pdp11 software handbook
The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use of a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1975, by Digital Equipment Corporation

The following are trademarks of Digital Equipment Corporation:

<table>
<thead>
<tr>
<th>CDP</th>
<th>DIGITAL</th>
<th>INDAC</th>
<th>QUICKPOINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTER LAB</td>
<td>DNC</td>
<td>KA10</td>
<td>RAD-8</td>
</tr>
<tr>
<td>COMSYST</td>
<td>EDGRIN</td>
<td>LAB-8</td>
<td>RSTS</td>
</tr>
<tr>
<td>COMTEX</td>
<td>EDUSYSTEM</td>
<td>LAB-8/e</td>
<td>RSX</td>
</tr>
<tr>
<td>DDT</td>
<td>FLIP CHIP</td>
<td>LAB-K</td>
<td>RTM</td>
</tr>
<tr>
<td>DEC</td>
<td>FOCAL</td>
<td>OMNIBUS</td>
<td>RT-11</td>
</tr>
<tr>
<td>DECCOMM</td>
<td>GLC-8</td>
<td>OS/8</td>
<td>SABR</td>
</tr>
<tr>
<td>DECTAPE</td>
<td>IDAC</td>
<td>PDP</td>
<td>TYPESET 8</td>
</tr>
<tr>
<td>DIBOL</td>
<td>IDACS</td>
<td>PHA</td>
<td>TYPESET 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PS/8</td>
<td>UNIBUS</td>
</tr>
</tbody>
</table>

This handbook was composed and edited with the aid of DIGITAL's
**TYPESET-11 System**
CONTENTS

SECTION I  PDP-11 SOFTWARE FAMILY

CHAPTER 1  INTRODUCTION ............................................. 1-1
  1.1 HARDWARE/SOFTWARE SYSTEMS ................................. 1-1
  1.2 OPERATING SYSTEMS ........................................... 1-1
  1.3 LANGUAGES AND LANGUAGE PROCESSORS ......................... 1-2
  1.4 SYSTEMS SUMMARY ............................................ 1-6

CHAPTER 2  OPERATING SYSTEMS ..................................... 2-1
  2.1 COMPONENTS AND FUNCTIONS ................................... 2-1
  2.2 PROCESSING MODES ............................................ 2-3
  2.3 DATA MANAGEMENT ............................................. 2-5
    2.3.1 Physical and Logical Units of Data ...................... 2-5
    2.3.2 Data Storage and Transfer Modes ....................... 2-6
    2.3.3 I/O Devices and Physical Data Access Characteristics 2-9
    2.3.4 Physical Device Characteristics and Logical Data Organizations 2-11
    2.3.5 File Structures and Access Methods .................... 2-13
    2.3.6 Directories and Directory Access Techniques ........... 2-16
    2.3.7 File Protection ........................................ 2-19
    2.3.8 File Naming ........................................... 2-20
  2.4 USER INTERFACES .............................................. 2-22
    2.4.1 Special Terminal Commands ............................. 2-22
    2.4.2 I/O Commands .......................................... 2-23
    2.4.3 Monitor and Command Language Commands ................. 2-24
  2.5 PROGRAMMED SYSTEM SERVICES ................................ 2-26
  2.6 SYSTEM UTILITIES ........................................... 2-28

CHAPTER 3  LANGUAGE PROCESSORS ................................ 3-1
  3.1 LANGUAGE TRANSLATION SYSTEMS .............................. 3-1
  3.2 PDP-11 ASSEMBLERS AND THE FORTRAN COMPILERS .............. 3-4
  3.3 INCREMENTAL COMPILERS ..................................... 3-7

SECTION II  PDP-11 OPERATING SYSTEMS

CHAPTER 1  CASSETTE PROGRAMMING SYSTEM CAPS-11 ............... 1-1
  1.1 OPERATING SYSTEM FUNCTIONS AND FEATURES ................. 1-1
  1.2 MONITOR ORGANIZATION ..................................... 1-2
    1.2.1 Monitor Components .................................. 1-3
    1.2.2 User Program Loading ................................ 1-5
  1.3 SYSTEM CONVENTIONS ........................................ 1-8
    1.3.1 I/O Devices .......................................... 1-8
    1.3.2 File Naming .......................................... 1-8
    1.3.3 Monitor/Operator Console Commands ..................... 1-10
    1.3.4 Entering I/O Specifications Through the CSI .......... 1-11
    1.3.5 Special Console Commands ............................. 1-12
    1.3.6 Cassette File and Data Formats ....................... 1-12
1.4 MONITOR I/O PROGRAMMING SERVICES .......................... 1-14
1.5 SYSTEM PROGRAMS .................................................. 1-17
1.6 BASIC LANGUAGE PROCESSOR ...................................... 1-20

CHAPTER 2 FOREGROUND/BACKGROUND OPERATING SYSTEM RT-11 .................................................. 2-1
2.1 OPERATING SYSTEM FUNCTIONS AND FEATURES .............. 2-1
2.2 MONITOR ORGANIZATIONS ........................................... 2-3
   2.2.1 Monitor Components ........................................... 2-3
   2.2.2 General Memory Layout and Component Sizes .............. 2-4
   2.2.3 I/O System Design and Operation.......................... 2-7
   2.2.4 Batch Processing ............................................. 2-7
   2.2.5 Switching Between Single-Job and Foreground/Background ........................................... 2-8
2.3 SYSTEM CONVENTIONS .............................................. 2-8
   2.3.1 Physical Device Names ........................................ 2-8
   2.3.2 Filenames and Extensions ..................................... 2-8
   2.3.3 Data Formats .................................................. 2-9
   2.3.4 File Structure ................................................ 2-10
2.4 COMMANDS ......................................................... 2-10
   2.4.1 Keyboard Communication ...................................... 2-10
   2.4.2 Entering I/O Information Using the CSI ................. 2-14
   2.4.3 BATCH Job Control Language ................................. 2-14
2.5 MONITOR PROGRAMMED REQUESTS ................................. 2-15
   2.5.1 Summary of Programmed Requests ......................... 2-17
   2.5.2 Program Environment Control ................................. 2-20
   2.5.3 Resource and System Interrogation ......................... 2-22
   2.5.4 Command Interpretation ..................................... 2-23
   2.5.5 File Operations .............................................. 2-24
   2.5.6 Input/Output ............................................... 2-24
   2.5.7 Interjob Communication .................................... 2-26
   2.5.8 Timer Support .............................................. 2-27
   2.5.9 Program Termination or Suspension ....................... 2-27
   2.5.10 Interrupt Service .......................................... 2-27
2.6 SYSTEM PROGRAMS ................................................. 2-28
   2.6.1 EDIT Interactive Editor .................................... 2-28
   2.6.2 LINK Linker ................................................. 2-28
   2.6.3 LIBR Librarian ............................................. 2-29
   2.6.4 ODT On-Line Debugger ...................................... 2-29
   2.6.5 PATCH Code Patch Utility .................................. 2-30
   2.6.6 PATCHO Object Patch Utility ............................... 2-30
   2.6.7 PIP Peripheral Interchange Program ....................... 2-30
   2.6.8 SRCCOM Source Compare Utility ............................ 2-30
   2.6.9 FILEX File Exchange Utility ............................... 2-30
   2.6.10 DUMP File Dump Utility .................................. 2-31
2.7 LANGUAGES .......................................................... 2-31
   2.7.1 MACRO Assembler ............................................ 2-31
   2.7.2 EXPAND Macro Expander and ASEMBL Assembler ............ 2-31
   2.7.3 FORTRAN .................................................... 2-32
   2.7.4 FOCAL ....................................................... 2-33
   2.7.5 Single-User BASIC .......................................... 2-33
   2.7.6 Multi-User BASIC .......................................... 2-35
CHAPTER 3  RESOURCE-SHARING TIMESHARING SYSTEM
RSTS/E ............................................. 3-1
  3.1 FUNCTIONS AND FEATURES ..................... 3-1
  3.2 SYSTEM CONFIGURATION AND OPERATION .......... 3-3
      3.2.1 Timesharing Operations Overview .......... 3-4
      3.2.2 System Generation ......................... 3-6
      3.2.3 System Initialization ...................... 3-7
      3.2.4 System Management Utility Programs .... 3-8
  3.3 DEVICE AND FILE CONVENTIONS .................. 3-10
      3.3.1 File Specifications ........................ 3-10
      3.3.2 Data Formats ............................. 3-11
      3.3.3 File Access Techniques ..................... 3-12
      3.3.4 Logical Disk Structures .................... 3-14
      3.3.5 System Accounts and Libraries ............ 3-15
      3.3.6 Privileged Accounts and Privileged Programs 3-16
  3.4 USER INTERFACE .................................. 3-17
      3.4.1 System Commands ........................... 3-17
      3.4.2 General System Utility Programs ........... 3-20
      3.4.3 Batch Processing .......................... 3-21
  3.5 SYSTEM FUNCTIONS AND THE PEEK FUNCTION .... 3-22
  3.6 BASIC-PLUS ..................................... 3-27
      3.6.1 Immediate Mode Operations .................. 3-32
      3.6.2 Data Formats and Operations ............... 3-33
      3.6.3 Matrix Manipulation ......................... 3-33
      3.6.4 Advanced Statement and Function Features 3-34
  3.7 COBOL ............................................ 3-34
  3.8 FORTRAN IV ...................................... 3-35

CHAPTER 4  MULTI-USER DATA BASE MANAGEMENT SYSTEM
MUMPS-11 ........................................... 4-1
  4.1 FUNCTIONS AND FEATURES ....................... 4-1
  4.2 SYSTEM OPERATION AND COMPONENTS ............... 4-2
      4.2.1 Executive ................................. 4-3
      4.2.2 I/O Monitor ................................ 4-4
      4.2.3 Log-in Processing and the Language Interpreter 4-5
      4.2.4 Data Base Supervisor ....................... 5-6
  4.3 DATA MANAGEMENT FACILITIES ..................... 4-7
      4.3.1 The MUMPS Disk Structure and Global Arrays 4-8
      4.3.2 Terminals and Ancillary I/O Devices .... 4-9
      4.3.3 Data Storage .............................. 4-11
  4.4 USER INTERFACE .................................. 4-11
  4.5 SYSTEM UTILITIES AND LIBRARY PROGRAMS ....... 4-13
  4.6 THE MUMPS LANGUAGE ............................. 4-15
      4.6.1 Data Storage Elements ..................... 4-16
      4.6.2 Expressions ............................... 4-18
      4.6.3 Commands .................................. 4-19
      4.6.4 Functions .................................. 4-23
      4.6.5 System Variables .......................... 4-24
CHAPTER 5 REAL-TIME MULTIPROGRAMMING FAMILY RSX-11

5.1 FUNCTIONS AND FEATURES ............................................. 5-1
5.2 COMMON RSX-11 FAMILY OPERATING SYSTEM CONCEPTS ............. 5-6
5.3 SYSTEM ORGANIZATION AND GENERATION ............................ 5-13
5.3.1 RSX-11D System Organization ..................................... 5-15
5.3.2 RSX-11M Executive and Memory Structures ....................... 5-18
5.3.3 RSX-11S System Components ...................................... 5-21
5.4 SYSTEM CONVENTIONS .................................................. 5-23
5.4.1 Devices .............................................................. 5-23
5.4.2 File Structures ..................................................... 5-24
5.4.3 File Specifiers ..................................................... 5-26
5.4.4 MCR Operator Commands and Terminal Control .................. 5-28
5.4.5 Indirect Files (Command Files) .................................... 5-33
5.4.6 Batch Processing (RSX-11D Systems Only) ....................... 5-36
5.5 SYSTEM DIRECTIVES ................................................... 5-37
5.6 FILE CONTROL SERVICES .............................................. 5-42
5.6.1 File Access Methods ............................................... 5-42
5.6.2 Data Formats for File-Structured Devices ....................... 5-43
5.6.3 Block I/O Operations .............................................. 5-43
5.6.4 Record I/O Operations ............................................ 5-43
5.6.5 Shared Access to Files ............................................ 5-45
5.6.6 Spooling Operations .............................................. 5-45
5.6.7 FCS Macros and Macro Usage .................................... 5-46
5.7 SYSTEM UTILITY PROGRAMS .......................................... 5-47
5.8 LANGUAGES ................................................................ 5-50
5.8.1 MACRO .................................................................. 5-50
5.8.2 FORTRAN IV ......................................................... 5-50
5.8.3 FORTRAN IV-PLUS .................................................. 5-50
5.8.4 COBOL ................................................................. 5-50

CHAPTER 6 INTERACTIVE APPLICATION SYSTEM IAS ..................... 6-1

6.1 OPERATING SYSTEM FUNCTIONS AND FEATURES .................... 6-1
6.2 IAS EXECUTIVE ORGANIZATION AND SERVICES ..................... 6-3
6.2.1 The Active Task List ............................................... 6-4
6.2.2 The Timesharing Scheduler ....................................... 6-4
6.2.3 Batch Processing .................................................... 6-6
6.2.4 Executive Data Structures ....................................... 6-7
6.2.5 I/O Services and Device Independence ......................... 6-7
6.2.6 Sharing of Common Routines ..................................... 6-7
6.2.7 System Generation and Initialization ........................... 6-8
6.3 COMMAND LANGUAGE INTERPRETERS ............................. 6-8
6.3.1 Program Development System (PDS) ............................... 6-9
6.3.2 System Control Interface (SCI) .................................. 6-14
6.4 TIMESHARING CONTROL PRIMITIVES .............................. 6-14
6.4.1 CLI Control Primitives ............................................. 6-15
6.4.2 Task Initiation and Control ...................................... 6-15
6.4.3 Buffer Management ............................................... 6-16
6.4.4 Terminal Event Control Primitives .............................. 6-16
6.4.5 Job Node Management ............................................ 6-16
6.4.6 System Management ............................................... 6-17
6.4.7 Device Management Primitives ................................... 6-18
6.4.8 CLI Service ......................................................... 6-18
6.5 SYSTEM TASKS AND SPECIAL UTILITIES .................................. 6-19
6.6 LANGUAGES .............................................................................. 6-19
  6.6.1 MACRO ............................................................................... 6-19
  6.6.2 FORTRAN IV ....................................................................... 6-20
  6.6.3 FORTRAN IV-PLUS ............................................................... 6-20
  6.6.4 COBOL ............................................................................... 6-20
  6.6.5 BASIC ............................................................................... 6-21

SECTION III PDP-11 LANGUAGE PROCESSORS

CHAPTER 1 MACRO ................................................................. 1-1
  1.1 FUNCTIONS AND FEATURES .............................................. 1-1
  1.2 LANGUAGE .......................................................................... 1-1
    1.2.1 Symbols and Symbol Definitions .................................. 1-2
    1.2.2 Directives ...................................................................... 1-4
  1.3 ASSEMBLER OPERATION .......................................................... 1-11
  1.4 ASSEMBLER ENVIRONMENTS .................................................. 1-15

CHAPTER 2 FORTRAN IV ....................................................... 2-1
  2.1 FUNCTIONS AND FEATURES .............................................. 2-1
  2.2 LANGUAGE .......................................................................... 2-3
  2.3 COMPILER OPERATION ............................................................ 2-9
  2.4 FORTRAN IV OPERATING ENVIRONMENTS .......................... 2-14

CHAPTER 3 FORTRAN IV-PLUS ........................................... 3-1
  3.1 FUNCTIONS AND FEATURES .............................................. 3-1
  3.2 LANGUAGE EXTENSIONS ..................................................... 3-2
  3.3 COMPILER OPERATION AND OPTIMIZATIONS ..................... 3-5
  3.4 ENVIRONMENTS ................................................................... 3-9

CHAPTER 4 COBOL ............................................................... 4-1
  4.1 FUNCTIONS AND FEATURES .............................................. 4-1
  4.2 LANGUAGE FEATURES .......................................................... 4-2
  4.3 COMPILER IMPLEMENTATION AND OPERATION ................ 4-6
  4.4 COBOL OPERATING SYSTEM ENVIRONMENTS .................. 4-10
  4.5 UTILITY PROGRAMS .............................................................. 4-10

CHAPTER 5 BASIC ............................................................... 5-1
  5.1 FUNCTION AND FEATURES .............................................. 5-1
  5.2 LANGUAGE .......................................................................... 5-1
    5.2.1 BASIC Files .................................................................... 5-7
    5.2.2 Creating, Modifying and Executing BASIC Programs ....... 5-8
  5.3 COMPILER OPERATION .......................................................... 5-9
  5.4 BASIC ENVIRONMENTS .......................................................... 5-11

APPENDIX A GLOSSARY ......................................................... A-1

APPENDIX B COMMONLY-USED MNEMONICS AND ABBREVIATIONS .... B-1
This handbook describes the major operating systems and programming languages available for the PDP-11 family of computers. It is intended for the system manager or programmer who needs a brief introduction to the range of PDP-11 software products and who is interested in determining which products best suit a particular processing environment.

The handbook includes brief technical descriptions of the major products only. It does not include descriptions for all PDP-11 software. In particular, it does not describe the variety of special systems or applications software available. The Catalog of Software Products and Services provides a list of all PDP-11 software products, including special systems and application software. The handbook is divided into three sections. Section I introduces the range of operating systems available, from small program development or dedicated application systems to large, multifunction systems. It describes their common features and characteristics and explains their differences. It also introduces the programming languages available on the systems.

Section II includes a chapter for each major operating system. Each chapter provides a summary of the system's features, characteristics, and hardware requirements. Further information is provided on the system's organization, file structures, user interface, and programming support features.

Section III includes a chapter for each of the major programming languages available on two or more systems. The chapters summarize the special features or characteristics of the languages. In addition, they discuss the implementation of the languages on the operating systems on which they are available.

The technical descriptions provided in this handbook are not intended to be functional descriptions or operating procedures. This handbook is intended to be used in conjunction with the PDP-11 Processor Handbooks and PDP-11 Peripherals Handbook to introduce users to the PDP-11 family's products. Complete technical information can be found in the set of software manuals that accompany each product.

Because DIGITAL is constantly developing new products and improving current ones, the information in this document is subject to change. Users should consult their sales and software support representatives to obtain the latest information about a product's new features and characteristics.
section 1
pdp 11
software family

CHAPTER 1 INTRODUCTION
CHAPTER 2 OPERATING SYSTEMS
CHAPTER 3 LANGUAGE PROCESSORS
CHAPTER 1

INTRODUCTION

1.1 HARDWARE/SOFTWARE SYSTEMS
The PDP-11 Computer Family is a wide range of compatible central processors complemented by a variety of peripheral devices, software and services. The PDP-11 Family processors are ordered in incremental steps of speed and size, and organized into four groups by their typical applications:

- PDP-11 microcomputers (LSI-11 based 11/03 and 11V03) for stable, programmable dedicated applications
- PDP-11 minicomputers (11/04, 11/05 and 11/10) for dedicated applications which may be planned for possible upward growth
- PDP-11 system computers (11/35, 11/40 and 11/45) for multiple application tasks
- PDP-11 high-throughput computer (PDP-11/70) for multiple-purpose simultaneous application tasks

This handbook uses these processor groups as the basis for discussing the range of hardware/software systems available in the PDP-11 Family. In particular, an operating system that is designed to make the most use of a particular processor is normally available on any processor in the same group. In addition, an operating system that runs on a particular group of processors can often also run on processors in the group above or below it.

Furthermore, some programming languages may be available on either large or small hardware/software systems, but not both. For example, COBOL is available only on the larger systems. Other languages may be available on a wide range of systems, but may vary in characteristics significant for a particular application. For example, FORTRAN IV is available on both large and small systems, but compilation speed may vary from system to system, depending on the hardware configuration.

The flexibility of capabilities among hardware/software systems allows the user to select both the most appropriate hardware for a particular application’s needs, and the operating system and languages that can serve immediate needs and still allow for possible growth.

DIGITAL offers a variety of operating systems, languages, special applications and utility software for the PDP-11 Computer Family. This handbook discusses the major operating systems and languages offered. It assumes that the reader is familiar with operating system software and programming languages in general.

1.2 OPERATING SYSTEMS
An operating system not only provides access to the features of a
processor in its size range, it also organizes a processor and peripherals into a useful tool for a certain range of applications. For example, the operating systems that run on the small processors are generally intended for dedicated applications. The operating systems that run on large processors are multi-purpose, and can provide a variety of services. The major operating systems to be discussed are:

**CAPS-11** A small, single-user program development system designed to give the user maximum price/performance in the low end of the PDP-11 Family.

**RT-11** A small, single-user foreground/background system that can support a real-time application job’s execution in the foreground and an interactive or batch program development job in the background.

**MUMPS-11** A small-to-large sized timesharing system that offers a unique fast access data storage and retrieval system for large data base processing.

**RSTS/E** A moderate-large sized timesharing system that can support up to 63 concurrent jobs, which includes interactive terminal user jobs, detached jobs, and batch processing.

**RSX-11D** A large, real-time multiprogramming system that can support multi-user interactive and batch program development.

**RSX-11M** A small-to-moderate sized real-time multiprogramming system compatible with RSX-11D that can be generated for a wide range of application environments—from small, dedicated systems to large, multi-purpose real-time application and program development systems.

**RSX-11S** A small, execute-only member of the RSX-11 family for dedicated real-time multigrogramming applications (requires a host RSX-11M system).

**IAS** A large, multi-user timesharing system, allowing real-time applications execution concurrent with timeshared interactive and batch processing.

Figure 1-1 illustrates the operating system’s relative sizes (using their minimum memory requirements), along with the systems’ general natures.

Section II of this handbook describes these operating systems in detail. Included in each chapter are a general description of the requirements for the system, the monitor/executive characteristics, the file structures and data handling facilities, the user interfaces, the programmed monitor services, the system utilities, and the language processors supported. For users who are not familiar with DIGITAL PDP-11 system components and services, Chapter 2 of Section I provides a summary description of the PDP-11 operating system’s components and common facilities.

**1.3 LANGUAGES AND LANGUAGE PROCESSORS**
All PDP-11 operating systems offer a variety of programming language processors. A programming language is a tool that enables the user to
state a problem that a computer can solve. A programming language is designed to be easily understood and manipulated by humans, while a language processor translates the instructions into the machine’s language.

In general, the language processors available to run under an operating system are commensurate with the kind of applications for which the operating system is designed. For example, a real-time application environment could be a laboratory in which a scientific programming language is useful for problem solving. The real-time application operating systems offer FORTRAN IV language processors.

The programming languages available on two or more systems that are discussed in this handbook are:

- **MACRO** The general-purpose assembly language for PDP-11 computers.
- **FORTRAN IV** The language most used for scientific problem solving.
- **FORTRAN IV-PLUS** An advanced version of FORTRAN IV available on some PDP-11 operating systems.
- **COBOL** The language most used for business data processing.
- **BASIC** A language well-known in the educational and scientific communities for its ease-of-use.

Section III of this handbook describes these language processors in detail. Each chapter in the Operating Systems section (Section II) provides a brief description of the language processors that can run under an operating system. If the language processor is common among several operating systems, the user can refer to its description in Section III for complete details.
<table>
<thead>
<tr>
<th>LSI-11 based</th>
<th>11/04,05,10</th>
<th>11/35,40</th>
<th>11/45</th>
<th>11/70</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CAPS-11 Cassette-based program development system</strong>&lt;br&gt;8K to 28K words of memory. 8K BASIC available; requires at least 16K for string or laboratory and graphics peripherals support.&lt;br&gt;<strong>Languages:</strong> PAL-11 assembly included; BASIC optional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RT-11 Foreground/Background or Single-Job operating system</strong>&lt;br&gt;8K to 28K words of memory. In 8K: Single-Job (S-J) operation; subset MACRO included; BASIC, FORTRAN IV, FOCAL as options. In 16K: Foreground/Background (F/B) or S-J operation; BASIC can support string operations, laboratory and graphics peripherals; full MACRO assembler included; multi-user BASIC available as option supporting as many as 4 users (under S-J monitor). MU BASIC supports as many as 8 users in 24K words under S-J monitor; and as many as 4 in 28K words under F/B monitor.&lt;br&gt;<strong>Languages:</strong> MACRO included; FORTRAN IV, BASIC, MU BASIC, FOCAL are options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MUMPS-11 Data management timesharing system</strong>&lt;br&gt;16K to 124K words of memory. 16K system supports 2 users. At least 32K words are needed to support 6 users. Up to 40 timesharing users can be supported.&lt;br&gt;<strong>Language:</strong> MUMPS included</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RSTS/E General-purpose timesharing system</strong>&lt;br&gt;32K to 124K words of memory or 128K to 1024K words on 11/70. Depending on disk and memory configuration, as many as 24 users can be supported on an 11/35,40; as many as 32 users on an 11/45; as many as 63 users on an 11/70.&lt;br&gt;<strong>Languages:</strong> BASIC-PLUS included; COBOL, FORTRAN IV and MACRO are options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LSI-11 based</td>
<td>11/04,05,10</td>
<td>11/35,40</td>
<td>11/45</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>-------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>RSX-11S</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute-only real-time multiprogramming system</td>
<td>8K to 124K words of memory or 128K to 1024K words on 11/70. 8K system allows 4K for user tasks. 16K required for on-line task loading or support for tasks written in FORTRAN.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Languages:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program development on host RSX-11M system only</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RSX-11M</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small- to moderate-sized real-time multiprogramming system</td>
<td>16K to 124K words of memory or 128K to 1024K words on 11/70. 16K system allows 8K for user tasks; includes a subset MACRO. At least 24K words are required for full MACRO support, concurrent program development and application tasks execution, or memory management support.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Languages:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACRO included; FORTRAN IV and FORTRAN IV-PLUS are options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RSX-11D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large real-time multiprogramming system</td>
<td>48K to 124K words of memory or 128K to 1024K words on 11/70. User accounting and error logging and analysis supported. At least 56K words are required for concurrent real-time applications execution and program development.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Languages:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACRO included; FORTRAN IV, FORTRAN IV-PLUS and COBOL are options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IAS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-purpose multiprogramming system</td>
<td>64K to 124K words of memory or 128K to 1024K words on 11/70. Timeshared interactive and batch job processing with concurrent real-time applications execution. Depending on disk and memory configuration, as many as 10 interactive users can be supported on an 11/45; as many as 20 interactive users on an 11/70.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Languages:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACRO included; FORTRAN IV, FORTRAN IV-PLUS, COBOL and BASIC are options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1.4 SYSTEMS SUMMARY
The PDP-11 software systems can run on a variety of hardware configurations. The processor support and memory requirements for a particular computing system vary according to how the system is to be used. Table 1-1 shows which operating systems can run on each processor in the PDP-11 Family. As a general guide to size range, each processor group supports certain amounts of memory. In PDP-11 systems, memory size is defined as the number of K words, where K is equivalent to 1024 (decimal) and a word is 16 bits (2 bytes).

The small PDP-11 processors (LSI-11 based, PDP-11/04, 11/05 and 11/10) can support up to 28K words of memory. The PDP-11/35, 11/40 and 11/45 can support up to 124K words of memory. The PDP-11/70 requires a minimum memory of 64K words, and can support up to 1024K words.

For each operating system, the table lists the minimum and maximum amounts of memory supported. The more memory available to a system, the greater the system’s capabilities. Some language features, for example, can not be supported by very small systems, as shown in the table. Where the amount of memory is significant to a major system facility, it is noted in the in the description as the minimum memory required to support the feature. The maximum memory available supports all features listed at lower memory ranges.

The features and capabilities listed are intended as general guidelines only and do not constitute strict configurations, since the performance characteristics of any system are largely dependent on the kinds of operations being performed as well as the kind of hardware in the system.
CHAPTER 2

OPERATING SYSTEMS

Operating systems have two basic functions: they provide services for application program development and act as an environment in which application programs run. The character that an operating system has, that is, the services and environment it supplies, is appropriate only for a certain range of program development and application requirements in order to serve selected needs efficiently. Operating systems for the PDP-11 family of computers, however, share many similar program development techniques and processing environments. This chapter describes some of the common features and characteristics of PDP-11 operating systems as well as some of their differences.

2.1 COMPONENTS AND FUNCTIONS
An operating system is a collection of programs that organizes a set of hardware devices into a working unit that people can use. Figure 2-1 illustrates the relationship between users, the operating system, and the hardware. PDP-11 operating systems basically consist of two sets of software: the monitor (or executive) software and the system utilities.

![Diagram of computer system components]

Figure 2-1  Computer System Components

An operating system monitor is an integrated set of routines that acts as the primary interface between the hardware and a program running on the system, and between the hardware and the people who use the system. The monitor's basic functions can be divided among the routines that provide the following services:

- device and data management
- user interface
- programmed processing services
- memory allocation
- processor time allocation
In general, a monitor can have two distinct operating components: a permanently-resident portion and a transient portion. When a monitor is loaded into memory and started, all of the monitor is resident in memory. Its first duty is to interface with the operator running the system. The monitor simply waits until an operator requests some service, and then performs that service. In general, these services include loading and starting programs, controlling program execution, modifying or retrieving system information, and setting system parameters. In most systems, these functions are serviced by transient portions of the monitor.

In some cases, when the monitor initiates another program's execution, the transient portion of the monitor can be over-written by the loaded program or swapped out. The permanently-resident portion remains in memory to act on requests from the program. These generally include I/O services such as file management, device dependent operations, blocking and unblocking data, allocating storage space, and managing memory areas. In large systems, these services might also include inter-task communication and coordination, memory protection and parity checking, and task execution scheduling.

The dividing line between permanently-resident and transient portions of the monitor, however, is not strictly based on user-interface functions and program-interface functions. In some systems, special monitor routines that service either the operator or programs might be stored on the system device, and are called into memory only as needed. The concern for space in small systems usually determines what portions of the monitor are resident at any time. The programmer or operator can control the size of the monitor, based on the needs for memory.

In some cases, the user can adjust the size of the monitor by eliminating certain features that are not needed in an application environment. RSTS/E, RSX-11M and RSX-11S are examples of such systems. The RSX-11S system's monitor (called an executive) is always permanently-resident when the system is operating. In this case, the user concerned with size can eliminate routines that perform unneeded operations. In general, however, all PDP-11 operating systems are designed to be flexible enough to operate in a relatively wide range of hardware environments.

System utilities are the individual programs that are run under control of the monitor to perform useful system-level operations such as source program assembly or compilation, object program linking, and file management.

System utility programs enhance the capabilities of an operating system by providing users with commonly-performed general services. There are three classes of system utilities: those used solely or primarily for program development, those used for file management, and those used to perform special system management functions.

Program development utilities include text editors, assemblers and compilers, linkers, program librarians, and debuggers. File management utilities include file copy, transfer, and deletion programs, file format
translators, and media verification and clean-up programs. System management utilities vary from system to system, depending on the purpose and functions the system services. Some examples are system information programs, user accounting programs, and error logging and on-line diagnostic programs.

2.2 PROCESSING METHODS
The basic distinctions among operating systems are the processing methods they use to execute programs. The basic distinctions among processing methods to be discussed here are:

- single-user and multi-user
- single-job and foreground/background
- foreground/background and multi-programming
- timesharing and event-driven multi-programming

A single-user operating system views demands upon its resources as emanating from a single source. It has only to manage the resources based on these demands. As an effect, these systems do not require account numbers to access the system or data files. CAPS-11 and RT-11 systems are single-user operating systems.

A multi-user operating system receives demands for its resources from many different individuals. The system must manage its resources based on these demands. For example, several users may want sole access to the same device at the same time. The system must control access to these devices. In addition, the individuals may be using the system for different purposes, implying that some privacy be maintained. As an effect, a multi-user system normally has an account system to manage different user's files. The IAS, RSTS/E, RSX-11D, and RSX-11M systems are all multi-user systems, and all provide device allocation control and file accounts. In the case of the IAS, RSTS/E and RSX-11D systems, the file account structure is also used to keep track of the amounts of system resources an individual uses. Furthermore, the RSTS/E system extends privacy by protecting individual users at a system level from the effects of any other users of the system.

As well as being a single-user system, the CAPS-11 system is also a single-job system. Only one program is active at any time. The program executes until either it is completed or it is interrupted by the operator. This program has complete use of all the system's resources.

An RT-11 system can operate in two modes: as a single-job system, or as a foreground/background system. In a foreground/background system, memory for user programs is divided into two separate regions. The foreground region is occupied by a program requiring fast response to its demands and priority on all resources while it is processing; for example, a real-time application program. The background region is available for a low-priority, pre-emptable program; for example, a compiler.

Two independent programs, therefore, can reside in memory, one in the foreground region and one in the background region. The foreground program is given priority and executes until it relinquishes control to
the background program. The background program is allowed to execute until the foreground program again requires control. The two programs effectively share the resources of the system. When the foreground program is idle, the system does not go unused. Yet, when the foreground program requires service, it is immediately ready to execute. I/O operations are processed independently of the requesting job to ensure that the processor is used efficiently as well as enable fast response to all I/O interrupts.

The basis of foreground/background processing is the sharing of a system's resources between two tasks. An extension of foreground/background processing is multi-programming. In multi-programmed processing, many jobs, instead of only two, compete for the system's resources. While it is still true that only one program can have control of the CPU at a time, concurrent execution of several tasks is achieved because other system resources, particularly I/O device operations, can execute in parallel. While one task is waiting for an I/O operation to complete, for example, another task can have control of the CPU.

The RSX-11 family of operating systems employs multi-programmed processing based on a priority-ordered queue of programs demanding system resources. In this case, memory is divided into several regions called partitions, and all tasks loaded in the partitions can execute in parallel. Program execution, as in the RT-11 foreground/background system, is event-driven. That is, a program retains control of the CPU until it declares a significant event—normally meaning that it can no longer run, either because it has finished processing, or because it is waiting for another operation to occur. When a significant event is declared, the RSX-11 executive gives control of the CPU to the highest priority task ready to execute. Furthermore, a high-priority task can interrupt a lower-priority task if it requires immediate service.

The RSTS/E and MUMPS-11 systems also perform concurrent execution of many independent jobs. RSTS/E and MUMPS-11, however, process jobs on a timesharing rather than an event-driven basis, since this is best suited for an interactive processing environment. Each job is guaranteed a certain amount of CPU time (a time slice), and jobs receive time one after another, in a round-robin fashion based on job priority levels set by the system. The system itself manages timesharing processing to obtain the best overall response depending generally on whether jobs are compute-bound or I/O-bound. The system manager or privileged users can also specify the minimum guaranteed time for a particular job when it gets service, as well as modifying its priority.

The IAS system effectively combines event-driven and timeshared processing in order to handle both real-time processing needs and interactive timesharing needs. In IAS, I/O tasks and any user-designated real-time tasks are assigned high priorities and receive service on an event-driven basis. All other tasks run at lower priorities on a timeshared basis, using any CPU time remaining after real-time high-priority tasks have been serviced.

As an option, RSX-11M can also perform time-slice round-robin processing among tasks at a user-selected range of priority levels. RSX-11D
can perform, as an option, time-based swapping in a given partition to share the resources of the system equally among the tasks running in the partition.

2.3 DATA MANAGEMENT
Digital computers deal with binary information only. The way in which people interpret and manipulate the binary information is called data management.

This section describes PDP-11 software data management structures and techniques, from the physical storage and transfer level to the logical organization and processing level. This includes:

- ASCII and binary storage formats—how binary data can be interpreted
- physical and logical data structures—the difference between how data storage devices operate and how people use them
- file structures—how physical units of data are logically organized for ease of user reference
- file directories—how files are located and retrieved
- file protection—how files are protected from unauthorized users
- file naming conventions—how files are identified

2.3.1 Physical and Logical Units of Data
Physical units of data are the elements which digital computer devices use to store, transfer and retrieve binary information.

A bit (binary digit) is the smallest unit of data that computer systems handle. An example of a bit is the magnetic core used in some processor memories that is polarized in one direction to represent the binary number 0 and in the opposite direction to represent the binary number 1.

In PDP-11 computers, a byte is the smallest memory-addressable unit of data. A byte consists of eight binary bits. An ASCII character code can be stored in one byte. Two bytes comprise a 16-bit word. A word is the largest memory-addressable unit of data. Some machine instructions are stored in one word.

The smallest unit of data that an I/O peripheral device can transfer is called its physical record. The size of a physical record is usually fixed and depends on the type of device being referenced. For example, a card reader can read and transfer 80 bytes of information, stored on an 80-column punched card. The card reader’s physical record length is 80 bytes.

A block is the name for the physical record of a mass-storage device such as disk, DECTape or magnetic tape. An RK05 disk block consists of 512 contiguous bytes. Its physical record length is 512 bytes.

Physical blocks can be grouped into a collection called a device or a physical volume. This collection generally has a size equal to the capacity of the device media. The term physical volume is generally used with removable media, such as a disk pack or magnetic tape.

Logical units of data are the elements manipulated by people and user programs to store, transfer and retrieve information. The information
floating-point numeric data are some examples of data stored in binary format.

Figure 2-3 illustrates the way in which binary data can be interpreted as either ASCII data or machine instructions. The figure shows two examples of a word of storage containing the same sequence of bits, interpreted first as two ASCII characters and second as a machine instruction. When a word of storage is interpreted as two ASCII characters, the binary digits are grouped into octal digits in a byte-wise manner. Each byte is grouped into three octal digits. The low-order two octal digits contain three binary digits. The high-order octal digit contains two binary digits. When a word of storage is interpreted as a machine instruction, the binary digits are grouped into six octal digits in a word-wise manner. Proceeding from the low-order binary digit, each group of three binary digits is interpreted as an octal digit. The single remaining high-order binary digit is interpreted as an octal digit.
Figure 2-3 ASCII and Binary Storage
In large, sophisticated systems such as RSTS/E, RSX-11, and IAS, the way in which data is stored on the byte or bit level is rarely a concern of the application programmer. The operating system handles all data storage and transfer operations. In small systems such as CAPS-11 and RT-11, the programmer can become involved in data storage formats. A particular application may require the selection of a particular storage format.

The data storage format is related to the way in which data is transferred in an I/O operation. In the CAPS-11 system, the assembly-language programmer can select any of four data transfer modes: unformatted or formatted ASCII or binary. Formatting refers to the use of special characters or bits to convey information. For example, formatted ASCII transfers do not use the high-order bit of a byte; a line feed or form feed character is used to signal the end of the data to be transferred. Formatted binary is used to transfer checksummed 8-bit bytes in blocks. All eight bits of a byte are transferred and a checksum is calculated to verify that data was transferred correctly.

Formatting can also be applied at a higher level to define the type of data file being processed. In the RT-11 system, there are four types of binary files; each type signifies that a special interpretation applies to the kind of binary data stored. For example, a memory image file is an exact picture of what memory will look like when the file is loaded to be executed. A relocatable image file, however, is an executable program image whose instructions have been linked as if the base address were zero. When the file is loaded for execution, the system has to change all the instructions according to the offset from base address zero.

2.3.3 I/O Devices and Physical Data Access Characteristics

In a PDP-11 computer system, data moves from external storage devices into memory, from memory into the CPU registers, and back out again. The “window” from external devices to memory and the CPU is called the I/O page. Each external I/O device in a computing system has an external page address assigned to it. Figure 2-4 illustrates the data movement path in a PDP-11 computing system.

Although all external devices transmit and receive data through the UNIBUS, devices differ in their ability to store, retrieve or transfer data. Almost all PDP-11 operating systems provide device independence between devices that have similar characteristics and, where possible, between differing devices in situations where the data manipulation operations are functionally identical. Primarily, PDP-11 operating systems differentiate between:

- file-structured and non-file-structured devices
- block-replacable and non-block-replacable devices

Terminals, card readers, paper tape readers, paper tape punches and line printers are examples of devices that do not provide any means to selectively store or retrieve physical records. They can transfer data only in the sequence in which it physically occurs.
In contrast, mass-storage devices such as disk, DECTape, floppy disk, magnetic tape and cassette have the ability to store and retrieve physical records selectively. For example, an operating system can select a file from among many logical collections of data stored on the medium.

Mass-storage devices are called file-structured devices, since a file, consisting of a group of physical records, can be stored on and retrieved from the device. Terminals, card readers, paper tape readers/punches and line printers are called non-file-structured devices because they do not have the ability to selectively read or write the physical records comprising a file.

Finally, mass-storage devices differ in their ability to read and write physical records. Disk and DECTape devices are block-replacable devices, because a given block can be read or written without accessing or disturbing all the other blocks on the medium. Magnetic tape and cassette are not block-replacable devices.
A device's physical data access characteristics determine which data transfer methods are possible for that device. Non-file-structured devices allow sequential read or write operations only. Non-block replaceable devices allow sequential or random read operations, but allow sequential write operations only. Block-replaceable devices allow both sequential and random read or write operations. Figure 2.5 summarizes the read/write capabilities of each category of I/O device.

![Image of device characteristics]

---

2.3.4 Physical Device Characteristics and Logical Data Organizations

One of the most important services an operating system provides is the mapping of physical device characteristics into logical data organizations. Users do not have to write the software needed to handle input and output to all standard peripheral devices, since appropriate routines are supplied with the operating system.

2.11
There are generally two sets of routines provided in any operating system, depending on its complexity. They are:

- device drivers or handlers
- file management services

Device drivers and handlers can perform the following operations to relieve the user of the burden of I/O services, file management, overlapping I/O considerations and device dependence:

- drive I/O devices
- provide device independence
- block and unblock data records for devices, if necessary
- allocate or deallocate storage space on the device
- manage memory buffers

These routines may exist in the system as part of the monitor or executive, as in CAPS-11, RT-11, MUMPS-11, RSTS/E, RSX-11M or RSX-11S, or they may be provided as separate tasks, as in RSX-11D and IAS.

An operating system can also provide a uniform set of file management services. For example, the RT-11 system provides file management services through the part of the monitor called the User Service Routine (USR). The User Service Routine provides support for the RT-11 file structure. USR loads device handlers, opens files for read/write operations, and closes, deletes and renames files.

In summary, an operating system maps physical device characteristics into logical file organizations by providing routines to drive I/O devices and interface with user programs. Figure 2-6 illustrates the transition between the user interface routines and the I/O devices.

Figure 2-6 Device Control and File Management Services

2-12
As an example of the mapping of physical characteristics into logical organizations, the RSX-11 and IAS systems’ device drivers and handlers and file management services allow the user application program to treat all file-structured devices in the same manner. All of these devices appear to the user program to be organized into files consisting of consecutive 512-byte blocks which are numbered starting from block 1 of the file to the last block of the file. In reality, the blocks may be scattered over the device and, in some cases, the device’s actual physical record length may not be 512 bytes.

In RSX-11/IAS terminology, the actual physical records on the device (for example, the sectors on a disk) are called physical blocks. At the device driver or handler level, the system maps these physical blocks into logical blocks. Logical blocks are numbered in the same relative way that physical blocks are numbered, starting sequentially at block zero as the first block on the device to the last block on the device. At the user interface level, the operating system maps logical blocks into virtual blocks. Virtual block numbers become file relative values, while logical block numbers are volume relative values.

