUE''INTEGER' TRACK);
13  'BEGIN'
14        'INTEGER' I, RECORD;
15        'FOR' RECORD:=1, RECORD*2 'WHILE' RECORD<=128 'DO'
16            'BEGIN'
17            'IF' ([TRACK] 'MASK' BOT BYTE(RECORD))=BOT BYTE(RECORD)
18               'THEN''BEGIN'
19                     [TRACK]:=[TRACK] 'DIFFER' BOT BYTE(RECORD);
20                     'FOR' I:=0,I+1 'WHILE' I<=7 'DO'
21                         'IF' SECTORS[I] = BOT BYTE(RECORD)
22                            'THEN''ANSWER' I
23                     'END'
24            'END';
25        'ANSWER' -1
26  'END';
27
28        SN := -1; (ASSUME DISC IS FULL)
29        ALLOCATE(SBS, ADDR);
30        DISKOP(1, UDISK, 1, ADDR);
31        'FOR' I:=ADDR+4,I+1 'WHILE' I<= ADDR+TOP TRACK 'DO'
32            'BEGIN'
33            'IF' [I] > 0
34               'THEN''BEGIN'
35                     TN := I-ADDR-4; (FOUND TRACK)
36                     SN := FIND FREE SECTOR(I);
37                     I := ADDR+500
38                     'END'
39            'END';
40        'IF' SN < 0 'THEN' TN:=0; (DISC FULL)
41        DISKOP(2, UDISK, 1, ADDR);
42        FREE(SBS, ADDR);
43        PAR1 := TN*8 + SN;
44        ACTIVATE(0)
45
46  'END'
47  'FINISH'
```

Appendix I cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (PARAMS);
 5  'SEGMENT' RELEASE SECTOR PRIMITIVE
 6  'BEGIN'
 7         'INTEGER' ADDR, TN, SN;
 8         'INTEGER''ARRAY' SECTORS[0:7] := 1, 2, 4, 8, 16, 32, 64, 128;
 9         'LIBRARY' (LINKS);
10
11         SN := PAR1;
12         TN := SN/8;
13         SN := SN - (TN*8);
14         ALLOCATE(SBS, ADDR);
15         DISKOP(1, UDISK, 1, ADDR); (DISC STATUS)
16         TN := TN + ADDR +4;  (INDEX TO CORRECT ENTRY)
17         [TN] := [TN] 'UNION' SECTORS[SN]; (UPDATE)
18         DISKOP(2, UDISK, 1, ADDR);
19         FREE(SBS, ADDR);
20         ACTIVATE(0)
21
22  'END'
23  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (PARAMS);
 6  'SEGMENT' EXIT PRIMITIVE
 7  'BEGIN'
 8        'DEFINE' STR DCODER
 9                 "'HEX'(C4C3),'HEX'(CFC4),'HEX'(C5D2),'HEX'(AEC2)";
10        'DEFINE' STR XER
11                 "'HEX'(D8C5),'HEX'(D2AE),'HEX'(C2A0),'HEX'(A0A0)";
12
13        'INTEGER' ADDR, SN;
14        'INTEGER''ARRAY' DCODER [1:4] := STR DCODER;
15        'INTEGER''ARRAY' XER [1:4] := STR XER;
16
17        'COMMENT'  **  SYSTEM  STARTUP & RECOVERY  **;
18
19  'PROCEDURE' FIND ENTRY('INTEGER''ARRAY' NAME);
20  'BEGIN'
21        'INTEGER' ENTRY, LOCK;
22        LOCK  := 0;
23        'FOR' ENTRY:=ADDR+4,ENTRY+7 'WHILE' LOCK=0 'DO'
24            'BEGIN'
25            'IF' [ENTRY] = 'LITERAL'(*)
26               'THEN' LOCK := 1
27               'ELSE''IF' [ENTRY+1]=NAME[1] 'AND' [ENTRY+2]=NAME[2] 'AND'
28                          [ENTRY+3]=NAME[3] 'AND' [ENTRY+4]=NAME[4]
29                      'THEN''BEGIN'
30                            SN := [ENTRY+6];
31                            LOCK := 1
32                            'END'
33            'END'
34  'END';
35
36        'IF' PAR8 > 0 'AND' PAR8 <> 111
37           'THEN''BEGIN'
38                 SHUTDOWN IO
39                 'END';
40        ACTIVE := 0;
41        RESET MEMORY;
42        ALLOCATE(SBS, ADDR);
43        DISKOP(1, SDISK, 3, ADDR); (READ PORTOS)
44        'IF' PAR8=0 'THEN' FIND ENTRY(DCODER)
45                    'ELSE' FIND ENTRY(XER);
46        FREE(SBS, ADDR);
47        LOAD(SDISK, SN)
48
49  'END'
50  'FINISH'
```

```
1   'CORAL'
2   'PROGRAM' KERNEL
3   'LIBRARY' (DBASE);
4   'LIBRARY' (LINKS);
5   'LIBRARY' (PARAMS);
6   'SEGMENT' LOAD PRIMITIVE
7   'BEGIN'
8         'INTEGER' ADDR, I, KEY, ILA, SIZE, SN, PC, WA;
9
10        'DEFINE'  EOF  "'OCTAL'(177771)";
11
12        SN := PAR1;
13        ALLOCATE(SBS, ADDR);
14        DISKOP(1,SDISK,SN,ADDR);
15        SIZE := [ADDR+5]-'OCTAL'(1000); (SIZE IN WORDS)
16        RESET MEMORY; ALLOCATE(SIZE, ILA);
17        ALLOCATE(SBS, ADDR); DISKOP(1,SDISK,SN,ADDR);
18        ILA := [ADDR+4]; PC := [ADDR+6];
19        WA := ILA; I := ADDR+13; KEY := 0;
20        'FOR' SIZE := 0 'WHILE' KEY = 0 'DO'
21            'BEGIN'
22            'IF' [I]=EOF
23               'THEN' KEY:=1
24               'ELSE''BEGIN'
25                     [WA] := [I];
26                     WA := WA + 1;
27                     I := I + 1;
28                     'IF' I >= ADDR+SBS
29                        'THEN''BEGIN'
30                              SN := [ADDR+1];
31                              DISKOP(1, SDISK, SN, ADDR);
32                              I := ADDR+4
33                              'END'
34                     'END'
35            'END';
36        FREE(SBS, ADDR);
37        'CODE''BEGIN'
38          ´   JMP   ´PC´,*      ;´
39              'END'
40
41  'END'
42  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'EXTERNAL' ('PROCEDURE' PUTCHAR('VALUE''INTEGER');
 7              'PROCEDURE' MESSAGE('VALUE''INTEGER');
 8              'INTEGER''PROCEDURE' GETCHAR  );
 9  'SEGMENT' GET RECORD PRIMITIVE
10  'BEGIN'
11        'INTEGER' DATA, I, N, PTR;
12
13        'COMMENT'    **  READ RECORD FROM TERMINAL  **;
14
15        PTR:=PAR1;
16        N:=PAR2;
17        PUTCHAR(BEL);
18        'FOR' I:=1,I+1 'WHILE' I<=N 'DO'
19            'BEGIN'
20            DATA:=GETCHAR;
21            'IF' DATA=DEL
22               'THEN''BEGIN'
23                     MESSAGE(" *CANCEL*!CL!");
24                     PUTCHAR(ERL); PUTCHAR(BEL);
25                     PTR:=PAR1; I:=0
26                     'END'
27               'ELSE''IF' DATA=KILL
28                        'THEN''BEGIN'
29                              I:=I-1;
30                              'IF' I < 0 'THEN' I:=0;
31                              PTR:=PTR-1;
32                              'IF' PTR < PAR1 'THEN' PTR:=PAR1
33                              'END'
34               'ELSE''BEGIN'
35                     [PTR]:=DATA;
36                     'IF' DATA=CR
37                        'THEN' I:=N
38                        'ELSE' PTR:=PTR+1
39                     'END'
40               'END';
41        ACTIVATE(0)
42
43  'END'
44  'FINISH'
```

Appendix I cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'EXTERNAL' ('PROCEDURE' PUTCHAR('VALUE''INTEGER')  );
 7  'SEGMENT' PUT RECORD PRIMITIVE
 8  'BEGIN'
 9        'INTEGER' DATA, I, N, PTR;
10
11        PTR:=PAR1;  (BUFFER ADDR)
12        N:=PAR2;    (NO.OF WORDS)
13        'FOR' I:=1,I+1 'WHILE' I<=N 'DO'
14            'BEGIN'
15            DATA:=[PTR];
16            PUTCHAR(DATA);
17            'IF' DATA=CR
18               'THEN''BEGIN'
19                     PUTCHAR(LF);
20                     I:=N
21                     'END'
22               'ELSE' PTR:=PTR+1
23             'END';
24        ACTIVATE(0)
25  'END'
26  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (PARAMS);
 6  'LIBRARY' (CHARS);
 7  'SEGMENT' SHUTDOWN IO PRIMITIVE
 8  'BEGIN'
 9        'INTEGER' FILEND := 'OCTAL'(147777);
10        'INTEGER' ACC, I;
11
12        'FOR' I:=0,I+1 'WHILE' I<=4 'DO'
13            'BEGIN'
14            'IF' CHANNEL[I] >= 0
15               'THEN''BEGIN'
16                     CHANNEL[I]:=-1;
17                     'IF' FUNCTION[I] = 2
18                        'THEN''BEGIN'
19                              ACC := BUFFER[I] + WORD[I];
20                              'IF' BYTE[I] = 1
21                                 'THEN''BEGIN'
22                                       BOT BYTE(WORD[I]) := 0;
23                                       ACC := ACC + 1
24                                       'END';
25                              [ACC] := FILEND;
26                              DISKOP(2,UDISK,SECTOR[I],BUFFER[I])
27                              'END';
28                      FREE(SBS, BUFFER[I])
29                      'END'
30             'END';
31        ACTIVATE(0)
32  'END'
33  'FINISH'
```

Appendix I cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'SEGMENT' FETCH TIME PRIMITIVE
 6  'BEGIN'
 7       'INTEGER' PTR;
 8
 9       PTR:=PAR1;
10       [PTR]:=DATE[1]; PTR:=PTR+1; (DAY)
11       [PTR]:=DATE[2]; PTR:=PTR+1; (MONTH)
12       [PTR]:=DATE[3]; PTR:=PTR+1;
13       [PTR]:=DATE[4]; PTR:=PTR+1; (YEAR)
14       [PTR]:=HOURS;   PTR:=PTR+1;
15       [PTR]:=MINS;    PTR:=PTR+1;
16       [PTR]:=SECS;
17       ACTIVATE(0)
18  'END'
19  'FINISH'
```

Appendix J

PORTOS Kernel Routines Source Listings

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' FETCH ENTRY(IA,VI,LI,LI,LI);
10              'PROCEDURE' COPY FILE(IA,IA,VI);
11              'PROCEDURE' DELETE FILE(IA,VI);
12              'PROCEDURE' ERROR(IA,VI);
13              'PROCEDURE' MAKE ENTRY (IA,VI,VI,VI,LI)  );
14  'DELETE' IA; 'DELETE' VI; 'DELETE' LI;
15  'SEGMENT' COPY FILE PRIMITIVE
16  'BEGIN'
17       'INTEGER' ADDR,KEY,XSN,SN,LOCK,S,TYPE,PROT;
18
19  'PROCEDURE' COPY FILE('INTEGER''ARRAY' FROM,TO;'VALUE''INTEGER' OUST);
20  'BEGIN'
21       FETCH ENTRY(FROM, UDISK, SN, TYPE, PROT);
22       'IF' SN < 0
23          'THEN' ERROR(FROM,3)  'COMMENT' SOURCE NOT FOUND;
24          'ELSE''BEGIN'                                                    ]
25               FETCH ENTRY(TO, UDISK, XSN, TYPE, PROT);                    ]
26               'IF' XSN >= 0
27                  'THEN''BEGIN'
28                        'IF' OUST = 0
29                           'THEN' ERROR(TO,4)  'COMMENT' TARGET NOT UNIQ
30                           'ELSE''BEGIN'
31                                 DELETE FILE(TO,UDISK);
32                                 MAKE ENTRY(TO,UDISK,TYPE,0,XSN)
33                                 'END'
34                         'END'
35                'END';
36        LOCK:=0;
37        ALLOCATE(SBS, ADDR);
38        'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
39            'BEGIN'
40            DISKOP(1, UDISK, SN, ADDR);  (READ FILE)
41            'IF' [ADDR+1]=0
42               'THEN''BEGIN'
43                     DISKOP(2, UDISK, XSN, ADDR);
44                     LOCK:=1
45                     'END'
46               'ELSE''BEGIN'
47                     SN:=[ADDR+1];
48                     GIVE SECTOR(UDISK,S);
49                     [ADDR+1]:=S;
50                     DISKOP(2, UDISK, XSN, ADDR);
```

Appendix J cont.