Figure 2-7 illustrates the mapping between physical, logical and virtual blocks in RSX-11 and IAS systems. The figure shows two disk device types which have different physical record lengths. In this case, the blocks comprising a file are scattered over the disk. The file is a total of 5 blocks long. At the logical block level, the operating system views the file as a set of non-contiguous blocks. At the virtual block level, the user software views the file as a set of contiguous, sequentially-numbered blocks.

![Diagram](image)

**Figure 2-7  Physical, Logical and Virtual Blocks**

### 2.3.5 File Structures and Access Methods

A file structure is a method of organizing logical records into files. It describes the relative physical locations of the blocks comprising a file. The file structure or structures a particular operating system employs is a product of the way in which the system views the particular I/O
devices and the kinds of data processing requirements the system fulfills.

File structure is important because a file can be effective in an application only if it meets specific requirements involving:

**SIZE**  
Growth of the file may require a change in the file structure or repositioning of the file.

**ACTIVITY**  
The need to access many different records in a file or frequently access the same file influences data retrieval efficiency.

**VOLATILITY**  
The number of additions or deletions made to a file may affect the access efficiency.

An access method is a set of rules for selecting logical records from a file. The simplest access method is sequential: each record is processed in the order in which it appears. Another common access method is direct access: any record can be named for the access. The non-block replaceable devices, such as paper tape and magnetic tape, can only be processed sequentially. The block replaceable devices, such as disk and DECtape, can be processed by either access method, but direct access takes fullest advantage of the device characteristics.

PDP-11 operating systems provide a variety of file structures and access methods appropriate to their processing services. All PDP-11 file structures are, however, based on some form of the following basic file structures:

<table>
<thead>
<tr>
<th>FILE STRUCTURE</th>
<th>ACCESS METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked</td>
<td>Sequential</td>
</tr>
<tr>
<td>Contiguous</td>
<td>Sequential or Direct Access</td>
</tr>
<tr>
<td>Mapped</td>
<td>Sequential or Direct Access</td>
</tr>
</tbody>
</table>

Linked files are self-expanding series of blocks which are not physically adjacent to one another on the device. The operating system records data blocks for a linked file by skipping several blocks between each recording. The system then has enough time to process one block while the medium moves to the next block to be used for recording. In order to connect the blocks together, each block contains a pointer to the next block of the file. Figure 2-8 shows the format of a linked file.

Linked files are especially suited for sequential processing where the final size of the file is not known. It readily allows later extension, since the user can add more blocks in the same way the file was created. In this way, linked files make efficient use of storage space. Linked files can also be easily joined together.

The blocks of contiguous files are physically adjacent on the recording medium. This format is especially suited for random (direct access) processing, since the order of the blocks is not relevant to the order in which the data is processed. The system can readily determine the physical location of a block without reference to any other blocks in the file. Figure 2-8 also shows the format of a contiguous file.
Mapped files are virtually contiguous files, that is, they appear to the user program to be directly-addressable sets of adjacent blocks. The files may not, however, actually occupy physically contiguous blocks on the device. The blocks can be scattered anywhere on the device. Separate information, called a file header block, is maintained to identify all the blocks comprising a file. This method provides an efficient use of storage space and allows files to be extended easily, while still maintaining a uniform program interface. Figure 2-9 illustrates a mapped file format.

If desired, a mapped file can be created as a contiguous file to ensure the fastest random accessing, in which case it is both virtually and physically contiguous.

The basic file structure discussed above can be modified or combined to extend the features of each type for special-purpose logical processing methods. Some examples are indexed files and global array files.

Indexed files are actually two contiguous files. One file acts as an ordered “map” of a second file containing the target data. The index
portion or "map" contains either an ordered list of key data selected from the target data records or pointers to data records in the second file, or both. The target data records can be processed in the order of the index portion, or the target data records can be selected by searching through the index portion for the key data identifying the records. These methods of logically processing the target data are called indexed sequential access and random access by key, respectively.

Global array files display a special form of linked file structure. The arrays themselves are a logical tree-structured organization consisting of one or more subscripted levels of elements. All elements on a particular subscripting level are stored in a single chain of linked blocks. At the end of each block in the chain is a pointer to the next block in the chain. The levels of the array (all the block chains) are linked together through pointers in the first block of each chain. This file structure ensures that the time it takes to access any element of the array is minimal.

2.3.6 Directories and Directory Access Techniques
Just as file structure and access methods are required to locate records
within files, directory structures and directory access techniques are required to locate files within volumes.

A directory is a system-maintained structure used to organize a volume into files. It allows the user to locate files without specifying the physical addresses of the files. It is a direct access method applied to the volume to locate files.

RT-11 supports the simplest kind of file directory. When disk and DEC-tape media are initialized for use, the system creates a directory on the device. Each time a file is created, an entry is made in the directory that identifies the name of the file, its location on the device and its length. When access to the file is requested thereafter, the system examines the directory to find out where the file is actually located. The system can access the file quickly without having to examine the entire device.

In multi-user systems such as RSTS/E, IAS, RSX-11D and RSX-11M, two different kinds of directories are used to enable the system to differentiate between files belonging to different users. They are the Master File Directory and the User File Directories. These directories are maintained as files themselves, stored on the volume for which they provide a directory.

A Master File Directory (MFD) is a directory file containing the names of all the possible users on of a particular device. A User File Directory (UFD) is a directory file containing the names of all the files created by a particular user on a device. The system first checks the Master File Directory to locate the User File Directory for the particular user, and then checks the User File Directory to locate the file. Figure 2-10 illustrates the use of the Master and User File Directories.

RSTS/E creates an MFD on each disk when it is initialized. On all disks except the system disk, the MFD catalogs other user accounts on the disk. The MFD on the system disk has a special purpose, since it maintains a catalog of the accounts under which users can log in, in addition to the user accounts on the disk. A UFD exists on each disk for each account under which files are created. A UFD for an account is not created until a file is created by the user under that account. DECTape devices are considered to be single-user devices, and the RSTS/E system maintains only a single directory on DECTapes.

The RSX-11D, RSX-11M, and IAS systems also employ MFD and UFD files on file-structured volumes. As with RSTS/E systems, the number of directory files required depends on the number of users of the volume. For single-user volumes, only an MFD is needed. For multiple-user volumes, an MFD and one UFD for each user are required. An MFD is automatically created when the volume is initialized for use. A UFD is created only by the system manager.

File access in RSX-11D, RSX-11M and IAS systems, however, is not limited to using the MFD and UFD files. The basis of file access using the MFD and UFD in these systems is a special file called the Index File. Like the MFD, an index file is created on each volume when it is ini-
Figure 2-10  Master and User File Directories

tialized. Files in these systems are mapped files, and the Index File contains the file header for each file stored on the volume, including the MFD. Each file is uniquely identified by a file ID. A file header contains the file’s ID and the physical location (logical record number) of each series of contiguous blocks comprising a file. By knowing a file’s ID and searching through the index file, a program can locate a file (and any block within the file) without having to use the MFD and UFD directories. Figure 2-11 illustrates how an Index File is used to access files on a volume.

All these operating systems also permit non-block replaceable media such as cassette and magnetic tape to be given a file structure. These media have no directory because a directory could not be updated and replaced. Instead, each file is preceded by one or more header records
which contain the directory information such as the file's name. The operating system can locate a file by scanning the volume and reading each file header until the correct one is found. The file can then be processed by a sequential access method.

2.3.7 File Protection
Master File and User File Directories form the basis for file access protection in multi-user systems. Unauthorized users can not access a file unless they know the account under which it is stored and can obtain access to that account. Account systems and file access protection techniques are related.

Multi-user systems identify the individuals who use the system by account numbers called User Identification Codes (UIC). The system manager normally gives a user an account number under which the user can log in to the system and obtain access to its services. In general, a UIC consists of two numbers: the first number is used to identify a group of users, the second number is used to uniquely identify an individual user in the group.

In RSTS/E systems, an individual file can be protected against read access or write access where distinctions are made on the basis of the UIC account number under which a file is stored. For example, a file can be read protected against all users who are not in the same account group and write protected against all users except the owner.

The RSX-11/IAS file system provides a protection scheme for both volumes and files. It is possible to specify protection attributes for an entire volume as well as for the files in the volume. A file or an entire volume can be read, write, extend or delete protected. Distinctions are made on the basis of account number, where the system recognizes four groups of users: privileged system users, owner, owner's group, and all others.
2.3.8 File Naming

The most common way users communicate their desire to process data is through file specifications. A file specification uniquely identifies and locates any logical collection of data which is on-line to a computer system.

A compiler, for example, needs to know the name and location of the source program file that it is to compile; it also needs to know the name that the user wants to use for the output object program and listing files it produces. Most PDP-11 operating systems share the same basic format for input and output file specifications.

In the CAPS-11 and RT-11 systems, a file specification consists of the name of the device on which the file resides, a filename, and a filename extension in the following format:

```
dev:filnam.ext
```

The colon is part of the device name, separating it from the filename on the right. The period is part of the filename extension, separating it from the filename on the left.

PDP-11 operating systems use the same device names for the devices they can access. A device name consists of a 2-letter mnemonic and, for multiple devices of the same kind, a one-digit number indicating the device unit number. For example, the name "DK1:" is used to identify the RK11 disk drive unit number 1. The name "DPO:" identifies the RP11 disk drive unit number 0.

In the CAPS-11 and RT-11 systems, a filename is a 1- to 6-character alphanumeric name designated by the user. For example, "SYMBOL", "RL12", and "NORT4" are examples of filenames. In the RSX-11D and RSX-11M systems, a filename can be up to nine characters long.

A filename extension is a 1- to 3-character alphanumeric name preceded by a period. The extension can either be assigned by the user or, if unspecified, assigned by the system. The extension generally indicates the format of a file. System-assigned and recognized extensions make it easy for the user and the system to distinguish between different forms of a file. For example, if a file having the extension ".FOR" is recognized by the FORTRAN compiler as a source program written in FORTRAN. A file with the extension ".OBJ" is recognized by the Linker as an object program, a legal input file. When in the process of compiling and linking a FORTRAN program, the user has only to specify a filename to the compiler and Linker. The FORTRAN compiler will compile the file whose extension is ".FOR" and produce a file with the same filename whose extension is ".OBJ". The Linker will link the file whose extension is ".OBJ".

In multi-user systems such as RSTS/E, RSX-11D and RSX-11M, a distinction must be made between files stored under various accounts on a device. Two different users can have a file named "REFER.OBJ" on a disk. In these systems, therefore, a file specification has an additional component to identify the user file directory or account under which the
file is stored. The basic file specification is expanded to use the fol-
lowing format:

    dev:[ufd]filnam.ext

The account number or user file directory is always enclosed in brackets. It consists of the project or group number followed by a comma and a programmer or user number. For example, "[12,4]" is an example of an account or user file director.

RSTS/E systems also include a protection code as part of the file speci-
fication, to indicate the protection that the file receives. A complete RSTS/E file specification could be:

    DK1:[200,210]BINFOR.DAT<60>

RSX-11 systems extend the basic file specification format by adding a version number identification after the filename extension. For example, when a file is first created using the Editor, it is assigned a version number of 1. If the file is subsequently opened for editing, the Editor keeps the first version for backup and creates a new file using the same file specification, but with a version number of 2. A complete RSX-11 file specification could be:

    DPO:[15,7]PREPT.MAC;1

In most cases, the user does not have to issue a complete file specifica-
tion. The PDP-11 operating systems use default values when a portion of a file specification is not supplied. The filename extension defaults, as indicated previously, depend on the kind of operation being performed.

The device name, if omitted, is normally assumed to be the system device. For example, the file specification "FILE.DAT" is equivalent to the specification "DK0:FILE.DAT", if the system device is RK11 drive unit 0. Most systems also allow the user to omit the unit number. If omitted, the unit number is assumed to be unit number 0. For example, "DT:" is equivalent to "DTO:"; it signifies DECTape drive unit 0.

If the account number is omitted from the file specification, the system assumes that it is the same as the UIC under which the user logged in or under which the operation is being performed. For example, if the user logged in under UIC 200,200 and issues a file specification "DK3:SAMPL.DAT", it is interpreted as "DK3:[200,200]SAMPL.DAT".

If the version number is omitted from an RSX-11/IAS file specification, the system assumes that the file specification refers to the latest version of the file.

For references to file-structured devices, a file specification must include a filename. The device mnemonics, however, are also used to refer to non-file-structured devices. In this case, a filename is irrelevant. For example, an operation to read through a deck of cards and print the information on a line printer is issued in most systems as follows:

    #LP: < CR:
The "\#" indicates that an input/output command is being issued; it is printed on the terminal by the program that requests the I/O command. The user types the command "LP: < CR:". The "<" separates the input file specification on the right from the output file specification on the left. The device name "LP:" signifies that the line printer is to be used as the output device, and the device name "CR:" signifies that the card reader is to be used as the input device. A filename, if used, would be ignored, since the system can not symbolically reference data on non-file-structured devices.

In addition to relying on defaults in the file specification, the user can also put an asterisk in place of a filename, filename extension, account number or version number to indicate a class of files. The asterisk convention, also called the wildcard convention, is commonly used in PDP-11 operating systems when performing the same operation on related files. For example, the file specification "DP1:[2,1]PROG." refers to all files on DP1: under account [2,1] with a filename PROG and any extension. The file specification "DK:[][FILE.SAV" refers to the files under all accounts on RK11 drive unit 0 named FILE.SAV. The file specification "DT:.[OBJ" refers to all files on the DECtape mounted on drive unit 0 that have the extension .OBJ.

2.4 USER INTERFACES
A user interface refers to both the software that passes information between an operator and a system and the "language" that a system and an operator use to communicate. In the latter sense, a user interface consists of commands and messages. Commands are the instructions that the user types on a terminal keyboard (or gives to a batch processor) to tell the system what to do. Messages are the text that a system prints on a terminal (or line printer) that tells the operator what is going on, for example, prompting messages, announcements and error messages. This section discusses commands, the portion of the user interface that tells the system what to do, and prompting messages, the messages the system prints when it is ready to receive commands.

There are basically four types of commands used in PDP-11 operating systems:

- monitor or command language commands—used to request services from the system as a whole
- I/O commands—used to direct any kind of I/O operation (often a part of monitor commands)
- special terminal commands—these use keys on a terminal for special functions
- system program commands—commands used in system programs that perform operations relevant only for the individual program

Since system program commands are relevant only for individual system programs, and not operating systems in general, this section discusses monitor and command language commands, I/O commands and special terminal commands only.

2.4.1 Special Terminal Commands
Special terminal commands are a set of keys or key combinations that,
when typed on a terminal, are used to perform special functions. For example, a user normally types the carriage return key at the end of an input command string to send the command to the system, which responds immediately by performing a carriage return and line feed on the terminal. The key labelled RUBOUT or DELETE is used to delete the last character typed on the input line.

The most significant special terminal commands are those used with the key labelled CTRL (control). When the CTRL key is held down (like the shift key) and another key is typed, a control character is sent to the system to indicate that an operation is to be performed.

For example, a line currently being entered (whether as part of a command or as text) will be ignored by the system by typing a CTRL/U combination (produced by holding down the CTRL key and typing a U key). The user can then enter a new input line. The CTRL/U function is the same as typing successive RUBOUT keys to the beginning of a line. CTRL/U is standard on PDP-11 operating systems.

Another example is the CTRL/O function. If, during the printing of a long message or a listing on the terminal, the user types a CTRL/O, the teleprinter output will stop. The program printing the output, however, will still continue. The user can type a CTRL/O again to resume output. CTRL/O is a standard function on PDP-11 operating systems.

2.4.2 I/O Commands
As mentioned in section 2.3.7, users communicate their intentions to process data files by issuing I/O commands consisting of at least one file specification. Normally, the I/O commands used in a system are standard throughout that system; in addition, most PDP-11 operating systems share the same basic I/O command string format.

For example, in CAPS-11 and RT-11 systems, the monitor includes a Command String Interpreter routine that parses and validates I/O command strings. The Command String Interpreter routine is used both by the monitor and the system programs to obtain a definition from the user of the input file or files to be processed and a definition of the output file or files to be created. User-written programs can also call the Command String Interpreter to obtain I/O specifications from the operator at a terminal.

A standard I/O command string consists basically of one or more input and/or output file specifications. In all systems except IAS, an I/O command string uses the following general format:

filespec=filespec

where "filespec" is a file specification (see section 2.3.7) and the equal sign (=) represents a character (usually equal sign or less-than sign) that separates an input file specification on the right from an output file specification on the left. If there is more than one input file specification or output file specification, they are separated from each other by commas. For example, if there are two output file specifications and three input file specifications:

filespec,filespec=filespec,filespec,filespec
If the program requesting an I/O command string does not need either an input or output file specification, the equal sign (or less than sign) is not present in the I/O command string.

As an example, the user can run the RT-11 operating system’s Linker system utility to link one or more object program files and produce an executable program file and a load map. The I/O command issued to the Linker could be:

```
*DK:RESTOR.SAV,DK1:RESTOR.MAP=DK:RESTOR.OBJ/B:500
```

Where:

* Is the prompting character printed by the Linker program indicating that it wants an I/O command string. After it is printed, the user types the remaining characters on the line.

DK:RESTOR.SAV  Is the name of the executable program file to be created. It will be stored on the disk cartridge mounted on the RK11 drive unit zero.

DK1:RESTOR.MAP  Is the name of the load map file to be created. It will be stored on the disk mounted on RK11 drive unit 1.

DK:RESTOR.OBJ  Is the name of the object module (input file) to be used to create RESTOR.SAV.

/B:500  Is a command string switch indicating that the RESTOR.SAV program is to be linked with its starting address at location 500.

Command string switches are simply ways of appending qualifying information to an I/O command string. The switches used vary from program to program. They are not usually required in an I/O command string, since most programs assume default values for any switch.

### 2.4.3 Monitor and Command Language Commands

The primary system/user interface is provided in PDP-11 operating systems by either monitor software or special command language interface programs that run under the monitor. The monitor software and command languages allow the user to request the system to set system parameters, load and run programs, and control program execution.

An input command line consists of the command name (an English word that describes the operation to be performed) followed by a space and a command argument. For example, the command to run a program is the word "RUN" followed by the name of the file containing the program. If the command name is long, it can usually be abbreviated. For example, the command to set the system’s date to August 27, 1975 could be "DATE 27-AUG-75." The system could also accept "DA 27-AUG-75." A command input line is normally terminated by typing the carriage return key on the console keyboard, although in some systems the key labelled ALTMODE is also used. Typing the carriage return key (or ALTMODE key) tells the system that the command line is ready to be processed.
In the CAPS-11 system, a monitor component called the Keyboard Listener prints a period (.) on the left column of the system’s console to indicate that the monitor is ready to accept commands. In the RT-11 system, a monitor component called the Keyboard Monitor performs the same operation. The user enters a command string on the same line following the period, and terminates the command string by typing the carriage return key.

In the RSTS/E system, the monitor and the BASIC-PLUS language processor share the responsibility for interpreting commands. The system prints the word “READY” on the terminal and then spaces down two lines. The user then enters a command on the new line and terminates the line by typing the carriage return key. There are three types of commands the user can issue: RSTS/E monitor commands, such as RUN, ASSIGN, or RENAME; BASIC-PLUS immediate mode statements, such as PRINT, INPUT or OPEN; or Concise Command Language commands.

A Concise Command Language (CCL) command is used to automatically run and pass arguments to designated programs stored in the system library. The programs can be system utilities supplied with the operating system, or can be user-written console routine programs that perform special application operations. For example, RSTS/E includes a system utility called PIP that performs a variety of file manipulation operations, including a file copy operation. The dialogue normally used to run the PIP utility and issue a copy command is:

```
READY
RUN $PIP
PIP Vnnn
*FILEB=FILEA
*↑C
```

READY

The system prints READY.

The system prints READY.
The user runs PIP.
PIP announces itself.
PIP prints an asterisk to request an I/O command and the user issues a copy command.
PIP prints an asterisk, indicating that the operation was performed and that it is ready to accept another command; the user types a CTRL/C to abort PIP and return to the monitor.

READY

The system prints READY.

The standard RSTS/E system also includes a CCL command named PIP that can be issued to perform any of PIP’s normal functions. If used as a CCL command, the dialogue to perform the same copy operation is:

```
READY
PIP FILEB=FILEA
```

READY

The user issues the CCL command and the argument that tells PIP to copy FILEA to FILEB.

READY

The system prints READY.

A CCL command not only provides an easy-to-use command interface, it can also provide protection from unauthorized use of certain programs. For example, if a particular program performs several operations, some of which should not be available to unauthorized users, the sys-
tem manager can prevent those users from issuing the RUN command to run the program, but can allow them to perform safe operations by using CCL commands.

In the RSX-11 systems, a command interface called the Monitor Console Routine (MCR) allows the user to perform system level operations. When MCR is activated, it prints the characters "MCR>" on the terminal. The user enters a command on the same line as the prompt, and terminates the line with a carriage return or an ALTMODE. If the line is terminated with a carriage return, MCR prints a prompt and is ready to receive another command. If the line is terminated with an ALTMODE, MCR does not reactivate. To reactivate MCR at a terminal, the user types a CTRL/C.

There are two kinds of commands that MCR accepts: general user commands and privileged user commands. General user commands provide system information, run programs, and mount and dismount devices. Privileged user commands control system operation and set system parameters.

To run a system utility, the user can type the utility's name in response to an MCR prompt. When the utility is loaded, it prints a prompt to request a command string. The user can then enter a command string. When it completes the operation, the user can enter another command or type CTRL/Z to terminate the program. For example, to run the PIP utility program:

\[
\text{MCR> PIP} \\
\text{PIP> command string} \\
\text{PIP> \uparrow Z} \\
\text{MCR>}
\]

If the user wants to issue only one command to the utility, the user can type the command string on the same line with the MCR request to run the utility. For example:

\[
\text{MCR> PIP command string} \\
\text{MCR>}
\]

In the IAS system, system/user interfaces are provided by programs called Command Language Interpreters (CLI). The standard system includes a CLI called the Program Development System. When it is activated, it prints the prompt "PDS>" on the terminal to indicate it is ready to accept and process commands. The user has several options for command string formats. If the user is uncertain about a command's syntax, the user can simply type the command name and a carriage return. PDS will ask the user to supply each portion of the command string individually. Users can write their own Command Language Interpreters.

2.5 PROGRAMMED SYSTEM SERVICES

All PDP-11 operating systems provide access to their services through requests that programs or tasks can issue during execution.
In the CAPS-11 system, the assembly language programmer can use the monitor's I/O routines by issuing a PDP-11 IOT (Input/Output Trap) instruction. Each I/O routine is associated with a command code that the monitor recognizes. When the program issues an IOT instruction, it specifies the code of the routine that the monitor is to execute. After performing the requested operation, the monitor continues the program's execution from the point where it passed control to the monitor.

The RT-11 system provides a variety of programmed requests. There are programmed requests that perform file manipulation, data transfer and other system services such as loading device handlers, setting a mark time for asynchronous routines, suspending a program, and calling the Command String Interpreter. Monitor services are requested through macro instructions in assembly language programs, or through calls to the system library in FORTRAN programs. The basis of the programmed requests in RT-11 are the EMT (Emulator Trap) instructions. When an EMT is executed, control is passed to the monitor, which extracts appropriate information from the EMT instruction and executes the operation requested. When the operation is performed, the monitor returns control to the program.

In the RSTS/E system, users writing BASIC-PLUS programs have access to the monitor's services through system function calls. The function calls allow a program to control terminal operation, to read and write core common strings, and to issue calls to the system file processor. The file processor calls enable a program to: set program run priority and privileges, scan a file specification, assign devices, set terminal characteristics, and perform directory operations. A system function is called in a manner similar to normal BASIC-PLUS language calls. When the function operation is performed, the program continues execution.

The RSX-11 and IAS executives include programmed services called executive directives. Directives can be executed in MACRO programs using system macro calls provided with the system. The FORTRAN programmer can invoke directives through a subroutine call. The system uses only the EMT 377 instruction to implement directives. The directives allow the program to obtain system information, control task execution, declare significant events, and perform I/O operations. After the directive is processed, control is normally returned to the instruction following the EMT.

The RSX-11D, RSX-11M and IAS systems also include programmed file control services. The file control services enable the programmer to perform record-oriented and block-oriented I/O operations. These services are provided as macro calls.

The IAS system includes a special set of programmed services called Timesharing Control Primitives. These are available for use by any program that is written as a Command Language Interpreter (CLI). They enable a CLI to start or control execution of other timesharing tasks, and share access to devices with other timesharing users.
2.6 SYSTEM UTILITIES

PDP-11 operating systems provide, in general, three kinds of system utility programs: program development utilities, file management utilities, and special system management utilities.

Most PDP-11 operating systems include the following kinds of program development utilities:

**Text Editor**
An editor is used for on-line interactive creating and editing of source programs or data files. An editor uses several sets of commands that search for character strings, insert, move or delete characters or lines, and insert, move, delete or append whole buffers of data. Although a text editor is designed for interactive use, it can also usually be run under a batch processor if the operating system supports batch processing.

**Assembler**
An assembler accepts a source program written in PDP-11 symbolic machine language and produces an object module as output.

**Linker**
A linker is a program that accepts relocatable object programs created by an assembler or compiler and produces an executable program module. Some linkers provide facilities for overlaid program segments to enable a large program to execute in a small memory area.

**Librarian**
A librarian is a program that enables a programmer to create, update, modify, list and maintain library files. A library file is an object module (or modules) that is used several times in a program, used by more than one program, or routines that are related and simply gathered together to incorporate easily into a program.

**Debugger**
A debugger is a program which enables a user to troubleshoot program errors dynamically through a terminal keyboard. It is normally linked with a program and runs as part of the program.

Some of the file management utilities available on many operating systems include:

**PIP**
The Peripheral Interchange Program (PIP) is a general-purpose file utility package for both the general user and programmer and the system manager. PIP normally handles all files with the operating system's standard data formats. In general, the program transfers data files from any device in the system to any other device in the system. PIP can also delete or rename any existing file. Some operating systems include special file management operations in the PIP utility, such as directory listings, device initialization and formatting, and account creation.

**FILEX**
The File Exchange program is a special-purpose file transfer utility similar in operation to PIP. It provides the
ability to copy files stored in one kind of format to another format. This enables a user to create data on one system in a special format and then transfer the data to a device in a format that another system can read.

DUMP

DUMP displays all or selected portions of a file on a terminal or line printer. In general, DUMP enables the user to inspect the file in any of three modes: ASCII, byte, and octal. In ASCII mode, the contents of each byte is printed as an ASCII character. In byte mode, the contents of each byte is printed as an octal value. In octal mode, the contents of each word is printed as an octal value.

VERIFY

In general, a VERIFY program checks the readability and validity of data on a file-structured device.

Most system management utilities included in an operating system are dependent on the function the operating system serves. The RSX-11D, RSX-11M, IAS and RSTS/E systems provide special system management utilities. For example, RSX-11M and RSTS/E include system error logging and report programs. RSTS/E, IAS and RSX-11D include user accounting programs.
CHAPTER 3

LANGUAGE PROCESSORS

3.1 LANGUAGE TRANSLATION SYSTEMS
A programming language is a system of symbols and syntax which can be used to describe a procedure that a computer can execute. A language processor is a program that translates one programming language into another. A language processor reads a program written in a language easily understood by people and translates it into a program written in the binary language of a digital computer. The program which the processor reads is called the source program. The program which the processor writes is called the object program.

Assemblers
An assembler is a language processor written for a particular digital computer. The source language it translates is called assembly language. There is a one-to-one correspondence between most of the mnemonics used as the assembly language operators and the binary instructions of the computer. Some exceptions are macro calls and assembler directives.

During the language translation process, an assembler performs a number of error checking operations. When an error is detected, the assembler notes the error and attempts to continue processing. At the end of processing, the assembler produces an error listing showing all the occurrences of errors, with substantial messages to the programmer. In addition to an error listing, the programmer can obtain an assembly listing in any of several formats and a symbol table listing. In addition, some assemblers can provide a Cross Reference listing for all symbols used in the program.

Most assemblers produce an object program by making one or more passes over the source program (reading the original source code several times). The resultant object program is in relocatable binary format. That is, the first instruction appears to be located in the first word of processor memory. Since in most cases the program is not to be loaded into the bottom of memory, the object program must be linked to the proper memory addresses before it can be executed.

The linking program is provided as a standard program development utility with an operating system. Figure 3-1 illustrates the fundamental steps in producing an executable program from assembly source code.

Compilers
A compiler is a language processor written to translate a higher-level language whose structure, syntax and symbols are independent of any particular machine. The higher-level language operators most often do not correspond directly to binary instructions. It is the compiler's job to provide algorithms for their translation.
Figure 3.1 Fundamental Assembly or Compilation Procedure

Most compilers do not translate the source code until the entire source program is read at least once. The translation of the source code into object code takes place during several passes over the source code or, if only one pass over the original source code is made, during several phases of the compilation process. This allows the compiler to examine the code it produces as a whole to eliminate unnecessary instructions (code optimization). In addition, the compiler can perform many levels of error checking and it can produce several kinds of compilation listings, including source code listings, code generation listings, and diagnostics.

An incremental compiler (also called an interpreter) is a compiler that immediately translates source statements into an internal format. Each source statement is translated (and therefore can be executed) before the following statement is translated. Although this method of source translation does not enable possible object code optimization, it allows the compiler to provide program development services not possible in multi-pass or multi-phase compilers. For example, a syntax error detected in a source statement can be reported to the programmer immediately, and the programmer can correct the statement before proceeding.

One significant difference between a general compiler and an incremental compiler are the characteristics of the resulting object program. The object code produced by the former type of compiler requires a separate step of linking before it can be executed, as shown in Figure 3.1. This approach enables the programmer to combine several object programs into one executable program. This provides several advantages:

Modularity
A source program may be too large to be compiled successfully as a single unit but, if divided into modular sections, can be compiled as
several separate units. The separate sections can be combined at the object level to produce the resultant program. In addition, programs that are extremely complex can be divided into several sections so that they can be easily manipulated, debugged or modified. A change in one module of the program will only require recompilation of that section.

Assembly Language Routines
The compiler’s object code can be combined with the object code produced by the operating system’s assembler. Algorithms which are most easily written in assembly language, such as user-defined I/O processing, can be incorporated into a program written primarily in a higher-level language.

Library Routines
Libraries of commonly-used routines and functions written in either assembly or the higher-level language can be maintained in object format. These routines can be selectively included in the resultant program by the linking utility. This not only eliminates repetitive source coding and associated errors, it also decreases the size of the source and object programs.

The object code produced by an incremental compiler does not require an intermediate step of linking before it can be executed. The incremental compiler actually serves two purposes: it translates the source code into object code and it provides the environment in which to execute the object code. That is, the steps of source code translation, linking and execution are all provided by the translator. Figure 3-2 illustrates this type of translator operation.

![Diagram showing the process of translation and linking](image)

**Figure 3-2** Fundamental Incremental Compiler Operation

Program Development Facilities
A complete language translation system requires facilities for creating and editing source programs, to link object programs into executable programs, and to debug programs. Most PDP-11 operating systems provide an editor utility for source program creation and editing, and a librarian utility for library file creation. Operating systems also provide a linker utility to link and combine object modules produced by multipass compilers and assemblers. Finally, operating systems also include debugging utilities.

Some of these facilities may or may not be incorporated into the language translator program itself. For example, an incremental compiler may include an editing facility as part of the language translation code.
This allows the programmer to edit the program interactively as it is being compiled and executed.

**Libraries and Object Time Systems**

Also included in most language translation systems is a library of the most commonly-used functions and routines. The system library is generally a part of the language processor’s Object Time System (OTS).

A multi-pass or multi-phase compiler does not usually generate all of the machine language code required by the program at run-time. Common sequences of code required by the program can be maintained in the OTS file. The compiler then flags the places where the desired sequences are needed. The linker utility, during its pass over the object program, selects those sequences from the OTS file and incorporates them into the executable program module.

An incremental compiler may also have an OTS. In this case, however, the OTS is generally part of the run-time code of the translator. When the object code is executed by the incremental compiler’s run-time code, the OTS is used to provide common library code sequences.

### 3.2 PDP-11 ASSEMBLERS AND THE FORTRAN COMPILERS

With three exceptions, all the operating systems described in this handbook include the MACRO assembly language. RSTS/E supports MACRO only in conjunction with the FORTRAN IV language option. The CAPS-11 operating system supports only the PAL-11 assembly language. MUMPS-11 does not support an assembly language processor.

Two FORTRAN IV compilers are available: FORTRAN IV and FORTRAN IV-PLUS. FORTRAN IV is available on all the operating systems described in this handbook except CAPS-11 and MUMPS-11. FORTRAN IV-PLUS is available on the RSX-11 and IAS operating systems.

The MACRO assembler, FORTRAN IV compiler and FORTRAN IV-PLUS compiler display the same external operating characteristics. In general, they accept source code from any valid input device and produce an object file on any valid file-structured device. If the input device is a file-structured device, the assembler or compiler can accept several source files. If desired, an assembly or compilation listing can also be produced as output, either as a file or on a line printer or terminal. MACRO can also generate a symbol table listing if requested. RT-11 MACRO (and MACRO available with the RSTS/E FORTRAN IV option) can generate a Cross Reference Listing (CREF) if desired.

As shown in Figure 3-3, there are several methods for creating sources. A source program can be punched on cards if a card reader is available, or it can, in some cases, be entered directly on the terminal. The common method is to create a file on a file-structured device. The file can be created from a deck of punched cards, using the PIP file transfer utility to copy it onto disk or DECTape. The file can also be created on a terminal, using the operating system’s editor utility to store it on disk or DECaPe.

In addition to source program files, the MACRO assembler accepts source library files as input. The operating system provides a system library for
MACRO containing the macro definitions for the system’s monitor calls or executive directives. The assembler selects those macro definitions required by the source program from the system library file.

In RSX-11 and IAS systems, the MACRO assembler can also accept a user-created macro library as input. The sources for the user-defined macro libraries are created in the same manner as normal source programs. The operating system’s librarian utility program is used to create the library files. Figure 3-4 illustrates this procedure.
Once the assembler or compiler produces an object file, the object file can be linked by the linker utility. The linker can accept several object files as input. In addition, when linking object files produced a FORTRAN compiler, the linker accepts the FORTRAN system object library for the given compiler as input. The linker automatically selects the required routines from the library.

Users can also create their own object library files. The source code is created in the same manner as normal source programs. The librarian utility is used to build the library file. Figure 3-5 illustrates the procedure.

PDP-11 assemblers and compilers differ in their internal operation. The MACRO assembler is a two-pass assembler. It makes a first pass over the source input to collect the symbol references, expand macros and produce preliminary object code. A second pass is made to resolve symbol references and produce the completed object code and listings.

The FORTRAN IV compiler is a multiple-phase compiler. Instead of making multiple passes over the source program, it reads the source program once and manipulates the source code in memory. The compiler operates in multiple phases. An overlay is read into memory for
each phase of the compilation process. This method enables the compiler to compile relatively large programs very quickly.

The FORTRAN IV-PLUS compiler is a multiple-pass compiler. It reads the source program several times, using a work file to build the object code. The work file is deleted when the compilation process is complete. Figure 3-6 illustrates the compilation methods of the two FORTRAN compilers.

The FORTRAN IV-PLUS compiler’s work file not only makes it possible to compile relatively large programs, but it also allows the compiler to thoroughly examine the object code it produces. The compiler refines the code it produced during the initial compilation to ensure that the program, when executed, will run as fast as possible. Therefore, although the disk accesses required to read and write a work file decrease the speed of the compiler, the work file enables the FORTRAN IV-PLUS compiler to produce highly optimized code.

3.3 INCREMENTAL COMPILERS

Four languages available in the PDP-11 software systems described in this handbook are processed by incremental compilers. The languages are:

- COBOL—the PDP-11 COBOL compiler is available as an option for the RSTS/E, RSX-11D and IAS operating systems.
MULTI-PHASE COMPILATION

SOURCE FILE  

MEMORY AREA  

SPACE FOR SOURCE MANIPULATION  

OBJECT FILE  

COMPILER FILE  

COMPILER OVERLAYS  

MULTI-PASS COMPILATION

SOURCE FILE  

MEMORY AREA  

BUFFERS FOR SOURCE MANIPULATION  

COMPILER CODE  

OBJECT FILE  

WORK FILE  

Figure 3-6 Compilation Methods

• BASIC—there are three BASIC language compilers available. Single-user BASIC is available on the CAPS-11,RT-11,IAS and PTS-11 systems (PTS-11 is not described in this handbook). Multi-user BASIC is available for the RT-11 operating systems. BASIC-PLUS is the primary language processor for the RSTS/E operating system.

• MUMPS—the MUMPS language processor is the sole language processor available on the MUMPS-11 operating system.

• FOCAL—the FOCAL language compiler is available for the RT-11 operating systems and the PTS-11 systems (FOCAL/PTS is not described in this handbook).

The external operating characteristics of these language processors vary slightly. The COBOL compiler can accept source input from cards, from the terminal, or from a file created using cards or an interactive editor. Figure 3-7 illustrates the COBOL compiler's external operating characteristics.

The BASIC language processors can accept source input from a terminal or from a file generated using an editor utility, as illustrated in Figure 3-8. The most common method of creating a source program is by giving the source statements directly to the compiler through an interactive
terminal. For this reason, the BASIC language processors include an editing facility. This allows the programmer to create, test, and modify the source program interactively.

The MUMPS and FOCAL languages are very similar in structure and syntax. Their language processors are also similar. Both include interactive editing facilities. The FOCAL language processor can accept source input either from a file created using the RT-11 editor or directly from the terminal. The MUMPS language processor can accept source input directly from the terminal or from a file previously created using the MUMPS language processor.
SECTION 2
pdp-11
operating systems

CHAPTER 1  CASSETTE PROGRAMMING SYSTEM CAPS-11

CHAPTER 2  FOREGROUND/BACKGROUND SYSTEM RT-11

CHAPTER 3  RESOURCE-SHARING TIME-SHARING SYSTEM RSTS/E

CHAPTER 4  MULTI-USER DATA BASE MANAGEMENT SYSTEM MUMPS-11

CHAPTER 5  REAL-TIME MULTIPROGRAMMING FAMILY RSX-11

CHAPTER 6  INTERACTIVE APPLICATION SYSTEM IAS
CHAPTER 1

CASSETTE PROGRAMMING SYSTEM CAPS-11

1.1 OPERATING SYSTEM FUNCTIONS AND FEATURES
CAPS-11 is a small, cassette-based program development system with true operating system characteristics. CAPS-11 includes a Monitor, an editor, assembler, linker, on-line debugging module, and file interchange utility. As an option, CAPS-11 supports a BASIC language processor, BASIC/CAPS.

CAPS-11 runs on a PDP-11 with between 8K and 28K words of memory. In addition, CAPS-11 requires an operator’s console and a TA11-TU60 dual cassette drive and control. It supports an optional line printer and high-speed paper tape reader/punch. The paper tape reader/punch is used to transfer files from cassette to the punch or from the reader to cassette only. Under the BASIC/CAPS language, the user can access graphic display and laboratory data peripherals.

The CAPS-11 Monitor is compatible with the IOX executive of the PTS-11 paper tape software, but uses magnetic tape cassettes for program and data storage. The Monitor includes a keyboard listener, command string interpreter, cassette loader and resident monitor. The keyboard listener allows the operator to request the Monitor to load and start user programs, run system programs, set the date, etc. The command string interpreter decodes the operator’s commands, including input and output file specifications, and passes the information to system or user programs. The cassette loader brings binary format programs into memory from cassette and starts their execution. The resident monitor performs all cassette, console and line printer input and output operations. All devices are interrupt-driven.

CAPS-11 allows the user to develop programs that use as many or as few of the Monitor’s facilities as desired. The fewer portions of the Monitor used, the more space available for the user’s program. In particular, CAPS-11 allows the user to create and load stand-alone programs. The user can develop a stand-alone application under the CAPS-11 program development system and then create a cassette on which the Monitor’s cassette loader and the user’s stand-alone application program are stored. The program can then be bootstrapped into memory just as the Monitor. In this case, all but approximately 400 words of memory are available to the stand-alone program.

CAPS-11 is upward compatible with the PTS-11 paper tape software. Programs written under PTS-11 can be converted to operate under CAPS-11. An optional CAPS-11 I/O mode which uses IOX compatible buffer structures simplifies converting programs that use IOX paper tape
executive commands for I/O. DIGITAL provides explicit routines for handling cassette I/O to aid in converting programs that handle their own I/O.

Table 1-1 summarizes the facilities of CAPS-11.

**Table 1-1  CAPS-11 System Summary**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System type</td>
<td>Single-user, interactive program development</td>
</tr>
<tr>
<td>CPU’s supported</td>
<td>PDP-11/04/05/10, PDP-11/35/40, PDP-11/45 (no memory management or parity support)</td>
</tr>
<tr>
<td>Memory size range</td>
<td>Minimum: 8K words  Maximum: 28K words</td>
</tr>
<tr>
<td>Additional CPU hardware supported</td>
<td>PDP-11/04/05/10 Extended Arithmetic Element (EAE)*</td>
</tr>
<tr>
<td></td>
<td>PDP-11/35/40 Extended Instruction Set (EIS)*</td>
</tr>
<tr>
<td></td>
<td>PDP-11/45 Floating-point processor (FPP)*</td>
</tr>
<tr>
<td>Minimum peripherals</td>
<td>Terminal and TA11/TU60 dual cassette drive and control</td>
</tr>
<tr>
<td>Additional peripherals</td>
<td>LS11 or LP11 line printer</td>
</tr>
<tr>
<td></td>
<td>PC11 paper tape reader/punch (under PIP program control)</td>
</tr>
<tr>
<td></td>
<td>VT11-A Graphics Display Processor*</td>
</tr>
<tr>
<td></td>
<td>LPS11 Laboratory Peripheral System*</td>
</tr>
<tr>
<td></td>
<td>AR11 Analog Real-time Subsystem*</td>
</tr>
<tr>
<td>Monitor size</td>
<td>2-3K words</td>
</tr>
<tr>
<td>Maximum Program Load Size</td>
<td>With all Monitor facilities: 25.2K words</td>
</tr>
<tr>
<td></td>
<td>With Monitor I/O facilities only: 26.0K words</td>
</tr>
<tr>
<td></td>
<td>With no Monitor facilities: 27.5K words</td>
</tr>
<tr>
<td>System Programs</td>
<td>EDIT interactive editor</td>
</tr>
<tr>
<td></td>
<td>PAL-11S assembler</td>
</tr>
<tr>
<td></td>
<td>LINK linker</td>
</tr>
<tr>
<td></td>
<td>ODT on-line debugger</td>
</tr>
<tr>
<td></td>
<td>PIP file utility</td>
</tr>
<tr>
<td>High-level Languages</td>
<td>Single-user BASIC (optional)</td>
</tr>
<tr>
<td>File system</td>
<td>DIGITAL standard labelled, contiguous cassette files</td>
</tr>
</tbody>
</table>

### 1.2 MONITOR ORGANIZATION

The CAPS-11 Monitor is composed of several subprograms that are responsible for various system and user interaction functions. To use the CAPS-11 Monitor, the user loads all of the Monitor’s components into memory and starts their execution. The user begins the loading process by starting the cassette bootstrap loader (CBOOT). When the Monitor has been completely loaded and is ready for use, it resides in memory as shown in Figure 1-1.