```
51                          XSN:=S
52                          'END'
53                  'END';
54            FREE(SBS, ADDR)
55      'END';
56      'END'
57      'FINISH'
```

Appendix J cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 8  'EXTERNAL' ('PROCEDURE' CHANGE PROTECT(IA,VI);
 9              'PROCEDURE' ERROR(IA,VI)  );
10  'DELETE' IA; 'DELE TE' VI;
11  'SEGMENT' CHANGE PROTECT PRIMITIVE
12  'BEGIN'
13
14  'PROCEDURE' CHANGE PROTECT('INTEGER''ARRAY' NAME;
15                             'VALUE''INTEGER' PROTKEY);
16  'BEGIN'
17        'INTEGER' ENTRY, KEY, ADDR, RESULT;
18
19        KEY:=0; RESULT:=3; (ASSUME ENTRY NOT FOUND)
20        ALLOCATE(SBS, ADDR);
21        DISKOP(1, UDISK, 2, ADDR);
22        'FOR' ENTRY:=ADDR+4,ENTRY+7 'WHILE' KEY=0 'DO'
23            'BEGIN'
24            'IF' [ENTRY] = 'LITERAL'(*)
25               'THEN' KEY := 1
26               'ELSE''IF' [ENTRY+1]=NAME[1] 'AND' [ENTRY+2]=NAME[2] 'AND'
27                          [ENTRY+3]=NAME[3] 'AND' [ENTRY+4]=NAME[4]
28                        'THEN''BEGIN'
29                              BOT BYTE([ENTRY]):=PROTKEY;
30                              DISKOP(2, UDISK, 2, ADDR);                   ]
31                              RESULT := 0;                                 ]
32                              KEY := 1
33                              'END'
34            'END';
35        FREE(SBS, ADDR);
36        'IF' RESULT > 0
37           'THEN' ERROR(NAME,RESULT)
38  'END';
39
40  'END'
41  'FINISH'
```

Appendix J cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' VI "'VALUE''INTEGER'";
 9  'DEFINE' LI "'LOCATION''INTEGER'";
10  'EXTERNAL' ('PROCEDURE' DELETE FILE(IA,VI);
11              'PROCEDURE' KILL ENTRY(IA,VI,LI)  );
12  'DELETE' IA; 'DELETE' LI; 'DELETE' VI;
13  'SEGMENT' DELETE FILE PRIMITIVE
14  'BEGIN'
15        'INTEGER' ADDR, SN;
16
17  'PROCEDURE' DELETE FILE('INTEGER''ARRAY' NAME;
18                          'VALUE''INTEGER' DEVICE);
19  'BEGIN'
20        'INTEGER' I, KEY;
21
22        KILL ENTRY(NAME,DEVICE,SN);
23        ALLOCATE(SBS, ADDR);
24        KEY := 0;
25        'FOR' I:=1 'WHILE' KEY = 0 'DO'
26            'BEGIN'
27            DISKOP(1,DEVICE,SN,ADDR); (1ST REC.OF FILE)
28            RELEASE SECTOR(DEVICE, SN);
29            'IF' [ADDR+1] = 0
30               'THEN' KEY := 1
31               'ELSE' SN := [ADDR+1]
32            'END';
33        FREE(SBS, ADDR)
34  'END';
35  'END'
36  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' RENAME FILE(IA,IA);
10              'PROCEDURE' ERROR(IA,VI);
11              'PROCEDURE' FETCH ENTRY(IA,VI,LI,LI,LI)    );
12  'DELETE' IA; 'DELETE' VI; 'DELETE' LI;
13  'SEGMENT' RENAME FILE PRIMITIVE
14  'BEGIN'
15
16  'PROCEDURE' RENAME FILE('INTEGER''ARRAY' FROM,TO);
17  'BEGIN'
18        'INTEGER' ENTRY, LOCK, TYPE, PROT, SN, ADDR, RESULT;
19
20        FETCH ENTRY(TO, UDISK, SN, TYPE, PROT);
21        'IF' SN >= 0
22           'THEN' ERROR(TO, 4)  'COMMENT' TARGET NOT UNIQUE;
23           'ELSE''BEGIN'
24                 ALLOCATE(SBS, ADDR);
25                 DISKOP(1,UDISK,2,ADDR);
26                 LOCK  := 0; RESULT := 3; (ASSUME SOURCE NOT FOUND)
27                 'FOR' ENTRY:=ADDR+4,ENTRY+7 'WHILE' LOCK=0 'DO'
28                     'BEGIN'
29                     'IF' [ENTRY] = 'LITERAL'(*)
30                        'THEN' LOCK := 1
31                        'ELSE''IF' [ENTRY+1]=FROM[1]'AND'[ENTRY+2]=FROM[2]
32                                   [ENTRY+3]=FROM[3]'AND'[ENTRY+4]=FROM[4]
33                                'THEN''BEGIN'
34                                      [ENTRY+1] := TO[1];
35                                      [ENTRY+2] := TO[2];
36                                      [ENTRY+3] := TO[3];
37                                      [ENTRY+4] := TO[4];
38                                      RESULT := 0;  (OK)
39                                      DISKOP(2,UDISK,2,ADDR);
40                                      LOCK := 1
41                                      'END'
42                     'END';
43                 FREE(SBS, ADDR)
44                 'END';
45        'IF' RESULT > 0
46           'THEN' ERROR(FROM, RESULT)
47  'END';
48  'END'
49  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL'('PROCEDURE' MAKE ENTRY(IA,VI,VI,VI,LI)  );
10  'DELETE' IA; 'DELETE' VI;
11  'SEGMENT' MAKE ENTRY PRIMITIVE
12  'BEGIN'
13
14  'PROCEDURE' MAKE ENTRY('INTEGER''ARRAY' NAME;
15                         'VALUE''INTEGER' DEVICE, TYPE, PROTECT;
16                         'LOCATION''INTEGER' SECT);
17  'BEGIN'
18        'INTEGER' ADDR, ENT, KEY, SN;
19
20        KEY := 0;
21        ALLOCATE(SBS, ADDR);
22        'IF' DEVICE=SDISK
23           'THEN''BEGIN'
24                 DISKOP(1,SDISK,3,ADDR)
25                 'END'
26           'ELSE''BEGIN'
27                 DISKOP(1,UDISK,2,ADDR)
28                 'END';
29        GIVE SECTOR(DEVICE,SN);
30        SECT:=SN;
31        'FOR' ENT:=ADDR+4,ENT+7 'WHILE' KEY = 0 'DO'
32             'BEGIN'
33             'IF' [ENT]='LITERAL'(#) 'OR' [ENT]='LITERAL'(*)
34                'THEN''BEGIN'
35                      KEY := 1;  (TERMINATE SEARCH)
36                      'IF' [ENT]='LITERAL'(*)'THEN'[ENT+7]:='LITERAL'(*);
37                      TOP BYTE([ENT]) := TYPE;
38                      BOT BYTE([ENT]) := PROTECT;
39                      [ENT+1] := NAME[1];
40                      [ENT+2] := NAME[2];
41                      [ENT+3] := NAME[3];
42                      [ENT+4] := NAME[4];
43                      [ENT+5] := 0;
44                      [ENT+6] := SN
45                      'END'
46             'END';
47        'IF' DEVICE=SDISK
48           'THEN''BEGIN'
49                 DISKOP(2,SDISK,3,ADDR)
50                 'END'
51           'ELSE''BEGIN'
52                 DISKOP(2,UDISK,2,ADDR)
53                 'END';
54        FREE(SBS, ADDR)
55  'END';
56
```

Appendix J cont.

57 'END'
58 'FINISH'

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' FETCH ENTRY(IA,VI,LI,LI,LI)  );
10  'DELETE' IA; 'DELETE' VI; 'DELETE' LI;
11  'SEGMENT' FETCH ENTRY PRIMITIVE
12  'BEGIN'
13
14  'PROCEDURE' FETCH ENTRY('INTEGER''ARRAY' NAME;
15                          'VALUE''INTEGER' DEVICE;
16                          'LOCATION''INTEGER' SN, TYPE, PROTKEY);
17  'BEGIN'
18        'INTEGER' ADDR, ENTRY, LOCK;
19
20        ALLOCATE(SBS, ADDR);
21        'IF' DEVICE=SDISK
22           'THEN''BEGIN'
23                 DISKOP(1,SDISK,3,ADDR)
24                 'END'
25           'ELSE''BEGIN'
26                 DISKOP(1,UDISK,2,ADDR)
27                 'END';
28        LOCK  := 0; SN := -1; (ASSUME ENTRY NOT FOUND)
29        'FOR' ENTRY:=ADDR+4,ENTRY+7 'WHILE' LOCK=0 'DO'
30             'BEGIN'
31             'IF' [ENTRY] = 'LITERAL'(*)
32                'THEN' LOCK := 1
33                'ELSE''IF' [ENTRY+1]=NAME[1] 'AND' [ENTRY+2]=NAME[2] 'AND'
34                           [ENTRY+3]=NAME[3] 'AND' [ENTRY+4]=NAME[4]
35                        'THEN''BEGIN'
36                              SN := [ENTRY+6];
37                              TYPE := TOP BYTE([ENTRY]);
38                              PROTKEY := BOT BYTE([ENTRY]);
39                              LOCK := 1
40                              'END'
41             'END';
42        FREE(SBS, ADDR)
43  'END';
44
45  'END'
46  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (CHARS);
 5  'LIBRARY' (LINKS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' VI "'VALUE''INTEGER'";
 9  'DEFINE' LI "'LOCATION''INTEGER'";
10  'EXTERNAL' ('PROCEDURE' KILL ENTRY(IA,VI,LI);
11              'PROCEDURE' ERROR(IA,VI)  );
12  'DELETE' IA; 'DELETE' VI; 'DELETE' LI;
13  'SEGMENT' KILL ENTRY PRIMITIVE
14  'BEGIN'
15
16  'PROCEDURE' KILL ENTRY('INTEGER''ARRAY' NAME;
17                         'VALUE''INTEGER' DEVICE;
18                         'LOCATION''INTEGER' SN);
19  'BEGIN'
20        'INTEGER' ADDR, ENTRY, LOCK, RESULT;
21
22        LOCK:=0; RESULT:=3; (ASSUME ENTRY NOT FOUND)
23        ALLOCATE(SBS, ADDR);
24        DISKOP(1, DEVICE, 2, ADDR);
25        'FOR' ENTRY:=ADDR+4,ENTRY+7 'WHILE' LOCK=0 'DO'
26            'BEGIN'
27            'IF' [ENTRY] = 'LITERAL'(*)
28               'THEN' LOCK := 1
29               'ELSE''BEGIN'
30                     'IF' [ENTRY+1]=NAME[1] 'AND' [ENTRY+2]=NAME[2] 'AND'
31                          [ENTRY+3]=NAME[3] 'AND' [ENTRY+4]=NAME[4]
32                        'THEN''BEGIN'
33                              'IF' BOT BYTE([ENTRY]) = 1
34                                 'THEN''BEGIN'
35                                       RESULT:=9;
36                                       LOCK:=1
37                                       'END'
38                                 'ELSE''BEGIN'
39                                       SN := [ENTRY+6];
40                                       [ENTRY]:='LITERAL'(#);
41                                       [ENTRY+1]:=0;
42                                       DISKOP(2,DEVICE,2,ADDR);
43                                       RESULT:=0;
44                                       LOCK := 1
45                                       'END'
46                              'END'
47                     'END'
48            'END';
49        FREE(SBS, ADDR);
50        'IF' RESULT > 0
51           'THEN' ERROR(NAME, RESULT)
52  'END';
53  'END'
54  'FINISH'
```