---

*Under BASIC language processor only.*
1.2.1 Monitor Components

**Cassette Bootstrap (CBOOT)**
The Cassette Bootstrap is used to load and start any program which is in "CBOOT Loader Format", such as the Monitor's loader (CTLOAD.SYS).

**Resident Monitor (RESMON)**
RESMON handles input and output operations. It contains routines for all file-structured cassette I/O, and all console and line printer I/O. All RESMON's routines are available to the user's program, if desired. Usually RESMON is not overwritten, but is always available in memory during a program's execution. The user who does not want its services, however, can load a program over RESMON to use the additional memory area.

RESMON also uses and maintains the System Communication Area (SYSCOM), which provides system and user programs with information concerning available memory and locations of important Monitor routines.

**Cassette Loader for CAPS-11 (CLOD11)**
CLOD11 is used to direct the loading of programs in memory for the RUN, LOAD and LOAD/G Monitor commands (see section 1.3.3). For example, in the case of the RUN command, CLOD11 allows the user to
enter an I/O command string while the program is being loaded. CLOD11 also handles error detection and reporting.

**Command String Interpreter (CSI)**
The Command String Interpreter (CSI) is used by all system programs except EDIT and ODT and can be used by any user program which is loaded and started by the RUN command. When the user runs a program, the CSI responds by printing an asterisk (*) at the left margin of the console terminal page. The user responds by entering all device and file I/O information needed by the program. The CSI then constructs a table which contains the information needed by the program.

**Cassette Absolute Loader (CABLDR)**
CABLDR is generally used to load stand-alone programs which do not use RESMON's I/O facilities. CABLDR loads programs written in absolute binary format (the format of all system programs and all programs created using the linker). CABLDR performs error detection and halts when an error occurs to allow the user to examine and correct the condition using the CPU switch register.

**Keyboard Listener (KBL)**
The Keyboard Listener indicates that the Monitor is ready to accept a Monitor command by printing the dot or period (.) at the left margin of the console. The KBL is also responsible for positioning the cassette tape for loading during a RUN, LOAD, LOAD/G, or LOAD/O command. It then passes control to CLOD11 or CABLDR (the latter for the LOAD/O command) which handle the actual loading during command processing.

**System Communication (SYSCOM)**
The System Communications Area (SYSCOM) resides in absolute locations 40 through 57 and is loaded into memory as part of the RESMON source code. This area provides a means of communication between the Monitor and other programs not linked with it, such as system and user programs.

SYSCOM locations used by the CAPS-11 system and of particular interest to the user are described below.

- **HIFREE**
  Contains the address of the highest location available to the user for program loading and storage which precedes the "expected" portion of the Monitor still residing in memory. At run time, HIFREE tells a program how much memory can be used without destroying the portions of the Monitor that are still resident.

- **DATPTR**
  Contains the address in RESMON of the current date as entered by the operator issuing the Monitor DATE command.

- **HLTERR**
  This is examined by the Monitor's cassette interrupt handler each time there is a cassette error. If this byte has been set by the user, the interrupt routine halts so that the user can examine the cassette status register. Pressing the CONTINUE key on the processor console resumes program execution. This byte is used primarily as a hardware debugging aid.

- **KBLRES**
  This is a flag indicating the state of the non-resident portion of the Monitor. When a program is loaded into memory, this
location is set to indicate that the entire Monitor may not be intact, and that it should be reloaded from cassette when the program terminates. If the program does not over-write the Monitor, however, it can set this location to indicate that the entire Monitor is intact. When the program terminates, the Monitor is simply restarted. For example, the system programs PIP, EDIT, LINK and ODT do not require the additional memory space made available when they are loaded and started. These programs each set KBLRES so that a fatal error or CTRL/C do not cause a reboot of the Monitor.

KBLADR Contains the starting address of the KBL (the lowest address of the Monitor). If KBLRES is non-zero when a fatal error occurs or a CTRL/C is issued, the Monitor is restarted at this address. The user can compare the address in KBLADR with a program’s use of memory to determine whether it can clear KBLRES to allow a simple restart of the Monitor.

FILWRD Contains information used by RESMON and ODT to handle different console terminals. Some terminals, such as the serial LA30 DECwriter and VT05 DECScope, require that certain characters be followed by a number of “fill” characters. The low-order byte of FILWRD contains the character which requires “fill” and the high-order byte contains the number of “fill” characters required. RESMON and ODT will type this number of nulls (ASCII 000) after the character defined in FILWRD.

1.2.2 User Program Loading
The CAPS-11 Monitor attempts to provide the user at all times with the maximum loading space and maximum storage space for system and user programs. It does this by allowing unneeded portions of the Monitor to be overwritten, and by moving necessary sections to higher positions in memory.

The user has four different methods available to load and execute a program. Each method differs in the amount of free memory it offers and the Monitor services available. The four methods described below are listed in order from least free memory and all Monitor services to most free memory and no Monitor services. The LINK program uses a default absolute location of 600 for the lowest program address, assuming that the program uses locations 400-600 for stack space. The user can slightly alter the amount of memory made available for a program using any of the load methods described below by specifying a different low address.

METHOD 1
The user can load a program up to the base address of the Monitor (start location of the KBL). None of the Monitor services are lost. If a fatal error occurs or the operator issues a CTRL/C to terminate the program, the Monitor is automatically restarted. This method is normally used during program development to test or debug a program. Program loading and execution is done using two commands: LOAD and START.
METHOD 2
The user can load a program over the KBL and CABLDR and, if necessary, over most of the CSI. In addition, the program has several options for increasing memory area available for run-time data storage. This method provides the most flexibility for memory use and still retains the Monitor's CSI and RESMON services. Normally, if the user chooses this method to load and execute a program, the Monitor must be rebooted after execution terminates. If, however, the program does not use the memory area above the start address of the KBL, the program can clear the KBLRES location in SYSCOM and the Monitor is simply restarted when the program terminates. Program loading and execution is done by issuing one command: RUN.

On receiving the command, the KBL initiates the load by positioning the cassette on which the program is stored. When this is done, the KBL and CABLDR are no longer needed and can be overwritten by CLOD11 as it loads the program. While the load is in progress, the operator can enter I/O specifications to the CSI. The CSI builds a table which contains all the I/O information entered. Initially, the CSI reserves 300 octal bytes for the table, but once the table is built and its size is known, it is moved to occupy memory just below CLOD11. HIFREE is then set to the bottom of the CSI table and CLOD11 allows the program to load up to this address. Figure 1-2 shows the memory map after this stage of program loading is reached. Note that if a program is large enough that it loads over the CSI, it can use the CSI services only once.

When loading is complete, CLOD11 is no longer needed. To free its memory locations for the user program, the CSI table is normally moved up over CLOD11 just below RESMON, as shown in Figure 1-3. The free memory locations can be used by the program for run-time data storage. The CSI table is not moved over CLOD11, however, if the program does not use memory above the start of the KBL and notifies the Monitor by clearing the KBLRES location in SYSCOM.

METHOD 3
The user can load a program up to the start address of CLOD11, and use the CLOD11 memory area after the load for run-time data storage. RESMON is preserved to handle I/O within the user's program. However, the rest of the Monitor is not preserved and no Monitor Keyboard commands are possible. Figure 1-4 illustrates the memory map after load. Loading and execution are requested with either of two commands: LOAD/G or RUN with a special filename extension.

METHOD 4
The user can load a program with maximum memory available for loading and storage. The KBL positions the cassette for the load and then moves the CABLDR to high memory just below CBOOT. CABLDR directs program loading. Loading is allowed up to the start address of CABLDR, and after loading the entire memory is available for storage. Since no part of the Monitor is preserved, the user program must handle its own

*The 300 octal (192 decimal) bytes originally reserved for the CSI table is a parameter which can be changed by the user during reassembly of the CAPS-11 Monitor.
Figure 1-2 Memory Map After Loading with CSI Services

Figure 1-3 Memory Map After CSI is Moved
I/O, and no further Monitor commands or functions are available for use. Figure 1-5 illustrates the memory map after load. Loading and execution are requested with either of two commands: LOAD/O or RUN with a special filename extension.

**Figure 1-4 Memory Map for RESMON Services Only**

### 1.3 SYSTEM CONVENTIONS

The following paragraphs discuss six aspects of the CAPS-11 system input/output conventions: I/O devices and device names, cassette files and file formats, file naming, Monitor commands, I/O command specifications, and special keyboard commands.

#### 1.3.1 I/O Devices

There are four categories of input/output device in the CAPS-11 system: these are the console terminal keyboard and printer, cassette drives 0 and 1, an optional line printer, and an optional high-speed paper tape reader and punch (which may be used only by PIP to copy paper tape data to cassette or vice versa). Each device is referenced by means of a standard permanent device name which is recognized by the CAPS-11 system when encountered in an I/O command string.

#### 1.3.2 File Naming

System and user cassette files are referenced symbolically by a name of as many as six alphanumeric characters followed by a period and an
optional extension. Most system programs assume certain default extensions. The CAPS-11 default extensions are listed below:

.BAS  BASIC language processor program file
.DAT  BASIC language processor data file
.LDA  Linker binary output load module
.LST  Assembler listing output file
.MAP  Linker load map output
.OBJ  Relocatable binary object module (Assembler output, Linker input)
.PAL  Assembler source file (Editor input and output, Assembler input)

NOTE
The next three extensions are default extensions for the Monitor RUN command.

.SLO  Absolute binary object file (default extension for RUN command, causing an automatic load and overlay of the Monitor as necessary up to CABLDR)
.SLG  Absolute binary object file (default extension for RUN command, causing an automatic Load and Go)
.SRU    Absolute binary object file (normal default extension for the RUN command)
.SYS    CAPS-11 system file (i.e., CTLOAD.SYS, CAPS11.SYS; the extension is reserved for these two files)

1.3.3 Monitor/Operator Console Commands
There are eight Monitor commands which can be typed in response to the dot printed by the Keyboard Listener; they are entered when the RETURN key is pressed. Monitor commands generally require only a single command line which specifies the device, filename(s), and switch(es) in the following format:

.COMMAND/SW DEV:FILENA.EXT

COMMAND represents one of the eight Monitor commands. SW represents a switch—an alphabetic character separated from the command and from another switch character by a slash (/). The device (DEV), if specified, is always a cassette; the user can enter just the drive number rather than the entire permanent device name. With the exception of the ZERO command, drive 0 is always assumed, so the user can omit the device specification entirely if CT0: is the device. FILENA.EXT represents the file being accessed; the filename must be separated from the drive number (if indicated) by a colon.

The CAPS-11 Monitor commands are listed below:

DATE    Allows the operator to enter the date to appear on link maps, assembler and directory listings and to be entered in the cassette file header as the file creation date.

DIRECTORY  Lists the directory of a cassette on the operator's console.

DIRECTORY/F  Lists a brief directory of a cassette by omitting the current and file creation dates.

LOAD    Instructs the Monitor to load a program into memory, but not to overlay the Monitor's keyboard listener (KBL). When the program is loaded into memory, the KBL is ready to accept another command, usually either START or RUN for ODT.

LOAD/G    Instructs the Monitor to load a program into memory and start its execution at its transfer address. The Monitor’s KBL, CABLDR and CSI are assumed to have been overwritten when the program was loaded. The Monitor’s RESMON is not overwritten and the loaded program can use its I/O routines.

LOAD/O    Instructs the Monitor to load a program into memory and start its execution at its transfer address. The loaded program cannot use RESMON’s I/O routines, since it has been overwritten.

RUN     Instructs the Monitor to load a program into memory and start its execution at its transfer address. If the program's filename extension is neither .SLG nor .SLO,
the program can be loaded over the KBL, CABLDR, and most of the CSI. The program can use the CSI and receive I/O specifications entered by the operator. If the program’s filename extension is .SLG, the operation is the same as the LOAD/G command’s. If the program’s filename extension is .SLO, the operation is the same as the LOAD/O command’s.

SENTINEL Allows the operator to delete all of the files on a cassette after the file specified in the command string. The Monitor writes a sentinel file (logical end of cassette marker) after the specified file.

START Instructs the Monitor to start executing a program loaded into memory at a specified address or its transfer address.

VERSION Requests the Monitor to print its version number on the operator’s console.

ZERO Instructs the Monitor to delete all files on a cassette and write a sentinel file at the beginning of the tape.

1.3.4 Entering I/O Specifications through the CSI
The Command String Interpreter (CSI) allows the user to enter command strings which provide necessary information concerning input and output files and devices, file formats to be used in I/O operations, and any other important information needed for the I/O process. The CSI prints an asterisk (*) at the left margin of the console terminal page as soon as it is ready to accept this information.

The CAPS-11 system programs, PAL, LINK, and PIP use the CSI routines to obtain I/O specifications from the operator. In addition, a user program can use the CSI in the same manner. The CSI scans the entered command string and builds a table containing the information. The command string can be entered while the program is being loaded.

The command string which the user enters in response to the asterisk contains all input and output specifications in the following general format:

\*DEV:OUTPUT.EXT/OPT = DEV:INPUT.EXT/OPT

where:

- DEV:OUTPUT.EXT represents an output file or device specification
- DEV:INPUT.EXT represents an input file or device specification
- /OPT represents an option switch

Input and output file or device specifications, if both exist in a single command string, are separated by an equals sign (=), a less-than sign (<), or a back-arrow (→). If there are two or more input or output specifications in a command string, they are separated by a comma.

There can be several options associated with an I/O specification. Options are separated from the rest of the command line and from one
another by a slash (/), and are indicated in the command string only when the user wants the associated action to occur. Option usage varies according to the program being used. Options include specifying high, low and transfer addresses to the LINK program, selecting delete, zero or ASCII options for the PIP program, etc. Of special interest to the CAPS-11 user are the following three options:

/F Forward—indicates that the cassette does not have to be rewound before searching for the specified file. That is, the specified file follows the last file accessed on the cassette. For example, if the user runs the LINK program and wants to link together three object programs named A, B, and C, each of which are stored on the cassette mounted on unit 0 in that order, the command string might be:

```
G1:OUTPUT = 0:A, B/F, C/F
```

The /F switches suppress rewinds on cassette unit 0.

/P Prompt—requests that the system prompt the operator to change cassettes on an indicated drive before attempting to access a file. The system prints “#?” on the console, where # represents the number of the appropriate drive (0 or 1). The user can specify several files in a command string that are stored on different cassettes and the system will prompt the user when a new cassette must be mounted.

/S Several—used after a LINK program input filename to indicate that the indicated file contains more than one input object module (where several object modules can be combined under one filename using PIP). This allows the user to combine files to simplify LINK instructions.

### 1.3.5 Special Console Commands

The CAPS-11 system allows the operator to control system execution and command correction using keyboard control characters. The control characters that CAPS-11 supports are:

- **CTRL/C**: Reloads or restarts the Monitor. Terminates execution of any program in progress. If the Monitor's KBL is not resident, CTRL/C prompts the operator to mount the system cassette and then reboots the Monitor. If the Monitor's KBL is resident, CTRL/C simply restarts the KBL.

- **CTRL/O**: Enables or disables console output, except important error messages.

- **CTRL/P**: Restarts the KBL or CSI when issued just after entering a command or command string that the user wants to change. If issued during execution of a user program, restarts the program if it has specified a restart address: otherwise, CTRL/P restarts the Monitor.

- **CTRL/U**: Instructs the system to ignore the line currently typed at the console; allows the operator to enter a new line.

### 1.3.6 Cassette File and Data Formats

A cassette consists of a sequence of one or more files, separated from each other by a single file gap. The first file on the cassette must be
preceded by a file gap; the last file must be followed by a file gap and a sentinel file or by clear trailer. Figure 1-6 illustrates the cassette tape format.

Each file consists of a sequence of a header record plus zero or more data records separated from each other by record gaps. The first record of a file is called the file header record, or file label. The header record is 32 (decimal) bytes long.

![Cassette Tape Format](image)

**Figure 1-6 Cassette Tape Format**

The header record contains the file's name, indicates the file's type, specifies the record length, and gives the file's creation date. Files can be stored in ASCII format or binary format.

Files in ASCII format conform to the American National Standard Code for Information Interchange in which alphanumeric characters are represented by an 8-bit code. Parity is not required and CAPS-11 ignores the high-order bit of ASCII data. Files in ASCII format are generally those created using the Editor or the BASIC language processor.

Files in binary format consist of 8-bit bytes represented data and PDP-11 machine language code. Binary files contain addresses and machine instructions and can be read directly into memory for immediate execution. System programs and object programs that the user creates using the Assembler and Linker are in binary format.

Record length is fixed at 128 bytes. Approximately 600 records can be stored on a cassette tape. The user can estimate the number of records remaining on a tape before beginning a file creation or transfer by examining the cassette tape window. Each mark on the bottom of the window represents about 50 inches of tape. Approximately two records fit on an inch of tape; each mark, therefore, represents about 100 data records.

The file creation date consists of six 7-bit ASCII digits specifying the day number (01-31), the month number (01-12), and the last two digits of the year, in that order.

The last file on a cassette tape is called the sentinel file. This file consists of only a 32 (decimal) byte header record and represents the logical end-of-tape (CAPS-11 also recognizes clear trailer as logical end-
of-tape). A sentinel file is identified by a null character (ASCII=000) as the first name character in the header record. A zeroed or blank cassette tape is one consisting of only the sentinel file.

1.4 MONITOR I/O PROGRAMMING SERVICES

The majority of I/O in the CAPS-11 system is done using RESMON, the part of the Monitor which contains routines to handle all file-structured cassette I/O and teleprinter, keyboard and line printer input and output. In addition, RESMON allows the user to access the basic routines necessary for doing cassette I/O. The user can, therefore, develop applications which use a non-standard cassette file structure.

RESMON is brought into memory by bootstrapping the system or by typing a CTRL/C while a CAPS-11 system program is running. RESMON loads the following interrupt and trap vectors: console terminal keyboard and printer, cassette, timeout, breakpoint, illegal memory reference, stack overflow, power fail, EMT, TRAP and IOT. The RESMON I/O handlers remain in memory unless the user does an overlay load (LOAD/O).

Simple I/O requests can be made by specifying devices and data forms for interrupt-controlled data transfers. These requests can be occurring concurrently with the execution of a running user program; multiple I/O devices may be running single or double buffered I/O processing simultaneously.

Communication with RESMON is accomplished by IOT (Input/Output Trap) instructions in the user's program. Each IOT is followed by two words consisting of one of the RESMON commands and its operands.

There are two types of RESMON commands: commands that establish the conditions for performing input and output, and commands that transfer data. When transfer of data is occurring, RESMON is operating at the priority level of the device. The calling program runs at its own priority level, either concurrently with the data transfer, or sequentially. Table 1-2 lists the RESMON IOT commands.

<table>
<thead>
<tr>
<th>Table 1-2 RESMON Monitor IOT's</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INITIALIZATION IOT's</strong></td>
</tr>
<tr>
<td><strong>RESET</strong></td>
</tr>
<tr>
<td><strong>RESTART</strong></td>
</tr>
<tr>
<td><strong>CNTRLO</strong></td>
</tr>
<tr>
<td><strong>CASSETTE FILE I/O INITIALIZATION IOT's</strong></td>
</tr>
<tr>
<td><strong>SEEK</strong></td>
</tr>
</tbody>
</table>

1-14
subsequent READ IOT’s by setting up input buffers. This is a cassette file lookup.

SEEKF  Opens a cassette file for input in the same manner as a seek, but does not issue a rewind before searching for the file.

ENTER  Opens a cassette file for output, writes a file header for the new file at the logical end of the specified cassette and prepares RESMON for subsequent WRITE IOT's by setting up output buffers. The user can safeguard against data loss by specifying a user-written overflow routine to continue on another cassette if the WRITE processor encounters the end of the cassette before the file is closed.

CLOSE  Specifies that a file currently open for output is to be closed and not referenced further. Frees the cassette unit so that it can be opened again with a SEEK, SEEKF or ENTER.

DATA TRANSFER IOT’s

READ  Reads data from a device and places it in a buffer. RESMON initiates the read and returns control to the user program. When the program needs the data, it issues a WAITR.

WRITE  Writes data to a device from a buffer. RESMON returns control to the calling program as soon as the transfer is initiated. The program issues a WAITR if it needs to know that the transfer is done before continuing.

WAITR  Tests the status of a device to determine if all initiated transfer for that device have been completed.

I/O devices in the CAPS-11 system are fixed. RESMON associates a specific code with each of the five possible devices: cassette drive 0, cassette drive 1, console printer, console keyboard, and line printer.

The operands for data transfer commands are a device code and an address. Addresses or codes can be specified symbolically. In non-data-transfer commands where an address or device code does not apply, the device code is set to zero; the address is ignored and can be any number.

The address operand for data transfer commands specifies the buffer for the data transfer. Most CAPS-11 I/O processing is record-oriented, so data transfer buffers generally include a buffer header. This type of record buffer is used not only to store data for processing, but is also used to hold information regarding the quantity, form and status of the data. Unformatted cassette I/O is block-oriented rather than record-oriented, so its buffers consist only of data and do not include a header.

The buffer header for record I/O specifies the buffer size, the mode of transfer for READs or WRITEs, and the byte count for the transfer. The user can select any of four transfer modes:

Formatted ASCII  READ transfers 7-bit characters (bit 8 is zero) until a line feed or form feed is encountered. RESMON sets the byte count to indicate the number of characters in the buffer. WRITE trans-
fers the number of 7-bit characters specified by the byte count. Bit 8 is always output as zero.

Unformatted ASCII

READ and WRITE transfer the number of 7-bit characters specified in the byte count.

Formatted binary

READ and WRITE transfer 8-bit characters to and from the device and buffer exactly as they appear. Formatted binary is used to transfer checksummed binary data in blocks. On WRITE, RESMON calculates a checksum and transfers it at the end of the data. On READ, RESMON calculates a new checksum and compares it with the checksum following the data block.

Unformatted binary

READ and WRITE transfer the number of 8-bit characters specified in the byte count. No formatting or character conversions occur.

The distinction between formatted and unformatted cassette I/O is made at the time a cassette file is opened for input or output (using SEEK, SEEKF or ENTER). The mode specified at that time governs the way subsequent READs or WRITEs are interpreted for the opened file.

For FORMATTED cassette READs or WRITEs (record I/O), all buffers are assumed to include a buffer header. The user defines an intermediate block buffer to be used by RESMON for record unblocking. The user can optionally define an additional block buffer and RESMON will automatically double buffer cassette I/O. This minimizes the time the program must wait for physical I/O transfers.

If the user specifies UNFORMATTED I/O to or from cassette, the buffer pointer in subsequent READ or WRITE IOT commands is assumed to point to a 128-byte block buffer. During an unformatted READ from cassette, a 128-byte data block is read directly into the buffer indicated. During an unformatted WRITE to cassette, 128 bytes of data are taken directly from the buffer indicated and transferred to cassette.

Users can implement their own double buffering scheme by using unformatted cassette I/O, since the location of cassette buffers is not fixed at SEEK, SEEKF or ENTER time, but can be varied with every READ or WRITE command simply by changing the buffer pointer in the command.

A user program handling cassette I/O can include special techniques to manage multiple cassettes in a similar manner to the CAPS-11 system. The commands SEEK, SEEKF, and ENTER have an additional feature which can aid the user who has his files on several cassettes. If, on entry to these commands, the Status/Error byte in the IOT parameter block is equal to 377 (octal) RESMON will prompt the user to mount a new cassette on the specified unit prior to executing the command. RESMON will type:

```
#?
```

where "#" is the unit number on which RESMON expects a new cassette to be mounted. RESMON then waits for the user to type any character on the keyboard. When the user has done this, RESMON as-
sumes that the proper cassette has been mounted and initiates the command. System programs that allow the operator to append a /P (prompt) option to a command string I/O specification use this feature of RESMON.

In addition, if the user program is performing formatted cassette WRITEs, it can specify an argument to the ENTER command to safeguard against data loss if there is not enough room remaining on the cassette to complete the WRITE. The user provides the name of an overflow subroutine to be called in case the WRITE processor encounters the end of cassette before the file is CLOSED. If an overflow subroutine was specified when the file was ENTERed, the WRITE processor will call it via a JSR PC, SUBR. The user’s subroutine tells the operator to mount a new cassette on the same drive that the file which overflowed was mounted on. It then ENTERs a file on that new cassette (using the same internal buffers as the original ENTER command) and returns to RESMON’s WRITE processor via an RTS PC. The WRITE processor will continue writing onto the new file; the two files should then be combined with PIP before being used further.

If users want to implement their own cassette data structures, RESMON allows the user to call the basic cassette I/O functions that RESMON itself uses to perform standard cassette I/O. These functions allow the user program to read and write data with any block size, write a file gap, space forward or reverse to a file or block, and rewind the tape. These functions are initiated by IOT calls; RESMON handles all the physical I/O, and all the I/O is interrupt driven.

1.5 SYSTEM PROGRAMS
The CAPS-11 operating system includes an interactive editor, assembler, linker, on-line debugger and file interchange utility. These programs are loaded and executed by issuing the Monitor’s RUN command when the Monitor is loaded into memory and the system cassette is mounted.

EDITOR
The CAPS-11 editor, EDIT, is an interactive, character-oriented text editor compatible with the RT-11 editor. Controlled by commands from the console, EDIT reads ASCII files from cassette, makes specified changes, and writes ASCII files back to cassette. To create a file, the operator simply opens a cassette file for output and enters the text from the console.

The Editor operates in one of two modes: Command Mode or Text Mode. In Command Mode all input typed on the console is interpreted as commands instructing the editor to perform some operation. In Text Mode all typed input is interpreted as data to replace, be inserted into, or be appended to text currently in an ASCII file.

There are five types of commands: input and output commands, text pointer relocation commands, text search commands, text modification commands, and utility commands.

Input commands are used to read text into a text buffer where it then becomes available for editing or listing. Output commands cause text to be written out to a cassette or be listed on the console terminal or
line printer. If an output cassette becomes full during any output operations, the Editor prompts the operator to mount another cassette. After the operator mounts the new cassette the output operation continues. The files can then later be combined under one filename using PIP.

Pointer relocation commands and text modification commands include both character commands for character-level operations and line-oriented commands for operations suited to lines.

Command features of particular interest are iteration brackets, which allow repetition of portions of the command as often as desired, command macros, which allow specification of a series of operations to be used over and over, and the ability to save and move blocks of text.

ASSEMBLER
The CAPS-11 Assembler is a two pass assembler (with an optional third pass) which allows the user to create a binary object file from a source program. In the first two passes, the source program (which is generated on-line using the Editor) is translated into an object module which can contain both absolute and relocatable code. Separately assembled object modules can reference one another using special symbols called global symbols. Object modules are then processed by the Linker, producing a load module which may be loaded into memory and executed. The Assembler produces a complete octal/symbolic listing of the assembled program during the second pass or, if it is to be stored on cassette during a third pass.

The CAPS-11 Assembler is a modified version of PAL-11S, the PTS-11 assembler. Its features are:

- Error listing on the console terminal
- Double-buffered and concurrent I/O
- Alphabetized, formatted symbol table listing
- Relocatable object modules
- Global symbols for linking between object modules
- Conditional assembly directives
- Program sectioning directives

LINKER
The CAPS-11 Linker converts object modules produced by the Assembler into a format suitable for loading and execution. This allows the user to assemble a large program in several small subprograms or to separately assemble a main program and each of its subroutines without assigning an absolute load address at assembly time. Object modules are processed by the Linker to:

1. Relocate each object module and assign absolute addresses.
2. Link the modules by correlating global symbols defined in one module and referenced in another module.
3. Print a load map which displays the assigned absolute addresses.
4. Output a load module which can subsequently be loaded and executed.
The CAPS-11 Linker is upward compatible from the PTS-11 Linker, LINK-11S.

Advantages of using the CAPS-11 Assembler and the Linker include the following:

1. A program is divided into segments (usually subroutines) which are assembled separately. If an error is discovered in one segment, only that segment need be edited and reassembled. The new object module is then linked with the other object module.

2. Absolute addresses need not be assigned at assembly time as the Linker automatically assigns absolute addresses. This keeps programs from overlapping each other and also allows subroutines to change size without influencing the contents of other routines.

3. Separate assemblies allow the total number of symbols to exceed the number allowed in a single assembly.

4. Since global symbols are usually referenced from more than one object module, the programmer must be sure that his names for such symbols are unique between object modules. However, this does not apply when the symbol is internal; since an internal symbol is referenced only from within the current assembly, the same symbol names may be used in several different modules.

5. Subroutines may be provided for general use in object module form to be linked into the user's program.

ON-LINE DEBUGGER

ODT (On-line Debugging Techniques) aids the user in debugging assembled and linked object programs. Using the console terminal keyboard, the user interacts with ODT and his object program to:

- Print the contents of any location for examination or alteration
- Run all or any portion of the object program using the breakpoint feature
- Search the object program for specific bit patterns
- Search the object program for words which reference a specific word
- Calculate offsets for relative addresses
- Fill a block of words or bytes with a designated value

ODT is upward compatible with the PTS-11 on-line debugger, ODT-11R. ODT is supplied as a relocatable object module and as an absolute binary program. The latter program is linked to be loaded just under the KBL; it is normally used by loading the user program into memory with the LOAD command and then running ODT. This is the most common form of ODT use, as it is expected that user programs will start in low memory and that the standard location of ODT will suffice. The user can, however, relink ODT using the CAPS-11 Linker, or link ODT along with the user program.

FILE INTERCHANGE UTILITY PROGRAM

The Peripheral Interchange Program (PIP) allows the user to copy cassette files, transfer cassette files to console terminal, line printer, or high-speed paper tape punch. PIP also allows the user to transfer paper
tapes to cassette files. These files can then be accessed by CAPS-11 system programs.

1.6 BASIC LANGUAGE PROCESSOR
As an option, CAPS-11 supports a BASIC language processor which is a single-user subset of the RT-11 MU BASIC language processor.

BASIC/CAPS provides sequential file capabilities and allows the user to save and retrieve files from cassettes. BASIC/CAPS allows user-defined functions, user-written assembly language routines, and chaining between BASIC programs with data passed in cassette files and/or in memory.

The BASIC/CAPS system is provided on three cassettes. Two running versions are provided on one cassette. These may be loaded by either the bootstrap loader or the CAPS-11 Monitor. One version is for PDP-11 systems with more than 8K of memory (referred to as the “greater than 8K version”) and the other will run in any PDP-11 with at most 8K (referred to as the “8K version”) of memory. The 8K version employs overlaying of the BASIC system from cassette and does not support string variables or the PRINT USING statement. The greater than 8K version is a non-overlaying BASIC and supports string variables and the PRINT USING statement.

BASIC-CAPS runs on any PDP-11 system with 8K or more of memory, a dual drive cassette and a console terminal. BASIC-CAPS also supports the following optional hardware: a line printer (LP11 or LS11), a high-speed paper tape reader/punch (PC11), a low-speed paper tape reader/punch (LT33), special arithmetic hardware (EAE, FPU, PDP-11/40 Extended Processor, and PDP-11/45 Processor), and up to 28K of memory.

For systems with more than 8K of memory, support is also provided for the Laboratory Peripheral System (LPS) and the GT40 display processor (VT11). Support is provided in the form of subroutines that may be linked with BASIC.

When BASIC/CAPS is first loaded into memory, the operator has the option of deleting certain BASIC language functions. Selectively excluding functions provides space for additional user program lines.

The optional functions are:

<table>
<thead>
<tr>
<th>String BASIC (Greater than 8K)</th>
<th>No String (8K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RND</td>
<td>RND</td>
</tr>
<tr>
<td>ABS</td>
<td>ABS</td>
</tr>
<tr>
<td>SGN</td>
<td>SGN</td>
</tr>
<tr>
<td>BIN</td>
<td>BIN</td>
</tr>
<tr>
<td>OCT</td>
<td>OCT</td>
</tr>
<tr>
<td>TAB</td>
<td></td>
</tr>
<tr>
<td>LEN</td>
<td></td>
</tr>
<tr>
<td>ASC</td>
<td></td>
</tr>
<tr>
<td>CHR$</td>
<td></td>
</tr>
<tr>
<td>POS</td>
<td></td>
</tr>
<tr>
<td>SEG$</td>
<td></td>
</tr>
<tr>
<td>VAL</td>
<td></td>
</tr>
<tr>
<td>TRM$</td>
<td></td>
</tr>
<tr>
<td>STR$</td>
<td></td>
</tr>
</tbody>
</table>

1-20
Each exclusion of a function other than POS or SEG$ provides room for between two and five additional program lines. Excluding the POS and SEG$ functions provides approximately ten additional lines each.

BASIC/CAPS has the ability to use cassettes for data file storage and for saving BASIC programs. Data files are controlled by the OPEN, CLOSE, PRINT#, PRINT# USING, INPUT#, IF END#, and RESTORE# statements.

BASIC program storage and retrieval is accomplished by use of the OVERLAY and CHAIN statements, and the OLD, SAVE, REPLACE, APPEND, RUN, and RUNNH command. In addition to cassettes, BASIC/CAPS supports a line printer, high-speed paper tape reader, and a high-speed paper tape punch via the same commands and statements used to manipulate cassette files. All I/O devices except the console terminal are treated as sequential files.

BASIC/CAPS has a facility which allows PDP-11 assembly language programmers to interface their own assembly language routines to BASIC. This facility permits the user to add functions to BASIC which can operate directly on special purpose peripheral devices, or can perform specialized computations.

The CALL statement is used to execute these assembly language routines from the BASIC program.

For a routine to be accessible from the CALL statement, it must be defined in the special System Function Table. This table allows BASIC to associate the name of an assembly language routine with the actual location of the routine in memory. The function table consists of a list containing the name and address of every routine that may be called from BASIC.

The software provided with the BASIC kit includes a function table, FTBL.PAL. User-written assembly language routines must be defined in this module. The routines provided with the BASIC peripheral support for graphics (GT) and for the Laboratory Peripheral System (LPS) are conditionally defined in FTBL.PAL.

The BASIC-CAPS documentation includes examples which show how to access BASIC variables from user assembly language routines. These examples show how to pass parameters to user assembly language routines and how assembly language routines can store results in BASIC variables for access from the main BASIC program.
2.1 OPERATING SYSTEM FUNCTIONS AND FEATURES

RT-11 is a single-user disk operating system designed for interactive program development and on-line applications. In a foreground/background environment, the user can run a real-time application in the foreground while interactively developing a program or running a batch stream in the background. RT-11 includes two compatible monitors and a variety of program development and system utilities. As options, RT-11 supports FOCAL, FORTRAN IV and BASIC languages. BASIC is offered in a single-user or multi-user version.

RT-11 runs on a PDP-11 with between 8K and 28K words of memory. In addition, RT-11 requires a console terminal and one of the following random-access devices: TC11 dual DECTape drive, RK11 disk cartridge drive, RX11 floppy disk drive, RF11 fixed-head disk with floppy disk, RK11 disk, DECTape or paper tape reader/punch, RP02 disk drive with floppy disk, RK11 disk or DECTape drive, or RS03 or RS04 disk drive with RK05 disk drive. RT-11 also supports a line printer, magnetic tape drives, a card reader and cassette drives. In addition, the high-level languages provide access to a variety of graphics and lab display peripherals.

RT-11 is a particularly easy-to-use system, and yet it provides a sophisticated set of programming tools, particularly for the lab application environment. The system is completely device-independent and configuration-independent. Programs can be directed to use a specific peripheral at run-time. The system can run unmodified on a variety of memory configurations.

The I/O system has been designed so that device handlers are files on the system device. Users can incorporate device drivers for special devices through simple file-oriented interfaces.

<table>
<thead>
<tr>
<th>Table 2-1 RT-11 System Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System type</strong></td>
</tr>
<tr>
<td><strong>CPU support</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Memory size range</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Table 2-1  RT-11 System Summary (Cont.)</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>Additional CPU hardware supported</strong></td>
</tr>
<tr>
<td>PDP-11/04,05,10 Extended Arithmetic Element*</td>
</tr>
<tr>
<td>PDP-11/35, 40 Extended Instruction Set*,</td>
</tr>
<tr>
<td>Floating Instruction Set*</td>
</tr>
<tr>
<td>PDP-11/45 Floating Point Processor*</td>
</tr>
<tr>
<td>* Under BASIC or FORTRAN and FOCAL</td>
</tr>
<tr>
<td><strong>Minimum peripherals</strong></td>
</tr>
<tr>
<td>Console terminal; TC11 dual DECtape drive; or</td>
</tr>
<tr>
<td>RX11 floppy disk drive; or RK11 disk drive; or</td>
</tr>
<tr>
<td>RF11 fixed-head disk drive with PC11 paper tape</td>
</tr>
<tr>
<td>reader/punch, DECtape, floppy disk or RK11 disk;</td>
</tr>
<tr>
<td>or RP02 disk with DECtape, floppy disk or</td>
</tr>
<tr>
<td>RK11 disk; or RS03 or RS04 with RK11 disk.</td>
</tr>
<tr>
<td><strong>Additional peripherals</strong></td>
</tr>
<tr>
<td>LP11, LS11, or LV11 line printer,</td>
</tr>
<tr>
<td>LV11 Plotter (under BASIC or FORTRAN),</td>
</tr>
<tr>
<td>CR11 card reader,</td>
</tr>
<tr>
<td>TM11 magnetic tape drive,</td>
</tr>
<tr>
<td>TA11 cassette drive,</td>
</tr>
<tr>
<td>VT11 Display Subsystem</td>
</tr>
<tr>
<td>(under BASIC or FORTRAN)</td>
</tr>
<tr>
<td>AR11 A/D Subsystem (under FORTRAN)</td>
</tr>
<tr>
<td>DR11K Digital I/O Peripheral (under FORTRAN)</td>
</tr>
<tr>
<td>LPS-11 Lab Peripheral Options</td>
</tr>
<tr>
<td>(under BASIC or FORTRAN)</td>
</tr>
<tr>
<td><strong>Monitor size</strong></td>
</tr>
<tr>
<td>Single Job Monitor:</td>
</tr>
<tr>
<td>Foreground/Background Monitor:</td>
</tr>
<tr>
<td>2K words</td>
</tr>
<tr>
<td>4K words</td>
</tr>
<tr>
<td><strong>Maximum space available for programs and device handlers</strong></td>
</tr>
<tr>
<td>Under Single Job Monitor:</td>
</tr>
<tr>
<td>Under Foreground/Background Monitor:</td>
</tr>
<tr>
<td>26K words</td>
</tr>
<tr>
<td>24K words</td>
</tr>
<tr>
<td><strong>System programs</strong></td>
</tr>
<tr>
<td>EDIT text editor</td>
</tr>
<tr>
<td>MACRO assembler (requires 12K system)</td>
</tr>
<tr>
<td>EXPAND macro expander</td>
</tr>
<tr>
<td>ASEMBL assembler</td>
</tr>
<tr>
<td>LINK linker</td>
</tr>
<tr>
<td>LIBR librarian</td>
</tr>
<tr>
<td>ODT on-line debugger</td>
</tr>
<tr>
<td>PATCH code patch utility</td>
</tr>
<tr>
<td>PATCHO object code patch utility</td>
</tr>
<tr>
<td>PIP peripheral interchange program</td>
</tr>
<tr>
<td>FILEX file exchange utility</td>
</tr>
<tr>
<td>SRCCOM text comparison utility</td>
</tr>
<tr>
<td>DUMP file dump utility</td>
</tr>
<tr>
<td>LAB APPLICATION-11 library (optional)</td>
</tr>
<tr>
<td>BATCH batch processor</td>
</tr>
<tr>
<td><strong>High-level languages</strong></td>
</tr>
<tr>
<td>FOCAL</td>
</tr>
<tr>
<td>Single-user BASIC or multi-user MU BASIC</td>
</tr>
<tr>
<td>FORTRAN IV</td>
</tr>
</tbody>
</table>
2.2 MONITOR ORGANIZATION

The RT-11 operating system actually provides two monitors: The single job (SJ) Monitor and the Foreground/Background (F/B) Monitor. The F/B Monitor allows two programs to operate: A foreground program and a background program. The real-time or on-line function is accomplished in the foreground which has priority on system resources. Functions which do not have critical response-time requirements (e.g., program development or batch operations) are accomplished in the background, which operates whenever the foreground is not busy. Both foreground and background have access to all the monitor services. The language processors and utilities are confined to the background. Although they operate independently, Foreground and Background can communicate through disk files and/or system message queuing facilities. If F/B operation is not required, the SJ Monitor—a totally compatible subset, which requires less memory and less overhead—can be booted from the system device instead of the F/B Monitor.

2.2.1 Monitor Components

To keep system overhead low, the RT-11 Monitor is designed in a modular fashion so that only those portions actually in use at a given time are resident in memory. The Monitor is made up of four major modules plus the device handlers available under the system. As shown in Figure 2-1, only the resident monitor (RMON) is permanently in memory. The keyboard monitor (KMON), the user service routine (USR), the command string interpreter (CSI), and the device handlers are loaded into memory only as needed.

![Figure 2-1 RT-11 MONITOR](image)

Resident Monitor (RMON)
The resident monitor is the only permanently memory-resident part of
RT-11. The programmed requests for all services of RT-11 are handled by RMON. RMON also contains the console terminal service, error processor, system device handler, and system tables.

**Keyboard Monitor (KMON)**
The keyboard monitor provides communication between the user at the console and the RT-11 system. Monitor commands allow the user to assign logical names to devices, run programs, load device handlers, and control F/B operations. A dot at the left margin of the console terminal page indicates that the Keyboard Monitor is in memory and is waiting for a user command.

**User Service Routine (USR)**
The User Service Routine provides support for the RT-11 file structure. It loads device handlers, opens files for read or write operations, deletes and renames files, and creates new files. The Command String Interpreter (CSI) is part of the USR and can be accessed by any program to interpret a character string as a command to perform a set of USR operations.

**Device Handlers (DH)**
As shown below, RT-11 device handlers are, for the most part, short, easily implemented routines. The cassette and magnetic tape handlers are larger because they include a complete file label processing system. The device handlers are merely files on the system device. This means they may be created by the normal editing and assembling process. Interfacing them to the monitor then requires only the modification of five to six parameters in the monitor.