Appendix J cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (CHARS);
 4  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 5  'EXTERNAL' ('PROCEDURE' NEW SUFFIX(IA,IA,VI)  );
 6  'DELETE' IA; 'DELETE' VI;
 7  'SEGMENT' NEW SUFFIX PRIMITIVE
 8  'BEGIN'
 9         'INTEGER' DOT:='LITERAL'(.);
10         'INTEGER''ARRAY' DUMMY[1:4];
11
12  'PROCEDURE' NEW SUFFIX('INTEGER''ARRAY' OLDN, NEWN; 'VALUE''INTEGER' SUF]
13  'BEGIN'
14         'INTEGER' I;
15
16         'FOR' I:=1,I+1 'WHILE' I<=4 'DO'
17              'BEGIN'
18              DUMMY[I]:=OLDN[I];
19              NEWN[I]:=BLANK
20              'END';
21         'FOR' I:=1,I+1 'WHILE' I<=4 'DO'
22              'BEGIN'
23              'IF' TOP BYTE(DUMMY[I])=DOT 'OR'
24                   TOP BYTE(DUMMY[I])=SP
25                  'THEN''BEGIN'
26                         'IF' SUFFIX > 0
27                            'THEN''BEGIN'
28                                  TOP BYTE(NEWN[I]):=DOT;
29                                  BOT BYTE(NEWN[I]):=SUFFIX;
30                                  'END';
31                         I:=10
32                         'END'
33                  'ELSE''BEGIN'
34                         TOP BYTE(NEWN[I]):=TOP BYTE(DUMMY[I]);
35                         'IF' BOT BYTE(DUMMY[I])=DOT 'OR'
36                               BOT BYTE(DUMMY[I])=SP
37                            'THEN''BEGIN'
38                                   'IF' SUFFIX > 0
39                                      'THEN''BEGIN'
40                                             BOT BYTE(NEWN[I]):=DOT;
41                                             TOP BYTE(NEWN[I+1]):=SUFFIX;
42                                             'END';
43                                   I:=10
44                                   'END'
45                            'ELSE''BEGIN'
46                                   BOT BYTE(NEWN[I]):=BOT BYTE(DUMMY[I])
47                                   'END'
48                         'END'
49                 'END'
50  'END';
51  'END'
52  'FINISH'
53
```

```
1   'CORAL'
2   'PROGRAM' KERNEL
3   'LIBRARY' (CHARS);
4   'DEFINE' IA "'INTEGER''ARRAY'";
5   'EXTERNAL' ('INTEGER''PROCEDURE' WHAT SUFFIX(IA)  );
6   'DELETE' IA;
7   'SEGMENT' WHAT SUFFIX PRIMITIVE
8   'BEGIN'
9
10  'INTEGER''PROCEDURE' WHAT SUFFIX('INTEGER''ARRAY' NAME);
11  'BEGIN'
12        'INTEGER' I;
13
14        'FOR' I:=1,I+1 'WHILE' I<=4 'DO'
15            'BEGIN'
16            'IF' TOP BYTE(NAME[I])=SP
17               'THEN''ANSWER' SP
18               'ELSE''IF' TOP BYTE(NAME[I])='LITERAL'(.)
19                        'THEN''ANSWER' BOT BYTE(NAME[I])
20               'ELSE''IF' BOT BYTE(NAME[I])=SP
21                        'THEN''ANSWER' SP
22               'ELSE''IF' BOT BYTE(NAME[I])='LITERAL'(.)
23                        'THEN''ANSWER' TOP BYTE(NAME[I+1])
24            'END';
25        'ANSWER' SP
26  'END';
27  'END'
28  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 8  'DEFINE' LI "'LOCATION''INTEGER'";
 9  'EXTERNAL' ('PROCEDURE' CLOSE(VI);
10              'PROCEDURE' OPEN(VI,IA,VI,VI,LI);
11     'INTEGER''PROCEDURE' SEEK IO ENTRY;
12     'INTEGER''PROCEDURE' LOOKUP(VI);
13              'PROCEDURE' FETCH ENTRY(IA,VI,LI,LI,LI);
14              'PROCEDURE' MAKE ENTRY(IA,VI,VI,VI,LI)  );
15  'DELETE' VI; 'DELETE' IA; 'DELETE' LI;
16  'SEGMENT' OPEN CLOSE PRIMITIVES
17  'BEGIN'
18        'INTEGER' FILEND := 'OCTAL'(147777);
19
20  'PROCEDURE' OPEN('VALUE''INTEGER' CN; 'INTEGER''ARRAY' FNAME;
21                   'VALUE''INTEGER' DEVICE;
22                   'VALUE''INTEGER' OPER; 'LOCATION''INTEGER' RESULT);
23  'BEGIN'
24        'INTEGER' ADDR, I, SN, TYPE, PROT;
25
26        I:=LOOKUP(CN);
27        'IF' I >= 0 'THEN' EXIT(22);  (CN ALREADY OPEN)
28        I:=SEEK IO ENTRY;
29        'IF' I < 0 'THEN' EXIT(24);  (IO TABLE FULL)
30        RESULT:=0;
31        FETCH ENTRY(FNAME, DEVICE, SN, TYPE, PROT);
32        'IF' SN < 0
33           'THEN''BEGIN'
34                 'IF' OPER=1
35                    'THEN''BEGIN'
36                          TEL1:=FNAME[1];
37                          TEL2:=FNAME[2];
38                          TEL3:=FNAME[3];
39                          TEL4:=FNAME[4];
40                          RESULT:=3; (FILE NOT FOUND)
41                          'END'
42                    'ELSE''BEGIN'
43                          MAKE ENTRY(FNAME, DEVICE, 0, 0, SN)
44                         'END'
45                 'END'
46           'ELSE''BEGIN'
47                 'IF' OPER = 2
48                    'THEN' RESULT:=4; (NAME NOT UNIQUE)
49                 'END';
50        ALLOCATE(SBS, ADDR);
51        CHANNEL[I]:=CN;
52        SECTOR[I]:=SN;
53        FUNCTION[I]:=OPER;
54        WORD[I]:=4; BYTE[I]:=0;
55        BUFFER[I]:=ADDR;
56        'IF' OPER = 1 'AND' RESULT = 0
```

```
57              'THEN''BEGIN'
58                     DISKOP(1, DEVICE,  SN, ADDR);
59                     'END'
60   'END';
61
62   'PROCEDURE' CLOSE('VALUE''INTEGER' CN);
63   'BEGIN'
64         'INTEGER' ACC, I;
65
66         I:=LOOK UP(CN);
67         'IF' I < 0 'THEN' EXIT(23);  (CN NOT OPEN)
68         'IF' FUNCTION[I] = 2
69            'THEN''BEGIN'
70                  ACC := BUFFER[I] + WORD[I];
71                  'IF' BYTE[I] = 1
72                     'THEN''BEGIN'
73                           BOT BYTE(WORD[I]) := 0;
74                           ACC := ACC + 1
75                           'END';
76                  [ACC] := FILEND;
77                  [BUFFER[I]] := 0;
78                  [BUFFER[I]+1] := 0;
79                  DISKOP(2,UDISK,SECTOR[I],BUFFER[I])
80                  'END';
81         CHANNEL[I]:=-1;
82         FREE(SBS, BUFFER[I]);
83         BUFFER[I]:=0
84   'END';
85
86   'END'
87   'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 7  'DEFINE' LI "'LOCATION''INTEGER'";
 8  'EXTERNAL' ('PROCEDURE' READ RECORD(VI,IA,VI,LI);
 9     'INTEGER''PROCEDURE' READ CHAR(VI)  );
10  'DELETE' VI; 'DELETE' IA; 'DELETE' LI;
11  'SEGMENT' READ RECORD PRIMITIVE
12  'BEGIN'
13        'INTEGER' FILEND:='OCTAL'(147777);
14
15  'PROCEDURE' READ RECORD('VALUE''INTEGER' CN; 'INTEGER''ARRAY' B;
16                          'VALUE''INTEGER' LIMIT;
17                          'LOCATION''INTEGER' RESULT);
18  'BEGIN'
19        'INTEGER' I, DATA;
20
21        'IF' CN = 0
22           'THEN''BEGIN'
23                 RESULT:=0;
24                 GET RECORD(B,LIMIT)
25                 'END'
26           'ELSE''BEGIN'
27                 'FOR' I:=1,I+1 'WHILE' I<=LIMIT 'DO'
28                     'BEGIN'
29                     DATA:=READ CHAR(CN);
30                     'IF' DATA = FILEND
31                        'THEN''BEGIN'
32                              I:=LIMIT;
33                              RESULT := 1
34                              'END'
35                        'ELSE''BEGIN'
36                              RESULT := 0;
37                              'IF' DATA=LF
38                                 'THEN' DATA:=CR;
39                              B[I] := DATA;
40                              'IF' DATA = CR
41                                 'THEN' I:=LIMIT
42                              'END'
43                     'END'
44                 'END'
45  'END';
46  'END'
47  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 7  'EXTERNAL' ('PROCEDURE' WRITE RECORD(VI,IA,VI);
 8     'INTEGER''PROCEDURE' WRITE CHAR(VI,VI) );
 9  'DELETE' VI; 'DELETE' IA;
10  'SEGMENT' WRITE RECORD PRIMITIVE
11  'BEGIN'
12
13  'PROCEDURE' WRITE RECORD('VALUE''INTEGER' CN; 'INTEGER''ARRAY' B;
14                           'VALUE''INTEGER' LIMIT);
15  'BEGIN'
16        'INTEGER' I;
17
18        'IF' CN < 2
19           'THEN''BEGIN'
20                 PUT RECORD(B,LIMIT)
21                 'END'
22           'ELSE''BEGIN'
23                 'FOR' I:=1,I+1 'WHILE' I<=LIMIT 'DO'
24                     'BEGIN'
25                     WRITE CHAR(CN, B[I]);
26                     'IF' B[I] = CR
27                        'THEN' I:=LIMIT
28                     'END'
29                  'END'
30  'END';
31  'END'
32  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (PARAMS);
 6  'LIBRARY' (CHARS);
 7  'DEFINE' VI "'VALUE''INTEGER'";
 8  'EXTERNAL'('INTEGER''PROCEDURE' READ CHAR(VI);
 9              'INTEGER''PROCEDURE' GET CHAR;
10              'INTEGER''PROCEDURE' LOOKUP(VI)   );
11  'DELETE' VI;
12  'SEGMENT' READ CHAR PRIMITIVE
13  'BEGIN'
14        'INTEGER' FILEND:='OCTAL'(147777);
15
16  'INTEGER''PROCEDURE' READ CHAR('VALUE''INTEGER' CN);
17  'BEGIN'
18        'INTEGER' ACC, DATA, I, SN;
19
20        'IF' CN = 0
21           'THEN' DATA:=GET CHAR
22           'ELSE''BEGIN'
23                 I:=LOOKUP(CN); DATA:=0;
24                 'IF' I < 0
25                    'THEN' EXIT(23);   'COMMENT' CN NOT OPEN;
26                 ACC:=BUFFER[I] + WORD[I];
27                 'IF' [ACC]=FILEND 'THEN''ANSWER' FILEND;
28                 'IF' BYTE[I]=0
29                    'THEN''BEGIN'
30                          DATA:=TOP BYTE([ACC]);
31                          BYTE[I]:=1
32                          'END'
33                    'ELSE''BEGIN'
34                          DATA:=BOT BYTE([ACC]);
35                          BYTE[I]:=0;
36                          WORD[I]:=WORD[I]+1;
37                          'IF' WORD[I] > SBSM1
38                             'THEN''BEGIN'
39                                   SN:=[BUFFER[I]+1];
40                                   DISKOP(1,UDISK,SN,BUFFER[I]);
41                                   SECTOR[I]:=SN;
42                                   WORD[I]:=4; BYTE[I]:=0
43                                   'END'
44                          'END'
45                 'END';
46        'ANSWER' DATA
47  'END';
48  'END'
49  'FINISH'
```

Appendix J cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (PARAMS);
 6  'LIBRARY' (CHARS);
 7  'DEFINE' VI "'VALUE''INTEGER'";
 8  'EXTERNAL' ('PROCEDURE' WRITE CHAR(VI,VI);
 9               'PROCEDURE' PUT CHAR(VI);
10     'INTEGER''PROCEDURE' LOOKUP(VI)  );
11  'DELETE' VI;
12  'SEGMENT' WRITE CHAR PRIMITIVE
13  'BEGIN'
14          'INTEGER' I, XSN;
15          'INTEGER' FILEND:='OCTAL'(147777);
16
17  'PROCEDURE' WRITE CHAR('VALUE''INTEGER' CN, DATA);
18  'BEGIN'
19          'INTEGER' ACC;
20
21          'IF' CN < 2
22              'THEN' PUT CHAR(DATA)
23              'ELSE''BEGIN'
24                      I:=LOOKUP(CN);
25                      'IF' I < 0
26                          'THEN' EXIT(23);    'COMMENT' CN NOT OPEN;
27                      ACC:=BUFFER[I] + WORD[I];
28                      'IF' DATA = FILEND
29                          'THEN' [ACC] := FILEND
30                          'ELSE''BEGIN'
31                                  'IF' BYTE[I]=0
32                                      'THEN''BEGIN'
33                                              TOP BYTE([ACC]):=DATA;
34                                              BYTE[I]:=1
35                                              'END'
36                                      'ELSE''BEGIN'
37                                              BOT BYTE([ACC]):=DATA;
38                                              BYTE[I]:=0;
39                                              WORD[I]:=WORD[I]+1;
40                                              'IF' WORD[I] > SBSM1
41                                                  'THEN''BEGIN'
42                                                          GIVE SECTOR(UDISK,XSN);
43                                                          [BUFFER[I]+1]:=XSN;
44                                                          DISKOP(2,UDISK,
45                                                                 SECTOR[I],BUFFER[I]);
46                                                          SECTOR[I]:=XSN;
47                                                          WORD[I]:=4; BYTE[I]:=0
48                                                          'END'
49                                              'END'
50                                  'END'
51                      'END'
52  'END';
53  'END'
54  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (CHARS);
 6  'LIBRARY' (PARAMS);
 7  'EXTERNAL' ('INTEGER''PROCEDURE' SEEK IO ENTRY  );
 8  'SEGMENT' SEEK IO ENTRY PRIMITIVE
 9  'BEGIN'
10
11  'INTEGER''PROCEDURE' SEEK IO ENTRY;
12  'BEGIN'
13        'INTEGER' I;
14
15        'FOR' I:=0,I+1 'WHILE' I<=4 'DO'
16            'IF' CHANNEL[I]=-1 'THEN''ANSWER' I;
17         'ANSWER' -1
18  'END';
19
20  'END'
21  'FINISH'
```

Appendix J cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'EXTERNAL'( 'INTEGER''PROCEDURE' LOOKUP('VALUE''INTEGER')  );
 5  'SEGMENT' LOOKUP PRIMITIVE
 6  'BEGIN'
 7
 8  'INTEGER''PROCEDURE' LOOK UP('VALUE''INTEGER' CN);
 9  'BEGIN'
10        'INTEGER' I;
11        'FOR' I:= 0,I+1 'WHILE' I<=4 'DO'
12             'IF' CHANNEL[I]=CN 'THEN''ANSWER' I;
13        'ANSWER' -1
14  'END';
15
16  'END'
17  'FINISH'
```

Appendix J cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
 6  'EXTERNAL'('PROCEDURE' ERROR(IA,VI)  );
 7  'DELETE' VI; 'DELETE' IA;
 8  'SEGMENT' ERROR PRIMITIVE
 9  'BEGIN'
10
11  'PROCEDURE' ERROR('INTEGER''ARRAY' PARAM; 'VALUE''INTEGER' CODE);
12  'BEGIN'
13        TEL1:=PARAM[1];
14        TEL2:=PARAM[2];
15        TEL3:=PARAM[3];
16        TEL4:=PARAM[4];
17        EXIT(CODE)
18  'END';
19
20  'END'
21  'FINISH'
```

```
1   'CORAL'
2   'PROGRAM' KERNEL
3   'LIBRARY' (DBASE);
4   'LIBRARY' (CHARS);
5   'LIBRARY' (LINKS);
6   'LIBRARY' (PARAMS);
7   'DEFINE' VI "'VALUE''INTEGER'"; 'DEFINE' IA "'INTEGER''ARRAY'";
8   'DEFINE' LI "'LOCATION''INTEGER'";
9   'EXTERNAL'('PROCEDURE' EXECUTE(IA,VI,LI);
10              'PROCEDURE' FETCH ENTRY(IA,VI,LI,LI,LI)  );
11  'DELETE' VI; 'DELETE' IA; 'DELETE' LI;
12  'SEGMENT' EXECUTE PRIMITIVE
13  'BEGIN'
14
15  'PROCEDURE' EXECUTE('INTEGER''ARRAY' NAME; 'VALUE''INTEGER' DEVICE;
16                      'LOCATION''INTEGER' RESULT);
17  'BEGIN'
18         'INTEGER' SN, TYPE, PROT;
19
20         FETCH ENTRY(NAME,DEVICE,SN,TYPE,PROT);
21         'IF' SN < 0
22            'THEN' RESULT:=3
23            'ELSE''IF' TYPE <> 2
24                   'THEN' RESULT:=10
25            'ELSE''BEGIN'
26                  LOAD(DEVICE, SN)
27                  'END'
28  'END';
29  'END'
30  'FINISH'
```

Appendix J cont.

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'EXTERNAL' ('PROCEDURE' GET PARAM('INTEGER''ARRAY')  );
 5  'SEGMENT' GET PARAM PRIMITIVE
 6  'BEGIN'
 7
 8  'PROCEDURE' GET PARAM('INTEGER''ARRAY' NAME);
 9  'BEGIN'
10        NAME[1] := [PPTR];
11        NAME[2] := [PPTR+1];
12        NAME[3] := [PPTR+2];
13        NAME[4] := [PPTR+3];
14        NAME[5] := [PPTR+4];
15        PPTR:=PPTR+5
16  'END';
17
18  'END'
19  'FINISH'
```

```
 1  'CORAL'
 2  'PROGRAM' KERNEL
 3  'LIBRARY' (DBASE);
 4  'LIBRARY' (LINKS);
 5  'LIBRARY' (PARAMS);
 6  'DEFINE' VI "'VALUE''INTEGER'";
 7  'EXTERNAL' ('INTEGER''PROCEDURE' LENGTH(VI,VI)  );
 8  'DELETE' VI;
 9  'SEGMENT' LENGTH PRIMITIVE
10  'BEGIN'
11
12  'INTEGER''PROCEDURE' LENGTH('VALUE''INTEGER' DEVICE,SN);
13  'BEGIN'
14         'INTEGER' ADDR, COUNT, KEY, LOCK;
15
16         ALLOCATE(SBS,ADDR);
17         COUNT:=0; LOCK:=0;
18         'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
19              'BEGIN'
20              DISKOP(1,DEVICE,SN,ADDR);
21              COUNT:=COUNT+1;
22              'IF' [ADDR+1]=0
23                 'THEN' LOCK:=1
24                 'ELSE' SN:=[ADDR+1]
25              'END';
26         FREE(SBS,ADDR);
27         'ANSWER' COUNT
28  'END';
29  'END'
30  'FINISH'
```

PORTOS Hardware Interface Source Listings

```
 1  * PROGRAM : HARDWARE INTERFACE
 2  *
 3  * SEGMENT : DISKOP PRIMITIVE
 4  *
 5  * HOST COMPUTER : PRIME 300
 6
 7  *          **  DISK DRIVER PRIMITIVE  **
 8
 9          ORG  '32234
10          RLIT
11          ELM
12  * OUTWARD EXTERNAL AND COMMON
13  * DATA AREA
14          JMP   B1
15  HEAD   DATA '30000
16  TAIL   DATA '30001
17  ACTIV  DATA '30002
18  TNUM   DATA '30003
19  PAR1   DATA '30050
20  PAR2   DATA '30051
21  PAR3   DATA '30052
22  PAR4   DATA '30053
23  PAR5   DATA '30054
24  PAR6   DATA '30055
25  PAR7   DATA '30056
26  PAR8   DATA '30057
27  TEMP   DATA 0
28  B1      EQU   *
29  * INWARD EXTERNAL AND COMMON
30  * SEGMENT ENTRY POINT
31          SGL
32  *
33              LDA  PAR1,*          FETCH  'OPER'
34              STA  OPER
35              LDA  PAR2,*          FETCH  'SECTOR'
36              STA  TEMP
37              LDA  PAR3,*          FETCH  'ADDR'
38              STA  ADDR
39  *                                ADJUST SECTOR -> TN, SN
40              CRL
41              LDA  TEMP
42              IAB
43              DIV  =8
44              STA  TRACK
45              STA  CHPRO + '3
46              IAB
47              STA  SECTOR
48              STA  CHPRO + '7
49  *
50  START       CRA
```