<table>
<thead>
<tr>
<th>HANDLER</th>
<th>SIZE (DECIMAL WORDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartridge Disk (RK11)</td>
<td>113</td>
</tr>
<tr>
<td>DECtape (TC11)</td>
<td>106</td>
</tr>
<tr>
<td>Fixed Head Disk (RF11)</td>
<td>92</td>
</tr>
<tr>
<td>Cassette (TA11)</td>
<td>996</td>
</tr>
<tr>
<td>Magtape (TM11)</td>
<td>1120</td>
</tr>
<tr>
<td>Line Printer (LP11)</td>
<td>99</td>
</tr>
<tr>
<td>Card Reader (CR11)</td>
<td>363</td>
</tr>
<tr>
<td>Console Terminal (LA36)</td>
<td>140</td>
</tr>
<tr>
<td>Paper Tape Reader</td>
<td>62</td>
</tr>
<tr>
<td>Paper Tape Punch</td>
<td>56</td>
</tr>
<tr>
<td>Floppy Disk (RX11)</td>
<td>216</td>
</tr>
<tr>
<td>Disk Pack (RP02)</td>
<td>142</td>
</tr>
<tr>
<td>Fixed Head Disk (RS03/4)</td>
<td>78</td>
</tr>
</tbody>
</table>

**2.2.2 General Memory Layout and Component Sizes**
When the RT-11 system is first bootstrapped from the system device, memory is arranged as shown in the left diagram of Figure 2-2 (this is the case for either the Single-Job or Foreground/Background Monitor, since no foreground job exists yet). Under the F/B Monitor the background job is the RT-11 module KMON.

When device handlers are loaded into memory, USR and KMON are moved down, as shown in the center diagram of Figure 2-2.
RT-11 maintains a free memory list to manage memory. Thus, when a handler is unloaded, the space the handler occupied is returned to the free memory list and is reclaimed by the background.

When an RT-11 foreground job is initiated, room is created for the foreground job to be loaded by decreasing the amount of space available to the background job, as shown in the right diagram of Figure 2-2.

Following are the approximate sizes (in decimal number of words) of the components for RT-11.

<table>
<thead>
<tr>
<th></th>
<th>F/B</th>
<th>Single-job</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMON</td>
<td>4000</td>
<td>2000</td>
</tr>
<tr>
<td>USR</td>
<td>2050</td>
<td>2050</td>
</tr>
<tr>
<td>KMON</td>
<td>1550</td>
<td>1550</td>
</tr>
</tbody>
</table>

In the F/B system, the background area must always be large enough to hold KMON and USR (3.5K words). The following list indicates the total space available for the loaded device handlers, the foreground job, and the display handler. Note that the low memory area from 0-477 is never used for executable programs. (These sizes also allow room for the 4K RMON).

<table>
<thead>
<tr>
<th>MACHINE SIZE (WORDS)</th>
<th>FOREGROUND SPACE AVAILABLE (WORDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16K</td>
<td>8.5K</td>
</tr>
<tr>
<td>24K</td>
<td>16.5K</td>
</tr>
<tr>
<td>28K</td>
<td>20.5K</td>
</tr>
</tbody>
</table>

With the Single-Job Monitor, RMON requires only 2K. The following list shows the amount of space available to users with the Single-Job Monitor:
When a background job or single-job program is initiated, RT-11 allows the program to be loaded in over the KMON and USR if it exceeds the free memory available. While the program is running, the KMON and USR are not resident. If the program issues a request that the USR must service, a portion of the program is swapped out to make room for the USR. The USR is swapped in and remains resident until the request is serviced. When the request is serviced, the portion of the program that was previously swapped out is swapped back in.

If the program terminates or if an operator types a CTRL/C to interrupt the program, both the KMON and USR are loaded into memory. That portion of the program that occupies the KMON/USR space is swapped out; it is held until the operator either continues the program or re-initializes the environment.

If a background or single-job program does not exceed the free memory available when it is loaded, the KMON and USR remain resident during its execution and are not reloaded when the program terminates. Swapping, if it does occur, is invisible to the user.

Typical memory layouts for both the Single-Job and Foreground/Background systems are shown in Figure 2-3.

![Figure 2-3 Typical Memory Layouts](image-url)
2.2.3 I/O System Design and Operation

All I/O is handled in block format directly to and from the user area. There is no intermediate buffering by the monitor to slow down throughput. Since most programs deal with data on a character-by-character basis, it was deemed superfluous to provide line-oriented handling in the monitor.

All I/O is processed by a queue manager. RMON sends I/O orders to the queue manager which directs the device handlers to carry out the order. If the device handler is busy, the order waits until the device is available and the order can be completed. If parallel processing is indicated, the order is placed in the I/O queue for later processing and control is returned to the requesting program. There is one queue for each device. In F/B operation, the foreground request is always given the highest priority because foreground requests are queued ahead of background requests as directed by the initial read/write order. The queue manager may initiate a completion routine when the device handler indicates completion of the I/O order.

There are three modes of I/O operation:

SYNCHRONOUS I/O
In this mode, the I/O transfer is initiated and control does not return to the requesting program until the transfer is complete or an error is detected. This allows processing to proceed knowing that the data transfer has been completed.

ASYNCHRONOUS I/O
In this mode, the transfer is started (the system will place the request in a queue automatically if the device is already in use) and control is returned immediately to the requesting program so that processing may continue. When the user program needs assurance that the I/O operation has been completed, it issues a WAIT request which returns control only when the requested operation has terminated. This option allows overlap of an I/O process with program execution and with other I/O processes. The processes are usually asynchronous, but can be synchronized when necessary.

EVENT-DRIVEN I/O
This mode utilizes the full flexibility of the PDP-11 and RT-11: The I/O transfer is initiated (automatically queued if the device is already in use) and control is returned to the requesting program. When the I/O operation has been completed, the user program is interrupted and control is passed to a “completion routine” (specified when the transfer was originally requested). When the completion routine exits, control is returned to the interrupted program at the point of interrupt. The completion function allows fully overlapped, asynchronous I/O, greatly simplifying the real-time task.

2.2.4 Batch Processing
RT-11 BATCH is a complete job control subsystem which provides batch mode processing of user jobs. In the foreground/background environment, BATCH processes job streams in the background partition, allowing real-time or other user jobs to run in the foreground. The BATCH run-time support package (which is resident when BATCH is running)
requires only 1K words of memory. RT-11 BATCH may be used in either single-job monitor configurations of 12K or more of memory, or in any foreground/background configuration.

RT-11 BATCH also has the unique capability of being interrupted. For example, a three-hour batch stream might be half done when an urgent need for a specific job arises. A simple command may be typed to RT-11, causing BATCH to give control to the keyboard monitor at a convenient breaking point. Commands may be entered and jobs may be run (even a batch job), after which a command may be given to continue running the interrupted batch stream to its normal completion.

The BATCH job control language is discussed in Section 2.4.3.

2.2.5 Switching Between Single-Job and Foreground/Background
The RT-11 user can use either the Single-Job or the Foreground/Background Monitor; the process of switching between the two requires no special programming considerations or elaborate operator procedures. For an application that requires frequent switching between the F/B and Single-Job monitors, the basic procedure is:
1. Store both monitors on the same device, one with the active monitor name, the other with an inactive name.
2. When the switch is desired, the operator runs the PIP system utility, preserves the running monitor by renaming it to an inactive name and renames the desired monitor to the active monitor name. The operator then uses two PIP commands to copy the bootstrap and reboot the system.

2.3 SYSTEM CONVENTIONS
The following sections describe the RT-11 system conventions for devices and device names, filenames and extensions, data formats and file structures.

2.3.1 Physical Device Names
Devices are referenced by means of a standard two-character device name. Devices can also be assigned logical names. A logical name takes precedence over a physical name and thus provides device independence. With this feature a program that is coded to use a specific device does not need to be rewritten if the device is unavailable.

2.3.2 Filenames and Extensions
Files are referred symbolically by a name of one to six alphanumeric characters followed, optionally, by a period and an extension of up to three alphanumeric characters. (Excess characters in a filename may cause an error message.) The extension to a filename generally indicates the format of a file. If an extension is not specified for an input or output file, most system programs assign appropriate default extensions. Table 2-2 lists the standard extensions used in RT-11.

<table>
<thead>
<tr>
<th>EXTENSION</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>.BAD</td>
<td>Files with bad (unreadable) blocks; this extension can be assigned by the user whenever bad areas occur on a device.</td>
</tr>
</tbody>
</table>
Table 2-2 Filename Extension (Cont.)

The .BAD extension makes the file permanent in that area, preventing other files from using it and consequently becoming unreadable.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.BAK</td>
<td>Editor backup file.</td>
</tr>
<tr>
<td>.BAS</td>
<td>BASIC source file (BASIC input).</td>
</tr>
<tr>
<td>.BAT</td>
<td>BATCH command file.</td>
</tr>
<tr>
<td>.CTL</td>
<td>BATCH control file generated by the BATCH compiler.</td>
</tr>
<tr>
<td>.CTT</td>
<td>BATCH internal temporary file.</td>
</tr>
<tr>
<td>.DAT</td>
<td>BASIC or FORTRAN data file.</td>
</tr>
<tr>
<td>.DIR</td>
<td>Directory listing file.</td>
</tr>
<tr>
<td>.DMP</td>
<td>DUMP output file.</td>
</tr>
<tr>
<td>.FOR</td>
<td>FORTRAN IV source file (FORTRAN input).</td>
</tr>
<tr>
<td>.LDA</td>
<td>Absolute binary file (optional Linker output).</td>
</tr>
<tr>
<td>.LLD</td>
<td>Library listing file.</td>
</tr>
<tr>
<td>.LOG</td>
<td>BATCH Log file.</td>
</tr>
<tr>
<td>.LST</td>
<td>Listing file (MACRO or FORTRAN output).</td>
</tr>
<tr>
<td>.MAC</td>
<td>MACRO or EXPAND source file (MACRO, EXPAND, SRCCOM input).</td>
</tr>
<tr>
<td>.MAP</td>
<td>Map file (Linker output).</td>
</tr>
<tr>
<td>.OBJ</td>
<td>Relocatable binary file (MACRO, ASEMBL, FORTRAN IV output, Linker input, LIBR input and output).</td>
</tr>
<tr>
<td>.PAL</td>
<td>Output file of EXPAND (the MACRO expander program), input file of ASEMBL.</td>
</tr>
<tr>
<td>.REL</td>
<td>Foreground job relocatable image (Linker output, default for monitor FRUN Command).</td>
</tr>
<tr>
<td>.SAV</td>
<td>Memory image or SAVE file; default for R, RUN, SAVE and GET Keyboard Monitor commands; also default for output of Linker.</td>
</tr>
<tr>
<td>.SOU</td>
<td>Temporary source file generated by BATCH.</td>
</tr>
<tr>
<td>.SYS</td>
<td>System files and handlers.</td>
</tr>
</tbody>
</table>

2.3.3 Data Formats
The RT-11 system makes use of five types of data formats: ASCII, object, memory image, relocatable image, and load image.

Files in ASCII format conform to the American National Standard Code for Information Interchange, in which each character is represented by a 7-bit code. Files in ASCII format include program source files created by the Editor, listing and map files created by various system programs, and data files consisting of alphanumerical characters.

Files in object format consist of data and PDP-11 machine language code. Object files are those output by the assembler or FORTRAN compiler and used as input to the Linker.
The Linker can output files in memory image format (.SAV), relocatable image format (.REL), or load image format (.LDA).

A memory image file (.SAV) is a "picture" of what memory will look like when a program is loaded. The file itself requires the same number of disk blocks as the corresponding number of 256-word memory blocks.

A relocatable image file (.REL) is one which can be run in the foreground. It differs from a memory image file in that the file is linked as though its bottom address were 0. When the program is called (using the monitor FRUN command), the file is relocated as it is loaded into memory. (A memory image file requires no such relocation.)

A load image (or .LDA) file may be produced for compatibility with the PDP-11 Paper Tape System and is loaded by the absolute binary loader. .LDA files can be loaded and executed in stand-alone environments without relocation.

2.3.4 File Structure

RT-11 uses a "contiguous" file structure. This type of structure implies that every file on the device is made up of a contiguous group of physical blocks. Thus, a file that is 9 blocks long occupies 9 contiguous blocks on the device. This file structure minimizes the number of disk accesses needed to transfer data and hence increases access speed.

A contiguous area on a device can be in one of the following categories:

1. Permanent file. This is a file which has been closed on a device. Any named files which appear in a PIP directory listing are permanent files.

2. Tentative file. Any file which has been created but not closed is a tentative file entry. When the file is closed, the tentative entry becomes a permanent file. If a permanent file already exists under the same name, the old file is deleted. If a file is never closed, the tentative file is treated like an empty entry.

3. Empty entry. When disk space is unused or a permanent file is deleted, an empty entry is created. Empty entries appear in a complete device directory listing as <UNUSED> N, where N is the decimal length of the empty area.

Since a contiguous structure does not automatically reclaim unused disk space, RT-11 PIP has an option which allows the user to collect all empty areas so that they occur at the end of a device.

2.4 COMMANDS

The operator controls and directs system operation through three different interfaces. The operator communicates directly with the Monitor using keyboard commands and special function keys, communicates indirectly with the Monitor or user programs by issuing an I/O command string. The following sections describe the commands and command formats.

2.4.1 Keyboard Communication

Keyboard commands and special function keys allow the operator to communicate with the RT-11 Monitor to allocate system resources, manipulate memory images, start programs, and use foreground/back-
ground services. The keyboard commands are interpreted by KMON. Note that in a Foreground/Background environment, KMON is always run as the background job. The special function keys are interpreted by the console terminal device handler. They can be used to communicate with any system or user program through the console terminal.

Table 2-3 lists the Monitor keyboard commands. The first set of commands are general commands used in either a Single-Job or Foreground/Background environment. The second set of commands are used in either environment to control single-job or background program operation. The last set of commands are only in a Foreground/Background system to control foreground program operation.

The special function of certain terminal keys used for communication with KMON are listed in Table 2-4. It is important to note that console input and output under F/B are independent functions; input can be typed to one job while output is printed by another. The operator may be in the process of typing input to one job when the other job is ready to print on the terminal, in which case the job which has output interrupts the operator and prints the output on the terminal. Input control is not redirected to this job, however, unless a CTRL/B or CTRL/F is explicitly issued. If input is typed to one job while the other has output control, echo of the input is suppressed until the job accepting input regains output control. At this point, all accumulated input is echoed. If the foreground job and background job are both ready to print output at the same time, the foreground job has priority.

Table 2-3  Monitor Keyboard Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>Enters or reports the system date. The specified date is assigned to newly-created files, new device directory entries, and listing output until a new DATE command is issued.</td>
</tr>
<tr>
<td>TIME</td>
<td>Resets or reports the current time of day kept by the system.</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>Assigns or deassigns a user-specified 1-3 character logical name to a physical device. This is useful, for example, to redirect 1/O to another device when a program to be executed refers to a device that is not available. It can also be used to assign FORTRAN logical units to device names.</td>
</tr>
<tr>
<td>SET</td>
<td>Used to change device handler characteristics and certain system configuration parameters. The user can set characteristics such as line printer or console line width, “device not ready” error handling, cassette read verification, card reader card codes, etc.</td>
</tr>
<tr>
<td>LOAD</td>
<td>Makes a device handler memory-resident for foreground and/or background jobs. Execution is faster when a handler is resident but memory area must be allocated for the handler. Any device handler to be used by a foreground job must be loaded before it can be used. A foreground or background job can own a device exclusively, or the jobs can share a device.</td>
</tr>
</tbody>
</table>
Table 2-3 Monitor Keyboard Commands (Cont.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNLOAD</td>
<td>Makes handlers that were previously loaded non-resident, freeing the memory they were using. UNLOAD clears ownership for all units of an indicated device type. Any memory freed is returned to a free memory list and eventually reclaimed for the background job. A special function of this command is to remove a terminated foreground job and reclaim memory, since the space occupied by the foreground job is not automatically returned to the free memory list when it finishes.</td>
</tr>
<tr>
<td>BASE</td>
<td>Sets a relocation base which is added to the address specified in subsequent Examine or Deposit commands to obtain the address of the location to be referenced. This command is useful when referencing linked modules; the base address can be set to the address where the module is loaded.</td>
</tr>
<tr>
<td>EXAMINE</td>
<td>Prints the contents of a specified location(s) in octal on the console.</td>
</tr>
<tr>
<td>DEPOSIT</td>
<td>Deposits a specified value at a given location.</td>
</tr>
<tr>
<td>SAVE</td>
<td>Writes a specified user memory area to a named file and device in save image format. This command allows the user to modify a background or single job program using the Examine and Deposit commands after it has been loaded into memory, and then save the modified program.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Causes currently open output files in the background job to become permanent files. The command is used after a CTRL/C is issued to abort a background job and the user wants to preserve any new files that job had open prior to its termination.</td>
</tr>
</tbody>
</table>

**Background or Single-Job Program Control Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Loads a specified memory image file (background program) into memory. It is normally used to load a background program into memory for modification or debugging with ODT or the Base, Examine, Deposit and START commands. Multiple GET commands can be used to combine programs.</td>
</tr>
<tr>
<td>START</td>
<td>Begins execution of the program currently in memory (loaded using the GET command) at a specified address. START does not clear or reset memory areas.</td>
</tr>
<tr>
<td>INITIALIZE</td>
<td>Resets background system tables and cleans up the background area. It makes non-resident any device handlers used by the background program that were not made permanently resident (using the LOAD command) and purges the background I/O channels. The command can be used prior to running a background program, or when the accumulated results of previously-issued GET commands are to be discarded.</td>
</tr>
<tr>
<td>RUN</td>
<td>Loads a specified memory image file (background program) into memory and starts execution at its start address. It is</td>
</tr>
</tbody>
</table>
Table 2-3 Monitor Keyboard Commands (Cont.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>The command is identical to the RUN command except that the file specified must reside on the system device.</td>
</tr>
<tr>
<td>REENTER</td>
<td>Starts a program at its reentry address (start address minus two). It is generally used to avoid reloading the same program for repetitive execution, or to restart a program interrupted by a CTRL/C.</td>
</tr>
<tr>
<td>FRUN</td>
<td>Used to initiate foreground jobs. Normally, FRUN loads the relocatable image file and starts it. The user can optionally specify additional space to be allocated for the job over and above the actual program size or specify additional stack space. In addition, the user can request that the program be loaded only. The user must then explicitly start the program using the RSUME command. If ODT is used with the foreground job, this feature allows the user to examine or modify the program before starting it.</td>
</tr>
<tr>
<td>SUSPEND</td>
<td>Stops the execution of a foreground job. The user can either resume the job, run another foreground job, or unload the stopped foreground job to release the memory.</td>
</tr>
<tr>
<td>RSUME</td>
<td>Resumes execution of the foreground job where it was suspended.</td>
</tr>
</tbody>
</table>

Table 2-4 Special Function Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL/A</td>
<td>Display next page of output; used only after CTRL/S (used only with GT ON).</td>
</tr>
<tr>
<td>CTRL/B</td>
<td>Under the F/B Monitor echoes B&gt; on the terminal (unless output is already coming from the background job) and causes all keyboard input to be directed to the background job. At least one line of output will be taken from the background job (the foreground job has priority, and control will revert to it if it has output). All typed input will be directed to the background job until control is redirected to the foreground job (via CTRL/F).</td>
</tr>
<tr>
<td>CTRL/C</td>
<td>Interrupts current program execution, and returns control to the Keyboard Monitor. Note that under the F/B Monitor, the job which is currently receiving input will be the job that is stopped (determined by whether a CTRL/F or CTRL/B was most recently typed). To ensure that the command is directed to the proper job, type CTRL/B or CTRL/F before typing CTRL/C. If a program is waiting for terminal input or is using the console terminal device handler for input, typing a single CTRL/C interrupts execution and returns control to the monitor command level; otherwise, two CTRL/C’s must be typed in order to interrupt execution.</td>
</tr>
<tr>
<td>CTRL/E</td>
<td>Display all I/O on the screen and console terminal simultaneously (with GT ON command only).</td>
</tr>
</tbody>
</table>
Table 2-4  Special Function Commands (Cont.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL/F</td>
<td>Under the F/B Monitor echoes F&gt; on the terminal and instructs that all keyboard input be directed to the foreground job and all output be taken from the foreground job. If no foreground job exists, F? is printed and control is directed to the background job. Otherwise, control remains with the foreground job until redirected to the background job (via CTRL/B) or until the foreground job terminates.</td>
</tr>
<tr>
<td>CTRL/O</td>
<td>Inhibit printing on console terminal.</td>
</tr>
<tr>
<td>CTRL/Q</td>
<td>Resume console output; used only after CTRL/S.</td>
</tr>
<tr>
<td>CTRL/S</td>
<td>Temporarily suspended terminal output until CTRL/A or CTRL/Q.</td>
</tr>
<tr>
<td>CTRL/U</td>
<td>Delete current line being entered.</td>
</tr>
<tr>
<td>CTRL/X</td>
<td>Delete entire command string (EDIT only).</td>
</tr>
<tr>
<td>CTRL/Z</td>
<td>End-of-file for terminal input.</td>
</tr>
</tbody>
</table>

2.4.2. Entering I/O Information Using the CSI

Once either monitor has been loaded and a system program (or any program which uses the Command String Interpreter) has been started, the Command String Interpreter prints an asterisk at the left margin. In response to the asterisk, a command string is entered providing information about devices, filenames and extensions, and switch options. The general format of this command line is:

```
*OUTPUT = INPUT/SWITCH
```

The = sign is a delimiter which separates the output and input fields; the < sign may be used in place of the = sign.

**OUTPUT** is entered in the format:

```
dev:filnam.ext[n], . . . dev:filnam.ext[n]
```

**INPUT** as:

```
dev:filnam.ext, . . . dev:filnam.ext
```

Command string switches vary with the system or user program.

2.4.3 BATCH Job Control Language

The RT-11 BATCH job control language provides complete system control capabilities through easy-to-learn commands:

**Job Control**

```
$JOB            Begin a job.
$SEQUENCE       Assign an identification to a job.
$EOJ            End a job.
```

**Data Control**

```
$DATA           Delimit the beginning of data in the input stream.
$EOD            Signal end of data in input stream.
```
Operator Communications

$MESSAGE
Write a message to the console terminal.

Device Associations

$MOUNT
Allows the operator to mount a requested volume and to associate the volume with a logical device name.

$DISMOUNT
Disassociates a logic device name from a physical unit.

Functional Commands

$COPY
Copies input files to output files.

$CREATE
Create a file with the data records in the input stream.

$DELETE
Delete specified files.

$DIRECTORY
Make directory listings.

$LINK
Create an executable program from object modules and libraries.

$PRINT
Print copies of specified files.

$RUN
Initiate execution of a specified program.

Language Processors

$BASIC
Execute BASIC.

$FORTRAN
Compile FORTRAN source programs.

$MACRO
Assemble MACRO source programs.

Additional Commands

$RT11
Accept RT-11 Keyboard Monitor commands directly.

$LIBRARY
Create a library list to be used in Link operations.

$CALL
CALL another BATCH command file as a subroutine.

$CHAIN
Transfer control to another BATCH command file.

All of the commands shown have several options each, allowing BATCH commands to specify many different operations. The following is an example of a runnable BATCH stream which will compile and execute a simple FORTRAN program:

$JOB
$FORTRAN/RUN
$EOJ

RT-11 BATCH also has a programmable mode. In this mode, BATCH supports such features as labels, variables, conditional branches, and substitute arguments.

2.5 MONITOR PROGRAMMED REQUESTS
The RT-11 monitor provides a complete set of services for user pro-
grams. Monitor services include calls used for program initialization, control of system operating characteristics, interrogation of system state and resources, command interpretation, file operations, I/O transfers, program termination, and interrupt servicing. Some features which are available only to the F/B user include:

1. Mark Time—This facility allows user programs to set clock timers to run for specified amounts of time. When the timer runs out, a routine specified by the user is entered. There may be as many mark time requests as desired, providing system queue space is reserved.

2. Timed Wait—This feature allows the user program to "sleep" until the specified time increment elapses. Typically, a program may need to sample data every few seconds or even minutes. While the program is idle, the other job can run. The timed wait accomplishes this; when the time has elapsed, the issuing job is again runnable.

3. Send Data/Receive—It is possible, under RT-11 F/B, to have the foreground and background programs communicate with one another. This is accomplished with the send/receive data functions. Using this facility, one program sends messages (or data) in variable size blocks to the other job. This can be used, for example, to pass data from a foreground collection program directly to a background analysis program.

For users of FORTRAN, the system subroutine library (SYSLIB) provides a set of functions for alphanumeric string manipulation as well as most monitor functions.

Communication with the monitor is accomplished through the EMT instruction or FORTRAN CALLS to the system library. The low-order byte of the EMT instruction is used as an operation code in the range 360(8) through 375(8). Those operations which use codes 360(8) through 374(8) either require no arguments or use Register 0 to pass arguments to or return arguments from the monitor. Operations that require more than one argument use EMT 375, and R0 is used to point to an argument list in memory. The argument list includes an operation code, the channel for that operation, and any other arguments necessary.

For example, the macro call to read 1000(8) words into a buffer from block 7 of file 3

```
.READW #AREA, #3, #BUFFER#, #1000, #7
```

is expanded into code which points R0 to the argument block, fills the argument block with the specified arguments and executes the instruction EMT 375.

This structure is fully re-entrant, and allows users to construct programmed requests either statically or dynamically from parameters specified at run-time. The same routine can process many monitor services, simply loading R0 with argument pointers from appropriate queues. FORTRAN users can issue monitor calls through function calls to SYSLIB, the FORTRAN system interface library. These calls execute FORTRAN object time system (OTS) routines which actually issue the appropriate EMT instructions to be executed with the arguments spec-
<table>
<thead>
<tr>
<th>AREA:</th>
<th>READ CODE</th>
<th>3</th>
<th>FUNCTION AND CHANNEL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BUFFER</td>
<td></td>
<td>BUFFER ADDRESS</td>
</tr>
<tr>
<td></td>
<td>1000B</td>
<td></td>
<td>WORD COUNT</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td>BLOCK #</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>USE WAIT MODE(READW)</td>
</tr>
</tbody>
</table>

ified. For example, the previous READ operation can be accomplished in a FORTRAN program via a CALL READW statement.

Most system resource control and interrogation is done through the programmed requests, but some communication is accomplished using two core areas, the system communication area and the monitor "fixed-offsets." The job communication area is located in locations 40(8)-57(8), and contains parameters which describe and control execution of the particular job running at the time (it is context-switched in F/B). Included in this area is the Job Status Word, starting address of the job, USR swapping address, and other parameters.

The second memory communication area is the "fixed-offset area," whose items are accessed by a fixed address offset from the start of the resident monitor. It contains system constants used to control monitor operation. The user program can interrogate these constants to determine characteristics of the operating environment while the job is running.

2.5.1 Summary of Programmed Requests
There are three types of services which the monitor makes available to the user through programmed requests. These are:

1. Requests for File Manipulation.
2. Requests for Data Transfer.
3. Requests for Miscellaneous Services

Table 2-5 summarizes the programmed requests in each of these categories alphabetically. Those marked with an asterisk function only in a F/B environment; they are ignored under the Single-Job Monitor.

<table>
<thead>
<tr>
<th>Table 2-5 Summary of Programmed Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Manipulation Requests</td>
</tr>
<tr>
<td>MNEMONIC</td>
</tr>
<tr>
<td>*CHCOPY</td>
</tr>
<tr>
<td>.CLOSE</td>
</tr>
<tr>
<td>.DELETE</td>
</tr>
</tbody>
</table>

2-17
<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.RCVD</td>
<td><strong>.RCVD</strong></td>
</tr>
<tr>
<td>*.RCVDC</td>
<td></td>
</tr>
<tr>
<td>*.RCVDW</td>
<td></td>
</tr>
<tr>
<td>.READ</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.</td>
</tr>
<tr>
<td>.READC</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers asynchronously to the routine specified in the .READC request.</td>
</tr>
<tr>
<td>.READW</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program only after the transfer is complete.</td>
</tr>
<tr>
<td>*.SDAT</td>
<td>Allows the user to send messages or data to the other job in an F/B environment. The three modes correspond to the .WRITE, .WRITEC, and WRITEW modes.</td>
</tr>
<tr>
<td>*.SDATC</td>
<td></td>
</tr>
<tr>
<td>*.SDATW</td>
<td></td>
</tr>
<tr>
<td>.TTYIN</td>
<td>Transfers one character from the keyboard buffer to R0.</td>
</tr>
<tr>
<td>.TTINR</td>
<td></td>
</tr>
<tr>
<td>.TTYOUT</td>
<td>Transfers one character from R0 to the terminal output buffer.</td>
</tr>
<tr>
<td>.TTOUTR</td>
<td></td>
</tr>
<tr>
<td>.WRITE</td>
<td>Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of the I/O.</td>
</tr>
<tr>
<td>.WRITC</td>
<td>Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers asynchronously to the routine specified in the .WRITIC request.</td>
</tr>
<tr>
<td>.WRITW</td>
<td>Transfers data via the specified channel to a device and returns control to the user program only after the transfer is complete.</td>
</tr>
</tbody>
</table>
Table 2-5  Summary of Programmed Requests (Cont.)

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CDFN</td>
<td>Defines additional channels for doing I/O.</td>
</tr>
<tr>
<td>.CHAIN</td>
<td>Chains to another program (in the background job only).</td>
</tr>
<tr>
<td>*.CMKT</td>
<td>Cancels an unexpired mark time request.</td>
</tr>
<tr>
<td>*.CNTXSW</td>
<td>Requests that the indicated memory locations be part of the F/B context switch process.</td>
</tr>
<tr>
<td>*.CSIGEN</td>
<td>Calls the Command String Interpreter (CSI) in general mode.</td>
</tr>
<tr>
<td>*.CSISPC</td>
<td>Calls the CSI in special mode.</td>
</tr>
<tr>
<td>*.CSTAT</td>
<td>Returns the status of the channel indicated.</td>
</tr>
<tr>
<td>.DATE</td>
<td>Moves the current date information into R0.</td>
</tr>
<tr>
<td>*.DEVICE</td>
<td>Allows user to turn off device interrupt enable in F/B upon program termination.</td>
</tr>
<tr>
<td>.DSTATUS</td>
<td>Returns the status of a particular device.</td>
</tr>
<tr>
<td>.EXIT</td>
<td>Exits the user program and returns control to the Keyboard Monitor.</td>
</tr>
<tr>
<td>.FETCH</td>
<td>Loads device handlers into memory.</td>
</tr>
<tr>
<td>.GTIM</td>
<td>Gets time of day.</td>
</tr>
<tr>
<td>.GTJB</td>
<td>Gets parameters of this job.</td>
</tr>
<tr>
<td>.HERR</td>
<td>Specifies termination of the job on fatal errors.</td>
</tr>
<tr>
<td>.HRESET</td>
<td>Terminates I/O transfers and does a .SRESET operations.</td>
</tr>
<tr>
<td>.INTEN</td>
<td>Notifies monitor that an interrupt has occurred and to switch to ”system state,” and sets the processor priority to the correct value.</td>
</tr>
<tr>
<td>.LOCK</td>
<td>Makes the monitor User Service Routines (USR) permanently resident until .EXIT or .UNLOCK is executed. The user program is swapped out if necessary.</td>
</tr>
<tr>
<td>*.MRKT</td>
<td>Marks time; i.e., sets asynchronous routine to occur after a specified interval.</td>
</tr>
<tr>
<td>*.MWAIT</td>
<td>Waits for messages to be processed.</td>
</tr>
<tr>
<td>.PRINT</td>
<td>Outputs an ASCII string to the terminal.</td>
</tr>
<tr>
<td>*.PROTECT</td>
<td>Requests that a vector or vectors in the area from 0-476 be given exclusively to this job.</td>
</tr>
<tr>
<td>.PURGE</td>
<td>Clears out a channel.</td>
</tr>
<tr>
<td>.QSET</td>
<td>Expands the size of the monitor I/O queue.</td>
</tr>
<tr>
<td>.RCTRLO</td>
<td>Enables output to the terminal.</td>
</tr>
<tr>
<td>.REGDEF</td>
<td>Defines the PDP-11 general registers.</td>
</tr>
<tr>
<td>.RELEAS</td>
<td>Removes device handlers from memory.</td>
</tr>
</tbody>
</table>

2-19
Table 2-5  Summary of Programmed Requests (Cont.)

* .RSUM  Causes the main line of the job to be resumed where it was suspended with .SPND.

.SERR  Inhibits most fatal errors from causing the job to be aborted.

.SETTOP  Specifies the highest memory location to be used by the user program.

.SFPA  Sets user interrupt for floating point processor exceptions.

.SPFUN  Performs special functions on magtape and cassette units.

* .SPND  Causes the running job to be suspended.

.SRESET  Resets all channels and releases the device handlers from memory.

.SYNCH  Causes the rest of the interrupt service routine to run as a completion routine.

* .TLOCK  Indicates if the USR is currently being used by another job and performs a .LOCK if available.

.TRPSET  Sets a user intercept for traps to locations 4 to 10.

* .TWAITS  Suspends the running job for a specified amount of time.

.UNLOCK  Releases USR if a LOCK was done. The user program is swapped in if required.

.WAIT  Waits for completion of all I/O on a specified channel.

2.5.2  Program Environment Control
A user program typically issues several programmed requests to control the operating environment in which it will run. These include memory use, I/O access, device control, and error processing.

The memory needs of a program are specified to the monitor by the .SETTOP request. When loaded, a program occupies the memory specified by its image created at link time. To obtain more memory, a .SETTOP request is executed, with R0 containing the highest address desired. The monitor returns the highest address available (resident handlers or foreground jobs may prevent all the memory desired from being available), and adjusts its core usage algorithms accordingly. If the memory requirements of the running program permit it, the monitor will retain the file system in resident memory (reducing swapping). If not, it will automatically swap the file system with part of the user program (invisible to that program). The .SETTOP request, then, is the main mechanism by which a user program can determine how much memory is available and can control monitor swapping characteristics.

If a program needs so much memory that the USR (file system) must swap, the swapping will automatically occur (invisible to the program) whenever a USR call is made. If a program knows what file operations are necessary, however, and these operations can be consolidated and performed in localized areas, the efficiency of the system can be en-
hanced in the following manner: request the USR to be swapped in, have it remain resident while a series of USR operations are performed one after the other, then swap the USR back out when the sequence is done. Three requests are provided for the control of USR swapping: LOCK causes the USR to be made resident for a series of file operations; UNLOCK causes it to swap again. F/B programs can use .TLOCK to make the USR resident, but only if the USR is not occupied servicing the other job’s file requests. This check can prevent a job from becoming blocked while the USR, which is a serial, synchronous resource, is processing a request. When a .TLOCK succeeds, the USR is ready to perform an operation immediately.

I/O in RT-11 is accomplished via “channels.” A “channel” is a logical link between a file or a device and a program doing I/O on that file or device. When a file is opened, a unique channel number is associated with it, and all operations on that file are identified via the channel number. The only time a file or device name is used is when the file is opened. RT-11 provides 16(10) channels as part of the resident monitor; that is, up to 16(10) files can be active at any given time. More channels can be activated (up to 255(10)) with the .CDFN request. This request sets aside memory inside the job area to provide storage necessary to accommodate status information for the extra channels. Once the .CDFN request has been executed, as many channels as space allows can be active simultaneously.

The Foreground/Background monitor context-switches critical items such as the machine registers, the job status area, and PDP-11/45 floating point processor (FPP) registers. The job can add memory locations to the list of items to be context-switched with the .CNTXW request. Typical use of .CNTXW is to include a special user device or arithmetic unit in normal context switches.

Special devices can be stopped and interrupt-disabled should the job have to be unexpectedly aborted. The .DEVICE request allows the user to specify a set of device control register addresses and a mask of bits to be set in that register on job exit. When a job is terminated (either normally, by error condition, or operator command), the specified bits are set in the specified locations. The normal use of this feature is to disable any special devices that may be in use by the running program.

Special devices also require vector locations in the area 0-500(8). Since RT-11 normally loads the unused vector space as part of a program load, the user specifies the use of a vector with the .PROTECT request. This will cause RT-11 to leave this vector intact, and prevent other jobs from obtaining the vector at the same time.

During the course of program execution, errors can occur which normally indicate that a program cannot continue, and the job is stopped by the monitor. Typical examples of such errors include directory I/O errors, monitor I/O errors on the system device, or I/O requests to non-existent devices. Some programs cannot afford, however, to allow the monitor to abort the job because of such errors; a typical example is RT-11 multi-user BASIC, which cannot be aborted because of a directory I/O error affecting only one of its users. For such applications, a pair of requests is provided, .HERR and .SERR. A .HERR request (the normal
default) indicates that severe errors are to be handled by the monitor and result in job abortion; a .SERR request causes the monitor to return most errors to the user program for appropriate action.

Each of the pending I/O, message, or timer requests must be queued in one of the monitor queues. Since queue length is variable from one job to another, memory for queue elements desired is set aside with the .QSET request. If no .QSET request is executed by the user program, the monitor uses queue elements set aside in the resident monitor. If only one element is available, all operations will be synchronous (any request issued when the queues are full automatically waits for an element to free up). To expand the size of the I/O queues, a .QSET is executed by the user program. The .QSET declares where in memory the additional queues will go and how many elements they are to contain.

Finally, in addition to using .HERR and .SERR to process I/O errors, the user program can specify that it wishes to handle traps to locations 4(8) and 10(8) (processor error traps) or PDP-11/45 FPP exception traps. Normally these fatal errors are reported by the monitor and the job is aborted. A .TRPSET request, however, will specify the address of a user routine that is to be entered when a trap to 4(8) or 10(8) occurs, and a .SFPA can be used to specify the address of a floating point exception routine for the FPP. If one of these requests is issued, the monitor takes no action on these errors: it merely calls the specified routines.

All the previous initialization and control requests allow the user complete control over the characteristics of his operating environment. The monitor offers a variety of services to make it easy to write simple programs; sophisticated programmers, however, have full control over all aspects of monitor operation, and can place the monitor in the subservient role an operating system should play.

2.5.3 Resource and System Interrogation
Many programs are statically written. The devices to be used are known at programming time and the code is written accordingly. Other programs depend on user commands to control their operation. They must interrogate the system to find out specific details about a device or file it may be using.

The date can be obtained with a .DATE request, and then printed on a report or entered as a data record in a file. The time-of-day can also be determined with a .GTIM request for much the same purposes. Information about whether the job is running in the foreground or background, as well as information on the memory limits of the job and the address of its channel areas, can be obtained with a .GTJB request.

The specifics of a given file (what block it starts at, its length, what device it is on) can be obtained with a .CSTATUS request, and the specifics of a device (what controller it is, whether or not it is file structured, its block length, etc.) can be obtained with a .DSTATUS request.

These requests are most often used by programs which alter their behavior if they are using a particular device or are running in the foreground.
2.5.4 Command Interpretation

Two of the most powerful requests are those for the system Command String Interpreter (CSI). The CSI, which is part of the USR, will process standard RT-11 command strings in the form

*DEV:OUTPUT/SWITCH = DEV:INPUT/SWITCH

These are the same command strings used by RT-11 system programs.

The most commonly used CSI request is .CSIGEN. When called via .CSIGEN, the CSI will obtain a command string from the user at the terminal (or process a string in memory if desired). The CSI will then analyze the string for correct syntax (if incorrect, an error will be reported to the user and another command will be automatically solicited). It will then load the specified device handlers into memory, open the specified files, and return to the calling programs with switch information and all I/O channels active and ready for I/O. This means that with one simple macro call

.CSIGEN #DEFEXT, #DEVSPC, #0

a program can obtain a command, create or locate the name files, open the appropriate channels, and be ready to perform I/O without ever having to worry about file names, devices, or the like. Many RT-11 programs never directly open files or manipulate devices; a simple call to .CSIGEN sets up I/O and starts the program. With one macro call, a language processor such as FORTRAN is ready to do I/O from the source file and output to the listing and binary files. All user-specified switches are available to control the language processor’s operation.

Some programs desire to do their own file and device manipulation, but do not wish to provide their own command processor. For these needs .CSISP is provided. When .CSISP is used, the CSI will obtain a command string, syntactically analyze it into tabular form, and pass the tables on to the user program for appropriate action. The calling program is saved the chore of analyzing the command, but retains complete control over any device or file activity. As an example of how powerful these CSI calls are, consider the following routine. It copies, in only nine statements, a user-specified file on any input device to a file on any output device. The file names and devices for input and output are specified by the user during program execution.

```
START: .CSIGEN #DSpace, #DEFEXT, #0
LOOP:  .READW #AREA, #3, #BUFFER, #400, BLOCK
       BCS DONE
       .WRITW #AREA, #0, #BUFFER, #400, BLOCK
       BCS HERR
       INC BLOCK
       BR LOOP
DONE:  .EXIT
HERR:
```

2-23
2.5.5 File Operations

The basis of all RT-11 I/O are the device handlers. Device handlers are the monitor's device-specific routines which actually receive an I/O command. The system device and console terminal handlers are part of the resident monitor and require no further attention. All other device handlers are loaded into the user area with a .FETCH request prior to any request which might access that device.

Once the handler has been loaded, existing files can be located and opened for access with a .LOOKUP request. New files are created with an .ENTER request, which allows allocation of a specific-size disk space for the file. The option also exists to request that the monitor automatically allocate as much space as possible. When file operations are completed, a .CLOSE is done to make a new file permanent in the directory, or a .PURGE may be performed to free the channel (close the file) without involving any directory operations. Existing files can be renamed with a .RENAME request, or deleted with a .DELETE. Once the I/O on a device is finished, a .RELEASE command will remove the device handler from memory and free the space it occupied for other use. Two other requests add to the flexibility of file operations. Once an existing file has been opened with a .LOOKUP request, it can be made temporarily inactive with a .SAVESTATUS, which "remembers" the current status of the file and frees the channel for use with another file. When the time comes to access the file again, it can be re-activated on any free channel with a .REOPEN request, and I/O can continue on that channel. This feature is used for two purposes; it enables more files to be open than there are channels to keep them active, and it increases system swapping efficiency. The user keeps more files open than there are channels by performing a .SAVESTATUS on those which do not have to be active, shuffling files between active and inactive status as demand dictates.USR efficiency can be increased by locking the USR into core, opening all the files a job needs at once, .SAVESTATUSing them as they are opened, releasing the USR from core, and .REOPENing the files one at a time as they are needed. Because .REOPEN does not require any I/O, all USR swapping and directory motion for a job can be isolated in non-real-time initialization code, and many files can be efficiently manipulated at once.