```
 51            STA  STATUS
 52            LDA  =448           NO.OF WORDS
 53            TCA
 54            ALL  4
 55            STA  '20            USE CHANNEL '20
 56            LDA  ADDR           DMA STARTING ADDRESS
 57            STA  '21
 58            LDA  OPER
 59            CAS  =1
 60            JMP  WRITE
 61            JMP  READ
 62            JST  MESS
 63   READ     LDA  ='50003        READ 8 SEC/TRACK
 64            STA  CHPRO + '6
 65            JMP  CON
 66   WRITE    LDA  ='60003        WRITE 8 SEC/TRACK
 67            STA  CHPRO + '6
 68   CON      LDA  CHPRO + '7
 69            ALL  10
 70            STA  CHPRO + 7
 71            LDA  CHPRO + '3
 72            CAS  =203
 73            JMP  BOT
 74            JMP  BOT
 75            JMP  TOP
 76   BOT      SUB  =203           BOTTOM SURFACE
 77            STA  CHPRO + '3
 78            STA  TRACK
 79            IRS  CHPRO + '7     RW HEAD := 1
 80   TOP      LDA  =STATUS        ADDR OF
 81            STA  CHPRO + 9
 82            OCP  '1721          RESET (DISC := IDLE)
 83            LDA  =CHPRO         CHANNEL PROG. ADDR
 84            OTA  '1721          START (DISC := BUSY)
 85            HLT                 HALT IF DEVICE INACTIVE
 86   *
 87            CRA
 88            STA  LOOPC          COUNT LOOPS
 89   WAIT     IRS  LOOPC
 90            JMP  CHECK          SHOULD HAVE FINISHED
 91            JST  MESS           IS CHANNEL PROG.RUNNING
 92   CHECK    INA  '1721          YES, WAIT
 93            JMP  WAIT           OK, CHECK STATUS
 94            LDA  STATUS
 95            CAS  ='100000       NOT OK
 96            JMP  *+2            OK
 97            JMP  EXIT           IS DISK PROTECTED
 98            CAS  ='120000
 99            JMP  *+2            YES
100            JMP  PROT           ERROR MESSAGE
101            JST  MESS
102   EXIT     CRA
103            STA  TNUM,*         NORMAL EXIT
104            SVC
105   PROT     LDA  PCODE
106            STA  PAR8,*
```

```
107              LDA  TASKN
108              STA  TNUM,*
109              SVC                  ERROR EXIT
110   *
111   LOOPC       DATA  0
112   *
113   *CHANNEL PROGRAM
114   *
115   OPER       DATA  0
116   TRACK      DATA  0
117   SECTOR     DATA  0
118   ADDR       DATA  0
119   STATUS     DATA  0
120   *
121   CHPRO      OCT  40000           SELECT
122              OCT  1               UNIT 1
123              OCT  30000           SEEK
124              OCT  0               REQD. TRACK
125              OCT  150000          USE CHANNEL
126              OCT  20              '20
127              OCT  0               READ-WRITE (8 SECT/TRACK)
128              OCT  10000           SECTOR / HEAD
129              OCT  110000          INPUT STATUS WORD
130              OCT  0               TO LOC. 'STATUS'
131              OCT  0               HALT CHPRO
132              OCT  0               DISC := IDLE
133   *
134   *
135   DKERM      DATA  C'FAIL= '
136   EOM        OCT   223
137   *
138              SUBR MESS
139   MESS       DAC  **
140              CRA
141              STA  '0        X:=0
142   MLOOP      LDA  DKERM,1
143              CAS  EOM
144              JMP  *+2
145              JMP  OK
146              ICA
147              OTA  '4
148              JMP  *-1
149              ICA
150              OTA  '4
151              JMP  *-1
152              IRX
153              JMP  MLOOP
154   OK         LDA  OPER
155              JST  OCT
156              LDA  TRACK
157              JST  OCT
158              LDA  SECTOR
159              JST  OCT
160              LDA  ADDR
161              JST  OCT
162              LDA  STATUS
```

```
163                JST  OCT
164                LDA  ='215
165                OTA  4
166                JMP  *-1
167                LDA  ='212
168                OTA  4
169                JMP  *-1
170                LDA  CODE
171                STA  PAR8,*
172                LDA  TASKN
173                STA  TNUM,*
174                SVC                    ERROR EXIT
175   *
176   PCODE        DATA  12
177   CODE         DATA 13
178   TASKN        DATA  8
179   *
180                SUBR OCT
181   OCT          DAC **
182                CRB
183                IAB           PUT IT IN B REG
184                CRA
185                LLL 1         SHIFT 1ST DIGIT
186                ADD BIAS
187                OTA '04       PRINT IT
188                JMP *-1
189                LDX FIVE      5 MORE DIGITS TO PRINT
190   AGAIN        CRA           GO INTO LOOP
191                LLL 3         GET NEXT 3 BITS
192                ADD BIAS
193                OTA '04       PRINT THE EQUIV.DIGIT
194                JMP *-1
195                DRX           REPEAT LOOP
196                JMP AGAIN
197                LDA  ='240
198                OTA  4
199                JMP  *-1
200                OTA  4
201                JMP  *-1
202                JMP* OCT
203   *
204   FIVE         DATA  5
205   BIAS         OCT  260
206   *
207                END
```

```
 1  *
 2  * PROGRAM : HARDWARE INTERFACE
 3  *
 4  * SEGMENT : INTERRUPT HANDLER (ILLEGAL INSTRUCTION)
 5  *
 6  * HOST HARDWARE : PRIME 300
 7  *
 8         ORG  '33516
 9         SUBR ILLEG
10  ILLEG  DAC **
11         LDA ILLEG
12         STA ADDR      GET ADDRESS
13         CRA
14         STA  '0          X:=0
15  MLOOP  LDA  MESS,1
16         CAS  EOM
17         JMP  *+2
18         JMP  OK
19         ICA
20         OTA  '4
21         JMP  *-1
22         ICA
23         OTA  '4
24         JMP  *-1
25         IRX
26         JMP  MLOOP
27  OK     NOP
28         CRL
29         LDA ADDR
30         IAB
31         LLL 1
32         ADD ='260
33         OTA 4          PRINT ADDRESS
34         JMP *-1
35         LDX =5
36  XD     CRA
37         LLL 3
38         ADD ='260
39         OTA 4
40         JMP *-1
41         DRX
42         JMP XD
43         LDA ='215
44         OTA 4
45         JMP *-1
46         LDA ='212
47         OTA 4
48         JMP *-1
49         OTA 4
50         JMP *-1
51         CRA
52         STA PAR8,*
53         LDA =6
54         STA TNUM,*     EXIT VIA TASK 6
55         SVC
56  *
```

```
57  TNUM  DATA '30003
58  PAR8  DATA '30057
59  ADDR  DATA 0
60  MESS  DATA  C'ILLEGAL INSTR.AT '
61  EOM   OCT   223
62  *
63        END
```

```
 1   *
 2   * PROGRAM  HARDWARE INTERFACE
 3   *
 4   * SEGMENT : INTERRUPT HANDLER (REAL-TIME CLOCK)
 5   *
 6   * HOST HARDWARE : PRIME 300
 7   *
 8          ORG '33616
 9          SUBR  TIMER
10   TIMER DAC   **
11          OCP  '220          STOP CLOCK
12          STA  TEMP          SAVE A REG.
13          LDA  =-50
14          STA  '61           RESET COUNTER
15          ENB
16          OCP  '20           START CLOCK
17          LDA  SECS,*
18          A1A
19          CAS  =60
20          NOP
21          JMP  NEWM
22          STA  SECS,*
23          LDA  TEMP          RESTORE AREG
24          JMP  TIMER,*       RETURN
25   NEWM   CRA
26          STA  SECS,*
27          LDA  MINS,*
28          A1A
29          CAS  =60
30          NOP
31          JMP  NEWH
32          STA  MINS,*
33          LDA  TEMP
34          JMP  TIMER,*       RETURN
35   NEWH   CRA
36          STA  MINS,*
37          LDA  HOURS,*
38          A1A
39          CAS  =24
40          NOP
41          JMP  NEWD
42          STA  HOURS,*
43          LDA  TEMP
44          JMP  TIMER,*       RETURN
45   NEWD   CRA
46          STA  HOURS,*
47          STA  MINS,*
48          STA  SECS,*
49          LDA  TEMP          RESTORE A REG.
50          JMP  TIMER,*       EXIT
51   *
52   TEMP   DATA  0
53   HOURS OCT    30110
54   MINS  OCT    30111
55   SECS  OCT    30112
56   *
```

```
 1  *
 2  * PROGRAM : HARDWARE INTERFACE
 3  *
 4  * SEGMENT : VDU DRIVER (OUTPUT)
 5  *
 6  * HOST COMPUTER : PRIME 300
 7  *
 8         REL
 9         SUBR  PUTCHAR
10  PUTCHAR DAC   **
11         OTA   '04          OUTPUT CHAR
12         JMP   *-1          WAIT UNTIL OK
13         JMP   PUTCHAR,*    RETURN
14         END
```

```
1   *
2   * PROGRAM : HARDWARE INTERFACE
3   *
4   * SEGMENT : VDU DRIVER (INPUT)
5   *
6   * HOST COMPUTER : PRIME 300
7   *
8            REL
9            SUBR GETCHAR
10  GETCHAR DAC **
11           INA  '1004          READ A CHAR
12           JMP  *-1            WAIT UNTIL OK
13           CAS  ='212          IS IT LF
14           JMP  *+2            NO, CARRY ON
15           LDA  ='215          YES, REPLACE WITH CR
16           JMP  GETCHAR,*      RETURN
17           END
```

```
 1  *
 2  * PROGRAM : HARDWARE INTERFACE
 3  *
 4  * SEGMENT : PRINTER DRIVER
 5  *
 6  * HOST COMPUTER : PRIME 300
 7  *
 8          REL
 9          SUBR  GETPRINTER
10  GETPRINTER DAC  **
11          SKS  '4
12          JMP  *-1
13          OCP  '1704
14          OCP  '1204
15          CRA
16          OTA  '404
17          LDA  CWORD
18          OTA  '104
19          JMP  *-1
20          JMP  GETPRINTER,*
21  *
22  CWORD  OCT  3514
23  *
24          SUBR  RELEASEPRINTER
25  RELEASEPRINTER DAC **
26          SKS  '4
27          JMP  *-1
28          CRA
29          OTA  '104
30          JMP  *-1
31          OCP  '1704
32          OCP  '1204
33          JMP  RELEASEPRINTER,*
34  *
35          NOP
36          JMP  *-1
37          END
```

```
 1  *
 2  * PROGRAM : HARDWARE INTERFACE
 3  *
 4  * SEGMENT : BOOTSTRAP LOADER
 5  *
 6  * HOST COMPUTER : PRIME 300
 7  *
 8  *..........BOOTSTRAP LOADER
 9  *              READS TASKS FROM SECTOR 4
10  *              AND PLACES THEM IN '30000
11  *              INITIATES TASK 6
12  *
13  *   RESET HARDWARE MACHINE
14  *
15         REL
16         RMC
17         E32R
18         INH                 INHIBIT INTERRUPTS
19         CRA
20         SMK  '20            DISABLE ALL EXT.DEVICES
21         OCP '1204           TTY NORMAL MODE
22  *
23  *  CLEAR DBASE IN MEMORY
24  XL     CRA
25         STA  SLOC,*
26         IRS  SLOC
27         LDA  SLOC
28         CAS  ELOC
29         JMP  XB
30         JMP  XL
31         JMP  XL
32  *
33  SLOC   OCT  30000
34  ELOC   OCT  30113
35  *
36  *  READ TASKS & PLACE IN MEMORY
37  *
38  XB     LDA FOUR
39         STA SECTOR
40         LDA TASKA
41         STA ILA            ILA := TASK AREA
42         JST INSTALL
43         LDA ILA
44         STA '65            PC FOR SVCALL
45         LDA SIZE
46         ADD ILA
47         A1A
48         STA ILA            ILA:=ILA+SIZE
49         IRS SECTOR          SECTOR:=SECTOR+1
50         CRA
51         STA TTX            TTX:=0
52  NTASK  JST INSTALL
53         LDX TTX
54         LDA ENTRY
55         STA* VECTOR,1      VECTOR[TTX]:=ENTRY
56         CRA
```