2.5.6 Input/Output

RT-11 I/O operates in three modes: synchronous, asynchronous, and event-driven or completion I/O. Synchronous I/O is the simplest and consists of .READW and .WRITW requests. These requests do not return control to the user program until the specified operation is complete. When a program resumes after one of these requests, it can process the buffer, as the specified operation has been completed. Asynchronous I/O is accomplished with .READ and .WRITE requests. These requests cause the I/O command to be queued. Control is returned immediately to the calling program. It is up to the program, then, to perform a .WAIT on that channel before operating on the buffer. It is asynchronous I/O which is most commonly used for double-buffering, with a typical algorithm of
START: .READ Buffer 1
LOOP: .READ Buffer 2
WAIT Buffer 1
process Buffer 1
.READ Buffer 1
.WAIT Buffer 2
process Buffer 2
GO TO LOOP

Event-driven or completion I/O is the mode most akin to the I/O structure of the PDP-11 itself. Initiated with a .READC or .WRITC, completion I/O requests specify an extra parameter: the address of a user-written service routine to be entered when the operation is complete. The request is queued and control returns immediately to the calling program. When the operation is completed, the user program is “interrupted,” and the completion routine is entered. Completion routines perform whatever operation is appropriate to the completion of an I/O request (they can even issue an I/O request of their own), and return to the monitor, which resumes the main program where it was left off.

Completion routines are one of the most powerful features of RT-11. They are most frequently used as “event detectors,” which keep I/O going independent of main processing or respond with service to a specific external stimulus. They are a software analog to the PDP-11 interrupt structure; they require no processor time until the completion occurs, at which point they become the highest priority code in the job.

Consider as an example of completion I/O a simple “spooler” which prints a disk file on the line printer.

Main Routine

Read disk with completion routine DC (start spooling)

(processing)

Disk Completion Routine
DC: Write line printer with completion routine LP
RETURN

Line Printer Routine
LP: Read disk with completion routine DC
RETURN
The main program starts the processing by reading the first buffer load from the disk, then goes to a compute-bound job. When the disk-read completes, the disk completion routine is entered. A request is issued to print the buffer on the line printer, then the disk returns to the interrupted program. When the line printer-write completes, the line printer completion routine is entered and a request is issued for the next disk block to be read. This loop of completion routine issuing a request which results in activation of another completion routine continues until the entire file is printed. Meanwhile, the main program continues its processing, unaware of the interrupt-driven activity except that it has slightly less CPU time available.

RT-11 provides an additional I/O capability for the console terminal. Although buffers can be read or written to the terminal in the same manner as any other device, an alternative mode of I/O permits the terminal to provide character-by-character I/O more in keeping with the nature of the device itself. A TTYIN request will obtain a character from the console; a TTYOUT will print one on the console. Whole lines can be output with a PRINT request. Real-time programs can issue TTINR and TTOUTR requests, which return an indication that a character is not available or the output buffer is full, rather than waiting for their availability. The program can then resume real-time operation and try again later. A RCTRL request will force the terminal output to be reactivated should the user have typed CTRL/O to suppress it, assuring that urgent messages will be printed. The console terminal handler that services these requests also makes all the special function commands available for input and output to user programs.

Finally, RT-11 provides SPFUN, a request for performing special functions on unique devices such as magnetic tape. SPFUN requests are passed to the handler, and are used for such things as rewind or space-forward operations on magnetic tape devices.

2.5.7 Interjob Communications
The foreground/background monitor provides a mechanism for sending and receiving messages that is analogous to normal I/O. The SDAT and RCVD requests also have three modes (synchronous, asynchronous, and event-driven) which allow transfer of buffers between the two jobs as if I/O were being done. The sending job treats SDAT requests as if they were writes, and the receiving job views RCVD as a read. Receiving jobs can be "activated" when messages are sent via RCVDC completion routines, and sending jobs via SDATC completion routines. MWAIT is provided as a synchronization tool for message requests, similar to WAIT for normal I/O.

It is a common practice for one job in F/B to wish to read or write data in a file opened by the other job (the background may be processing data collected by the foreground, for example). This is accomplished via the CHCOPY request, which allows the user to obtain channel information from the other job and use that channel information to control a read or write request. Thus a background job processing foreground data might get a message from the foreground stating that the information is available on a given channel. It would then perform a CHCOPY on that channel and read from the file which was opened by the foreground.

2-26
2.5.8 Timer Support
Completion routines are also the mechanism used to provide timer support in the F/B monitor. With the .MRKT request, the user specifies the address of a routine that is to be entered after a specified number of clock ticks. Similar to an I/O completion routine, .MRKT routines are asynchronous and independent of the main program. After the specified time elapses, the main program is interrupted, the timer completion routine executes, and when done, returns control to the interrupted program.

Pending .MRKT requests—as many as the queue can hold—are numbered for identification purposes. Pending timer requests can be cancelled with a .CMKT request. .MRKT requests are normally used as a scheduling tool; multi-function jobs can “schedule” subjobs on the basis of clock “events,” detected by timer completion routines.

RT-11 allows a job to suspend itself for a specified time interval with a .TWAIT request. .TWAIT allows a compute-bound job to relinquish slices of time to the rest of the system, permitting other components to run.

2.5.9 Program Termination or Suspension
Many jobs come to an execution point where there is no further processing necessary until an external event occurs. In the F/B environment, such a job can issue a .SPND request, which suspends the execution of that job until it later executes a .RSUM request in a completion routine. While the foreground is suspended, the background will be running. When the desired external event occurs, it is often detected by a completion routine, which executes a .RSUM to continue the job where it was suspended.

When a job is ready to terminate or reaches a serious error condition, it can reset the system with .SRESET and .HRESET directives. .SRESET is a soft reset: the monitor data base is re-initialized, but queued I/O is allowed to run to completion. .HRESET is a hard reset: all I/O is stopped by a RESET instruction in the single job monitor or by calls to the handlers in F/B.

For actual termination, a job can return to the keyboard monitor with .EXIT, or initiate the execution of another program with a .CHAIN request. MACRO, for example, chains to CREF when finished to provide for the Cross REFERence listings. Files can remain open across a .CHAIN, and information is passed in memory to the chained job, so that it can adjust processing accordingly.

2.5.10 Interrupt Service
RT-11 does not require hardware for memory management or I/O management, so two program requests are used in interrupt service routines to provide necessary links to the monitor I/O system. .INTEN is the first command in every interrupt routine; it causes the system to use the system stack for interrupt service and allows the scheduler to make note of the interrupt. If device service is all the routine does, .INTEN is the only call it need make. If the interrupt routine is to do any other program requests, however (such as .READ or .WRITE), it must first force a context switch with a .SYNCH call. .SYNCH causes the remainder of the interrupt routine to be scheduled as a completion routine. When the
.SYNCH is finished the completion routine can execute programmed requests, initiate I/O, resume the mainline code or schedule a subjob.

2.6 SYSTEM PROGRAMS
As a comprehensive program development and operating system, RT-11 provides an impressive set of system programs to assist in user program development and debugging. These include a text editor, macro expander, language translators (MACRO, ASEMML, FORTRAN, BASIC, FOCAL), linker, librarian, and debugging utilities (ODT, PATCH, PATCHO). In addition, the system programs include several file utility programs. This section describes the standard system programs. Section 2.7 describes the languages.

2.6.1 EDIT Interactive Editor
The RT-11 editor, EDIT, is an interactive, character-oriented text editor with all the advantages of character searches and string manipulation that a character-oriented editor offers.

EDIT has essentially three modes: character commands for character-level operations, line-oriented commands for those operations suited to lines, and scope mode, which allows users of VT11 displays to interactively edit text with the aid of the screen. Features of EDIT above and beyond normal text editors include iteration brackets, which allow repetition of portions of the command as often as desired, command macros, which allow specification of a series of operations to be used over and over again as a discrete command, and the ability to save and move blocks of text.

In scope mode, EDIT displays a window of text around the area currently being edited, and provides single-stroke commands for the manipulation of the text file. Characters typed are entered directly into the file and changes appear simultaneously on the screen. Scope users can also employ normal editing commands, enjoying the added benefit of visual verification of text modifications and cursor positioning.

2.6.2 LINK Linker
The outstanding feature of the RT-11 linker is its transparent overlay scheme. The linker provides standard capabilities such as automatic library searches, the ability to output core image, foreground-relocatable or absolute binary program files, and a variety of link-time library manipulation capabilities.

Overlays in RT-11, however, require no assembly language instructions or macro calls. The overlay structure of a program is specified at link time and can be altered simply by altering the linker command strings (no source changes involved). It is only necessary to follow a few simple conventions as routines are written; (the restrictions amount to good modular programming rules). The routines are then combined in any number of possible overlay structures.

For example, assume main program A, subroutines B and C which are called by A, and subroutines D and E, called by subroutine B.

The user can generate a program image which includes all five modules if abundant memory is available with the command:

*PROG = A,B,C,D,E
As memory gets more restricted, D and E could share an overlay region by relinking with the command string:

\[
\begin{align*}
\ast \text{PROG} & = A,B,C/C \\
\ast D/O:1/C & \\
\ast E/O:1 & \\
\end{align*}
\]

Finally, the program could be made as small as possible by overlaying B and C as well as D and E with command strings

\[
\begin{align*}
\ast \text{PROG} & = A/C \\
\ast B/O:1/C & \\
\ast C/O:1/C & \\
\ast D/O:2/C & \\
\ast E/O:2 & \\
\end{align*}
\]

2.6.3 LIBR Librarian
The RT-11 Librarian (LIBR) allows the user to create, update, modify, list, and maintain library files.

LIBR provides the user with the capability of maintaining libraries composed of commonly-used functions and routines.

Each library is a file containing a library header, library directory (or entry point table), and one or more object modules. The object modules in a library file may be routines which are repeatedly used in a program, routines which are used by more than one program, or routines which are related and simply gathered together for ease in usage. The contents of the library file are determined by the user’s needs. An example of a typical library file is the FORTRAN library provided with the FORTRAN package, which contains all the mathematical functions needed for normal usage.

A program requests a library module through a subprogram call to a global entry point. If this global entry point (subprogram name) is defined in another module passed to the Linker, the request is said to be satisfied. When a library is passed to the Linker, it is searched for entry points to match unsatisfied requests. The modules which satisfy requests are linked in; all other modules in the library are ignored.

2.6.4 ODT On-Line Debugger
RT-11 On-Line Debugging Technique (ODT) is a system program that aids in debugging assembled and linked object programs. From the keyboard, the user interacts with ODT and the object program to:

- Examine and modify the contents of any location in main memory, the UNIBUS, the registers, the processor status register, or ODT internal registers.
- Run all or any portion of an object program using the breakpoint feature.
- Search the object program for specific bit patterns.
- Search the object program for words which reference a specific word.
- Calculate offsets for relative addresses.
- Fill a single word, block of words, byte or block of bytes with a designated value.
The assembly listing of the program to be debugged should be readily available when ODT is being used. Minor corrections to the program can be made on-line during the debugging session, and the program may then be run under control of ODT to verify any changes made.

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the Single-Job Monitor, ODT can be linked with the program.

To debug a program in the foreground area, it is recommended that ODT be run in the background while the program to be debugged is in the foreground.

2.6.5 PATCH Code Patch Utility
The PATCH utility program is used to make code modifications to memory image (.SAV) files, including overlay-structured and monitor files. PATCH, like ODT, can be used to interrogate, and then to change, words or bytes in the file.

2.6.6 PATCHO Object Patch Utility
The RT-11 PATCHO program is used to correct and update object modules (files output by the assemblers or by the FORTRAN compiler). It is particularly useful when making corrections to routines that are in .OBJ format for which the source files are not available. PATCHO cannot be used to patch libraries built by LIBR, but it can be used to patch the .OBJ modules from which a library is built.

2.6.7 PIP Peripheral Interchange Program Utility
A very commonly used RT-11 system program is PIP, the file transfer and maintenance program. PIP is the "system utility"; besides providing file-oriented operations, PIP offers several functions which are used in system building or control. PIP functions include the ability to copy files individually or in groups; operations which can extend, delete, or rename files; commands which list device directories; commands to initialize devices with directories or system bootstraps; the ability to bootstrap any supported system device; and the ability to scan a disk for bad blocks. Finally, one of the more important functions provided by PIP is the ability to consolidate the free space on a device into one area, making more efficient use of RT-11's contiguous file structure.

2.6.8 SRCCOM Source Compare Utility
The RT-11 Source Compare program (SRCCOM) is used to compare two ASCII files and to output any differences to a specified output device. It is particularly useful when the two files are different versions of a single program, in which case SRCCOM prints all the editing changes which transpired between the two versions.

2.6.9 FILEX File Exchange Utility
FILEX is a general file transfer program used to convert files among file-structured devices for various operating systems. Transfers may be initiated between any file-structured RT-11 device and PDP-11 DOS/BATCH DECTape (as input or output), DOS/BATCH disk, (as input only), RSTS-11 DECTape (as input or output), and DECSYSTEM-10 DECTape (as input only). Files are transferred as 16-bit binary data. No processing is
done on the data itself except that which is necessary to convert between various data representations.

2.6.10 DUMP File Dump Utility
RT-11 DUMP prints the contents of all or any part of a file on the console or the line printer. DUMP can print the file contents in any one of four selected formats: octal words, octal bytes, ASCII characters, or RAD50 characters.

2.7 LANGUAGES
RT-11 supports a wide variety of programming languages. The user can select any one of four languages to solve a particular application problem. Users can implement entire problem solutions using FORTRAN IV, BASIC or FOCAL, or can combine any of these languages with MACRO assembly language modules.

2.7.1 MACRO Assembler
The RT-11 MACRO assembler is a powerful, general-purpose macro assembler, the primary implementation tool for assembler language programs. MACRO is a two-pass assembler requiring an RT-11 system configuration (or background partition) of 12K or more words. Some notable features of MACRO are:

- program control of assembly functions
- device and filename specifications for input and output files
- error listing on command output device
- alphabetized, formatted symbol table listing
- relocatable object modules
- global symbols declaration for linking among object modules
- conditional assembly directives
- program sectioning directives
- instruction repetition directives
- macro definition directives for user-defined macros
- comprehensive set of system macros
- extensive listing control, including cross-reference listing; can be specified in the command string or in source program

MACRO also allows an ASCII file or macros, called a macro library, to be edited and used. Normally, DIGITAL’s system macro library is used to support macro calls to the monitor; these can be conveniently combined with user-defined macro libraries with a simple editing operation.

2.7.2 EXPAND Macro Expander and ASEMBL Assembler
Because the MACRO assembler requires 12K words of memory to operate, a pair of programs is provided which gives a subset of macro capabilities to 8K users. EXPAND is a one-pass macro expansion utility that takes a macro source file as input, expands the macro calls into simple assembler source statements, and outputs a macro-free assembly language program as an ASCII file. This source file is then input to the two-pass assembler, ASEMBL, which runs in 8K. The effect is that EXPAND and ASEMBL combine to provide a considerable subset of
MACRO's capability; enough of a subset to handle the system macro library and user macros which use a subset of the macro language.

2.7.3 FORTRAN
FORTRAN/RT-11 is an extended, optimizing FORTRAN IV system which operates on any RT-11 system. The FORTRAN IV compiler processes source programs extremely rapidly; typical 300-line programs compile in less than nine seconds on a PDP-11/45 and in less than 25 seconds on a PDP-11/10.

Extensive optimizations such as common subexpression elimination, array vectoring, and "peephole" local code sequence tailoring, decrease the size and increase the speed of object programs. The compiler produces the object code directly, without using temporary files, and does not require an intermediate assembly step, thus speeding program development time.

The FORTRAN IV system is designed to minimize the size of executable programs. The entire system (including the compiler and optimization capabilities) is completely functional in the minimum 8K RT-11 environment. Any optional arithmetic hardware (EAE, EIS, FIS, FPP) may be utilized.

FORTRAN programs may be developed under RT-11 and output in absolute binary format for execution on a satellite machine with minimum peripherals. Only a device such as a paper tape reader is required for program loading.

Using SYSLIB, the RT-11 FORTRAN system subroutine library, all features and services of the RT-11 monitor are available to the FORTRAN programmer without the need for assembly language coding. FORTRAN programs may schedule subroutines to be executed when an external event occurs (the receipt of a message from the other job, the completion of an I/O transfer, the lapping of a specified time interval); perform all types of monitor-level input/output and system informational calls; handle interrupts with a FORTRAN subprogram.

SYSLIB also contains extensive string manipulation routines in addition to routines for calling the monitor functions previously described. These routines create strings in LOGICAL*1 arrays, and allows their manipulation.

CONCAT to concatenate two strings together.
GETSTR to input a string.
INDEX for locating substring X in string Y.
INSERT for inserting one string into another.
LEN for determining the length of a string.
PUTSTR will output a string.
REPEAT will repeat strings.
SCOMP for string comparison and sorting.
SCOPY will copy a string.
STRPAD to pad a string with blanks to a specified length.
SUBSTR extracts a substring from a larger string.

TRANSL will replace one string with another after directed character modification.

TRIM to remove trailing blanks.

VERIFY to test whether characters in one string appear in another.

The real power of these functions is that they operate on variable-length strings. The strings can be manipulated fully without knowledge of their length, adding a new dimension to FORTRAN capabilities.

FORTRAN/RT-11 also provides a complete set of graphics subroutines for use with the VT11 Graphics Display Processor or the GT40, GT42, or GT44 Graphics Display Systems. An optional plotting package allows hardcopy graphics with the LV11 Electrostatic Printer/Plotter.

A complete library of FORTRAN-callable subroutines supports laboratory data acquisition and control using the LPS-11 or AR11. Maximum control of parameters and the asynchronous operation of the library routines allow the laboratory user to write applications completely in FORTRAN. An optional subroutine package provides mathematical and statistical routines commonly required in scientific programming.

2.7.4 FOCAL
FOCAL is an easy-to-learn interactive language. The RT-11 implementation is as an interpreter, which provides both stored program and immediate mode operations. Commands may be abbreviated to a single letter. Alphanumeric symbol names are provided, with up to six characters carried in the symbol table, only two of which are significant to FOCAL. FOCAL utilizes the same floating point package as does FORTRAN/RT-11, so all arithmetic options are supported; a double-precision version of FOCAL supports up to 17 digits of accuracy.

Any peripheral supported under RT-11 is available to the FOCAL user. The LIBRARY command allows the user to access any RT-11 file-structured device. Programs may be saved, loaded, started and/or CHAIN’d. Data may be saved/accessed in sequential files and/or virtual files. (Files may be treated by the FOCAL user as a virtual array; data types include floating, double-precision floating, signed or unsigned integers, or byte data.)

Other features include scheduling up to eight asynchronous tasks from the clock; processing interrupts in the FOCAL language; user-controlled error processing, and the facility for one or more user-written assembly language functions.

2.7.5 Single-User BASIC
BASIC/RT-11 is best suited for interactive applications. Machine resources (memory and execution time) are expended with resulting savings in programming time. BASIC/RT-11 has several features which add to its usefulness as a development tool. It is implemented as an incremental compiler; source statements are translated into a more compact, easily executed code and stored directly in memory. When the RUN command is issued, this easily interpreted internal language is executed.
as a program. When the LIST or SAVE commands are used, the internal form is translated back to the original ASCII for output. Like all Dartmouth-compatible BASIC implementations, statements are entered directly to BASIC one at a time (or the editor can be used), then executed as a program. Statements can also be executed immediately simply by typing commands without statement numbers. This immediate mode can be used for debugging, development, or even as a calculator. Immediate mode is a valuable debugging tool, because statements like

```
PRINT V1,Q
```

or

```
LET V1=3
```

allow the user to investigate the state of various program variables, change them, or insert statements in the middle of program execution.

BASIC/RT-11 supports two types of files, sequential and virtual. Sequential files are ASCII files of string or numeric data that are accessed sequentially. To get to item N, items 1 through N-1 have to be read first. Virtual files behave like standard arrays; string or numeric elements are accessed by subscript as if they were memory-resident. If the desired element is in the buffer, no I/O takes place; if not, the disk block containing the desired element is read, invisible to the program. Virtual files have a 256-word "cache;" access of elements near each other minimizes overhead, while I/O overhead increases as elements become more random.

Note that virtual files in immediate mode can be used to manipulate or "edit" a data base. If, for example, a file of 2000 data points is taken from A/D and is to be examined, commands like

```
OPEN "DATA" AS FILE VF1(2000)
```

and

```
PRINT VF1(137)
```

allow points to be selectively examined, and commands like

```
LET VF1(137)=0
```

are used to change them.

Beside numeric data, BASIC allows variable-length alphanumeric string variables. These string variables can also be used in both types of files.

For interface of user-written and special-purpose functions, BASIC/RT-11 supports the CALL statement. CALL allows the user to invoke a function by name and pass it any number of arguments. For example

```
CALL "RDOT" (S, Y)
```

is the command used to place a point on a VT11 display. CALL is a convenient interface for user-written assembly-language functions; it is also the mechanism used to support several special-purpose devices in BASIC. The VT11 display processor, LPS11 laboratory peripheral sys-
tem, and LV11 printer/plotter are all supported by a set of CALL func-
tions which make these devices easy to manipulate in a BASIC program.

In addition to these features, BASIC/RT-11 programs execute much
faster than traditional interpreters, although programs written under
FORTRAN execute considerably faster.

2.7.6 Multi-User BASIC
MU BASIC/RT-11 is an extended version of BASIC/RT-11 that allows an
RT-11 system to support up to eight interactive BASIC users on single-
job systems with 24K words of memory, or up to four users on single-job
systems with 16K words of memory. Under the foreground/background
monitor, up to four BASIC terminals can be supported on 28K word
systems.

All terminal I/O is performed by the MU BASIC language processor.
Users are optionally provided with accounts and passwords for file pro-
tection.

MU BASIC extends the BASIC/RT-11 language with additional statements
as well as commands and functions. In particular, some additional state-
ment features are:

COMMON Allows a program to pass information to a chained
       program.
PRINT-USING Provides extensive output formatting capabilities, in-
       cluding the ability to print exponential numbers and
dollar amounts with asterisk-fill protection.
ON-GOTO Transfers control to one of several lines of the pro-
            gram based on one or more conditions.
ON-GOSUB Transfers control to a subroutine based on the con-
            dition specified in the statement.

Unlike single-user BASIC/RT-11, however, MU BASIC/RT-11 does not
provide language support for the LV11 Printer/Plotter, the LPS11 Lab
Peripheral Subsystem, or the VT11 Display Subsystem.

2-35
CHAPTER 3

RESOURCE-SHARING TIMESHARING SYSTEM
RSTS/E

3.1 FUNCTIONS AND FEATURES
RSTS/E is a timesharing system that allows up to 63 simultaneous users to process data in interactive mode using the BASIC-PLUS language processor. As options, the system supports a COBOL language processor and PDP-11 SORT file sort utility and a FORTRAN IV compiler system. BASIC-PLUS jobs can range from small desk calculator jobs to large 16K word programs and can use chaining and interjob communication features to execute even larger programs. COBOL programs can have a maximum size of 28K words. RSTS/E also includes a comprehensive set of easy-to-use system utilities for the system manager and timesharing users.

As a general rule, RSTS/E can support up to 63 jobs on a PDP-11/70, up to 32 jobs on a PDP-11/45 and up to 24 jobs on a PDP-11/40 central processor. The actual number of jobs a configuration can support depends on the application's characteristics and the amount of memory available. RSTS/E requires at least 32K words of parity memory to run BASIC-PLUS jobs only, or at least 64K words to run both BASIC-PLUS and COBOL or FORTRAN IV jobs. A minimum peripheral complement includes a console terminal and a disk system. The system device can be a single RP02, RP03 or RP04 disk system or a dual drive RK11 disk system. A TU16 or TM11 magnetic tape or TC11 DECTape system is required for system generation and back-up unless the disk system is a dual drive RK11 system.

On a PDP-11/40 or PDP-11/45, memory can expand up to 124K words; on a PDP-11/70, up to 1024K words. In addition, RSTS/E can support multiple disk, DECTape, magnetic tape and floppy disk drives, multiple line printers, a card reader, a paper tape reader/punch, and a variety of terminal interfaces. All of these devices can be available to any terminal user. The terminals can be accessed under program control for input and output. A single program can control any number of terminals up to a maximum of 64.

RSTS/E users can expect efficient operation because the operating system dynamically allocates processor time, memory space, file space and peripherals to best suit changing demands. The system manager and designated privileged users have access to the monitor's system management commands either interactively using system utilities or under program control using BASIC-PLUS. Additional system commands and utility programs are also available to all users.

The RSTS/E file system provides a wide range of on-line processing capabilities. Files can be accessed randomly or sequentially. They can
contain alphanumeric string, integer numeric, or floating-point numeric data. Files can be created, updated, extended or deleted interactively from the user’s terminal or under program control. Files can be protected from access on an individual, group or system basis. Files can also be accessed by many users while being updated on-line.

RSTS/E provides the ability to back-up files selectively or totally. Backup can be done on-line without disrupting users or it can be done off-line.

Table 3-1 summarizes the components of RSTS/E.

<table>
<thead>
<tr>
<th>Table 3-1</th>
<th>RSTS/E System Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>System type</td>
<td>General timesharing system using the BASIC-PLUS interpreter with optional COBOL and FORTRAN IV language support.</td>
</tr>
</tbody>
</table>
| CPU’s supported | PDP-11/40 with Extended Instruction Set and Memory Management  
PDP-11/45 with Memory Management Unit  
PDP-11/70 |
| Memory ranges | Minimum: 32K words parity memory for BASIC-PLUS only  
64K words parity memory for BASIC-PLUS, FORTRAN IV and COBOL  
Maximum: 124K words on PDP-11/40,45  
1024K words on PDP-11/70 |
| Minimum peripherals | Console terminal  
Disk system: RP02, RP03, RP04, dual drive RK11 TU16 magnetic tape, TM11 magnetic tape or DECTape systems (not required with two RK05 drives) |
| Additional CPU hardware | PDP-11/45 or PDP-11/70 FP11-B Floating Point Processor  
PDP-11/40 KE11-F Floating Instruction Set |
| Additional peripherals | Up to eight line printers (LP11, LP05, LS11 or LV11)  
Up to eight RK05, RP02, RP03, or RP04 disk drives  
Up to four RS03 or RS04 disk drives for swapping  
Card reader (CR11 punched, CM11 marked or CD11 high-speed punched card reader)  
PC11 paper tape reader/punch  
Up to 16 KL11, DL11-A, B, C, D or LC11 local lines or DC11, DL11-E auto-answer dataset interfaces  
Up to 63 lines on DH11-AA terminal multiplexer with DM11-BB modem controls or EIA line adaptors |
| Optional software | COBOL Language Processor with PDP-11 SORT  
FORTRAN IV Language Processing System |
Table 3-1  RSTS/E System Summary (Cont.)

RSTS-11 SORT File and Index Sort Program
   (written in BASIC-PLUS)
RSTS/E Commercial Extensions Package
RSTS/2780 Remote Job Communications
   Package
DECAL Computer-aided Instruction Package
WISE College Administration DBMS Package
PICTURE BOOK Graphics Development System

3.2 SYSTEM CONFIGURATION AND OPERATION
RSTS/E system software exists as either system code, language code, or system program code. The system code and part of the language code is tailored at system generation time according to the hardware configuration on which the system runs and the software features which are chosen by the system manager. Once the system is generated, the system code and part of the language code are frozen and alterable only by patching or generating new code. The system program code exists in a library of programs executable by the system software or by individual users on the system. The library of programs is alterable and expandable during timesharing without requiring regeneration of the system.

System Code
The RSTS/E system code is stored on the system disk as a core image library (CIL). A core image library, when loaded into memory, is immediately executable by the PDP-11 computer. The system code comprises many distinct elements which are either resident in memory or on disk during timesharing. Permanently resident elements are the following:

- interrupt and trap vectors
- small and large system buffers
- system information and data tables
- disk and device drivers
- file processor modules

The following are disk resident (overlay) elements.

- file processor modules
- infrequently used utility routines
- system initialization code

RSTS/E operations start when the system disk is bootstrapped. The bootstrap routine loads the initialization code which performs many consistency checks to ensure the integrity of the software. When checking is completed, the initialization code remains resident and allows many options, some of which are described in section 3.2.3.

When timesharing operations are started, the initialization code is overlaid by the permanently resident system code and the BASIC-PLUS compiler and Run Time System. As timesharing operations proceed, in-
frequently used overlay code and system and user programs are loaded from disk as needed.

**Language Processors**
The language processors reside on the system disk in machine executable form and can be either permanently resident in memory or temporarily resident (swappable). The following are permanently resident elements.

- BASIC-PLUS text editor and analyzer
- BASIC-PLUS incremental compiler
- BASIC-PLUS run time system

The temporarily resident elements are the following:

- RTSLIB auxiliary run time system for COBOL
- COBOL compiler and object time system
- PDP-11 SORT program
- FORRTS auxiliary run time system for FORTRAN
- FORTRAN IV compiler, object time system and utilities

The BASIC-PLUS code is loaded into memory at the start of timesharing operations and remains resident during the session. The code analyzes all BASIC-PLUS statements and generates and executes intermediate (compiled) code. Many monitor services are available to a BASIC-PLUS program through system function calls.

The auxiliary run time system RTSLIB is loaded into memory only when a request is made to execute the COBOL compiler and object time system or the SORT program. RTSLIB remains resident until all COBOL or SORT requests are satisfied and afterwards is overlaid by system and user programs. The COBOL compiler and SORT program are swapped out to disk with the user job image.

The auxiliary run time system FORRTS is loaded into memory to execute the FORTRAN compiler or FORTRAN utility programs. It requires 4K words of memory and is shareable among users.

**System Program Code**
A library of programs is produced and stored on disk during the system library build procedures of system generation. Both the system and users execute these programs to perform system housekeeping and common utility functions. The system manager can use the programs to monitor and regulate system usage. Some library programs can be tailored by altering the source statements supplied by DIGITAL and recompiling to replace the current copy on the system disk.

**3.2.1 Timesharing Operations Overview**
To begin a timesharing session, a user logs into the system by entering an account number and password at a terminal. The user is assigned an account number and password by the system manager.

Immediately after logging into the system, a user's terminal is in edit mode (BASIC-PLUS command level) and is returned to edit mode when
any program execution is compiled or whenever a CTRL/C is typed at
the terminal. In edit mode, the system examines each ASCII text line
entered by the user and determines whether that line is a system com-
mand, an immediate mode statement, or a program statement. System
commands are executed immediately after being entered. Immediate
mode statements are first translated into an intermediate code, which is
placed in the user's job area, and are executed immediately by the Run
Time System. Program statements (lines of ASCII text preceded by line
numbers) are stored in their ASCII form in a temporary disk file under
the user's account. Each program statement is also compiled into its
intermediate code representation, which is placed in the user's area of
memory.

A user job area is initialized at log in time and set to a size of 2K words.
The job area can grow in increments of 1K words to a maximum size
set by the system manager at the start of timesharing operations. Inter-
mediate code created in the user's job area upon entry of program state-
ments in edit mode is not executed automatically. The related program
statements being created can be changed. A copy of the intermediate
code of the program can be transferred to disk storage or to an external
storage medium.

A user changes from edit mode to run mode by typing the RUN system
command or the CHAIN immediate mode statement. In run mode, the
Run Time System interpretively executes the intermediate code stored
in the user's job area. When a program finishes execution, the terminal
is returned to edit mode, signalled by printing of the READY message.
The user can interrupt the Run Time System by typing CTRL/C, which
also returns the terminal to edit mode.

When the terminal is in run mode, the user can detach the running job
from the terminal. This allows the user to log in again, open up another
job area, and run another job. The detached job runs unattended, but is
still associated with the account under which the user logged in. To re-
gain control of a detached job running under account, the user can log
in on any free terminal and attach the job to that terminal.

The RSTS/E system allows jobs to run (in either edit mode or run mode)
one at a time. A job runs until it either enters an I/O wait state or ex-
hausts the time quantum which either the system or the system manager
has assigned to it. At the point when the currently running job ceases
to run, the scheduler finds the next job that is ready to run and begins
running that job. Meanwhile, the interrupt-driven I/O device handlers
are processing requested data transfers. Upon completion of a transfer,
the scheduler marks the job that requested the transfer as ready to run
again and starts it from the point at which execution ceased.

RSTS/E attempts to keep as many jobs in memory as possible. When
more memory is required to run a job than is available, the system tem-
porarily swaps some jobs out of memory and stores them in one of the
swap files defined by the system manager.

When it is again their turn to run, the jobs in one of the swap files are
swapped back into memory. Jobs waiting for keyboard input and jobs
waiting for device I/O completion are most likely stored in the swap
files, while jobs currently running or involved in disk or magtape data transfers are necessarily in memory.

As system processes each job, it maintains accounting information in memory concerning that job. When the job is logged off the system, this information is used to update the accounting information stored on the disk for that account.

3.2.2 System Generation
System generation is normally a one-time operation in which the system manager defines the hardware configuration and selects the basic software options. The system manager only needs to perform a system generation when the system is first installed or when the hardware configuration changes. Software options can be included in the system to increase processing power or can be excluded from the system to conserve memory.

In addition to defining the number and kinds of peripherals and processing hardware during system generation, the system manager defines special configuration options. Some of these options are discussed below.

Pseudo Keyboards
The system manager can define the system to have one or more pseudo keyboards. A pseudo keyboard is a non-physical device that has the characteristics of a physical terminal but that has no terminal associated with it. As such, a pseudo keyboard has both input and output buffers to which a program can extract output. Using a pseudo keyboard as a communications device, a user can write a program to control other jobs. In addition, each copy of the BATCH system program requires one pseudo keyboard to run jobs in a batch stream. If the installation plans to run several copies of BATCH simultaneously, at least that number of pseudo keyboards must be defined.

Multiple Terminal Service
The multiple terminal service option allows one BASIC-PLUS program to simultaneously interact with several users by servicing their terminals on one I/O channel. This eliminates the need to run separate copies of the same program when several terminals must perform a similar function.

Maximum Number of Jobs
With sufficient hardware, RSTS/E can start up to 63 simultaneous jobs. The maximum number of jobs that can be run efficiently depends on the available memory space and the number and types of disks on the system. When a job is started, it is given a number by the system. Jobs are numbered sequentially from one to the maximum number of jobs the system can handle. Jobs include both attached jobs and detached jobs. The maximum number of jobs must be specified at system generation since it determines the size of some monitor tables. The number can be lowered during system initialization to adjust to changing requirements, but it can not be increased above the configured maximum unless the system is regenerated.
Concise Command Language (CCL)

CCL commands allow a user to enter one command that runs a system utility and specifies a single command for the utility to execute (refer to section 3.4). The system can have a total of 20 CCL commands defined. The standard system includes 10 commands for the RSTS/E utility programs. During system generation, the system manager has the option of including the standard set of commands; define a completely new set of commands; or add new commands to the standard set.

The precedence of CCL commands is above that of RSTS/E commands and BASIC-PLUS immediate mode statements. As a result, the system manager can control the use of a command or immediate mode statement. For example, the system manager could define a CCL command named BYE that performs certain operations before allowing a user to log off the system. As another example, the system manager could define a PRINT command that performs operations different than those of the BASIC-PLUS immediate mode PRINT statement. The CCL command has no effect on a BASIC-PLUS statement preceded by a line number since a line numbered statement has a higher precedence than the CCL command.

Floating Point Precision and Scaled Arithmetic

The system manager can select either single precision (2-word) or double precision (4-word) floating point numeric format. If the system has floating point hardware, the system manager can select floating point math packages that increase processing speed by using the hardware instructions. The scaled arithmetic feature is included in all 4-word floating point math packages. Scaled arithmetic avoids any loss of precision in floating point calculations; it is therefore very useful in calculating sums of money that cannot be manipulated easily as integer quantities.

3.2.3 System Initialization

After generating the system, the system manager boots the RISTS/E system to load the initialization code into memory. The (INIT) code is a collection of routines used to create the file structures, system files, and start up conditions required for normal operation of the RISTS/E system. The INIT code is essentially one large stand-alone program with many functions. Immediately after a system generation, several options must be used before the RISTS/E system can be brought up for timesharing. Thereafter, the initialization code provides the mechanism for altering critical system files and parameters as installation requirements change. INIT includes routines which ensure the integrity of the system disk file structure and provide many checks on the hardware configuration. Options are provided which enable the system to function even when certain hardware elements are inoperative. Finally, the initialization code is responsible for loading the RISTS/E Monitor and BASIC-PLUS Run Time System into memory for normal timesharing operations.

Once the default system initialization and start-up parameters are set up, the system manager does not have to repeat manual start-up each time the system is started. The system manager can create control files that perform timesharing start-up automatically. When the system is
started in automatic restart mode, control by-passes all parts of the start-up code that call for operator intervention. Using the restart feature, the RSTS/E system can recover and restart the timesharing session automatically after a system malfunction or power failure.

After system generation, however, or if the system manager chooses to reset system parameters, the system manager must run the initialization code options. Some of these options are:

REFRESH Allows the user to create or rebuild the system files such as the monitor core image, storage allocation table, system error messages texts, and swap files.

DEFAULT Allows the user to establish or change the default start up conditions such as the maximum number of jobs which can be run and the maximum size of a job. It also allows the user to relocate the BASIC-PLUS Run Time System into high-speed semi-conductor memory if it is available.

START Brings the RSTS/E system up for normal timesharing operations. In addition, START allows the user to set the maximum number of jobs, maximum job size and memory relocations parameters to override the DEFAULT specifications for this timesharing session only.

3.2.4 System Management Utility Programs
RSTS/E includes utility programs for both the system manager and general user. Some system management utilities are privileged programs and can be run only by the system manager. Other utilities are not privileged and can be run by the general user, but have privileged features that can be executed only by the system manager.

System management utilities include: initialization and maintenance programs, resource management and accounting programs, system error logging and analysis programs, spooling programs, and user communication programs.

System Initialization and Maintenance

INIT Controls system startup operations. This includes mounting disks, enabling LOGINs, starting system programs, and sending messages to terminals.

SHUTUP Performs an orderly system shut down operation.

UTILITY Allows the system manager to: enable/disable LOGINs, broadcast messages, kill a job, reset system date and time, mount and dismount private disks, lock and unlock disks, clean disks, zero a user's account.

TTYSET Sets terminal characteristics.

SYSTAT Monitor system status, including active jobs, device assignments, and detached jobs.

VT5DPY or VT50PY Displays the system status on a VT05 or VT50 DECscope and updates the status at given intervals.

PRIOR Reports and allows the system manager to change the priority, run burst and maximum size assigned to an existing job.
Resource Management and Accounting

DSKINT Initializes a disk for use on a RSTS/E system.
REACT Creates or deletes user accounts on disks.
UMOUNT Allows the user to mount or dismount disk packs.
SYSCAT Prints a current directory listing of any disk.
MONEY Extracts system accounting information for any selected or all accounts; accounting information includes amount of CPU time used, the KCT factor (usage of 1K words of memory per one-tenth of a second), amount of connect-time, device usage time, and disk storage usage.

Spooling Programs

QUEMAN Manages the queuing of jobs to spooling programs. Collects queue requests, maintains job queues and allows the system manager to check the status of spooled jobs currently running of the system.
SPOOL Transfers files from disk, DECtape or magtape to a line printer, as determined by the job queue maintained by QUEMAN.
BATCH Executes files containing batch job commands that have been queued on a batch device, as maintained by QUEMAN.

Error Logging and Analysis

ERRCPY Retrieves error-related data logged automatically by the RSTS/E monitor. Upon occurrence of a hardware error, Monitor routines save the contents of the device registers and send a message to ERRCPY to retrieve the data.
ERRCRS Retrieves error-related data saved following a system crash.
ERRDIS Produces summaries of error-related data and formats it for output to a terminal or line printer. Allows the system manager to obtain a full or partial history or summary of the error-related data preserved by the ERRCPY and ERRCRS programs.
ANALYS Retrieves and reports on the critical contents of memory obtained when a system crash occurs.
ODT Allows the system manager to open a file, a device, or memory as an address space and examine or change word or byte contents. The user can also list the contents of system table locations.

User Communication

GRIPE Allows the general user to communicate comments about the system to the system manager.
PLEASE Enables the user to send a message to the system console terminal. For example, a user can request to the operator to physically mount a particular tape on a tape drive.
TALK Enables users to broadcast messages to each other’s terminals.
3.3 DEVICE AND FILE CONVENTIONS

RSTS/E provides a device access structure that allows many users to share the resources of the system in a consistent manner. This section describes the device and file naming conventions, the public and private disk structures, and the account system used by RSTS/E.

3.3.1 File Specifications

The file specification for any user-identifiable collection of data is completely described by some or all of the following information:

dev:[proj.prog]filnam.ext<prot>

where "dev:" is a physical or logical device name, "[proj.prog]" is a user account number, "filnam" is a user-specified filename, "ext" is a filename extension, and "<prot>" is a file protection code.

For non-file-structured devices such as paper tape, line printer, or terminal devices, only the device designator is required in a file specification. For file-structured devices such as disk, DECTape or magnetic tape, RSTS/E requires that the user at least specify a filename in addition to the device designator. Filename extension, account number and protection code all have system defaults, and need only be specified if the system default is not to be used to identify the file.

RSTS/E recognizes the following default extensions:

- .BAS BASIC-PLUS source program file (ASCII format)
- .BAC BASIC-PLUS compiled program (binary format)
- .CBL COBOL source program file (ASCII format)
- .FOR FORTRAN source program file (ASCII format)
- .MAC MACRO source subprogram file (ASCII format)
- .OBJ COBOL, FORTRAN or MACRO compiled object program file (binary format)
- .SAV FORTRAN executable program file (binary format)
- .LST Listing file
- .DAT Data file
- .DIR Directory file
- .CTL Batch control file containing batch commands
- .LOG Batch log file
- .TMP Temporary file

The account number field (containing the project and programmer numbers) identifies the owner of the file. If it is omitted, the owner is assumed to be the current user. This field is meaningful only for disk and magtape files; it has no significance for DECTape files or files on non-file-structured devices.

The account number can be represented by special characters to indicate special system or user-defined accounts. For example, use of the $ character (dollar sign) in the project programmer field indicates that the file is stored under the system library account ([1, 2]), where all standard utility programs are stored. Other special account number characters are:
Account [1, 3] or installation-defined account
Account [1, 4] or installation-defined account
Account [1, 5] or installation-defined account
Account [n, 0] or "n" is the current account project number
Assignable account

The accounts associated with the !, % and & characters can be changed during system installation. The # character is unique because the system interprets it according to the account under which the user is running. For example, if the user is running under account [10, 20] and specifies the # character, the system interprets it to mean account [10, 0]. This feature allows each project on the system to have its own library of files.

When creating or renaming a file, a protection field can be specified. Files can be read and/or write protected against three classes of users where distinctions are made on basis of the project and programmer number of the user attempting to access the file. The three classes of users are:

owner;

group, all users having the same project number as the owner (termed the owner's group); and

others, all other users not in the owner's group

The protection code assigned to a file consists of a selected sum of the following numbers:

1  Read protect against owner
2  Write protect against owner
4  Read protect against owner's group
8  Write protect against owner's group
16 Read protect against all others
32 Write protect against all others
64 Compiled run-only program file
128 Privileged program file

For example, in creating a compiled BASIC-PLUS file, a default protection code of <124> is supplied. This permits only the owner to access the file, since 124 = 64 + 32 + 16 + 8 + 4.