```
 57          STA* RETURN,1    RETURN[TTX]:=0
 58          STA* FATHER,1    FATHER[TTX]:=0
 59          LDA ILA
 60          ADD SIZE
 61          A1A
 62          STA ILA           ILA:=ILA+SIZE
 63          IRS SECTOR       SECTOR:=SECTOR+1
 64          LDA TTX
 65          A1A
 66          STA TTX
 67          CAS TMAX         IF TTX > 11
 68          JMP INT          THEN LOAD HANDY
 69          NOP
 70          JMP NTASK        ELSE NEXT TASK
 71   INT    JST INSTALL      ELSE HANDY
 72          LDA ILA
 73          STA '72          PC FOR HANDY
 74          LDA  SIZE
 75          ADD  ILA
 76          A1A
 77          STA  ILA         ILA:=ILA+SIZE
 78          IRS  SECTOR       SECTOR:=SECTOR+1
 79   OK     JST  INSTALL      LOAD TIMER
 80          LDA  ILA
 81          STA  '63          PC FOR TIMER
 82          OCP  '1720        RESET CLOCK
 83          OCP  '1520        SET CLOCK INTERRUPT BIT
 84          LDA  INTERV
 85          STA  '61          SET INTERVAL TO 1 SEC
 86          LDA SUPM
 87          STA PAR8,*       PAR8:=111
 88          LDA SIX
 89          STA TNUM,*       TNUM:=6
 90          LDA  QMQM
 91          STA  MNAME1,*
 92          LDA  QMSP
 93          STA  MNAME2,*
 94          ESIM            ENTER STANDARD MODE
 95          ENB             ENABLE INTERRUPTS
 96          OCP  '20        START CLOCK
 97          SVC             STARTUP
 98    *
 99    *
100    *  PROGRAM PARAMETERS & VARIABLES
101    *
102    HEAD    DATA '30000  )
103    TAIL    DATA '30001  )
104    ACTIVE  DATA '30002  )
105    TNUM    DATA '30003  )
106    VECTOR  DATA '30004  )
107    RETURN  DATA '30020   >  DATA BASE (TASK TABLE)
108    FATHER  DATA '30034  )
109    PAR1    DATA '30050  )
110    PAR2    DATA '30051  )
111    PAR3    DATA '30052  )
112    PAR4    DATA '30053  )
```

```
113  PAR5    DATA '30054  )
114  PAR6    DATA '30055  )
115  PAR7    DATA '30056  )
116  PAR8    DATA '30057  )
117  MNAME1 DATA '30105  )
118  MNAME2 DATA '30106  )
119  TASKA   DATA '30113  )
120  TTX     DATA 0        TASK TABLE INDEX
121  EA      DATA '7005    LOC.GIVING EA OF BIN.MODULE
122  SA      DATA '7004      "     "  SA    "     "
123  FROM    DATA '7015    ADDR.FROM WHICH TO MOVE
124  ENTRY DATA 0
125  EOF    DATA '203
126  ILA    DATA 0            INITIAL LOAD ADDR
127  WA     DATA 0            WORKING ADDR
128  SIX    DATA  6
129  SEVEN  DATA 7
130  SUPM    DATA 111      STARTUP MESSAGE
131  FIVE    DATA  5
132  FOUR    DATA 4
133  TMAX    DATA 11          NO.OF TASKS-1
134  TCOUNT DATA 0
135  SIZE    DATA 0
136  XSN    DATA '7001
137  PC     DATA '7006
138  BINP1 DATA '7015
139  BINPX DATA '7004
140  ENDR   DATA '7000+447     END-OF-SECTOR
141  BIAS   DATA  '260
142  ID     OCT   300
143  INTERV DATA -50
144  QMQM    OCT    137677
145  QMSP    OCT    137400
146  *
147  *  PROCEDURE INSTALL
148  *
149          SUBR INSTALL
150  INSTALL DAC **
151          JST DISKOP
152          LDA EA,*
153          SUB SA,*
154          STA SIZE        SIZE:=EA-SA
155          LDA ILA
156          STA ENTRY       ENTRY:=PC (ILA)
157          STA WA          WA:=ILA
158          LDA BINP1
159          STA FROM        FROM=START OF BIN.PROG.
160  XFER    LDA FROM,*
161          CAS EOF
162          JMP *+2          IF EOF, THEN EXIT
163          JMP INSTALL,*   [WA] := [FROM]
164          STA WA,*        WA:=WA+1
165          IRS WA
166          IRS FROM
167          LDA FROM
168          CAS ENDR        IF FROM > '7000+447
```

```
169          JMP NEXT       THEN READ NEXT SECTOR
170          NOP
171          JMP XFER       ELSE CARRY ON
172   NEXT   LDA XSN,*
173          STA SN         SN := XTN
174          JST DISKOP
175          LDA BINPX
176          STA FROM       FROM := '7004
177          JMP XFER
178   *
179   *  DISC IO INFORMATION
180   TN     DATA 0       TRACK
181   SN     DATA 0      SECTOR
182   ADDR   DATA '7000
183   SECTOR DATA  0
184   RWHD   DATA 0     READ-WRITE HEAD
185   XX     DATA 0
186   ZZ     DATA 0
187   CW     DATA 0       CONTROL WORD
188   CHAN   DATA '20      CHANNEL NUMBER
189   *
190   *  PROCEDURE DISKOP
191   *
192          SUBR DISKOP
193   DISKOP DAC **
194             CRL
195             LDA  SECTOR
196             IAB
197             DIV  =8
198             STA  TN
199             IAB
200             STA  SN
201   *
202             CRA
203             STA  STATUS
204   START     LDA  =448          NO.OF WORDS
205             TCA
206             ALL  4
207             STA  '20           USE CHANNEL '20
208             LDA  ADDR          DMA STARTING ADDRESS
209             STA  '21
210             LDA  TN
211             STA  CHPRO + 3
212             LDA  SN
213             ALL  10
214             STA  CHPRO + 7
215             LDA  =STATUS
216             STA  CHPRO + 9
217             OCP  '1721         RESET (DISC := IDLE)
218             LDA  =CHPRO        CHANNEL PROG. ADDR
219             OTA  '1721         START (DISC := BUSY)
220             HLT                HALT IF DEVICE INACTIVE
221   *
222             INA  '1721         IS CHANNEL PROG.RUNNING
223             JMP  *-1           YES, WAIT
224             LDA  STATUS        OK, CHECK STATUS
```

```
225              CAS  ='100000
226              JMP  *+2          NOT OK
227              JMP  DISKOP,*     OK, RETURN
228              JST  MESS         PRINT ERROR MESSAGE
229              JMP  START        TRY AGAIN
230  *
231              SUBR MESS
232  MESS        DAC  **
233              LDA  ='330
234              LDX  =5
235              OTA  4
236              JMP  *-1
237              DRX
238              JMP  *-3
239              LDA  ='215
240              OTA  4
241              JMP  *-1
242              LDA  ='212
243              OTA  4
244              JMP  *-1
245              JMP  MESS,*
246  *
247  *CHANNEL PROGRAM
248  *
249  STATUS      DATA  0
250  *
251  CHPRO       OCT  40000        SELECT
252              OCT  1            UNIT 1
253              OCT  30000        SEEK
254              OCT  0            REQD. TRACK
255              OCT  150000       USE CHANNEL
256              OCT  20           '20
257              OCT  50003        READ 8 SEC/TRACK
258              OCT  10000        SECTOR / HEAD
259              OCT  110000       INPUT STATUS WORD
260              OCT  0            TO LOC. 'STATUS'
261              OCT  0            HALT CHPRO
262              OCT  0            DISC := IDLE
263  *
264         END
```

PORTOS Disk Build Program (PRIME 300)

```
1   'CORAL'
2   'PROGRAM' BUILD
3   'COMMON' ('INTEGER' ICHAR;
4             'INTEGER''ARRAY' MODN [1:3];
5             'INTEGER''ARRAY' L [1:80];
6             'INTEGER' ICOUNT  );
7   'DEFINE' IA "'INTEGER''ARRAY'"; 'DEFINE' VI "'VALUE''INTEGER'";
8   'EXTERNAL' ('PROCEDURE' OPENM;
9               'PROCEDURE' NEXTM;
10              'PROCEDURE' FETCHM;
11              'PROCEDURE' DISCOP(VI,VI,VI);
12              'PROCEDURE' NEW SUFFIX(IA,IA,VI);
13     'INTEGER''PROCEDURE' GET CHAR;
14              'PROCEDURE' WRITEN(VI,VI,VI);
15              'PROCEDURE' SHUTM  );
16  'DELETE' IA; 'DELETE' VI;
17  'SEGMENT' PORTOS DISK BUILD ON PRIME 300
18  'BEGIN'
19        'INTEGER''ARRAY' TRACK[0:405];
20        'INTEGER''ARRAY' SECTOR[0:7] := 1, 2, 4, 8, 16, 32, 64, 128;
21        'INTEGER''ARRAY' D[0:447];(DISC BUFFER)
22        'INTEGER''ARRAY' CFIL[0:447];(CATALOG BUFFER)
23        'INTEGER''ARRAY' ID [1:3] := BLANK, BLANK, BLANK;
24        'INTEGER' SOURCEFILE := 'OCTAL'(150244);
25        'INTEGER' TASK COUNT := 15;
26        'INTEGER' ADDR, I, J, LOCK, TN, SN, XTN, XSN,
27                  CFA, MOD COUNT, KEY, WORDS;
28
29
30  'PROCEDURE' INITIALIZE SECTOR;
31  'BEGIN'
32        'INTEGER' K;
33        'FOR' K:=0 'STEP' 1 'UNTIL' 447 'DO'
34            D [K] := 0; (SECTOR HEADER)
35        D [3] := 4; (RW PTR)
36        J := 4; (1ST.WORD OF BINARY PROG)
37  'END';
38
39        'FOR' I:=0,I+1 'WHILE' I <= 405 'DO'
40            TRACK [I] := 'HEX'(FF);
41        OUTSTR(1,"DISK ID? "); WRITCH(1,BEL); KEY:=GETCHAR;
42        I:=1; J:=1; LOCK:=0;
43        'FOR' KEY:=0 'WHILE' LOCK=0 'DO'
44            'BEGIN'
45            KEY:=GETCHAR;
46            'IF' KEY=CR
47               'THEN' LOCK:=1
48               'ELSE''BEGIN'
49                    'IF' J=1
50                       'THEN''BEGIN'
```