3.3.2 Data Formats
Under BASIC-PLUS, RSTS/E allows users to store data in any of three formats:

STRING  A sequence of ASCII characters treated as a unit. One ASCII character is stored in one byte and strings are normally variable-length.

INTEGER A number in the range -32768 to +32767. Integers are stored in two bytes in two's complement representation. Integer operations provide economies in space as well as increases in processing speed over floating-point operations.
FLOATING-POINT A number approximately in the range of $10^{-38}$ to $10^{+38}$. Floating point numbers can either be stored in 2-word format, which allows up to seven decimal digits of precision, or 4-word format, which allows up to 17 decimal digits of precision.

To perform decimal calculation on a system having 4-word floating point numeric storage, the user has an option to scale the numbers stored in the system. The user can specify the number of decimal places in fractional numbers by use of the SCALE system command (section 3.4.1).

With the scaled arithmetic feature, the scale factor can be set to an integer value between 0 and 6. The system uses the scale factor to preserve the accuracy of fractional numbers to the selected number of decimal places. The value 0 is used to disable the scale factor, and allow the system to perform calculations using standard double precision floating point arithmetic.

With a scale factor between 1 and 6 in effect, the system, upon input of a floating point number, internally moves the decimal place the selected number of places to the right and rounds it to an integer. The system performs all subsequent calculations with the floating point integers and, in turn, translates the result of each arithmetic operation into a floating point integer with the selected scale factor. On output the system moves the decimal point to the left of the selected number of places and passes the result to the output format routines.

Scaled arithmetic conversion thus avoids the loss of precision inherent in representing fractional numbers in binary notation since the system can represent the integer accurately in floating point format.

3.3.3 File Access Techniques

Under BASIC-PLUS, RSTS/E provides three methods of file access:

Formatted ASCII For standard sequential I/O operations.

Virtual Arrays For random access of large data files. A virtual array is stored on disk and can contain string, integer and floating point matrices.

Record I/O Allows the user to have complete control over I/O operations.

Formatted ASCII data files are the simplest method of data storage, involving a logical extension of the BASIC-PLUS PRINT and INPUT statements. The INPUT statements allows data to be entered to a running program from an external device, for example, the user's keyboard, a disk, DECTape, or paper tape reader. The PRINT statement causes the output of a specified string of characters to a selecting device.

The PRINT-USING statement allows the user to control output formatting. A special set of formatting characters allows the user to format strings and numeric fields with tabs, special characters and punctuation. For example, the user can format check amounts with asterisk-fill for protection.

The RSTS/E virtual array facility provides the means for a BASIC-PLUS
program to operate on data structures that require fast random access processing yet are too large to be accommodated in memory at one time. To accomplish this, RSTS/E uses the disk file system for storage of data arrays, and maintains only portions of these files in memory at any given time.

All references to virtual arrays are ultimately located via file addresses relative to the start of the file. No symbolic information concerning array names, dimensions, or data types is stored within the file. Thus, different programs may use different array names to refer to the data contained within a single virtual array file.

Virtual arrays are stored as unformatted binary data. This means that no I/O conversions (internal form to ASCII) need to be performed in storing or retrieving elements in virtual storage. Thus, there is no loss of precision in these arrays, and no time wasted performing conversions.

Any data element in a virtual array is completely contained within a single element (256 words) of disk storage. This restriction has no effect on integers and floating-point items, where the size of data items is fixed (1-word integer, 2- or 4-word floating point numbers), but does limit the maximum length of a virtual string to 512 characters (512 bytes). The number of data elements stored in each disk segment is a function of the size of each element.

Strings in virtual storage occupy pre-allocated space in the virtual file, and thus differ from strings in memory, where space is allocated dynamically. A disk segment containing virtual strings can be considered to be a succession of fields, each of the maximum string length. When a virtual string is assigned a new value, it is stored left-justified in the appropriate field. If the new string value is shorter than the maximum length, the remainder of the field is filled with zeros. When the string is retrieved, its length is computed as the maximum string length minus the number of zero-filled bytes.

The third type of I/O, Record I/O, permits a program to have complete control of I/O operations. Record I/O is the most flexible and efficient technique of data transfer available under BASIC-PLUS, although it is less simple to use than formatted ASCII I/O or virtual array I/O.

Input and output to Record I/O files is performed by the BASIC-PLUS GET and PUT statements. These statements allow the user to read or write specific blocks (physical records) of a file, where the block size is dependent on the type of device being accessed. For example, disk blocks are always 512 bytes long, while records from a keyboard device are one line long, where a line is delimited by a carriage return or similar terminating character. With disk files, the program has the capability of performing random access I/O to any block of the file. Furthermore, using Record I/O operations, the user can create a logical organization for file formats by controlling record length.

Normally, the system permits only one user at a time to have write privileges on any given files, to prevent loss of data if two users try to write the same block of a file. However, in certain applications (for ex-
ample, sales order-entry applications) it might be normal for several
users to be updating a single master file. For this reason, a special UP-
DATE option is available with RSTS/E Record I/O operations that permits
multiple users to have write access to a file while guarding against
writing of a single physical record. In this case, write privileges are
gained on a record-by-record basis, and no two users can have write
access to the same record simultaneously, although multiple users can
open the file for write operations.

3.3.4 Logical Disk Structures
Access to all executable code and to system and user data on the
RSTS/E system is accomplished through a logical structure of files.

The logical disk structure is divided into two types: public and private.
The file structure on a disk, whether it is designated public or private,
is the same.

A public disk is a disk on which any user can create files. Every user
has an account on a public disk. There is always at least one public disk
on the system, which is called the "system disk." All public disks to-
gether on a system are called the "public structure" because the system
itself treats all of the public disks together as a unit. For example, when
a program creates a file in the public structure, that file is placed on the
public disk with the most space available. This is done to ensure proper
distribution of files across the disks in the public structure. The actual
determination of which disks on a particular system are public and which
are private is left to the system manager. Therefore, this allocation will
vary from system to system.

The system disk contains the system code, language processors, and
the library of system programs. The system disk can also be used for
storage of active user jobs which are temporarily swapped out of mem-
ory. If the system disk is a moving head device, an auxiliary fixed head
disk can be used as the swapping device. In such cases, the swapping
device is a logical extension of the system disk and can be configured
to contain, in addition to the swapping files, other frequently-used sys-
tem files to improve speed of access. Remaining space on the system
disk and all space on other public disks is available for general storage of
user programs and data files.

Any remaining disk drives in the RSTS/E disk structure can be devoted
to private disk packs or disk cartridges. A private disk is one that belongs
to a few user accounts, conceivably to a single user account. Files can
be created only under these accounts, and can be read (or written) by
other users only if the protection code of the file permits. A user who
does not have an account on a private disk cannot create a file on it.

Private disks are always referenced by a physical or logical device name,
for example, "DK1:" for the RK11 disk drive unit 1, or "CREDIT:" for
the device assigned the logical name CREDIT. The public structure is
normally referenced by default; when no device name is given, the sys-
tem assumes the public structure. It also has the specific name "SY:".
The system will not allow two files of the same name for a single user
to exist in the public structure.

3-14
All public disks must be physically on-line and logically mounted whenever the system is running and must be accessible to all users during time sharing operations. Private disks can be logically mounted and dismounted and interchanged as needed during timesharing operations.

Control of and access to files in the RSTS/E system is accomplished by two structures called a Master File Directory and a User File Directory. A Master File Directory, or MFD, exists on each disk initialized for use on the RSTS/E system. The MFD is treated as an account on the disk, has a project-programmer number \([1,1]\), and catalogs other accounts on the disk. The MFD on the system disk is a special case, since it maintains a catalog of the accounts which can be used to log into the system. MFD accounts on other disks contain entries of accounts which can create files on that disk. Any user gains access to any file on a private disk if the protection code of the file permits. However, only those users whose accounts are entered in the MFD of the private disk can create files on the disk.

A user File Directory, or UFD, exists for each account under which files are created. The UFD contains accounting and retrieval information for each file stored under that account. A UFD for an account on a disk is not created until a file is created under that account on the disk.

### 3.3.5 System Accounts and Libraries

RSTS/E systems have three system accounts that are integral to the operation of the system and have auxiliary accounts for more efficient operation of the system. The MFD account is used on the system device and other disk devices in the system to control system access. The system library account is used by the RSTS/E system to manage a library of generally available and restricted use system programs and message and control files. A third special system account contains RSTS/E Monitor files and routines which are critical to the operation of the system.

Of particular interest to the system manager is the accounting information maintained on each user account in the MFD on the system device. This accounting information is normally accessed through the system accounting utility programs. The system manager or privileged users can also access and change this information in BASIC-PLUS using the SYS monitor functions (section 3.5). Table 3-2 summarizes the accounting information maintained in the MFD.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DESCRIPTION</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Project-programmer number</td>
<td>Account number under which a user logs in and creates files.</td>
</tr>
<tr>
<td></td>
<td>Password</td>
<td>Password required to gain access to the system.</td>
</tr>
<tr>
<td>Accumulated Usage</td>
<td>CPU time</td>
<td>Processor time the account used to date.</td>
</tr>
</tbody>
</table>
Table 3-2  Account Information Stored in the MFD on the System Device (Cont.)

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect time</td>
<td>Number of minutes the user has been connected to the system via a terminal or remote line.</td>
</tr>
<tr>
<td>Kilo-core ticks</td>
<td>Core usage factor. One KCT is the usage of 1K words of memory for one tenth of a second.</td>
</tr>
<tr>
<td>Device time</td>
<td>Peripheral device time the account has used.</td>
</tr>
<tr>
<td>Disk Storage</td>
<td>Quota</td>
</tr>
<tr>
<td></td>
<td>Number of 256-word blocks the user is allowed to retain at logout time.</td>
</tr>
</tbody>
</table>

3.3.6 Privileged Accounts and Privileged Programs

Certain accounts in the RSTS/E system have special capabilities. These accounts are called privileged accounts and are recognized as privileged if their project numbers are 1. The system library account [1,2] is an example of a privileged account.

Privileged accounts have the following special capabilities:

- Unlimited file access
- Creation and modification of privileged programs
- Use of privileged aspects of system programs
- Use of privileged SYS system functions and the PEEK function (see section 3.5).

Unlimited File Access

No file in the RSTS/E system can be protected against the user of a privileged account. A privileged user can create and delete files under any account number on any disks.

Creation and Modification of Privileged Programs

A program is privileged when it has a .BAC filename extension and a protection code of <192> or greater. Protection code <192> means that the privileged protection <128> and the compiled file protection <64> are set. Only the system manager or users running under privileged accounts can create or modify privileged programs.

Use of Privileged Features of System Programs

If a program is designated privileged, any user can run the program in privileged program status. Privileged program status means that system operations normally reserved to a user of a privileged account can be executed while running under a non-privileged account. The privileged status exists only for the duration of the program run.

The ability to designate a program as privileged allows the system manager to extend use of privileged functions to non-privileged users. For example, the program TTYSET allows general users to change characteristics of their terminals. Such an action is a privileged system function executable only by owners of privileged accounts. With the privileged program status, execution of the function by the owner of a non-privileged account does not cause the normal program trap.

3-16
The same TTYSET program additionally allows a privileged user to change characteristics of other terminals. A check is built into the program to ensure that a user attempting to change the characteristics of another terminal is indeed a privileged user. In effect, the execution of some privileged functions is made available to the non-privileged user but other privileged features are available to only those users logged in under privileged accounts. The system manager likewise can control the user of privileged operations.

3.4 USER INTERFACE
This section describes the system facilities available to the general user, including the system commands, the system utility programs and the batch processing commands.

3.4.1 System Commands
The RSTS/E system commands issued by the user at a terminal are easy-to-use English words or abbreviations. The system accepts both long and short command formats for inexperienced and experienced users. It responds with understandable statements and, if a command does not supply complete information, prompts the user for remaining data.

RSTS/E system commands include the following:
- system information and control commands
- login/logout commands
- program creation and modification commands
- program execution commands

Table 3-3 lists the standard system commands.

<table>
<thead>
<tr>
<th>Table 3-3 RSTS/E System Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Login/Logout Commands</strong></td>
</tr>
<tr>
<td>HELLO</td>
</tr>
<tr>
<td>LOGIN</td>
</tr>
<tr>
<td>LOG</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>ATTACH</td>
</tr>
<tr>
<td>ATT</td>
</tr>
<tr>
<td><strong>These commands allow the user to log into the system by specifying an account number and password. The system also notifies the user what job number is assigned, whether any other users are logged in under the same account, and what, if any, jobs are running detached under the account. The user can choose to attach to a detached job. If the user is already logged in and issues a HELLO, ATTACH or ATT command, the user can change accounts or attach to another job without logging off the system.</strong></td>
</tr>
<tr>
<td><strong>BYE</strong></td>
</tr>
<tr>
<td><strong>Allows the user to log off the system. Checks the user’s disk quota to ensure that the user does not exceed the limit allowed by the system manager. Closes and saves any files remaining open.</strong></td>
</tr>
<tr>
<td><strong>System Information and Control Commands</strong></td>
</tr>
<tr>
<td>ASSIGN</td>
</tr>
<tr>
<td><strong>Allows the user to reserve a device for use by a single job, associate one or more logical names with a particular</strong></td>
</tr>
</tbody>
</table>

3-17
### Table 3-3  RSTS/E System Commands (Cont.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEASSIGN</td>
<td>Allows the user to release a device or all devices previously reserved for user by a job, cancel a logical name for a device, or cancel the association between the @ account and a specific account.</td>
</tr>
<tr>
<td>REASSIGN</td>
<td>Transfers the control of a particular device to another job.</td>
</tr>
<tr>
<td>SCALE</td>
<td>Allows the user to set the double-precision floating-point numeric format scale factor to be used in all programs subsequently compiled for the account.</td>
</tr>
<tr>
<td>CATALOG CAT</td>
<td>These commands print a listing of any account's device directory on the terminal.</td>
</tr>
<tr>
<td>TAPE</td>
<td>Disables the terminal echo feature when reading a low-speed paper tape on the terminal.</td>
</tr>
<tr>
<td>KEY</td>
<td>Enables the terminal echo feature after reading a low-speed paper tape on the terminal.</td>
</tr>
</tbody>
</table>

### Basic-Plus Program Development Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>Allows the user to create a BASIC-PLUS program by entering it on the terminal.</td>
</tr>
<tr>
<td>OLD</td>
<td>Retrieves the source file of a previously saved BASIC-PLUS program and places it in the user's job area.</td>
</tr>
<tr>
<td>APPEND</td>
<td>Merges the contents of a previously saved program into a program currently in the job area.</td>
</tr>
<tr>
<td>COMPILE</td>
<td>Stores a compiled version of the source program currently in the user's job area on a selected device.</td>
</tr>
<tr>
<td>SAVE</td>
<td>Stores the current source program in the user's area on a selected device.</td>
</tr>
<tr>
<td>REPLACE</td>
<td>Same as SAVE, but allows the user to replace a program currently saved under the same name as the program currently in the job area.</td>
</tr>
<tr>
<td>UNSAVE</td>
<td>Deletes a file from a selected device.</td>
</tr>
<tr>
<td>RENAME</td>
<td>Changes the name of the program currently in memory to a new name.</td>
</tr>
<tr>
<td>NAME-AS</td>
<td>Renames or assigns protection codes to a disk or DECTape file.</td>
</tr>
<tr>
<td>LIST</td>
<td>Lists all or selected lines of a BASIC-PLUS program currently in the user's area on the terminal.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes one or more selected lines of a program currently in the user's area.</td>
</tr>
<tr>
<td>LENGTH</td>
<td>Prints the length, in 1K word increments, of the current program in the user's area.</td>
</tr>
</tbody>
</table>
Table 3-3 RSTS/E System Commands (Cont.)

Program Execution and File Manipulation Commands

**RUN**
Executes a specified compiled program. If the program does not exist as a compiled program, RUN loads the BASIC-PLUS source program into the job area, compiles and runs it.

**RUNNH**
Executes the program currently in the job area.

**CHAIN**
Allows a user to execute a program segment; it is similar to the RUN command except that the user can specify a starting line number.

**CONT**
Restarts execution of the program currently in the user’s job area where it was interrupted (either by a STOP statement in the program or a CTRL/C issued from the terminal).

In addition to the standard commands, some system programs can be run by typing a unique system command called a Concise Command Language (CCL) command. Use of CCL commands requires that the concise command language option be included on the system at system generation time. If CCL is included on the system, up to 20 such commands are available.

CCL commands allow a user to run a system program and to specify a single command for the program to execute. The user types the CCL command and the program command on one line and enters it to the system. For example, the user can run the PIP system utility to print a copy of a file on the line printer in either of two ways:

1. **RUN $PIP**
The user issues the RUN command for the PIP program stored in the system library account ($ = [1,2]).

   **PIP-RSTS V06A-02**
   PIP prints an identification message and requests a command by printing “#”. The user issues the request to print a copy of FILE on the line printer. When PIP finishes the request, it prints another “#” to prompt another command.

   **#↑C**
The user types a CTRL/C to terminate PIP and return to system command level.

   **READY**
The system prints READY on the terminal to indicate that it is ready to accept a system command.

2. **PIP LP:<FILE**
The user issues the CCL command PIP to run the PIP program and issues the request to print a copy of FILE on the line printer.

   **READY**
When PIP finishes executing the request, the system prints READY on the terminal to indicate that it is ready to accept a system command.

Although CCL commands are installation dependent, DIGITAL provides a standard set of commands which are listed below. Note that the UMOUNT
The system program is designed to be run only through the MOUNT and DISMOUNT CCL commands.

<table>
<thead>
<tr>
<th>CCL COMMAND</th>
<th>ASSOCIATED PROGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIP</td>
<td>PIP</td>
</tr>
<tr>
<td>HELP</td>
<td>PIP</td>
</tr>
<tr>
<td>QUE</td>
<td>QUE</td>
</tr>
<tr>
<td>EDIT</td>
<td>EDIT</td>
</tr>
<tr>
<td>CREATE</td>
<td>DIRECT</td>
</tr>
<tr>
<td>DIR</td>
<td>SYSTAT</td>
</tr>
<tr>
<td>SYS</td>
<td>TTYSET</td>
</tr>
<tr>
<td>SET</td>
<td>UMOUNT</td>
</tr>
<tr>
<td>MOUNT</td>
<td>U Mount</td>
</tr>
<tr>
<td>DISMOUNT</td>
<td>COBOL</td>
</tr>
<tr>
<td>COBOL</td>
<td>SORT</td>
</tr>
<tr>
<td>SORT</td>
<td>FORTRAN</td>
</tr>
<tr>
<td>FOR</td>
<td>LINK</td>
</tr>
<tr>
<td>LINK</td>
<td>MACRO</td>
</tr>
<tr>
<td>MACRO</td>
<td>EXEC</td>
</tr>
</tbody>
</table>

In addition to the system commands and CCL commands, RSTS/E supports the following special control character commands:

CTRL/C  Stops any current program execution and returns the system to command mode.
CTRL/O  Suppresses or enables output to the user terminal.
CTRL/S  Suspends output on a terminal until a CTRL/Q is received.
CTRL/Q  Resumes output interrupted by a CTRL/S.
CTRL/U  Deletes the current line entered.
CTRL/Z  Used as an end-of-file character.

### 3.4.2 General System Utility Programs

In addition to the system management utility programs, RSTS/E includes several utility programs available to the general user. These programs include system information and terminal utility programs, file utility programs, and special service programs. Like the system management utilities, they are stored in the system library account and are called and executed by issuing the RUN system command or, if available, the appropriate CCL command.

General system utilities include the following:

### System Information Programs

**SYSTAT**  Provides current system information concerning job, device, and buffer status. This includes identifying the active jobs in the system, the accounts under which they are running, their size, their associated keyboard if attached, and their current activity. It also identifies which devices are assigned and to which job they are assigned.

**QUOLST**  Provides current system information, including the number of free blocks remaining on the system structure, the number
of blocks used by an account, the number of free blocks remaining in an account, and its disk quota.

**MONEY** Prints the current account status, including the amount of CPU time, connect time, kilo-core ticks and disk blocks used.

**GRIPE** Allows the user to communicate comments to the system manager.

**TTYSET** Allows a user to establish terminal characteristics for the terminal. The user can call a macro command that establishes the standard characteristics for a selected type of terminal or select an individual combination of characteristics.

**INUSE** Prints the message “IN USE” at a terminal to allow a user to leave the terminal momentarily.

**File Manipulation Programs**

**EDIT** Allows the user to create or modify text or program files.

**PIP** Allows the user to transfer files from one device to another, merge files, delete files, zero a device directory or list a device directory.

**COPY** Fast copies all the information on a disk, DECTape or magtape device.

**BACKUP** Allows the user to preserve and recall files stored under one or more user accounts by transferring multiple files from the private or public disk structure to a private disk, a DECTape or a magtape.

**DIRECT** Prints fast directories of selected file-structured devices.

**FILCOM** Compares two text files line by line and prints any differences found.

**Special Service Programs**

**QUE** Creates jobs that are to be executed by spooling programs such as BATCH and SPOOL. It also lists pending requests and kills pending requests.

**RUNOFF** Generates a formatted listing of a text file containing special RUNOFF text format commands.

3.4.3 **Batch Processing**

The capability to execute a batch of commands allows the user to submit jobs to be run without terminal dialogue. Batch processing is particularly useful in executing large data processing operations for which interactive requirements are not a factor.

Batch input can be submitted from standard job control files on a random access device or from an I/O device such as the card reader. Such input consists of elements of the batch control language and is collectively referred to as a batch stream. It is possible to execute multiple streams simultaneously by running multiple copies of the BATCH program. The capability to run more than a single batch stream is controlled by the system manager.
To request the running of a batch job, the user runs the library program QUE and specifies the batch control file or files as in the following example:

```
RUN $QUE
QUE Vnnnn=RSTS Vnnnn
#QUE BA:BATCHOB = FILE1,FILE2,FILE3.DAT
#
```

Or, if QUE is available as a CCL command:

```
QUE BA:BATCHOB = FILE1,FILE2,FILE3.DAT
```

The user normally queues a batch job to device BA:. The job and log files in this example will be named BATJOB, and the files FILE1.CTL, FILE2.CTL, and FILE3.DAT will be concatenated to form the batch control file. The log file BATJOB.LOG will be printed after the job is complete.

The BATCH command set consists of the following control commands:

- `$JOB` Marks the beginning of a job and renames a job.
- `$EOJ` Marks the end of a job.
- `$BASIC` Calls the BASIC-PLUS compiler to compile a source program.
- `$COBOL` Calls the COBOL compiler to compile a source program.
- `$RUN` Executes a specified program.
- `$CREATE` Creates a file consisting of data in the input stream.
- `$DATA` Marks the beginning of an input stream.
- `$EOD` Marks the end of an input stream.
- `$DELETE` Deletes a specified file.
- `$COPY` Copies a specified file.
- `$PRINT` Prints a specified file on the line printer using the spooler.
- `$DIRECTORY` Produces a directory listing.
- `$MESSAGE` Logs a message on the system console terminal.
- `$MOUNT` Request system operator to logically mount a device.
- `$DISMOUNT` Cancels a logical device assignment and request operator to dismount a device.

### 3.5 SYS SYSTEM FUNCTIONS AND THE PEEK FUNCTION

SYS system function calls allow a user program to perform special I/O functions, to establish special characteristics for a job, to set terminal characteristics, and to request execution of special monitor operations. The function calls are unique to the BASIC-PLUS implementation of the BASIC language. They are system dependent and their format allows a variable number of parameters, unlike BASIC language function calls.

There are ten SYS functions. With one exception, all the functions can be called by non-privileged user programs. A special SYS function can be used to issue calls to FIP, the file processor. SYS calls to FIP allow the
user to select a FIP function. Some of the FIP functions can be called only by privileged user programs.

The ten SYS functions are listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel CTRL/O</td>
<td>Cancels the effect of the user typing a CTRL/O on the program’s console terminal.</td>
</tr>
<tr>
<td>Enter tape mode</td>
<td>Disables the terminal echo feature when reading a paper tape with the low-speed teletype-writer paper tape reader.</td>
</tr>
<tr>
<td>Enable echoing</td>
<td>Reverses the effects of an enter tape mode function call or a disable echoing function call.</td>
</tr>
<tr>
<td>Disable echoing</td>
<td>Prevents the system from echoing information typed at the console terminal. For example, information such as a password is not displayed but is accepted as input by the system.</td>
</tr>
<tr>
<td>Enable single character input mode</td>
<td>Allows a single character to be accepted as input from the terminal. Normally, the system waits until a line terminated by a carriage return, line feed or escape character has been typed before accepting input. In single character mode, a character typed at the terminal is passed immediately to the program by the next keyboard input request statement.</td>
</tr>
<tr>
<td>Exit to Editor with no READY message</td>
<td>Has the same effect as the END statement in a program except that it can appear anywhere in a program and does not cause a READY message to be printed at the terminal.</td>
</tr>
<tr>
<td>Get core common</td>
<td>Allows a program to extract a single string from a data area loaded by another program previously run by the same job. The data area is called the core common area.</td>
</tr>
<tr>
<td>Put core common</td>
<td>Allows a program to load a single string in a string common data area called core common. This string can be extracted later by another program, running under the same job and called by the CHAIN statement. This function allows a program to pass a limited amount of information when a CHAIN statement is executed.</td>
</tr>
<tr>
<td>Exit to Editor and setup NONAME program</td>
<td>Terminates the program (same action as an END statement) and clears the program from memory (same action as a NEW NONAME command). This is the proper method of stopping a program that is not to be rerun.</td>
</tr>
<tr>
<td>Call to FIP</td>
<td>Causes a dispatch call to the system file processor.</td>
</tr>
</tbody>
</table>
FIP calls allow the user program to perform a variety of file, device, job and system operations. Non-privileged user programs can issue the following FIP function calls:

**Monitor Information Calls**

**Read Accounting Data**
Reports the following accounting data for the program's account:
- Account number—project number, program number
- CPU time—amount of processor time used
- KCT usage—one KCT (kilo-core tick) is the usage of 1K words of memory for one-tenth of a second
- Connect time—amount of time the terminal has been connected
- Device usage time—amount of time spent using devices excluding the public disks
- Disk storage—number of disk blocks allocated
- Logout quota—number of disk blocks allowed to retain at logout time

**Get Monitor Tables**
Reports Monitor information such as the number of configured terminals, maximum number of jobs, address of the memory allocation table, address of the job status tables, etc.

**Return Error Message**
Extracts the error message text corresponding to an error code.

**Device Assignments**

**Assign/Reassign Device**
Reserves an I/O device for the use by a job, if it is available.
Reassign transfers device control to another job.

**Deassign a Device**
Releases a device to the device pool for use by other jobs.

**Deassign all Devices**
Releases all devices previously assigned to a job.

**Directory and File Control Calls**

**Filename String Scan**
Determines whether file naming syntax is valid. For example, it can check whether a given filename is valid or whether a certain disk is mounted.

**Directory Lookup on Index**
Searches for and reports a directory entry by its index position in the directory.

**Magnetic Directory Lookup**
Searches for and reports a directory entry on a magnetic tape device.

**Disk Directory Lookup by Filename**
Searches for and reports a directory entry on a disk device for a specified file.
**Disk Wild Card Directory Lookup**
Searches for and reports directory entries on a disk device for all files with a specified character(s) occurring in the filename or extension.

**Job Control Call**

**CTRL/C Trap Enable**
Allows a program to control processing when a CTRL/C is typed on the terminal.

**Communications Call**

**Send a Message**
Allows a job to send a brief message to an eligible receiving job.

FIP calls that can be used by privileged program's are:

**Monitor Information Calls**

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read of Reset Accounting data</td>
<td>Allows a program to reset accounting data for any job after reading the data.</td>
</tr>
<tr>
<td>Accounting Dump</td>
<td>Allows a program to dump accumulated accounting data.</td>
</tr>
<tr>
<td>Change Date and Time</td>
<td>Changes the date and time values maintained by the system.</td>
</tr>
</tbody>
</table>

**Job Control Calls**

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Password or Quota</td>
<td>Allows a program to change a user's password or logout disk space quota.</td>
</tr>
<tr>
<td>Change priority, run burst or maximum size</td>
<td>Allows a privileged user to give a running job an increased or decreased chance of gaining run time in relation to other running jobs, and to determine how much CPU time a job can have if it is compute-bound.</td>
</tr>
<tr>
<td>Set Special Run Priority</td>
<td>Allows a program to raise the priority of a job slightly above that of other jobs in its priority class.</td>
</tr>
<tr>
<td>Lock/Unlock Job in Memory</td>
<td>Prevents unnecessary swapping by forcing the job executing the call to remain in memory. The call eliminates swapping time between run bursts.</td>
</tr>
<tr>
<td>Drop Temporary Privileges</td>
<td>Allows a compiled program that has a privileged protection code to drop temporary privileges. A program normally issues this call when privileged status is no longer necessary to take advantage of the monitor protections otherwise overridden.</td>
</tr>
<tr>
<td>Kill Job</td>
<td>Terminates a job under programmed control.</td>
</tr>
<tr>
<td>Login</td>
<td>Logs a job in to the system.</td>
</tr>
<tr>
<td>Logout</td>
<td>Logs a job out that was initiated by a user at a terminal.</td>
</tr>
</tbody>
</table>
Detach

Disassociates a job from its terminal. This frees the terminal for other usage and makes the non-interactive job immune from interruption by someone typing a CTRL/C at the terminal.

Reattach

Attaches a detached job to a terminal.

Set Terminal Characteristics

Performs the same functions as the system program TTYSET. Allows a user to set lower case, baud rate, scope operation, etc., on a specified terminal.

System Control Calls

Logins

Sets the maximum allowable number of logins to the system.

Enable Logins

Allows the users to login to the system.

Disable Logins

Sets the number of logins allowed on the system to one. If no jobs are active on the system, one user can log in. Once one user is logged in, no other users can log into the system.

System Shutdown

Logs the current (and only) job off the system and brings the CPU to an orderly halt after the job is logged off the system.

Broadcast

Allows the user program to print a message on another user's terminal.

Force Terminal Input

Allows the user program to force data entry on another user's terminal. The forced data is seen as input by the system.

File Management Calls

Create User Account

Allows the user program to create an entry in the MFD on a disk for an account.

Delete User Account

Allows the user program to remove a MFD entry for an account on a disk.

Change File Statistics

Allows the program to change a file's creation date or time or date of last access in the UFD entry for the file.

Set Disk Access

Allows the program to logically mount or dismount a disk pack and to lock or unlock a disk pack (allow or prevent access).

Clean a Disk

Rebuilds the Storage Allocation Table on a disk.

Communication Calls

Declare a Receiver and Receive a Message

Notifies the system that an eligible receiver job is ready to receive messages. The system sets up a message queue and relays messages sent from other jobs when the program asks for a message.
Remove a Receiver Notifies the system that a receiving job is no longer eligible to receive messages.

The PEEK function allows a privileged user to examine any word location in the monitor part of memory. The program can examine words in small or large buffers, in the resident portion of the file processor, and in the low core and tables section of memory. The function does not allow a user program to examine the contents of another user’s program.

The PEEK function is normally used to examine either addresses returned by get monitor table calls or addresses of fixed monitor locations.

3.6 BASIC-PLUS

Users write executable programs in BASIC-PLUS as well as interact with the system through the BASIC-PLUS language compiler. BASIC-PLUS is an extension of Dartmouth BASIC. BASIC-PLUS under RSTS/E offers many features not found in standard Dartmouth BASIC or most other versions of BASIC.

BASIC-PLUS incorporates the following special features:

Immediate Mode of Operation: commands can be immediately executed by BASIC-PLUS instead of being stored for later execution.

Program Editing Facilities: an existing program can be edited by adding or deleting lines, or renaming the program. The user can combine two programs into a single program and request the listing of a program, either in whole or in part on his terminal or on a line printer.

Program Control and Storage Facilities: facilities are included for storing both programs and data on any mass storage device and later retrieving them for use during program execution. Programs can also be entered from a terminal paper tape reader as well as from the high-speed paper tape reader available on the computer.

Documentation and Debugging Aids: the insertion of remarks and comments within a program is easy in this version of BASIC. Debugging of programs is aided by the printing of meaningful diagnostic messages which pinpoint errors detected during the program’s execution.

Access to System Peripheral Equipment: the user program is able to perform input and output with various equipment, such as paper tape reader/punch, disk, DECTape, industry-compatible magnetic tape, line print, floppy disks, and card reader.

Record I/O: language extensions provide a means of handling records composed of fixed-length fields in a highly efficient manner.

Matrix Computations: a set of 13 commands is available for performing matrix I/O, addition, subtraction, multiplication, inversion, and transposition.

Alphanumeric String Capabilities: alphabetic and/or alphanumeric strings can be manipulated with the same ease as numeric data. Individual characters within these strings can be accessed by the user.
Formatting of Output: the PRINT-USING statement includes facilities for tabs, spaces, and the printing of column headings, as well as precise specifications of the output line formatting and floating dollar sign, asterisk fill, and comma insertion in numeric output.

The following sections describe some of the advanced BASIC-PLUS features, including immediate mode operations, data formats and operations, matrix manipulation, and advanced BASIC-PLUS statements. Table 3-4 provides a summary of the BASIC-PLUS language.

### Table 3-4 BASIC-PLUS Language Summary

#### Summary of Variable Types

<table>
<thead>
<tr>
<th>TYPE</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>X3</td>
</tr>
<tr>
<td>Integer</td>
<td>B%</td>
</tr>
<tr>
<td></td>
<td>D7%</td>
</tr>
<tr>
<td>Character string</td>
<td>M$</td>
</tr>
<tr>
<td></td>
<td>R1$</td>
</tr>
<tr>
<td>Floating point matrix</td>
<td>S(4)</td>
</tr>
<tr>
<td></td>
<td>E(5,1)</td>
</tr>
<tr>
<td></td>
<td>N2(8)</td>
</tr>
<tr>
<td></td>
<td>V8(3,3)</td>
</tr>
<tr>
<td>Integer matrix</td>
<td>A%(2)</td>
</tr>
<tr>
<td></td>
<td>I%(3,5)</td>
</tr>
<tr>
<td></td>
<td>E3%(4)</td>
</tr>
<tr>
<td></td>
<td>R2%(2,1)</td>
</tr>
<tr>
<td>Character string matrix</td>
<td>C$(1)</td>
</tr>
<tr>
<td></td>
<td>S$(8,5)</td>
</tr>
<tr>
<td></td>
<td>A2$(8)</td>
</tr>
<tr>
<td></td>
<td>V1$(4,2)</td>
</tr>
</tbody>
</table>

#### Summary of Operators

<table>
<thead>
<tr>
<th>TYPE</th>
<th>OPERATOR</th>
<th>OPERATES ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>—</td>
<td>unary minus</td>
</tr>
<tr>
<td></td>
<td>↑</td>
<td>exponentiation</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>subtraction</td>
</tr>
<tr>
<td>Relational</td>
<td>=</td>
<td>equals</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td></td>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;</td>
<td>not equal to</td>
</tr>
<tr>
<td></td>
<td>~</td>
<td>approximately equal to</td>
</tr>
<tr>
<td>Logical</td>
<td>NOT</td>
<td>logical negation</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>logical product</td>
</tr>
<tr>
<td></td>
<td>OR</td>
<td>logical sum</td>
</tr>
<tr>
<td></td>
<td>XOR</td>
<td>logical exclusive or</td>
</tr>
<tr>
<td></td>
<td>IMP</td>
<td>logical implication</td>
</tr>
<tr>
<td></td>
<td>EQV</td>
<td>logical equivalence</td>
</tr>
</tbody>
</table>

numeric variables or constants

string or numeric variables or constants

composed of string or numeric elements, integer variables

or integer valued expressions

3-28
<table>
<thead>
<tr>
<th>String</th>
<th>+</th>
<th>concatenation</th>
<th>string variables or constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>+,−</td>
<td>addition and subtraction</td>
<td>dimensioned variables of equal dimensions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>multiplication of conformable matrices</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* scalar multiplication</td>
</tr>
</tbody>
</table>

**Summary of Functions**

<table>
<thead>
<tr>
<th>Mathematical</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(X)</td>
<td></td>
<td>Absolute value</td>
</tr>
<tr>
<td>ATN(X)</td>
<td></td>
<td>Arctangent in radians</td>
</tr>
<tr>
<td>COS(X)</td>
<td></td>
<td>Cosine in radians</td>
</tr>
<tr>
<td>EXP(X)</td>
<td></td>
<td>Exponent</td>
</tr>
<tr>
<td>FIX(X)</td>
<td></td>
<td>Truncated value</td>
</tr>
<tr>
<td>INT(X)</td>
<td></td>
<td>Greatest integer</td>
</tr>
<tr>
<td>LOG(X)</td>
<td></td>
<td>Natural log</td>
</tr>
<tr>
<td>LOG10(X)</td>
<td></td>
<td>Common log</td>
</tr>
<tr>
<td>PI</td>
<td></td>
<td>Constant pi</td>
</tr>
<tr>
<td>RND</td>
<td></td>
<td>Random number between 0 and 1</td>
</tr>
<tr>
<td>SGN(X)</td>
<td></td>
<td>Sign</td>
</tr>
<tr>
<td>SIN(X)</td>
<td></td>
<td>Sine in radians</td>
</tr>
<tr>
<td>SQR(X)</td>
<td></td>
<td>Square root</td>
</tr>
<tr>
<td>TAN(X)</td>
<td></td>
<td>Tangent in radians</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Print</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS(X%)</td>
<td></td>
<td>Current print head position</td>
</tr>
<tr>
<td>TAB(X%)</td>
<td></td>
<td>Move print head position</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII(A$)</td>
<td></td>
<td>ASCII value in decimal</td>
</tr>
<tr>
<td>CHR$(X%)</td>
<td></td>
<td>ASCII character</td>
</tr>
<tr>
<td>CVT%$(I%)</td>
<td></td>
<td>Maps integer to string</td>
</tr>
<tr>
<td>CVTF$(X)</td>
<td></td>
<td>Maps floating point to string</td>
</tr>
<tr>
<td>CVT$(A$)</td>
<td></td>
<td>Maps string to integer</td>
</tr>
<tr>
<td>CVT$(F(A$)</td>
<td></td>
<td>Maps string to floating point</td>
</tr>
<tr>
<td>CVT$(S(A$,I%)</td>
<td></td>
<td>Converts string</td>
</tr>
<tr>
<td>STRING$(N1,N2)</td>
<td></td>
<td>Creates string</td>
</tr>
<tr>
<td>LEFT(A$,N%)</td>
<td></td>
<td>Returns left-most substring</td>
</tr>
<tr>
<td>RIGHT(A$,N%)</td>
<td></td>
<td>Returns right-most substring</td>
</tr>
<tr>
<td>MID(A$,N1%,N2%)</td>
<td></td>
<td>Returns middle substring</td>
</tr>
<tr>
<td>LEN(A$)</td>
<td></td>
<td>Returns string length</td>
</tr>
<tr>
<td>INSTR(N%,A$,B$)</td>
<td></td>
<td>Search for substring</td>
</tr>
<tr>
<td>SPACE$(N%)</td>
<td></td>
<td>String of spaces</td>
</tr>
<tr>
<td>NUM$(N%)</td>
<td></td>
<td>Strings of numerals</td>
</tr>
<tr>
<td>VAL(A$)</td>
<td></td>
<td>Computes numeric value</td>
</tr>
<tr>
<td>Xlate(A$,B$)</td>
<td></td>
<td>Translate string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE$(0%)</td>
<td></td>
<td>Current date</td>
</tr>
<tr>
<td>DATE$(N%)</td>
<td></td>
<td>Calendar date</td>
</tr>
<tr>
<td>TIME$(0%)</td>
<td></td>
<td>Current time</td>
</tr>
<tr>
<td>TIME$(N%)</td>
<td></td>
<td>Time of day</td>
</tr>
<tr>
<td>TIME(0%)</td>
<td></td>
<td>Clock time</td>
</tr>
<tr>
<td>TIME(1%)</td>
<td></td>
<td>CPU time used</td>
</tr>
<tr>
<td>TIME(2%)</td>
<td></td>
<td>Connect time</td>
</tr>
</tbody>
</table>

3-29
Table 3-4  BASIC-PLUS Language Summary (Cont.)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME(3%)</td>
<td>KCT's used</td>
</tr>
<tr>
<td>TIME(4%)</td>
<td>Device time used</td>
</tr>
<tr>
<td>STATUS</td>
<td>Statement status</td>
</tr>
<tr>
<td>BUFSIZE(N)</td>
<td>Opened device buffer size</td>
</tr>
<tr>
<td>LINE</td>
<td>Interrupted line number</td>
</tr>
<tr>
<td>ERR</td>
<td>Error code received</td>
</tr>
<tr>
<td>ERL</td>
<td>Line number on error receipt</td>
</tr>
<tr>
<td>SWAP%(N%)</td>
<td>Byte swap</td>
</tr>
<tr>
<td>RAD$(N%)</td>
<td>RAD50 conversion has been defined.</td>
</tr>
</tbody>
</table>

Matrix

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRN(X)</td>
<td>Transpose matrix</td>
</tr>
<tr>
<td>INV(X)</td>
<td>Inverse of matrix</td>
</tr>
<tr>
<td>DET</td>
<td>Determinant after inverse</td>
</tr>
<tr>
<td>NUM</td>
<td>Number of rows on input</td>
</tr>
<tr>
<td>NUM2</td>
<td>Number of row elements on input</td>
</tr>
</tbody>
</table>

Magtape

<table>
<thead>
<tr>
<th>Magtape</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAGTAPE</td>
<td>Program control of magtape</td>
</tr>
<tr>
<td>RECOUNT</td>
<td>Input characters read</td>
</tr>
</tbody>
</table>

Summary of Basic-Plus Statements

General Program Statements

<table>
<thead>
<tr>
<th>STATEMENT NAME</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>REM</td>
<td>Identifies comments in a program.</td>
</tr>
<tr>
<td>LET</td>
<td>Assigns a numeric value to a variable.</td>
</tr>
<tr>
<td>DIM</td>
<td>Defines the maximum number of elements in a matrix.</td>
</tr>
<tr>
<td>RANDOMIZE</td>
<td>Causes the random number function (RND) to choose a random starting value.</td>
</tr>
<tr>
<td>IF-THEN, IF-GOTO</td>
<td>Transfer conditionally from the normal consecutive order of statement numbers, depending on the truth of some mathematical relation or relations.</td>
</tr>
<tr>
<td>IF-THEN-ELSE</td>
<td>Same as the IF-THEN statement, except that rather than executing the line following the IF statement when the condition is not met, another line number can be specified for execution.</td>
</tr>
<tr>
<td>FOR</td>
<td>Causes the program to cycle through a designated loop using a control variable to count the number of repetitions.</td>
</tr>
<tr>
<td>FOR-WHILE, FOR-UNTIL</td>
<td>Specifies that a loop is to be reiterated as long as a certain condition remains true (FOR-WHILE) or false (FOR-UNTIL).</td>
</tr>
<tr>
<td>NEXT</td>
<td>Signals the end of the loop which began with the FOR statement.</td>
</tr>
<tr>
<td>DEF, single line</td>
<td>Defines a single-line function.</td>
</tr>
<tr>
<td>DEF, multiple line</td>
<td>Defines a multiple-line function.</td>
</tr>
<tr>
<td>GOTO</td>
<td>Unconditionally transfers execution to some line other than the next sequential line in the program.</td>
</tr>
<tr>
<td>ON-GOTO</td>
<td>Transfers control to one of several lines depending on the value of an expression.</td>
</tr>
</tbody>
</table>

3-30
<table>
<thead>
<tr>
<th>STATEMENT NAME</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOSUB</td>
<td>Transfers control to the first line of a subroutine.</td>
</tr>
<tr>
<td>ON-GOSUB</td>
<td>Transfers control of a program to one of several subroutines or to one of several entry points of one or more subroutines.</td>
</tr>
<tr>
<td>RETURN</td>
<td>Signals the end of a subroutine and transfers control to the line following the GOSUB statement line.</td>
</tr>
<tr>
<td>CHANGE</td>
<td>Transforms a character string into a list of numeric values or a list of numeric values into a character string.</td>
</tr>
<tr>
<td>OPEN</td>
<td>Associates a file on a file-structured device or some non-file structured device with an I/O channel number internal to the program.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Terminates I/O between the program and a peripheral device.</td>
</tr>
<tr>
<td>READ</td>
<td>Assigns values defined in DATA statements to specified variables.</td>
</tr>
<tr>
<td>DATA</td>
<td>Supplies the values to be used in READ statements.</td>
</tr>
<tr>
<td>RESTORE</td>
<td>Resets the data values list; a subsequent READ statement obtains the value defined by the first DATA statement in the program.</td>
</tr>
<tr>
<td>PRINT- USING</td>
<td>Performs formatted output.</td>
</tr>
<tr>
<td>INPUT</td>
<td>Enters data to a program from an external device such as the terminal, disk, DECTape, or paper tape reader.</td>
</tr>
<tr>
<td>INPUT LINE</td>
<td>Enters an entire line of data as a single character string without any formatting.</td>
</tr>
<tr>
<td>NAME- AS</td>
<td>Renames or assigns protection codes to a disk or DECTape file.</td>
</tr>
<tr>
<td>KILL</td>
<td>Deletes a file from an account area.</td>
</tr>
<tr>
<td>ON ERROR GOTO</td>
<td>Transfers control to a user-written subroutine that handles a normally fatal program error.</td>
</tr>
<tr>
<td>RESUME</td>
<td>Resumes execution of a program after a user-written error routine is executed.</td>
</tr>
<tr>
<td>CHAIN</td>
<td>Transfers control of execution to another program.</td>
</tr>
<tr>
<td>STOP</td>
<td>Suspends program execution and returns terminal to immediate mode.</td>
</tr>
<tr>
<td>END Matrix Statements</td>
<td>Identifies the end of a program.</td>
</tr>
<tr>
<td>MAT READ</td>
<td>Reads data into a matrix from DATA statements in a program.</td>
</tr>
<tr>
<td>MAT PRINT</td>
<td>Prints each element of a one or two dimensional matrix.</td>
</tr>
</tbody>
</table>
Table 3-4  BASIC-PLUS Language Summary (Cont.)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAT INPUT</strong></td>
<td>Enters the value of each element of a predimensioned matrix from the keyboard.</td>
</tr>
<tr>
<td><strong>MAT</strong></td>
<td>Creates initial values for the elements of a matrix (excluding row zero or column zero), which can be all zeros, all ones, or an identify matrix (all diagonal elements are ones).</td>
</tr>
</tbody>
</table>

**Statement Modifiers**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong></td>
<td>The statement is executed only if the condition specified following the IF is true.</td>
</tr>
<tr>
<td><strong>UNLESS</strong></td>
<td>The statement is executed only if the condition specified following the UNLESS statement is false.</td>
</tr>
<tr>
<td><strong>FOR</strong></td>
<td>Performs reiterative execution of a single line based on a control variable.</td>
</tr>
<tr>
<td><strong>WHILE</strong></td>
<td>Performs reiterative execution of a single line as long as the condition specified following the WHILE remains true.</td>
</tr>
<tr>
<td><strong>UNTIL</strong></td>
<td>Performs reiterative execution of a single line as long as the condition specified following the UNTIL remains false.</td>
</tr>
</tbody>
</table>

**System Statements**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SLEEP</strong></td>
<td>Suspends a program's execution for a specified interval.</td>
</tr>
<tr>
<td><strong>WAIT</strong></td>
<td>Sets a maximum period for the system to wait for input from a terminal before generating a trappable error.</td>
</tr>
</tbody>
</table>

**Record I/O Statements**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LSET</strong></td>
<td>Stores values in a string left-justified.</td>
</tr>
<tr>
<td><strong>RSET</strong></td>
<td>Stores values in a string right-justified.</td>
</tr>
<tr>
<td><strong>FIELD</strong></td>
<td>Associates a string name with an I/O buffer.</td>
</tr>
<tr>
<td><strong>GET</strong></td>
<td>Reads data from a file into an I/O buffer.</td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td>Writes data from an I/O buffer into a file.</td>
</tr>
<tr>
<td><strong>UNLOCK</strong></td>
<td>Allows another program to obtain write privileges to a record currently open for write operations.</td>
</tr>
</tbody>
</table>

**3.6.1 Immediate Mode Operations**

Most BASIC-PLUS statements can either be included in a program for later execution or be issued on-line at the terminal as commands which are immediately executed by the BASIC language processor. Immediate mode operation is especially useful in two ways: to perform simple calculations that do not justify writing a complete program, or for program debugging.

To make program debugging easier, a user can insert several STOP statements in the program. When run, each STOP statement causes the
program to halt and identify the line in the program at which the pro-
gram was interrupted. The user can then examine the current contents
of variables and change them if necessary, by using immediate mode
commands. The user can then type the CONTINUE command to resume
program execution from the line at which it stopped.

3.6.2 Data Formats and Operations
BASIC-PLUS allows the user to manipulate string, integer numeric or
floating point numeric data.

String data are sequences of ASCII characters treated as units. The user
can define string constants and string variables (including subscripted
variables). In addition, relational operators can be applied to string oper-
ands to compare and indicate alphabetic (ASCII) sequence. Using the
BASIC-PLUS CHANGE statement, the user can convert individual string
characters to their equivalent ASCII code (in decimal) and vice-versa.
BASIC-PLUS provides a variety of string functions that allow the user to
concatenate two strings, access part of a string, determine the number
of characters in a string, search for substrings, and convert strings to
compact storage formats. The user can also define new string functions.

Normally, all numeric values (variables and constants) specified in a
BASIC program are stored internally as floating point numbers. If opera-
tions to be performed deal with integer numbers, significant economies
in storage space can be achieved by the use of integer data type, which
requires only one word of storage per value. Integer arithmetic is also
significantly faster than floating point arithmetic.

BASIC-PLUS permits a user program to combine integer variables or
integer-valued expressions using a logical operator to give a bit-wise in-
teger result. The logical operators AND, OR, NOT, XOR, IMP, and EQV
operate on integer data in a bit-wise manner.

Floating point numeric operations are the default BASIC-PLUS numeric
type. BASIC-PLUS users working with floating point numbers can increase
accuracy of operations involving fractional numbers by using the scaled
arithmetic feature (section 3.3.2). Furthermore, the user can perform
arithmetic operations using a mix of integer and floating point numbers.
If both operands of an arithmetic operation are either explicitly integer
or floating point, the system automatically generates integer or floating
point results respectively. If one operand is an integer and another is
floating point, the system converts the integer to a floating point repre-
sentation and generates a floating point result. If one operand is an in-
teger and the other operand is a constant that can be interpreted either
as a floating point number or an integer, the system generates an in-
teger result. The user can explicitly impose the formats and thereby
control the result of the operation.

3.6.3 Matrix Manipulation
Matrices are arrays of data which are implicitly or explicitly dimensioned
by the user. Matrices can be composed of variables of any type. A single
matrix, however, is composed of a single type of data: floating point,
integer or string. Dimensioning a matrix establishes the default number
of elements in each row and column and the number of elements in the
matrix. Implicitly dimensioned matrices are assumed to have ten ele-
ments in each dimension referenced (size 10 for one-dimensional matrix, size 10 by 10 for two-dimensional matrix, with each dimension also having a zero row and column). Explicit dimensioning is done using the DIM dimension statement.

By using the BASIC-PLUS MAT statements, a user program can alter the number of elements in each row and the number of columns in the matrix, as long as the total number of elements does not exceed the number defined when the matrix was dimensioned. The MAT operations do not set row zero or column zero, nor do they initialize values in the space allocated to the matrix unless specific MAT functions are executed.

The operations of addition, subtraction, and multiplication (including scalar multiplication) can be performed on matrices using the common BASIC mathematical operators. The matrices indicated for any operation must be conformable to that operation. In addition, functions exist for the perfonname of transposition and inversion of matrices.

3.6.4 Advanced Statement and Function Features
BASIC-PLUS extends the BASIC language by including several additional statements for easier logic flow control, function definitions, and time-sharing response. The ON-GOTO, ON-GOSUB, IF-THEN-ELSE, FOR-WHILE, and FOR-UNTIL statements provide a variety of conditional control over looping and subroutine execution. The ON ERROR GOTO statement allows the programmer to write subroutines that handle error conditions normally considered fatal. The program can test a system variable named ERR to determine which error occurred, and can examine a system variable named ERL to determine the line number at which the error occurred. The SLEEP and WAIT statements allow program suspension, either for a specified time interval or until input from a terminal is received. The PRINT- USING statement provides special output formatting, including exponential representation, dollar signs, commas, trailing minus sign, and asterisk fill. The DEF statement allows multiple-line function definitions. Multiple-line function definitions can be nested, can be written in any data type and can contain any variety of argument types.

To increase the flexibility and ease of expression within BASIC-PLUS, five statement modifiers are available; IF, UNLESS, FOR (including FOR-WHILE and FOR-UNTIL), WHILE and UNTIL. These modifiers are appended to program statements to indicate conditional execution of the statement or the creation of implied FOR loops.

RSTS/E also includes several system functions and statements that allow program access to system information and conversion routines. The program can obtain the current date and time, the CPU time, connect time, KCT's, and device time used for the job. The program can convert a numeric value to a string date or time or vice versa, can swap bytes, or convert an integer in RADIX-50 format to a character string.

3.7 COBOL
RSTS/E offers PDP-11 COBOL language support as an option. PDP-11 COBOL conforms to the ANS-74 COBOL low-level standard, with many high-level extensions. The compiler supports the following modules:

Nucleus All language elements necessary for internal processing.
Table Handling Defining and manipulating tabular data.
Sequential I/O
Defining and accessing sequential files.

Relative I/O
Defining and accessing relative files including dynamic access.

Segmentation
Specifying overlay of the Procedure Division at object time.

Library
Copying predefined COBOL text into the source program.

The PDP-11 COBOL compiler is a compile-and-go system. A program is compiled, the COBOL object time system is loaded and the program is executed when the user enters the single command ‘’COBOL.’’ Several utility programs are provided with COBOL, including a report-generating program, the PDP-11 SORT utility, and a reformatting program.

For situations where the terminal is the only input device, PDP-11 COBOL provides a simple, terminal-oriented line format. Source programs and data can also be compiled from cards. PDP-11 COBOL programs can also use ANSI standard magnetic tape formats for magnetic tape files.

3.8 FORTRAN IV

RSTS/E FORTRAN IV is an extended, optimizing FORTRAN IV system that operates in interactive or batch mode under the RSTS/E executive. The FORTRAN IV system includes the FORTRAN IV compiler, the Object Time System (OTS), and several utility programs.

The FORTRAN IV compiler is designed to process source programs extremely rapidly. It uses no temporary files and requires no intermediate assembly step, thereby speeding program development. Extensive optimizations, such as common subexpression elimination, array vectoring, and local code sequence tailoring, decrease the size and increase the speed of programs.

The FORTRAN IV system minimizes the size of executable prgorams. The entire system (including compiler and optimization components) is completely functional in an 8K word user area. A system interface occupying 4K words of memory is shareable among all FORTRAN IV users on the system. In addition, the FORTRAN IV system provides overlay support for programs and data, allowing extremely large programs to be run in a small region of memory.

FORTRAN IV supports all processor arithmetic options available on RSTS/E systems, including Floating Point Processor option and the Floating Instruction Set option.

RSTS/E FORTRAN IV provides assembly language subprogram support, using the MACRO assembler. Although the assembly language subprogram can not issue any monitor calls, MACRO provides the experienced user with a tool to further enhance computational performance.

The FORTRAN IV compiler processes FORTRAN source programs, producing relocatable object files and program listings. The object files are later combined with the required modules of the Object Time System to create an executable program. The Object Time System is the library of FORTRAN run-time support routines.

In addition to the FORTRAN IV compiler and OTS, the RSTS/E FORTRAN IV system includes the following utility programs:
MACRO Assembler
The MACRO assembler is used to translate PDP-11 assembly language subprograms into object files and optionally produce assembly listings with cross-reference tables.

LINK Linker
The LINK utility combines relocatable object modules created by the FORTRAN IV compiler or MACRO assembler with library files to create executable program images.

LIBR Librarian
The LIBR utility creates and updates libraries of object programs. This utility is primarily used by the system manager to build the FORTRAN OTS library at system installation time, and to update the library as necessary.

ODT On-line Debugging Technique
ODT is provided as a debugging tool for those users writing MACRO subprograms.

PATCH
The PATCH utility allows the system manager to modify the FORTRAN IV compiler or utility programs as required for maintenance purposes.

PATCHO
The PATCHO program is used by the system manager modify the FORTRAN IV OTS modules to maintain the FORTRAN library.

EXEC
The EXEC program is used to load and run an executable FORTRAN program. It provides the capability to specify a maximum memory size requirement in which the program is to run.
CHAPTER 4

MULTI-USER DATA BASE MANAGEMENT SYSTEM
MUMPS-11

4.1 FUNCTIONS AND FEATURES
MUMPS-11 is an interactive multi-user data base management operating system. The capabilities of the system are heavily oriented toward string manipulation using the high-level MUMPS language. The system relieves the user of any concern for programming peripheral devices or for structuring data bases in the traditional sense.

Language processing by the system is interpretive. Each line of code undergoes identical processing each time it is executed (intermediate code is not generated). The MUMPS application programmer is relieved of assembly language programming. The major concerns of the application programmer are: developing the proper logical hierarchy for a data base, and developing efficient logic for the data processing requirements.

The MUMPS language is provided with its own stand-alone operating system. In addition to supporting the MUMPS language and providing all operating system capabilities, the system affords the user a unique data base structure and access method. Data which is referred to symbolically is automatically stored and linked in a hierarchical tree structure. The physical allocation of mass storage for the tree-structured data base is accomplished by the operating system. The data base thus created can be made available either to all system users or restricted to a class of users.

The MUMPS-11 operating system runs on any of the PDP-11 central processors, excluding the LSI-11 based processors. The system permits up to 40 simultaneous users, operating on any of up to 49 terminals, to interact with a common data base. The system is specifically designed to manipulate strings of data and to expand or contract the data storage areas through dynamic, problem-oriented procedures.

Additional features include:

- variable-sized data elements and logical records
- random access of data using multiple keys
- a variety of terminal and peripheral devices
- system utilities for backup, validation and reporting
- ease of writing, storing and debugging programs
- on-line modification of system configuration, system utilities and system library

4-1
• inter-task memory-to-memory communication facilities
• choice of ANSI standard and EBCDIC magnetic tape labelling

Both fixed-head and removable pack disk systems can be used for on-
line storage of user programs, the data base, and system utility pro-
grams. The maximum size system can provide more than 346.88 million
bytes of on-line storage.

A variety of terminals are supported. In addition, the system can use
DL11, DC11 or DH11 controllers.

Standard peripheral devices include industry-compatible magnetic tape,
DECTape, paper tape reader/punch, and line printer.

Table 4-1 summarizes the features of the MUMPS-11 operating system.

<table>
<thead>
<tr>
<th>Table 4-1</th>
<th>MUMPS-11 Operating System Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>System type</td>
<td>Single-language multi-user timesharing system with data management facilities</td>
</tr>
<tr>
<td>CPU’s supported</td>
<td>PDP-11/04,05,10 with Extended Arithmetic Element</td>
</tr>
<tr>
<td></td>
<td>PDP-11/35,40 with Extended Instruction Set</td>
</tr>
<tr>
<td></td>
<td>PDP-11/45 or PDP-11/70</td>
</tr>
<tr>
<td>Memory ranges</td>
<td>Minimum: 16K words (allows 2 users)</td>
</tr>
<tr>
<td></td>
<td>Maximum: 124K words</td>
</tr>
<tr>
<td>Additional CPU hardware supported</td>
<td>PDP-11/45 Floating Point Processor</td>
</tr>
<tr>
<td></td>
<td>Memory Management Unit (KT11-C or -D)</td>
</tr>
<tr>
<td>Minimum peripherals</td>
<td>Console terminal</td>
</tr>
<tr>
<td></td>
<td>Disk system (RK11, RF11, or RP02, RP03 or RP04)</td>
</tr>
<tr>
<td></td>
<td>Tape system (DECTape or TS03, TU10 or TU16 magtic tape system)</td>
</tr>
<tr>
<td>Additional peripherals</td>
<td>Maximum 16 single-line controllers</td>
</tr>
<tr>
<td></td>
<td>Maximum 3 multiplexors (DH11) with a maximum of 16 terminals per multiplexor</td>
</tr>
<tr>
<td></td>
<td>PC11 paper tape reader/punch</td>
</tr>
<tr>
<td></td>
<td>LP11 line printer</td>
</tr>
<tr>
<td></td>
<td>TC11 DECTape system (up to 2 dual drives)</td>
</tr>
<tr>
<td></td>
<td>TS03, TU10 or TU16 magtape system (up to 4 drives)</td>
</tr>
<tr>
<td>Language</td>
<td>MUMPS</td>
</tr>
<tr>
<td>File System</td>
<td>Hierarchically-structured global arrays</td>
</tr>
</tbody>
</table>

4.2 SYSTEM OPERATION AND COMPONENTS
The operating system is highly modular and resides permanently in
memory (Figure 4-1). The system uses between 22K and 48K bytes of
memory, depending on the hardware configuration and system software
options selecting during system generation. During system generation,
the remaining memory is subdivided into user partitions. Machines with
no more than 28K words of memory can have a maximum of 18 user
partitions. Machines with more than 28K words can have a maximum
of 40 partitions.
A partition holds one active user's program, local data and system overhead data. There is no fixed correspondence between terminals and partitions. Indeed, jobs can run without having terminals associated with them.

Partition assignment is performed dynamically at log-in time, and is also permitted during execution. The recommended size for partitions is approximately 4K bytes each, but they do not all have to be the same size. When logging in, a user is assigned the next available partition.

Each active user requiring CPU time obtains a time slice in turn. A checkpoint form of timesharing is used whereby a program is allowed to execute until its time slice has expired, plus any additional time required to complete a current operation. Control then passes to the next job (in priority order) requiring service.

The operating system contains all software necessary to operate MUMPS-11 in the hardware environment of the PDP-11. The software is entirely memory-resident and consists of the four subsystems described below:

- **Executive** Supervises the timesharing/multiprogramming operations of the system.
- **I/O Monitor** Supervises terminal and peripheral device I/O and interrupt processing.
- **Language Interpreter** Implements and provides execution control of MUMPS language programs.
- **Data Base Supervisor** Performs all logical and physical control of the data base.

The following paragraphs describe the operational features of each.

**4.2.1 Executive**
The Executive implements the timesharing aspects of the system and permits partitioned multiprogramming using dynamic assignment of
memory-resident user partitions. In a timesharing environment, jobs are
generally highly interactive and normally require little processing time
between I/O requests. The Executive passes control from one user to
another in order to use the central processor as much as possible. Be-
cause jobs are resident in memory partitions, the Executive can switch
from user to user in minimum time.

The Executive uses a set of priority-weighted queues to administer its
scheduling algorithm. These queues include one or more Wait-Queues
and a Run-Queue. The Executive passes control to each job listed in the
Run-Queue in turn. As each job receives its time slice, it is transferred
to a Wait-Queue to await its next turn for service. Jobs are transferred
from Wait-Queues to the Run-Queue when all the jobs in the Run-Queue
have received their time slice.

Initially, a job starts at the end of the highest priority Wait-Queue. Upon
reaching the top of this queue, the job is placed in the Run-Queue and
allowed to execute for the duration of its time slice. If the job is pro-
cessor-bound at the end of its time slice, the Execution drops it from
the Run-Queue and places it at the end of the next lower priority Wait-
Queue. When the job reaches the front of this queue, the Executive
doubles the job’s time slice and places it in the Run-Queue. If the job
remains processor-bound upon expiration of its time slice, it is placed
in the lowest priority Wait-Queue. When it reaches the top of this queue,
it is allocated a triple time slice and is placed in the Run-Queue. There-
after the Executive circulates the job between the lowest priority Wait-
Queue and the Run-Queue.

When the job becomes I/O bound, the Executive places the job in an
I/O hung state to await completion of the requested I/O operation.
Completion of the operation causes the Executive to place the job at
the end of the highest priority Wait-Queue.

The Executive transfers jobs from lower priority Wait-Queues to the Run-
Queue only when the higher priority Wait-Queues are empty. This tech-
nique produces the most favorable response time for interactive opera-
tions by servicing I/O bound tasks very quickly but taking longer to
service CPU-bound operations.

4.2.2 I/O Monitor
When a job becomes I/O bound, the Executive places the job in the ap-
propriate hung state that signals the I/O Monitor to start its processing.
The I/O Monitor initiates and processes the I/O activity through its inter-
rupt handlers.

The MUMPS Interpreter and the I/O Monitor communicate through buf-
fers for terminal I/O character processing, but the I/O Monitor supervises
the asynchronous filling and emptying of these buffers to overlap output
with that program’s processing whenever possible.

The I/O Monitor creates a terminal-independent environment in which an
application program can run with any terminal of the hardware system
regardless of its specific speed and formatting characteristics. At terminal
log-in, a partition initially "owns" one terminal. It may subsequently acquire other terminals in the system or it may release the original terminal and continue as a detached job.

The I/O Monitor also supervises the peripheral I/O devices of the system, including the magtape and DECTape drives, the paper tape reader/punch and line printer.

4.2.3 Log-in Processing and the Language Interpreter
During terminal log-in, a user is assigned an available partition. User Class Identifier (UCI) codes and the Programmer Access Code (PAC) are checked for validity, resulting in either authorization or denial of access to associated programs and global files. Since terminal programming of application packages is allowed, stringent checks are performed by the Interpreter to safeguard the system's service operations from all programming activities.

If the user intends to program, the partition is initialized and control is passed to the Interpreter for the subsequent programming session. If the user desires activation of a service program, the requested program is loaded from the disk into the partition and execution of that program commences. In either case, the user retains the partition until logging-off the system or until the requested program finishes executing.

The system also enables the system manager to tie a terminal to a specific task. An attempt to log-in at a tied terminal activates the task to which the terminal is tied and limits the user to the resources associated with that task. Normally, the user gains access to the system by typing a CTRL/C, entering a UCI or UCI and PAC code, and then selecting a program or command to execute. When the user types a CTRL/C at a tied terminal, the task to which the terminal is tied immediately begin executing.

All application programs, system utilities and library programs are written in the MUMPS language. This language allows an application programmer to write a program and debug, edit, run and modify it in a single interactive session at a terminal. This minimizes the programmer's time in solving a problem, the computer time needed in checking it out, and the elapsed time required to obtain a final running program. The Interpreter is that part of the operating system responsible for these services. The Executive and the I/O Monitor serve to enable the Interpreter to operate efficiently.

The Interpreter examines and analyzes all MUMPS language statements, executing in turn the desired operations. Each MUMPS language statement undergoes identical processing each time it is executed by the Interpreter. Intermediate code is not generated. Comprehensive error checking is also performed to ensure proper language syntax.

In addition, the Interpreter stores and loads programs through the disk storage system. During program execution, the Interpreter automatically overlays external program segments invoked by an active program. Proper linkages are set up to return to the invoking program when execution of the segments terminates.
A number of major advantages are obtained from the use of the Interpreter as the major component of the MUMPS system. First, programs written in an interpretive language do not require any compiling or assembling. Error comments during execution are printed at the programmer's terminal and allow quick recovery program modification and re-execution. All program debugging and modification operations are performed in the MUMPS language directly at the terminal. This makes modification convenient, particularly in an environment where the troubleshooting necessary to interface a program with an application area is a time-consuming process. The MUMPS environment allows a programming session to take the form of a conversational dialogue between the programmer and the terminal device.

4.2.4 Data Base Supervisor
The Data Base Supervisor consists of a group of routines which provide physical as well as logical control of the various disk systems which store the data base.

In MUMPS, all file information is referenced symbolically, in the context of hierarchical global variables and arrays. This replaces the standard technique of sequentially accessing the blocks comprising files on secondary memory devices. Instead, the content and structure of the tree-structured symbol tables is logically mapped into the physical storage medium of the system.

The Data Base Supervisor maps logical information at a specific level of an array into directories of fixed-size blocks. These blocks are linearly chained together to contain all the data values (logical records or fields) at that specific subscripting level, as well as pointers which link it to chains at the next lower level. The first block in a chain is called the header block. Blocks that are chained to other blocks are called the continuation blocks.

Maps of addresses of unused blocks (storage allocation bit maps) are maintained to facilitate the dynamic allocation of disk storage space to files. Whenever a continuation or header block is to be allocated, a block in the bit map whose segment address is a few segments away is used. This block allocation method ensures that the time required to retrieve a particular datum is kept to a minimum. This means that the logical processing of data in a chain is extremely rapid. Furthermore, when data is updated, care is given to repacking, and sometimes reorganizing the individual data elements within a chain, to ensure maximum use of space for variable length data.

Once a block of data accommodating a given level of subscripting is referenced, its address is placed in the partition's overhead area and the block remains in memory until a reference to a different block is made. When a level is reached, often no further disk access need be made to reference associated information.

At system generation, the system manager has the option to establish a buffer pool of up to the equivalent of 32 disk block buffers. Disk data blocks will be kept in the buffer pool as a function of frequency of use. Frequently-used blocks will tend to remain in memory, thus reducing the number of disk accesses.
When a part of a global structure is deleted, it is attached to a "Garbage Chain." The Garbage Collector routine removes blocks from the tree-structured chain and refills the storage allocation map with the addresses of the deleted blocks. This is done during periods of low CPU activity to avoid competition with user programs.

### 4.3 DATA MANAGEMENT FACILITIES

The data management features of the system allow local data used by a program to be referenced symbolically. Storage space for this data is allocated by the system as needed. Local data is that set of variables established within the domain of a particular partition and is defined only for programs within that partition. This form of storage is used for scratch or transient data. These local data arrays are treated as if they are intended to be sparse. That is, only subscripts for which data are defined are allocated space. Other implied subscripts of the array are not allocated space. A symbolic variable used in a program can be given either a numeric or a variable length string value. When it has the string value, only that space actually required to store the string is allocated.

This local data storage philosophy is extended to the management of data on the random access disk system. All elements stored in data files are referenced symbolically. The file name is similar to that of a symbolic local variable name in a program. Records in the file are treated as array elements and are referenced by subscripts. Subrecords (or fields) are referenced by appending additional subscripts. Files on disk thus comprise an external system of arrays, which provide a common data base available to all programs within a given user class. The arrays which make up this external system are called global arrays. Each global array is identified by a unique name.

The structure of global arrays is hierarchical. Any element within an array tree can contain a numeric or string data value and (or) be a pointer to a lower subscripting level in the tree. Data can be stored at any level. There are no constraints on the dimension or size of any array. In addition, the number of subscripts in an array is dynamic, so that its content and structure can change during usage.

In addition to storing global data files, the disk is also used to contain MUMPS language programs, which include both user-created programs and system utility programs.

The availability of programs and global data to users is controlled by the system’s protection scheme. Up to 16 classes of user can be defined within the system. Each user class has access only to those programs and globals residing in that class. In addition, specially named library programs residing in UCI #1 (the system UCI) can be accessed by all users.

A password issued at log-in time allows access to an associated user class. This password, called a User Class Identifier (UCI), allows Indirect Mode access to the system. Indirect Mode operations allow filed programs to be run but not changed, and global data can be read or written as required by these pre-stored programs.

If a terminal is a tied terminal, a program is automatically started in the
user partition and the normal log-in procedure is by-passed. The user is allowed to perform only the programmed functions.

An additional code called the Programmer Access Code (PAC) can be used with any UCI code to permit Direct Mode access to the system. Direct Mode operations allow programs and global files in a particular user class to be created and modified.

4.3.1 The MUMPS Disk Structure and Global Arrays
The primary devices used by the MUMPS-11 system are the disk units allocated to the storage of MUMPS globals and MUMPS programs. Each UCI defined by the system manager has two directories associated with it: the global directory (that is, the file directory), and the program directory.

Directories for programs and globals are normally stored on the system disk. Storage area for programs and globals usually begins on the same disk unit as the associated directories. As programs and globals increase in size and number, storage area will ultimately flow across physical disk unit boundaries. This is completely transparent to the user. The general user does not have to be concerned with any MUMPS-11 disk device unit naming to retrieve globals or programs from any of the disks allocated for this purpose.

The system manager can locate the directories on any disk unit in the system. The system manager can also limit program and global storage to specific disk units in the system.

Globals are logically organized as multidimensional tree-structured arrays. An element of an array has a logical name consisting of the global name and the subscript(s) uniquely identifying the element. For example, $\uparrow ABC(2,3,4,6)$ is the name of the element in the global called “ABC” whose first subscript is 2, whose second subscript is 3,4, and whose third subscript is 6. An element of the array can contain data and/or a pointer to an element at the next lower level of subscripting. The elements of a global array are called nodes.

The user's global directory contains the names of all the globals it can reference, together with the pointers to the header block for the first subscripting level of each global.

At the first subscripting level, all nodes which are initially created (either by storing data in them or storing data in nodes at a lower subscripting level) are stored in the header block. Since a block is a fixed size, the amount of data it can contain is limited. A block can hold a large number of nodes containing little data, or a few nodes containing a large amount of data. The amount of space each node occupies varies directly with the amount of data it contains.

When an added node is too large to fit into the space remaining in the header block, MUMPS stores it in an empty block called a continuation block. The address of the continuation block is placed at the end of the preceding block. This header/continuation block relationship is the fundamental structure used in global storage. As more blocks are required to store additional nodes at particular subscripting level, they are linked to-
gether in the same manner. Blocks linked together in this way are called a chain. Each logical section of an array resides in a block chain. In particular, all nodes having a common first subscript reside in the same chain of continuation blocks.

Suppose the user stores data in a node using a second level of subscripting. If a node already exists in the first level whose subscript is the same as the first subscript of the second-level node, a pointer to the header block of the second subscripting level is entered in that node. If no first-level node already exists, it is created. As more second level nodes are created, they are stored in the header block until it is full. If additional nodes are created, a continuation block is allocated, and its address is placed at the end of the header block, and so on.

Figure 4-2 illustrates the logical tree-structured arrangement of an array called ∪NAM. It has two levels of nodes. All nodes at the first level are stored in a single block. Nodes at the second level are stored in a chain of continuation blocks. The node ∪NAM(12) at the first level contains a logical pointer to the second level header block, which contains the node ∪NAM(12,11). This node contains the datum F. The node ∪NAM(15) contains logical pointers to the nodes at the second level called ∪NAM(15,01), ∪NAM(15,07), and ∪NAM(15,19). These nodes contain the data M, M, and F, respectively.

![Logical Structure of A Global Array File](image)

Figure 4-3 illustrates the physical arrangement of the same global array. The first level of nodes point to the second level of nodes. If continuation blocks are needed, the pointer is placed at the end of the header block for each subscripting level.

4.3.2 Terminals and Ancillary I/O Devices
In addition to the disk devices reserved for use by the MUMPS data base supervisor, MUMPS allows users to have access to terminals and ancillary...
I/O devices such as the paper tape reader/punch and magnetic tape devices. Each I/O device has a unique identification number in the system.

Ownership of terminals and ancillary I/O devices is established using the ASSIGN command (see section 4.6). Once ownership is established, I/O may proceed using the I/O commands available. In general, the programmer need not be concerned with specific characteristics of I/O devices, since data transfers consist of ASCII strings not greater than 132 characters. There are, however, certain physical operating characteristics of these devices which may be of interest to the programmer: for example, rewinding a magtape or access to a particular location on DECTape. There are also logical characteristics such as use of special characters to indicate end of a logical record or end-of-medium (EOM). The omission of
such characters can result in logical records of unlimited length (except for physical device limitations such as length of tape).

The unique identification number of each I/O device always represents the same device regardless of the hardware configuration of the particular system. For example, the console terminal is always device #1, the paper tape reader/punch is always device #2, the line printer is always device #3, etc. If a particular system does not have a line printer, then device #3 is non-existent, and any attempt to reference it generates an error.

The commands which effect input and output operations to the terminals and ancillary devices are: TYPE, READ, PRINT, WRITE and LOAD. The TYPE command is used to output both local and global data, as well as literals, constants and format control characters. The PRINT command is used primarily to take advantage of special features of I/O devices, which are specified, generally, by non-printing ASCII codes. The PRINT command accepts numeric arguments, the low-order seven bits of which are taken as the decimal representation of the ASCII code. For example, the command PRINT 12 is used to output a line feed character.

In addition to the standard I/O peripheral devices such as the line printer and magnetic tape drives, MUMPS has two special “devices”. They are the Sequential Disk Processor and the CPU-CPU device.

The Sequential Disk Processor (SDP) allows the user to physically access the disk as an assignable sequential I/O device. The SDP can only access disk space that is explicitly set aside for its use. Other disk space, including the global data base structure, can not be accessed. SDP allows the user to impose any file structure on the SDP space.

The CPU-CPU device allows a MUMPS program to communicate with a program running on another central processor. This device is an asynchronous, half-duplex, serial communications line that connects the MUMPS-11 CPU to another CPU. The other CPU does not necessarily have to be a MUMPS-11 system, or even a PDP-11 central processor.

The CPU-CPU device has two operating modes or states: terminal state and message state. In terminal state, the device operates exactly as if the remote CPU were a MUMPS-11 terminal. In message state, the device transmits and receives data as formatted messages which are acknowledged or retransmitted under system control.

4.3.3 Data Storage
The MUMPS-11 system handles two types of data: string and numeric. A string is any sequence of up to 132 ASCII characters. A MUMPS number is a signed, fixed-point, two-place decimal value in the range $-21474836.47$ to $+21474836.47$.

MUMPS can also handle 4-word floating point numbers approximately in the range $10^{-38}$ to $10^{+38}$ (up to 17 decimal digits of precision).

4.4 USER INTERFACE
Each user of the MUMPS-11 system gains access to the system’s programs using a special log-in sequence which involves one or two access codes (depending on the privileges of the user). These codes, provided
by the system manager, are the User Class Identifier code or UCI, and the Programmer Access Code or PAC.

The MUMPS-11 system can have up to 16 UCI's (classes of user). The UCI code must be entered by everyone who wishes to use the system. It allows access to the programs and globals listed in the program and global directories for that UCI. A user who is permitted simply to run programs needs to know only the UCI and the name of the programs for that UCI.

Users who are allowed to create or modify programs and global files must know the system's PAC. This code permits system operation in Direct Mode. In Direct Mode, a programmer can issue MUMPS commands at the keyboard, as well as create, modify and delete global data and programs associated with the UCI under which the user logged-in.

To log-in to the system, the user types the CTRL/C keys or the BREAK key on the terminal. If the terminal is not tied, MUMPS responds by requesting a UCI code. The terminal user can respond in either of two ways. If the user is not a privileged user, the response consists of a UCI code followed by the name of the program to be executed. In this case, MUMPS logs-in the user if the UCI is valid, executes the named programs and logs-off the user.

If the user is a privileged user, the response consists of a UCI code followed by the PAC. In the latter case, MUMPS enters Direct Mode, indicated by its printing a "greater than" character (>) on the terminal. In Direct Mode, the programmer can:

- execute MUMPS commands immediately
- enter the steps of a program
- run programs and access global files listed in the UCI directories
- run library utility programs

Almost any MUMPS command or function can be executed from the keyboard in direct mode (refer to section 4.6 for a description of the MUMPS language). When a command is entered, the MUMPS language interpreter executes the command immediately and gives the appropriate response to the programmer. A command line can consist of several MUMPS commands and arguments, comments and data. For example, the programmer can enter the command line:

> TYPE "7+5=",7+5

This command tells MUMPS to print the characters 7+5= on the terminal, evaluate the arithmetic expression 7+5 and print the result on the terminal. MUMPS therefore responds by immediately printing:

7+5=12
>

To create a program, the programmer simply enters a step or part number at the beginning of the command line. This signals the system to store the line in the program buffer of the user's partition rather than to
execute it immediately. For example, to store the previous command line, the user can enter the line:

>1.03 TYPE "7+5=",7+5
>
MUMPS responds only by printing the > character. The programmer must explicitly request MUMPS to execute the stored command line. For example:

>DO 1
7+5=12
>
The DO command tells MUMPS to execute part 1 of the stored program. Since step 1.03 is the only statement in part 1, MUMPS executes the command line and returns.

Once a program has been created, the programmer can store the contents of the partition's program buffer on disk or on a secondary storage device such as magnetic tape or DECtape, or punch the data on paper tape. The program can then be reloaded into the program buffer from the disk, secondary storage or paper tape.

A program can be modified when it is loaded in the program buffer by adding new steps or by replacing, deleting, or modifying existing steps.

4.5 SYSTEM UTILITIES AND LIBRARY PROGRAMS
A set of MUMPS language utility programs provides the user with the tools to maintain and service the system efficiently. All these utilities are written as MUMPS language programs, and as such can be easily modified and extended to suit the needs of a particular installation.

The utility programs consist of two operationally distinct groups: system utility programs and library utility programs. The system utility programs provide functions for use by the system manager. They reside on the disk under the control of the system UCI (UCI ≠ 1), and are accessible only to those individuals possessing the system UCI code.

Library utility programs provide general services which are available to all system users, regardless of UCI. These programs also reside under the system UCI but employ a naming convention which distinguishes them from system utilities.

Table 4-2 and 4-3 briefly describe the system utility and library programs.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
</table>
| BCS     | Broadcast Program  
          | Allows the operator to send messages to all or specified terminals. |
| CTK     | System Caretaker Program  
          | Collects system error statistics. |
| DAT     | Date Routine (or Date Set)  
          | Sets the system date. |
Table 4-2  System Utility Program Summary (Cont.)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| DBT | Disk Block Tally Program  
Calculates the number of disk blocks available for each disk (logical and physical), verifies system maintenance, tallies and reports errors. |
| DMP | Disk Block Dump Program  
Lists the contents of selected disk blocks or gives an analysis report of the system's last recorded crash block. |
| KTR | Caretaker Reporter Program  
Lists error statistics collected by the caretaker. |
| MSP | Modify System Parameters Program  
Alters UCI codes, PAC codes, terminal types, and number and size of partitions. |
| RSJ | Restore Job  
Allows jobs that are either in a wait queue or in an I/O hung state to be restored to the system. |
| SDP | Sequential Disk Processor  
Space allocation/deallocation program. |
| SIF | Status Information Program  
Provides system status information (calls the SS program) and lists system partition size assignments. |
| SS | System Status Program  
Provides information about the current users in the system, the status of their jobs, and utilization of system resources. |
| SSD | System Shutdown  
Gives instructions to the operator on how and when to halt the system. |
| STO | System Startup  
Initializes the system when disk bootstrap loading is performed. |
| SYSGEN | System Generator Program  
Tailors the basic MUMPS-11 operating system for a specific hardware configuration. |
| TIM | Time Routine (Time Set)  
Sets the system time. |
| TP1 through TP8 | System Test Package  
Provides basic test programs to help verify that a MUMPS system is operational. |
| %GP | Global Place  
Allows the system user to position global files on a specific unit and cylinder of a disk device. |

Table 4-3  Library Utility Program Summary

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| %D | Date Subroutine  
Reports the current date on the specified I/O device. |
### Table 4-3  Library Utility Program Summary (Cont.)

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Program Description</th>
</tr>
</thead>
</table>
| %FD          | Fast Program Directory Lister  
An abbreviated high-speed version of %PD, %FD lists only program names. |
| %GD          | Global Directory  
Lists the names of all global files for the current UCI onto the designated I/O device. |
| %GL          | Global Lister  
Lists the structure and content of a specified global file on the designated I/O device. |
| %GR          | Global File Restore  
Restores all or specified global files onto the data base, entering their names in the global directory of the current UCI. |
| %GS          | Global File Save  
Copies all or specified global files listed in the global directory of the current UCI onto the designated output device. |
| %GT          | Global Trace Program  
Lists global nodes, their location, level, data type and contents for the current UCI. |
| %IO          | I/O Device Assignment Routine  
Assigns the specified I/O device if available and informs the calling program of the result. |
| %IU          | IN USE Message Program  
Displays the text “IN USE” on the currently assigned device. |
| %OD          | Octal/Decimal Conversion Program  
Converts octal or decimal values to their decimal or octal equivalents. |
| %OP          | User to Operator Communication Program  
Allows a terminal user to send messages to the console terminal. |
| %PD          | Program Directory Lister  
Lists the contents of the program directory, the starting disk block number, and the length of each program of the current UCI on the designated I/O device. |
| %PL          | Program Load  
Loads programs residing on paper tape, DECtape, or magtape which were saved using %PS, onto the disk, and enters their names in the program directory of the current UCI. |
| %PS          | Program Save  
Copies specified programs listed in the program directory of the current UCI onto the designated I/O device. |
| %T           | Time Subroutine  
Reports the current time on the specified I/O device. |

#### 4.6  THE MUMPS LANGUAGE
The MUMPS language contains a large repertoire of capabilities. The basic orientation of MUMPS is procedural, much like FORTRAN or

4-15
COBOL. Its capabilities are primarily directed toward the processing of variable-length string data. In addition, standard algebraic and Boolean operations are available. Data is represented in either string or numeric form, and mixed mode operations are expressly permitted. The language also allows assembly language-like bit manipulation operations.