```
51                          TOP BYTE(ID[I]):=KEY; J:=2
52                          'END'
53                    'ELSE''BEGIN'
54                          BOT BYTE(ID[I]):=KEY;
55                          I:=I+1; J:=1
56                          'END'
57                  'END'
58          'END'; OUTSTR(1,"!CLL!");
59     GIVE SECTOR(0); (XBOOT 0)
60     GIVE SECTOR(0); (STATUS 1)
61     GIVE SECTOR(0); (DIRECTORY 2)
62     GIVE SECTOR(0); (PORTOS 3)
63     CLEAR CFIL;
64     CFA := 'LOCATION'(CFIL[0]);
65     DISCOP(2, 2, CFA); (WRITE DIRECTORY)
66     DISCOP(2, 3, CFA); (WRITE PORTOS)
67     ADDR := 'LOCATION'(D[0]);
68     OPENM; ICHAR := 0;
69     'FOR' MOD COUNT := 0, MOD COUNT+1 'WHILE' ICHAR=0 'DO'
70         'BEGIN'
71         INITIALIZE SECTOR;
72         NEXTM; WORDS:=0;
73         'IF' ICHAR=0
74            'THEN''BEGIN'
75                  'IF' MOD COUNT=0
76                     'THEN''BEGIN'
77                           TN:=0; SN:=0; J:=0
78                           'END'
79                     'ELSE' GIVE SECTOR(0);
80                  'IF' MOD COUNT > TASK COUNT
81                     'THEN''BEGIN'
82                           DISCOP(1,3,CFA); (PORTOS)
83                           'IF' MODN[1] = SOURCEFILE
84                              'THEN' MAKE ENTRY(0,1)
85                              'ELSE' MAKE ENTRY(2,1);
86                           DISCOP(2,3,CFA)
87                           'END';
88                  KEY := 0;
89                  'FOR' I:=0 'WHILE' KEY=0 'DO'
90                       'BEGIN'
91                       FETCHM; LOCK:=0;
92                       'FOR' I:=1,I+1 'WHILE' LOCK=0 'DO'
93                            'BEGIN'
94                            'IF' L[I]='OCTAL'(170707)
95                               'THEN''BEGIN'
96                                      'COMMENT' FILEND;
97                                      'IF' MOD COUNT <= TASK COUNT
98                                         'THEN' D[J]:='OCTAL'(203)
99                                         'ELSE''BEGIN'
100                                              'IF' MODN[1]=SOURCEFILE
101                                                 'THEN' D[J]:='OCTAL'(14
102                                                 'ELSE' D[J]:='OCTAL'(17
103                                              'END';
104                                     LOCK:=1; KEY:=1
105                                     'END'
106                               'ELSE''IF' I > 80
```

```
107                                     'THEN' LOCK:=1
108                                     'ELSE''BEGIN'
109                                           D[J]:=L[I];
110                                           'IF' MODN[1]=SOURCEFILE 'A
111                                                L[I]='OCTAL'(106400)
112                                              'THEN'LOCK:=1;
113                                           J:=J+1; WORDS:=WORDS+1;
114                                           'IF' J > 447
115                                              'THEN''BEGIN'
116                                                     GIVE SECTOR(1);
117                                                     D[1]:=XSN;
118                                                     DISCOP(2,SN,ADDR)
119                                                     WRITEN(SN,4,1);
120                                                     SN:=XSN;
121                                                     INITIALIZE SECTOR
122                                                     'END'
123                                           'END'
124                               'END'
125                           'END';
126                     'IF' MODCOUNT = 0
127                        'THEN''BEGIN'
128                              'FOR'I:=3,I+1'WHILE'I<=446'DO'D[I]:=D[I+1];
129                              D[0]:=ID[1]; D[1]:=ID[2]; D[2]:=ID[3]
130                              'END';
131                     DISCOP(2,SN,ADDR);
132                     WRITEN(SN,4,1);
133                     OUTSTR(1," :SIZE"); WRITEN(WORDS,5,1);
134                     SHUTM
135                     'END'
136           'END';
137       INITIALIZE SECTOR;
138       'FOR' I:=0,I+1 'WHILE' I <= TOP TRACK 'DO'
139           'BEGIN'
140           D[J] := TRACK[I];
141           J := J+1
142           'END';
143       D[3] := TOP TRACK;
144       DISCOP(2,1,ADDR); (DISC STATUS)
145       OUTSTR(1,"!CLLS4!**BUILD OK!LLL!");
146       'CODE''BEGIN'
147          ´   HLT        ´;
148              'END'
149   'END'
150   'FINISH'
```

PORTOS Supplementary Software (PRIME 300)

```
 1  C
 2  C.........PAPER TAPE READ
 3  C         (TO LOAD, TYPE "CO 200000")
 4  C
 5  C   READ ASCII FILES ON PAPER TAPE AND WRITE THEM
 6  C   TO NAMED FILES ON DISC ONE AT A TIME
 7  C
 8        COMMON ICHAR
 9        DIMENSION NOF(3), L(80)
10        INTEGER BUF(10400)
11        INTEGER CR, LF, EOF, SP
12        DATA CR/30215/, LF/30212/, EOF/30223/, SP/30240/
13        I=1
14        LOBUF=10400
15  C  FETCH FILE NAME
16        CALL TNOUA('FILENAME? ',10)
17        READ(1, 5)(NOF(J), J=1, 3)
18      5 FORMAT(3A2)
19  C  CREATE IT FOR INPUT
20        CALL CONTRL(2, NOF, 5)
21  C  START PAPER TAPE READER
22  C  READ CHARACTERS AND FILL BUFFER
23  C  UNTIL EOF CHAR IS READ
24     10 CALL PTRON
25        DO 16 J=1, LOBUF
26        CALL INP
27        IF(ICHAR.EQ.CR) ICHAR=SP
28        BUF(J)=ICHAR
29        IF(ICHAR.EQ.EOF) J=LOBUF
30     16 CONTINUE
31        CALL PTROFF
32  C  SCAN BUF, MOVE A RECORD AT A TIME
33  C  TO DISC SHIFTING CHARS LEFT SO AS
34  C  TO MAKE THEM COMPATIBLE WITH IO RTNS.
35     20 J=1
36     23 ICHAR=BUF(J)
37        IF(ICHAR.EQ.EOF) GOTO 50
38        IF(ICHAR.EQ.LF) GOTO 40
39        CALL MOVEL
40        L(I)=ICHAR
41        J=J+1
42        I=I+1
43        IF(J.GT.LOBUF) GOTO 10
44        GOTO 23
45     40 N=I-1
46        IF(N.EQ.0) L(1)=SP
47        WRITE(5, 45)(L(I), I=1, N)
48     45 FORMAT(80A1)
49        I=1
50        J=J+1
```

```
51          IF(J.GT.LOBUF) GOTO 10
52          GOTO 23
53    C   END OF INPUT. CLOSE FILE & EXIT
54       50 N=I-1
55          IF(N.GT.1) WRITE(5, 45)(L(I), I=1, N)
56          CALL CONTRL(4, NOF, 5)
57          WRITE(1, 60)
58       60 FORMAT(/'DONE !')
59          CALL EXIT
60          END
```

PORTOS Supplementary Software (TI 990/10)

```
1   'CORAL'
2   'PROGRAM' FORMAT
3   'DEFINE' VI "'VALUE''INTEGER'";
4   'EXTERNAL' ('PROCEDURE' MESSAGE(VI);
5               'PROCEDURE' PUT CHAR(VI)  );
6   'DELETE' VI;
7   'SEGMENT' DISK BUILD
8   'BEGIN'
9         'INTEGER' RES, I;
10        'INTEGER' DISC STATUS, COMMAND, FORMAT, CYLINDER,
11                  COUNT, ADDRESS, SELECT, CONT STATUS;
12
13        'CODE''BEGIN'
14          ´   LWPI  C$WS      ´;
15          ´   RSET            ´;
16          ´   LI    R12,0     ´;
17          ´   SBO   9         ´;
18          ´   SBO   10        ´;
19          ´   SBO   11        ´;
20          ´   SBO   12        ´;
21          ´   SBO   13        ´;
22              'END';
23        RES := 'HEX'(2000); (CLEAR 464 WORDS FROM >4000)
24        'FOR' I:=1 'STEP' 1 'UNTIL' 464 'DO'
25            'BEGIN'
26            [ RES ] := 0; (CLEAR WORD)
27            RES := RES+1
28            'END';
29        CLEAR;
30        TOP BYTE(COMMAND) := 'HEX'(07); (RESTORE)
31        EXECUTE;
32        RES := WAIT;
33        'IF' RES = 0
34           'THEN''BEGIN'
35                'FOR' I:=0,I+1 'WHILE' I<=202 'DO'
36                    'BEGIN'
37                    CLEAR; DISPLAY(I);
38                    CYLINDER := I;
39                    TOP BYTE(COMMAND):='HEX'(06); (SEEK)
40                    EXECUTE;
41                    RES := WAIT;
42                    'IF' RES = 0
43                       'THEN''BEGIN'
44                             CLEAR;
45                             TOP BYTE(COMMAND):='HEX'(01);(WR.FORMAT)
46                             BOT BYTE(COMMAND):=0; (SURFACE)
47                             FORMAT:='HEX'(0300); (3 SECTORS/RECORD)
48                             CYLINDER := I;
49                             COUNT := 'HEX'(3A0); (MAX)
50                             ADDRESS := 'HEX'(4000);
```

```
51                              EXECUTE;
52                              RES := WAIT;
53                              'IF' RES = 0
54                                'THEN''BEGIN'
55                                    BOT BYTE(COMMAND):=1;(SURFACE)
56                                    EXECUTE;
57                                    RES := WAIT
58                                    'END'
59                              'END';
60                        'IF' RES <> 0 'THEN' I:=300
61                        'END'
62                     'END';
63         DISPLAY(RES);
64         'CODE''BEGIN'
65              ´    IDLE      ´;
66                   'END'
67     'END'
68     'FINISH'
```

REFERENCES

1) APPLE Computer Inc.
1979
"DOS Version 3.2 Instructional and Reference Manual"

2) Baig, W. G.
Symp.Microprocessors At Work Sussex University 1976 p49
"Microprocessor Program Development Support"

3) Baker, K.
Microprocessors and Microsystems 1979 V3 N2 p87
"Microprocessors and Software Design Tools"

4) Balzer, R. M.
Comm.of ACM 1973 V16 N2 p117
"An Overview of the ISPL Computer System Design"

5) Barron, D. W. & Jackson, I. R.
SOFTWARE Practice & Experience 1972 V2 p143
"The Evolution of Job Control Languages"

6) Bennett, J. L.
Ann.Review of Information Science and Technology 1972 V7 p159
"The User Interface in Interactive Systems"

7) Bennett-Novak, G.
SIGPLAN Notices 1976 V11 N4 p40
"Machine-Independent Extended FORTRAN for Minicomputers"

8) Bergeron, R. D. et al
Advances in Computers 1972 V12 p175
"Systems Programming Languages"

9) Boies, S. J.
IBM Systems Journal 1974 V13 N1 p2
"User Behaviour on an Interactive Computer System"

10) Brown, P. J.
Comm.of ACM 1972 V15 N12 p1059
"Levels of Language for Portable Software"

11) Brown, P. J.
Annual Review in Automatic Programming 1969 V6 N2 p37
"A Survey of Macro Processors"

12) Brown, P. J.
SOFTWARE Practice & Experience 1974 V4 p139
"Writing Software in ALGOL"

13) Brown, W.
COMPUTER 1978 V11 N3 p40
"Modular Programming in PL/M"

14) Brunt, R. F. & Tuffs, D. E.
SOFTWARE Practice & Experience 1976 V6 p93
"A User-Orientated Approach to Control Languages"

References

15) Campbell-Kelly, M.
Macdonald & Co. Ltd. 1974
"Introduction to Macros"

16) Cheriton, D. R. et al
Comm.of ACM 1979 V22 N2 p105
"THOTH, A Portable Real-Time Operating System"

17) Clark, B. L. & Horning, J. J.
ACM SIGPLAN-SIGOPS 1973 V8 N9 p59
"Reflections on a Language Designed to Write an Operating System"

18) Coleman, S. S. et al
SOFTWARE Practice & Experience 1974 V4 p5
"The Mobile Programming System JANUS"

19) Colijn, A. W.
SOFTWARE Practice & Experience 1976 V6 p133
"Experiments with the KRONOS Control Language"

20) Colin, A. J. T.
SOFTWARE Practice & Experience 1972 V2 N2 p137
"The Implementation of STAB-1"

21) Commodore PET 2001
1979
"Personal Computer User Manual"

22) Conradi, R.
SIGPLAN Notices 1976 V11 N4 p8
"Critical Comments on PASCAL"

23) Corbato, F. J.
Datamation 1969 V15 N5 p68
"PL/I As A Tool For System Programming"

24) Coulouris, G.
Microprocessors and Microsystems 1979 V3 N2 p69
"Personal Computers in Offices of the Future"

25) Coulouris, G. F.
Annual Review in Automatic Programming 1969 V6 N2 p89
"A Machine-Independent Assembly Language for System Programming"

26) Dubois, P. J. & Martin, J. L.
SIGPLAN Symp.on System Implementation Languages 1971 V6 N9 p92
"The LRLTRAN language Used in FROST & FLOE Operating Systems"

27) Eadie, D.
Reston Publishing Co. Inc. ISBN 0-8359-4387-9
"Minicomputers Theory & Operation"

28) Eason, K. D.
Computer Journal 1976 V19 N1 p3
"Understanding the Naive Computer User"

References
----

29) Editorial
SOFTWARE Practice & Experience 1974 V4 p3
"Programming Languages for Writing System Programs"

30) Evershed, D. G. & Rippon, G. E.
Computer Journal 1971 V14 p87
"High Level Languages for Low Level Users"

31) Fletcher, J. G.
SIGPLAN Notices 1975 V10 N11 p5
"Should High-Level Languages Be Used To Write System Software"

32) Fletcher, J. G. et al
SIGPLAN Notices 1972 V7 N7 p28
"On The Appropriate Language for Systems Programming"

33) Frank, G. R.
CREST Portability Conference 1976
"A Portable Operating System"

34) Frenzel Jr., L. E.
Howard W. Sams & Co. Inc. 1978 ISBN 0-672-21486-5
"Getting Acquainted With Microcomputers"

35) Futuredata 2300 Series
Advanced Microprocessor Development System 1978
"Reference Manual"

36) Good, J. & Moon, B. A. M.
SOFTWARE Practice & Experience 1973 V3 p9
"FORTRAN As Provided by Some Major Manufacturers in 1970"

37) Gurski, A.
SIGPLAN Notices 1973 March p18
"Job Control Languages as Machine Orientated Languages"

38) Hansen, P. B.
SOFTWARE Practice & Experience 1976 V6 p141
"The SOLO Operating System: A Concurrent PASCAL Program"

39) Hertweck, F. R.
IFIP Conf.on Command Languages Ed. C.Unger 1975 North Holland p43
"Command Languages : Design Consideration and Basic Concepts"

40) Hewlett-Packard HP2000
1976
"Access BASIC Reference Manual"

41) Holdsworth, D.
SOFTWARE Practice & Experience 1977 V7 p331
"System Implementation in Algol68-R"

42) Hopkins, M.
SIGPLAN Symp.on System Implementation Languages 1971 V6 N9 p89
"Problems of PL/I for Systems Programming"

References

43) Hugh Pushman Associates (Compilers) Limited
1977
"The RMCS CORAL66 Compiler for the PDP-11"

44) Hunter, J. M. D.
RMCS Computing Science Branch Technical Note 15 1973
"Manual For Enhanced CORAL66 For ICL 1900 Series"

45) Isaacson, P. et al
COMPUTER 1978 V11 N9 p86
"Personal Computing"

46) Jackson, K. & Simpson, H. R.
RRE Technical Note 1975 No.778
"MASCOT-Modular Approach to Software Construction, Operation & Test"

47) Krayl, H. et al
IFIP Conf.on Command Languages Ed.C. Unger 1975 North Holland p293
"Portability of JCL Programs"

48) Landy, B.
BCS Symp.on JCL Ed. D. Simpson 1974 NCC ISBN 0-85012-119-1 p53
"Development of a New Command Language for the IBM System/370"

49) Lang, S. R.
SERT Symp.Minicomputers At Work 1976 p113
"Resident CORAL66 Compiler For The Intel 8080"

50) Lauesen, S.
BIT 1973 V13 N3 p323
"Program Control of Operating Systems"

51) Lister, A. M. & Sayer, P. J.
SOFTWARE Practice & Experience 1977 V7 p613
"Hierarchical Monitors"

52) Lyle, D. M.
SIGPLAN Symp.on System Implementation Languages 1971 V6 N9 p73
"A Hierarchy of High-Order Languages for System Programming"

53) Lynch, W. C.
SIGPLAN Conf. on Reliable Software 1975 p252
"CHI Operating System"

54) M68MDOS3 EXORdisk II/III Operating System
1978
"User's Guide"

55) Madsen, J.
SOFTWARE Practice & Experience 1979 V9 p25
"CCL - A High-Level Command Language"

56) Mann, W. C.
AFIPS 1975 V44 p785
"Why Things Are So Bad For The Computer-naive User"

References

57) Marcotty, M. & Schults, H.
SOFTWARE Practice & Experience 1974 V4 p79
"The Systems Programming Language MALUS"

58) Miller, L. H.
AFIPS 1977 V46 p409
"A Study in Man-Machine Interaction"

59) Miller, R.
ACM OS Review 1978 V12 N3 p32
"UNIX - A Portable Operating System?"

60) NORTH STAR DOS for Horizon Micro Disk System
SOFT-DOC Revision 2.1 1979
"NORTH STAR System Software Manual"

61) Neal, D. & Wallentive, V.
SOFTWARE Practice & Experience 1978 V8 p341
"Experiences With The Portability of Concurrent PASCAL"

62) Newell, G. B.
BCS Symp.on JCL Ed. D. Simpson 1974 NCC ISBN 0-85012-119-1 p61
"The Family of Operating Systems Called GEORGE"

63) Newey, M. C. et al
SOFTWARE Practice & Experience 1972 V2 p107
"Abstract Machine Modelling to Produce Portable Software"

64) Newman, I. A.
Proc. Datafair 1973 V2 p353
"The UNIQUE Command Language - Portable Job Control"

65) Nicholls, J. E.
BCS Symp.on JCL Ed. D. Simpson 1974 NCC ISBN 0-85012-119-1 p39
"Job Control in IBM System/360 & 370"

66) Oestreicher, M. D. et. al.
Comm. ACM 1976 N11 p685
"GEORGE3 - A General Purpose Time Sharing Operating System"

67) Olivetti P6060 Personal Minicomputer
1977
"Reference Manual"

68) Peschke, J.
SIGPLAN Notices 1971 V6 N5 p16
"PL/I Subsets for Software Writing"

69) Powell, M. S.
SOFTWARE Practice & Experience 1979 V9 p561
"Transporting and Using The SOLO Operating System"

70) Pyster, A. & Dutta, A.
SOFTWARE Practice & Experience 1978 V8 N1 P99
"Error Checking Compilers and Portability"

References

71) Rao, G. V.
Van Nostrand Reinhold Co. 1978 ISBN 0-442-22000-6
"Microprocessors and Microcomputer Systems"

72) Rayner, D.
SOFTWARE Practice & Experience 1975 V5 p375
"Recent Developments in Machine-Independent Job Control Languages"

73) Richards, M.
Proc.of AFIPS 1969 V34 p557
"BCPL: A Tool For Compiler Writing and Sytem Programming"

74) Richards, M. et al
SOFTWARE Practice & Experience 1979 V9 p513
"TRIPOS - Portable Operating System for Minicomputers"

75) Sabin, M. A.
SOFTWARE Practice & Experience 1976 V6 p393
"Portability - Some Experiences With FORTRAN"

76) Sammet, J. E.
SIGPLAN Symp.on System Implementation Languages 1971 V6 N9 p1
"Brief Survey of Languages Used in Systems Implementations"

77) Sammet, J. E.
Comm.of ACM 1976 V19 N12 p655
"Roster of Programming Languages for 1974-75"

78) Sapper, G. R.
SIGPLAN Symp.on System Implementation Languages 1971 V6 N9 p37
"PS440 As A Tool For Implementing A Time-Sharing System"

79) Shearing, B. H.
BCS Symp.on JCL Ed.D. Simpson 1974 NCC ISBN 0 85012 119 1 p15
"JCL - As They Are And As They Might Be"

80) Slimick, J.
SIGPLAN Symp.on System Implementation Languages 1971 V6 N9 p20
"Current Systems Implementation Languages: One User's View"

81) Snow, C. R.
SOFTWARE Practice & Experience 1978 V8 N1 p41
"An Exercise in the Transportation of an Operating System"

82) Stephenson, C. J.
ACM SIGOPS Operating Systems Review 1973 V7 N4 p22
"On The Structure and Control of Commands"

83) Stoy, J. E. & Strachey, C.
Computer Journal 1972 V15 N2 p117
"OS6-An Experimental Operating System for a Small Computer"

84) Stuart, T.
SIGPLAN Notices 1976 V11 N4 p144
"Adapting Large Systems To Small Machines"

References

85) System Designers Limited
September 1977
"Portable CORAL Compiler for the Texas 990 DX10 2.2 Computer"

86) Tanenbaum, A. S. et al
SOFTWARE Practice & Experience 1978 V8 p681
"Guidelines for Software Portability"

87) Terashima, N.
SIGPLAN Notices 1974 V9 N12 p35
"SYSL - System Description Language"

88) Texas Instruments TX990 Operating Systems
1979
"TX990 Operating System Programmer's Guide"

89) Tsichritzis, D.
Software Engineering Ed.F.L.Bauer 1973 Springer-Verlag p374
"Project Management"

90) Waite, W. M.
Computer Journal 1970 V13 N1 p28
"Building A Mobile Programming System"

91) Waite, W. M.
SOFTWARE Practice & Experience 1975 V5 N3 p295
"Hints on Distributing Portable Software"

92) Walther, G. H. & O'Neill,Jr., H. F.
AFIPS 1974 V43 p379
"Online User-Computer Interface: Effects of Interface Flexibility,..."

93) Webb, J. T.
NCC Punlications 1978 ISBN 0 85012 193 0
"CORAL66 Programming"

94) Welsh, J. E. et al
SOFTWARE Practice & Experience 1977 V7 p685
"Ambiguities and Insecurities in PASCAL"

95) Wichman, B. A.
Computer Journal 1972 V15 p8
"Five ALGOL Compilers"

96) Wiederhold, G. C. M.
SIGPLAN SIGOPS Interface Meeting V8 p140
"The Need and Techniques to Obliterate Control Languages"

97) Woodward, P. M. et al
HMSO 1974 ISBN 0 11 470221 7
"Official Definition of CORAL66"

98) Wortman, D. B. et al
SOFTWARE Practice & Experience 1976 V6 p411
"Six PL/I Compilers"

References
----

99) Wulf, W. A.
SIGPLAN Symp.System Implementation Languages 1971 V6 N9 p42
"Reflections on a Systems Programming Language"

100) Wulf, W. A. et al
COMM.of ACM 1971 V14 N12 p780
"BLISS: A Language For Systems Programming"