Language processing is in every sense interpretive. Each line of MUMPS code undergoes identical processing each time it is executed. The language interpreter has two operating modes: program execution mode (Indirect Mode) and program creation mode (Direct Mode). In Direct Mode, programs can be created, modified, debugged, stored and executed in whole or in part. Indirect Mode permits the execution of these programs.

The following paragraphs discuss some of the major elements of the MUMPS language.

4.6.1 Data Storage Elements
MUMPS interprets all data in one of two ways: as numeric values or as strings. Program data values can be expressed as literals, constants or variables. Three types of variables can be created in MUMPS programs: simple variables, subscripted variables and global variables. Variables can be created, modified and deleted using the SET, READ, KILL and XKILL commands.

System variables are a fourth type of variable. These variables, maintained by the operating system, contain general information for use by all MUMPS programs. With one exception, system variables are read-only and cannot be altered as can normal variables. Section 4.6.5 describes the system variables.

A subscript is a number enclosed in parentheses which is appended to a variable name to uniquely identify a number of data elements which are to reside under that variable name. All the subscripted variables residing under a common name are collectively referred to as an array. A global array can consist of variables which have more than one level of subscripting. When more than one level is used for global array subscripts, they are separated by commas.

A sparse array is an array in which only those elements which are explicitly defined or which are required to support the array structure actually exist. Unlike other languages which may require a declaration of the maximum size of an array to preallocate space, MUMPS dynamically allocates storage for all arrays only as needed, thus conserving storage space.

Local variables are variables which reside in the same partition as the commands which created them. These variables are accessible only to programs running in the same partition. Simple variables have no subscript, for example, ABC, R45, X, %D. Subscripted variables can have a single level of subscripting, for example, ABC(2), R49(D), ABC(4+B(C*D)/0.89).

Global variables are multисubscripted arrays. Unlike local variables, they are external to a program's partition. There is no logical limit to the num-
ber of subscripts that can be used. The physical limit is the line length of the MUMPS command line. Like subscripted local variables, global arrays also reside in sparse arrays and are created simply by reference in a program. Global names are always preceded by an up-arrow character (↑).

Array elements, which are often called nodes, can contain either a numeric or string data value. Some nodes may be simply a pointer to a level lower in the array tree. Nodes can also contain both data and a pointer to a lower level.

For example, assume that the program defines the following elements of a global named ↑ABC:

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑ABC(1)</td>
<td>&quot;ABC&quot;</td>
</tr>
<tr>
<td>↑ABC(1,2,1)</td>
<td>&quot;AGE&quot;</td>
</tr>
<tr>
<td>↑ABC(1,2,2)</td>
<td>&quot;NAME&quot;</td>
</tr>
<tr>
<td>↑ABC(2)</td>
<td>&quot;VALUES&quot;</td>
</tr>
<tr>
<td>↑ABC(2,1)</td>
<td>365.47</td>
</tr>
<tr>
<td>↑ABC(2,4,5)</td>
<td>832.56</td>
</tr>
<tr>
<td>↑ABC(3,87)</td>
<td>&quot;ZZZ&quot;</td>
</tr>
</tbody>
</table>

Two global nodes are created that contain only pointers to nodes at a lower level: ↑ABC(1,2) and ↑ABC(3). These nodes are defined implicitly. Other variables such as ↑ABC(1) and ↑ABC(2) contain both data and a pointer to data at a lower level. Still other variables, those at the lowest level of a branch in the tree, contain only data. Such is the case with ↑ABC(1,2,1) ↑ABC(1,2,2), etc.

A global variable node can be referenced in a program using a special abbreviated syntax called naked syntax. The naked syntax facility permits the programmer to avoid excessive disk accesses during program operation.

Each time a regular global reference is made (e.g., SET A=↑ABC(1,2,1)), several physical disk accesses may be performed to bring the contents of the disk block containing that global variable into memory. This is due to the hierarchical nature of the global structures. In fact, the physical access is required only when the desired block is not found in the buffer pool.

Since global variables at the same level of subscripting reside in the same or a related disk block, a physical disk access is not always necessary when accessing globals at the same level. Using naked syntax, disk accesses are made only when the subscripting level is changed or when a continuation block at the same level must be read in to locate the desired variable (transparent to the user). The use of naked syntax affects the determination of which disk blocks are to be kept in the buffer pool, since data remains in the pool as a function of frequency of use.

In form, only the up-arrow and subscripts are explicitly stated. The global name is assumed from the last global reference made. Thus, if a reference to ↑ABC(2) is to be made after referencing ↑ABC(1), only the subscript is specified: ↑(2).
4.6.2 Expressions

An expression is a value description that can be made in the MUMPS language. An expression is any legal combination of operands and operators. Expression elements include such basic language elements as literals, constants, simple variables and subscripted variables. Also included are function references and subexpressions, which are simply legal expressions enclosed in parentheses. The following are examples of expression elements:

- 123.34 constant
- ABC simple variable
- "ABCD" literal
- MX(5) local subscripted variable
- †XYZ(2,5) global variable
- $ROOT(25) function reference
- (A+B-(C/D)) subexpression

The operators in an expression serve to represent the various arithmetic and logical computations of the MUMPS language. Table 4-4 lists the MUMPS expression operators.

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Subtraction or Unary Minus</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td></td>
<td>#</td>
<td>Modulo</td>
</tr>
<tr>
<td></td>
<td>\</td>
<td>Integer divide</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td></td>
<td>=&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>Relational</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td>!</td>
<td>OR</td>
</tr>
<tr>
<td></td>
<td>^</td>
<td>NOT</td>
</tr>
<tr>
<td>String Relational</td>
<td>[</td>
<td>Contains</td>
</tr>
<tr>
<td></td>
<td>]</td>
<td>Follows</td>
</tr>
<tr>
<td></td>
<td>?</td>
<td>Pattern verification</td>
</tr>
<tr>
<td></td>
<td>#</td>
<td>Equality</td>
</tr>
<tr>
<td>String Concatenation</td>
<td>@</td>
<td>Concatenation</td>
</tr>
<tr>
<td>Data Mode Conversion</td>
<td>@</td>
<td>Convert mode to string value</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>Convert mode to numeric value</td>
</tr>
</tbody>
</table>

Of special importance are the relational string operators. They provide facilities for determining the characteristics of string data. The operators return true or false results. They are:
String Contains ([])—The string specified by the left operand is examined for the occurrence of the string specified by the right operand. If a match is found, the result is true.

String Follows ([])—The string specified by the left operand is compared character-for-character with the string specified by the right operand to establish relative position according to the ASCII collating sequence. If the string specified by the left operand follows that specified by the right operand, the result is true.

Pattern Verification (?)—The string specified by the left operand is examined for the occurrence of the character patterns specified by the Pattern Specification Codes. If a matching condition exits, the result is true. The Pattern Specification Codes can be preceded by a single decimal integer to specify the number of occurrences of a particular character type. The Pattern Specification Codes are:

A  Verify upper case alphabetics
B  Verify lower case alphabetics
C  Verify upper and lower case alphabetics
D  Verify numerics
M  Verify numerics and upper case alphabetics
N  Verify numerics and lower case alphabetics
O  Verify numerics and upper and lower case alphabetics
P  Verify punctuation
Q  Verify punctuation and upper case alphabetics
R  Verify punctuation and lower case alphabetics
S  Verify punctuation and upper and lower case alphabetics
T  Verify numerics and punctuation
U  Verify numerics, punctuation and upper case alphabetics
V  Verify numerics, punctuation and lower case alphabetics
W  Verify any character

4.6.3 Commands
A command is the basic unit of expression in the MUMPS language. A command is a mnemonic which symbolizes the action to be performed, for example GOTO or SET. The command name can be abbreviated to one letter. It usually takes one or more arguments which specify the objects of the action to be performed. Several MUMPS commands can be present on a command line. Program comments can be appended to any command line using a semicolon to separate the command line from the comment text.

A step number is used to identify each line of a MUMPS program. Step numbers establish the fundamental sequence of command line execution. MUMPS commands are executed from left to right within a line and sequentially from one line to the next in ascending step number order (assuming no control commands are encountered). For example, 2.03 is a step number.

All step numbers having a common integer base form a part. Parts are used to form program modules, each module specifying a particular procedure within a program. A program can have one part or many parts. Each part is a distinct entity with regard to program execution.
Execution control is limited to those steps within a part, and control commands such as GOTO, DO, OVERLAY, etc., must be used to transfer control outside a part.

Certain commands permit the optional use of an argument or argument list. The indirection syntax operator, symbolized by underscore or back-arrow, provides dynamic argument definition. In form, the command argument is replaced by the indirection syntax operator immediately followed by a variable name. During execution, the contents of that variable name are taken as the argument. For example:

1.15 S ARG="15+3/6" ;variable ARG is set to value
1.20 T -ARG ;contents of ARG are evaluated and output

An optional Boolean valued expression preceded by a colon can be used as part of an argument to specify conditional execution. For example:

2.03 GOTO 3:A>B ;control is transferred to part 3 if A is greater than B

The MUMPS commands fall into six functional groups as shown below. Commands can be issued in either Indirect Mode or Direct Mode unless specified otherwise. Table 4-5 summarizes the MUMPS commands.

<table>
<thead>
<tr>
<th>Table 4-5 MUMPS Commands Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assignment Commands</strong></td>
</tr>
<tr>
<td>SET</td>
</tr>
<tr>
<td>KILL</td>
</tr>
<tr>
<td>XKILL</td>
</tr>
</tbody>
</table>

| **Control Commands**             |
| GOTO                             | Transfers flow of execution from the current step sequence to the specified step or part number. This command can be issued only in Indirect Mode. |
| DO                               | Initiates execution of the specified argument. |
| IF                               | Effects a change in a program's operation based on the validity of one or more Boolean-valued expressions. If all expressions are true, the remainder of the command line is processed. If any expression is false, the next step is executed. IF can be used without arguments; the condition tested is the sense of the last IF statement. The ELSE command is used to test the logical reverse of an IF. |
| FOR                              | Produces efficient looping by repeating commands residing on the same line for a specific set of variable values. WHILE and UNTIL clauses can be used to test the status of logical conditions external to the FOR loop. |
### Table 4-5  MUMPS Commands Summary (Cont.)

- **ELSE**
  Provides the means for testing the sense of the previously executed IF command. When the sense of the preceding IF is false, commands following the ELSE on the line are executed. Otherwise, control passes to the next program step. This command can be issued in Indirect Mode only.

- **CALL**
  Initiated execution of one or more programs residing in the user's program directory or the system library. When the called program is finished, the original calling program is read back into the program buffer and re-entered at the point immediately following the invoking CALL command. When a program is called all local variables in the partition are preserved and made available to it.

- **OVERLAY**
  Loads and starts programs residing in the user's program directory or system library. Local variables are preserved unless the overlaying program changes them. Control is not automatically returned to the command following the OVERLAY command when the overlaying program finishes. This command can be issued in Indirect Mode only.

- **START**
  Permits a currently executing program to load and start one or more programs which are to run concurrently in separate partitions. Optional arguments can specify the partition size and the starting step or part number.

- **QUIT**
  Terminates the execution of a logical process, including the execution of a step, part or program. QUIT is often used to prematurely terminate operations which are executed within the range of the DO, FOR and CALL commands.

- **HANG**
  Suspends program execution for a specified time interval. When the interval is up, program execution resumes at the command following the HANG.

- **HALT**
  Unconditionally terminates a MUMPS job and causes terminal sign-off.

- **LOCK**
  Program convention for restricting access to a particular node of a global and all nodes to which it points (all its descendants). This allows the program to protect global data which may be accessed by several programs simultaneously for updating.

- **UNLOCK**
  Releases all global nodes previously locked by a particular job.

#### Input/Output Commands

- **TYPE**
  Writes data to the currently assigned device. Arguments can be expressions or the format control characters # (page or form feed), ! (carriage return, line
Table 4-5  MUMPS Commands Summary (Cont.)

feed), or ? (horizontal tab). If no arguments are specified, the contents of all local and system variables are written.

**READ**

Reads one or more lines of characters into specified local variables. Additional optional arguments are a message to be written and the format control characters, and timing information. A timed READ enables the program to continue processing if the time interval elapses before any input is received. This is particularly useful in applications where terminals are either infrequently attended or unattended.

**PRINT**

Used primarily to write device dependent control characters to the currently assigned I/O device. This command allows the programmer to take advantage of the control functions of a particular device. For example, this command can be used to ring the bell on a terminal or move the cursor on a video display terminal. In addition, special arguments of the PRINT command allow the program to change the system protection parameters of the $J system variable: enabling/disabling of library global or program updates; enabling/disabling of the use of the VIEW command to write in memory or system disk locations.

**WRITE**

Writes MUMPS programs individual steps or parts in the program buffer to the currently assigned I/O device. WRITE is essentially the opposite of the LOAD command for non-global-structured devices.

**ASSIGN**

Permits one or more I/O devices (DECtape, magtape, paper tape, line printer, sequential disk processor and terminals) to be reserved for the exclusive use of a program (Indirect Mode) or programmer (Direct Mode).

**UNASSIGN**

Releases the specified I/O device from the ownership of the current job for use by others.

**Editing Commands**

**MODIFY**

Provides program editing capabilities by altering the contents of a step.

**ERASE**

Deletes an individual step or part, a range of steps or parts, or an entire program in the program buffer of a user partition. This command can be used in Direct Mode only.

**LOAD**

Loads, but does not initiate, a program in the program buffer of a user's partition. This command can be used in Direct Mode only.
Table 4-5  MUMPS Commands Summary (Cont.)

**FILE**  
Stores the program steps currently residing in the user's program buffer on the disk. This command can be used in Direct Mode only.

**Debugging Commands**

**BREAK**  
Stops a program when control reaches the BREAK command. This allows the programmer to examine the contents of variables. This command can be used in Indirect Mode only.

**GO**  
Restarts a program interrupted by the BREAK command. This command can be issued in Direct Mode only.

**System I/O Command**

**VIEW**  
Permits the reading and writing of memory locations and disk storage blocks in the system's data base. The use of the VIEW command is restricted by several levels of protection.

### 4.6.4 Functions

Functions are predefined procedures that are an integral part of the MUMPS language. The type of each function and its arguments are predefined. There are two types of functions: string and numeric. Numeric functions return numeric values, while string functions return string values. Table 4-6 summarizes the MUMPS functions.

Table 4-6  MUMPS Functions Summary

**Numeric Functions**

**$CREATE**  
Creates a unique positive number from the first three characters of a specified ASCII string. For example, $C can be used to create alphabetic subscripts thus allowing data to be stored in subscript form. $T can be used to convert the number back into ASCII.

**$DEFINE**  
Returns a code identifying the data type of a specified local or global variable.

**$FIND**  
Searches for a given character substring within a string and returns a number representing the position of the character in the string following the substring.

**$HIGH**  
Returns the next higher subscript of an existing array element, given any reference subscript. $H can be used for sequentially processing the elements of an array in the order in which they are subscripted.

**$INTEGER**  
Returns the integer portion (numeric value) of a given numeric-valued expression. The fractional portion is truncated.
Table 4-6  MUMPS Functions Summary (Cont.)

$LENGTH Returns the number of characters in a specified string.
$M Allows standard arithmetic and relational numeric operations to be performed on 4-word floating point values.
$NEXT Returns the step number of the first step following any given reference step number.
$QUERY Allows global nodes at a given level to be sequentially processed in the physical order in which they appear.
$ROOT Returns the square root of a given numeric value, to two decimal places of accuracy.
$VIEW Returns the contents of a specified memory location. Except when the specified memory location is a device address register, the user or program using $VIEW must be logged-in under the system UCI or be a library program.

String Functions

$ALTERCASE Converts alphabetic characters from lower case to upper case and vice versa.
$EXTRACT Extracts all the characters from the specified string that are between two given character positions.
$PIECE Examines a string which is assumed to be divided into fields delimited by given characters and returns the string value contained in the field specified by the arguments.
$STEP Returns the contents of the given step number. This function is useful when programs need to store certain data in non-executable parts.
$TEXT Translates the numeric argument into ASCII characters. $TEXT is primarily for use by system programmers who are familiar with the internal data formats of the MUMPS-11 system. It is often used with the VIEW command to convert the contents of a location known to contain ASCII data.

4.6.5 System Variables
A number of special “reference only” variables are defined within the system control the flow of information and to provide system information to MUMPS programmers. These variables are maintained and updated by the system for each job partition. They can be examined by various MUMPS commands (TYPE, SET, etc.) but, with the exception of the $E variable, cannot be altered by the program. Table 4-7 summarizes the MUMPS system variables.

Table 4-7  MUMPS System Variables

$Address $A is used when performing device dependent I/O operations. MUMPS reports the hardware status for
Table 4-7  MUMPS System Variables (Cont.)

- **$Date**  
  Contains the current system date.

- **$Error**  
  When a normally fatal error occurs during processing, MUMPS sets the $E variable to a code which denotes the type of error incurred. The program can set this variable to a step or part number that is the beginning of a user-written error processing routine. If an error occurs, control is transferred to the given step or part number.

- **$IO device**  
  Contains the number of the currently assigned device as specified by the latest ASSIGN command.

- **$Job Status**  
  Contains a number, the binary digits of which specify the current status of: the programming mode, terminal interrupts and timed READ overruns. The ASSIGN and PRINT commands can be used to alter the reception or inhibition of terminal interrupts (CTRL/C or BREAK), the updating of library programs or globals, and the writing of memory or disk locations with the VIEW command.

- **$Location**  
  Contains the number of the program step currently being executed.

- **$R**  
  Random number generator.

- **$Storage**  
  Contains the number of free bytes remaining in the user's partition.

- **$Time**  
  Contains the current system time.

- **$Where**  
  If the program has its own error processing routine (as indicated by the $E system variable) and an error occurs, $W contains the number of the step at which the error occurred.

- **$X coordinate**  
  Contain the coordinates of the print head or cursor position (output only) on non-mass storage devices such as terminals and line printers. When a disk device is being used as a non-global-structured device, these variables contain the address of the next character to be read or written. When a CPU-CPU device is being used, $X contains the current message number.

- **$Y coordinate**  
  Contains the coordinates of the print head or cursor position (output only) on non-mass storage devices such as terminals and line printers. When a disk device is being used as a non-global-structured device, these variables contain the address of the next character to be read or written. When a CPU-CPU device is being used, $X contains the current message number.
CHAPTER 5

REAL-TIME MULTIPROGRAMMING FAMILY
RSX-11

5.1 FUNCTIONS AND FEATURES
RSX-11 is a unique family of compatible real-time multiprogramming operating systems for the PDP-11 computers. The RSX-11 family includes: RSX-11D, a large, multi-purpose operating system for dynamic environments; RSX-11M, a compact, efficient operating system for relatively stable environments; and RSX-11S, a small, execute-only operating system for dedicated application environments.

The RSX-11 operating systems comprise a compatible hierarchy. The program interface to RSX-11M is a proper subset of RSX-11D. Programs written to run under RSX-11M will run under RSX-11D following a re-link of the object form of the program on an RSX-11D system. RSX-11S is a memory-based proper subset of RSX-11M fully compatible internally. A program written to execute as a task-under RSX-11S will execute under RSX-11M without change. It will also execute under RSX-11D following a re-link of the object form of the program.

RSX-11D includes an Executive to monitor and control system operation, a set of Monitor Console Routines (MCR) to enable the user to communicate with the system, a set of File Control Services (FCS), I/O device handler tasks, and a variety of system utility programs. Programs can be written in MACRO, FORTRAN IV, FORTRAN IV-PLUS or COBOL.

RSX-11M includes an Executive, MCR services, FCS file system, and system utility programs. Under RSX-11M, programs can be written in a subset MACRO, in a fully-compatible RSX-11D MACRO, or in FORTRAN IV or FORTRAN IV-PLUS. RSX-11M's MCR services and FCS file system are compatible to RSX-11D's. Device handlers under RSX-11D are loadable tasks to provide system flexibility, while device drivers in RSX-11M are built into the Executive during system generation to provide faster response.

Both RSX-11D and RSX-11M are multi-user systems. More than one terminal user can interface with MCR services simultaneously. An MCR facility allows users to create a file containing executable commands to control common sequences of operations. The MCR facility in RSX-11M systems allow the user to create indirect command files using a procedure control language to effect a multi-stream batch capability. RSX-11D provides a batch processor to handle single-stream batch operations.

RSX-11S requires a host RSX-11M system for program development and system generation. Tasks can be written in MACRO, FORTRAN IV or
FORTRAN IV-PLUS, assembled or compiled and subsequently linked on the host system, and then transported to an RSX-11S system for execution. The minimum RSX-11S system includes an Executive (with incorporated device drivers) and a special FCS that contains no support for file-structured devices. The user can also add a subset of RSX-11M's MCR services if the hardware configuration includes a terminal. If on-line task loading is desired, the user can include an On-line Task Loader (OTL) utility. If the user wants to save a system image for subsequent re-booting, the user can include the System Image Preservation (SIP) utility.

Since RSX-11S is a memory-only system, it does not support a file system, non-resident tasks, task checkpointing, dynamic memory allocation or program development. It does, however, support data storage on all devices supported by RSX-11M. Its purpose is to provide a run-time environment for the execution of task on a small system with a very modest complement of peripherals.

RSX-11D runs on a PDP-11/40 with the Extended Instruction Set and Memory Management Unit, or a PDP-11/45 with the Memory Management Unit, or a PDP-11/70. The system requires a console terminal and either an RK11 disk system with two RK05 disk drives or an RP03 or RP04 disk system with a TM11 or TU16 magnetic tape system. The minimum memory requirement is 48K words; memory can expand up to 124K words on a PDP-11/40 or 11/45 and up to 1024K words on a PDP-11/70.

RSX-11D systems are distributed as bootable 48K word systems which include:

- EXEC Resident Kernel Executive,
- SCOM Tables, lists, code and nodes,
- SYDISK Partition for system disk handler,
- TTY Partition for single terminal,
- MCR Partition for MCR and some MCR functions,
- GEN General purpose partition for user tasks and libraries.

This configuration is the minimum recommended for a single command system. Non-batch program development is possible with this configuration, but it is not recommended because of the limited symbol capabilities with the assembler and compiler and running in this smaller environment.

For a multi-user RSX-11D system, a 56K word configuration with at least two million words of disk storage is recommended for simultaneous operations. This provides adequate space for FORTRAN IV libraries, compilation of large FORTRAN programs, and residency of several device handler tasks.

RSX-11M runs on any of the PDP-11 processors except the LSI-11. The minimum system requires a console terminal and either an RP02, RP03 or RP04 disk system with TM11 or TU16 magnetic tape drive system, or an RK11 disk system with one of the following: an additional RK05 disk drive, a TC11 DECtape system, a TA11 cassette system, an RX11 floppy disk system, a TM11 magnetic tape system, or a TU16 magnetic tape

5-2
system. Without a Memory Management Unit, the system can support between 16K and 28K words of memory. With memory management, memory can range between 24K and 1024K words. At least 24K words of memory are required for concurrent applications execution and program development.

The minimum configuration for an RSX-11S system is a PDP-11 processor (including the LSI-11) with at least 8 words of memory and one of the following load devices: TA11 cassette, PR11 paper tape reader, PC11 paper tape reader/punch, TC11 DECTape, TM11 magtape, TU16 magtape or RX11 floppy disk. At least 16K words are required for on-line task loading or the execution of tasks written in FORTRAN IV or FORTRAN IV-PLUS. With a Memory Management Unit, memory can expand up to 1024K words.

All three operating systems support a broad range of peripherals, including card readers, line printers, fixed-head disks and a variety of laboratory, industrial control, and communications equipment. Note that although the maximum configuration for an RSX-11S system is the same as that for RSX-11M, RSX-11S is a memory-based system and does not support disks or magnetic tape as file-structured devices.

The following tables summarize the components of the RSX-11 family. Table 5-1 covers RSX-11D, Table 5-2 covers RSX-11M, and Table 5-3 covers RSX-11S.

### Table 5-1  RSX-11D System Summary

<table>
<thead>
<tr>
<th>System type</th>
<th>Large, multi-user, general-purpose system for concurrent real-time applications, program development and general data processing.</th>
</tr>
</thead>
</table>
| CPU’s supported | PDP-11/40 with Extended Instruction Set and Memory Management  
PDP-11/45 with Memory Management  
PDP-11/70 |
| Memory ranges | Minimums:  
48K words for little or no program development  
56K words for simultaneous applications execution and program development  
Maximum:  
124K words on PDP-11/40 and PDP-11/45  
1024K words on a PDP-11/70 |
| Additional CPU hardware supported | PDP-11/40 Floating Point Unit  
PDP-11/45 Floating Point Processor  
PDP-11/70 Floating Point Processor |
| Minimum peripherals | Console terminal plus one of: an RK11 disk system with two RK05 disk drives; an RP03 disk system with a TM11 or TU16 magnetic tape system; or an RP04 disk system with a TM11 or TU16 magtape system |
| Additional peripherals | RF11, RS03 or RS04 fixed-head disk system  
TC11 DECTape system  
TA11 cassette system |
### Table 5-1  RSX-11D System Summary (Cont.)

- LP11, LS11 or LV11 line printer
- CR11, CD11 or CM11 card reader
- PC11 paper tape reader/punch
- LPS11 Laboratory Peripheral System
- ICR/ICS Industrial Control System
- AD01, AFC11 Analog/Digital Converter
- DH11, DL11, DJ11 Terminal/Line Interfaces
- KW11-P Programmable Clock

<table>
<thead>
<tr>
<th>System utilities</th>
<th>EDI</th>
<th>Line Text Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SLP</td>
<td>Source Language Input Editor</td>
</tr>
<tr>
<td></td>
<td>TKB</td>
<td>Task Builder with Global Cross Reference</td>
</tr>
<tr>
<td></td>
<td>LBR</td>
<td>Library Management Utility</td>
</tr>
<tr>
<td></td>
<td>ODT</td>
<td>User and Executive On-line Debugger</td>
</tr>
<tr>
<td></td>
<td>ZAP</td>
<td>Program Patch Utility</td>
</tr>
<tr>
<td></td>
<td>CDA</td>
<td>Core Dump Analyzer</td>
</tr>
<tr>
<td></td>
<td>PIP</td>
<td>Peripheral Interchange Utility</td>
</tr>
<tr>
<td></td>
<td>PRE</td>
<td>Media Backup Utility</td>
</tr>
<tr>
<td></td>
<td>FLX</td>
<td>File Exchange Utility</td>
</tr>
<tr>
<td></td>
<td>DMP</td>
<td>File Dump Utility</td>
</tr>
<tr>
<td></td>
<td>VFY</td>
<td>File Verification Utility</td>
</tr>
<tr>
<td></td>
<td>SORT</td>
<td>File and Index Sort Utility (option)</td>
</tr>
<tr>
<td></td>
<td>TRACE</td>
<td>User and Executive Trace Utility</td>
</tr>
</tbody>
</table>

- Task Accounting and Reporting Package
- On-line Device Error Logging and Analysis Package
- Multiplexed I/O Spooling Package

### Languages

- MACRO
- FORTRAN IV
- FORTRAN IV-PLUS
- COBOL

### Batch facilities

- Single-stream batch processor

### File systems

- Files-11: Hierarchical directory structure and volume and file protection; sequential and random access, fixed and variable length records; dynamic file creation, extension and deletion
- ANSI Standard Magtape Format: multi-volume, multi-reel support

### Table 5-2  RSX-11M System Summary

<table>
<thead>
<tr>
<th>System type</th>
<th>Compact, efficient real-time applications and program development system</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU's supported</td>
<td>PDP-11/04, 05, 10  PDP-11/35, 40, 45  PDP-11/70</td>
</tr>
<tr>
<td>Memory ranges</td>
<td>Minimums: 16K words without concurrent program development  24K words with concurrent applications execution and program development</td>
</tr>
</tbody>
</table>
Table 5-2  RSX-11M System Summary (Cont.)

<table>
<thead>
<tr>
<th>Additional CPU hardware supported</th>
<th>Minimum peripherals</th>
<th>Additional peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP-11/04, 05, 10 Extended Arithmetic Element</td>
<td>Console terminal and either: 1) an RK11 disk system with additional RK05 drive, TC11 DECtape, TA11 cassette, RX11 floppy disk, TM11 magnetic tape, TJU16 or TWU16 magnetic tape drive; or 2) an RP11 or RJP04 disk system with TM11 or TJU16 (TWU16) magnetic tape drive system</td>
<td>RF11, RJS03, RJS04, RWS03, or RWS04 fixed-head disk system</td>
</tr>
<tr>
<td>PDP-11/35, 40 Extended Instruction Set</td>
<td></td>
<td>RX11 floppy disk system</td>
</tr>
<tr>
<td>PDP-11/35, 40 Floating Point Unit</td>
<td></td>
<td>PC11 paper tape reader/punch</td>
</tr>
<tr>
<td>PDP-11/45, 70 Floating Point Processor</td>
<td></td>
<td>PR11 paper tape reader</td>
</tr>
<tr>
<td>PDP-11/35, 40, 45 KT11 Memory Management Unit</td>
<td></td>
<td>CR11 or CM11 card reader</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TA11 cassette system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LP11, LS11 or LV11 line printer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KW11-P programmable clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KW11-Y watchdog timer clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AR11 Laboratory Peripheral System</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LPS11 Laboratory Peripheral System</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICR/ICS Industrial Control Systems (local and remote)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DA11-B, DH11 with or without DM11-BB, DJ11, DL11-A/B/C/D/E.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DPI1, DQ11 communication interfaces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System utilities</th>
<th>Languages</th>
<th>Batch facility</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDI Line Text Editor</td>
<td>MACRO</td>
<td>Indirect MCR command file with procedure control language</td>
</tr>
<tr>
<td>SLP Source Language Input Editor</td>
<td>FORTRAN IV</td>
<td></td>
</tr>
<tr>
<td>TKB Task Builder with Global Cross Reference</td>
<td>FORTRAN IV-PLUS</td>
<td></td>
</tr>
<tr>
<td>CRF Cross Reference Program for task maps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBR Library Management Utility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODT On-line Debugging Tool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZAP Program Patch Utility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XDT Executive Debugging Tool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMD Post Mortem Dump and Snap Shot Dump</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLX File Exchange Utility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIP Peripheral Interchange Utility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRESRV Media Backup Utility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMP File Dump Utility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VFY File Verification Utility</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5-2  RSX-11M System Summary (Cont.)

| File system       | Files-11: Identical to RSX-11D file system, except for multi-header files and the truncate function |

Table 5-3  RSX-11S System Summary

| System type       | Execute-only real-time applications system; requires RSX-11M system for generation and program development |
| CPU's supported   | LSI-11 products                                      |
|                   | PDP-11/04, 05, 10                                   |
|                   | PDP-11/35, 40, 45                                   |
|                   | PDP-11/70                                           |
| Memory ranges     | Minimums:                                           |
|                   | 8K words without on-line task loading option        |
|                   | 16K words for one-line task loading or execution of |
|                   | tasks written in FORTRAN                           |
|                   | Maximum:                                            |
|                   | 124K words on all but PDP-11/70                    |
|                   | 1024K words on a PDP-11/70                         |
| Additional CPU    | Same as RSX-11M                                    |
| hardware supported|                                                     |
| Minimum peripherals| One of the following load devices:                  |
|                   | TA11 cassette system                                |
|                   | PR11 paper tape reader                              |
|                   | PC11 paper tape reader/punch                        |
|                   | TC11 DECtape system                                 |
|                   | TM11 magnetic tape system                           |
|                   | TJU16 or TWU16 magnetic tape system                 |
|                   | RX11 floppy disk system                             |
| Additional        | Same as RSX-11M                                    |
| peripherals       |                                                     |
| System utilities  | OTL On-line Task Loader                            |
|                   | SIP System Image Preservation Program               |
|                   | MCR Subset Console Interface                       |

5.2 COMMON RSX-11 FAMILY OPERATING SYSTEM CONCEPTS

The RSX-11 family of operating systems is designed to provide a resource-sharing environment ideal for multiple real-time activities. The basic facilities that the RSX-11 family provides for handling multiple requests for services while maintaining real-time response to each request are:

- multiprogramming
- priority scheduling
- contingency exits
- power-fail shutdown and auto-restart
In addition, both RSX-11D and RSX-11M provide:

- disk-based operation
- checkpointing
- dynamic memory allocation (optional in RSX-11M)

The basic unit of work which these operating system facilities service is called the task. A task consists of one or more programs written in a source language such as MACRO or FORTRAN, assembled or compiled into an object format, and then built into a task image by the linker utility called the Task Builder. In addition to the normal linkage functions of combining object modules or creating overlays, the Task Builder sets up the basic task attributes that determine the task’s resource requirements and relationship to other tasks in the system. The significant task attributes that affect a task’s operation in a real-time multiprogramming environment are:

- partition—the section of memory where the task will reside when it executes
- priority—the task’s relationship to other tasks competing for system resources
- checkpointability—the task’s ability to be swapped out of memory when it is not executing to make room for a task of higher priority that is ready to run (applicable only to RSX-11D and RSX-11M tasks)

Once a task is built, it can be installed in the system and executed. Task installation simply registers a task’s attributes with the system. The task is not in memory, nor is it in competition for system resources. An installed task can be put in active competition for system resources by the operator or by another active task in the system.

When an installed task is activated, the system will allocate necessary resources, bring the task into memory for execution, and place it in competition with other active tasks. Task installation is the basis for efficient task operation. An installed task uses very little memory resources; yet, when the task is needed to service a real-time event, it can be introduced into the system quickly since its basic parameters are already known to the system.

Tasks can also share code and data among themselves through the Shareable Global Areas facility. A Shareable Global Area (SGA) is made accessible to the system and to tasks by installing the SGA and the task which intends to use it.

The following paragraphs describe how the RSX-11 family’s real-time facilities handle task execution.

**Multiprogramming**

Multiprogramming is the concurrent execution of two or more tasks residing in memory. In a single processor, only one task can have control of the CPU at a time. When that task does not need CPU time (for example, when it is waiting for input from a terminal) another task that needs CPU time can execute. In the RSX-11 family, the multiprogramming of tasks is accomplished by logically dividing available memory into a number of named partitions. Tasks are built to execute out of a
specific partition, and all partitions in the system can operate in parallel.

In general, RSX-11 systems can have two kinds of partitions: system controlled and user controlled. System controlled partitions are intended for the execution of tasks where the user wishes the system to implicitly handle the allocation of memory. User controlled partitions are intended for the execution of tasks where the user wants to handle the allocation of memory.

A system controlled partition is dynamically allocated by the system to contain as many tasks as will fit simultaneously in the partition. Tasks are allocated a contiguous region in the partition, and are relocated using the hardware Memory Management Unit. The Memory Management Unit provides the facilities necessary for memory management and task relocation and protection. Systems using the Memory Management Unit are called mapped systems because the hardware allows the system to map virtual memory addresses into direct physical addresses. Only mapped RSX-11 systems can have system controlled partitions.

A user controlled partition is allocated to only one task at a time. The user has complete control over system activity in this type of partition. As a result, it provides an ideal environment for a real-time task's execution.

In RSX-11M or RSX-11S systems, a user controlled partition can be subdivided into as many as seven non-overlapping subpartitions. The subpartitions occupy the identical physical memory occupied by the main partition. Tasks built to execute in the subpartitions can execute in parallel. Tasks can not, however be resident in a main partition and its subpartitions simultaneously. If a main partition is occupied, the subpartitions can not be. All subpartitions can have tasks residing in them; therefore, up to seven potentially parallel task executions can exist within a pre-empted user controlled main partition. The goal of subpartitioning is to reclaim large memory areas when a task requiring a main partition is no longer active.

Furthermore, RSX-11M and RSX-11S systems can be mapped or unmapped systems. If the hardware configuration does not include a Memory Management Unit, the RSX-11M system is an unmapped system. If a Memory Management Unit is available, the RSX-11M or RSX-11S system can be a mapped system. Mapped systems can have both system controlled and user controlled partitions. Unmapped systems can have only user controlled partitions.

From the operator's point of view, almost no differences exist between mapped and unmapped RSX-11M or RSX-11S systems. One difference exists, however, in installing tasks into a partition. In unmapped systems, a task is linked to be installed and run in a partition with a specific base address. It can not run in any partition whose base address is not the same. In mapped systems, a task can be installed into any partition large enough to contain it, but it can only be run in the partition into which it was installed.

RSX-11D systems (which are always mapped systems) and mapped RSX-11M or RSX-11S systems provide automatic memory protection.
The memory area assigned to a task is protected from other tasks executing in the system. Each task has an absolute address range in which to execute. A task can reference and alter memory only within that specific task area which it owns. In an unmapped RSX-11M or RSX-11S system, a task cannot be prevented from referencing all available memory.

**Priority Scheduling**

Task scheduling in the RSX-11 family is primarily event-driven, in contrast to systems which use a time slice mechanism for determining a task's eligibility to execute. The basis of event-driven task scheduling is the software priority assigned to each active task. A task's default priority is set when the task is built. It can be altered once it is installed (though not active) by an MCR command from the console. RSX-11D also allows priorities to be changed dynamically from within a task.

Tasks are run at a software priority level ranging from a low of 1 to a high of 250. The Executive grants central processor resources to the highest priority task capable of execution. That task retains control of the central processor until it declares a significant event.

A significant event occurs when a task issues a system directive that implicitly or explicitly suspends a task's execution, or when an external interrupt occurs that can affect a task's execution. For example, a task can issue a directive that indicates it wants to wait until an I/O operation is complete before continuing execution; a significant event is declared when the I/O operation is complete. A special system directive also exists that allows a task to stimulate the event-driven task scheduling mechanism explicitly.

When a significant event is declared, the Executive interrupts the executing task and searches for a task capable of executing. The highest priority task that has all the resources it needs to run, and can make use of the resources it needs, will be the task that gains control of the CPU.

Event flags are associated with significant events. When a significant event occurs, the event flag indicates the specific cause of the interrupt.

There are 64 event flags. Flags 1 through 32 are local to the task, while event flags 33 through 64 are common to all tasks. A task can set, clear, test, and wait for any event flag or combination of event flags to achieve efficient synchronization between itself and other tasks in the system.

For example, upon completion of I/O requests, a device handler normally sets a requestor-indicated event flag and declares a significant event. If a requesting task instructs the system that it cannot run until an event flag is set (signalling task I/O completion), other eligible tasks of lower priority may run. In the scan of the active task list, a task that is awaiting I/O completion is bypassed until a significant event is declared through the setting of a event flag upon task I/O completion.

Although event-driven scheduling is the primary RSX-11 task scheduling mechanism, it is not the only mechanism available. As an option during system generation, RSX-11 systems allow the user to supplement event-driven task scheduling with time-sliced or time-based scheduling.
In RSX-11D systems, a system controlled partition can be defined at system generation time to use a time-based scheduling algorithm that swaps checkpointable tasks within the designated partition. The time scheduler uses the system clock to become active at specified intervals. When activated, the time scheduler performs the following for each time-scheduled partition: determines whether a checkpointable task is active in the time-scheduled partition; swaps out the active task and gives it an intermediate priority; swaps in the highest-priority ready-to-run task waiting to run in the partition; and increases the priority of all tasks currently waiting to run in the partition. In addition, the time scheduler remembers which task was most recently swapped out so that the same task is not always swapped out.

The time-based scheduling applies only to the checkpointable tasks in a time-scheduled system controlled partition. Tasks that are not checkpointable in a time-scheduled partition are processed by the RSX-11D priority scheduler. As a result, device handlers and real-time tasks can coexist in a partition with tasks that are time scheduled.

The time-based scheduling algorithm has the effect of scheduling the system's resources equally among checkpointable tasks in a time-based sequence without affecting real-time tasks in the partition. There can be several such partitions in an RSX-11D system.

In RSX-11M and RSX-11S systems, an optional time-slice scheduling mechanism is available. The time-slice scheduling is based on a priority range specified by the user during system generation. All tasks that have priorities within the specified range are scheduled using a time-slice algorithm. Tasks with higher or lower priorities than the specified range receive service in an event-driven manner. As a whole, the task range also receives service in an event-driven manner, but CPU time among the tasks within the range is shared.

**Contingency Exits (System Traps)**

Subroutines automatically entered as the result of an unanticipated synchronous condition (for example, an attempt to execute an illegal instruction) or as the result of an asynchronous condition anticipated or unanticipated (for example, an I/O termination) are called contingency exit or system trap routines.

System traps are another means of governing task execution. While significant events have a system-wide scope, traps are local to a task. Traps interrupt the sequence of instruction execution in the task and cause control to be transferred to a prespecified point in the program. In this way, system traps provide the ability to service certain conditions without continuously testing for their existence.

When a task plans to use the system trap facility, it must contain a trap service routine. This routine is automatically entered when the trap occurs using the task's normal priority and privilege. If a service routine is not supplied, the action taken by the Executive is dependent upon the type of trap.

There are two types of system traps: Synchronous System Traps (SST's) and Asynchronous System Traps (AST's).
SST's provide a means of servicing fault conditions within a task, such as memory protection violation and PDP-11/40 floating point unit exceptions. These conditions, which are internal to a task and are not significant events, occur synchronously with respect to task execution. In these cases, if an SST service routine is not included in the task, the task's execution is aborted.

AST's commonly occur as the result of a significant event and thus occur asynchronously with respect to a task's execution. A task does not have direct or complete control over when AST's occur. A characteristic of AST's is that they are for information purposes, such as signifying an I/O completion that a task wants to know about immediately, and PDP-11/45 and PDP-11/70 floating point processor exceptions.

If an AST service routine is not provided, a trap does not occur and task execution is not interrupted.

It should be emphasized that SST's are only initiated by the Executive; no further action is taken. That is, they appear to the Executive just like normal task execution. The Executive, having initiated an SST, cannot determine that the task is in the SST service routine. Thus, an SST service routine can be interrupted by another SST or an AST. SST's can be nested.

SST's are caused by activities internal to the task, while AST's occur as a result of an external event. The Executive keeps track of all AST's, queues them first-in, first-out, and is aware that a task is executing an AST.

**Power Failure Restart**

Power failure restart is the ability of a system to smooth out intermittent short-term power fluctuations with no apparent loss of service and without losing data, all the while maintaining logical consistency within the system itself and the application tasks. Power failure affects absolute response time and peak load capacity differently from the facilities previously discussed, since it applies to the aggregate system performance rather than increasing performance when the system is actually in operation. A system is not performing when it is shut down, and if the Executive can reduce the shutdown periods due to power failure restart, aggregate performance is increased.

Power failure restart functions in the RSX-11 family differ slightly between RSX-11D and RSX-11M or RSX-11S systems due to the fact that RSX-11D's I/O device handlers are included in the system as tasks, while in RSX-11M and RSX-11S systems, device drivers are part of the Executive. All RSX-11 systems perform the first three phases of power failure restart:

1. When power begins to fail, the processor traps to the Executive which stores all register contents.
2. When power is restored, the Executive again receives control and restores the previously preserved state of the system.
3. The Executive then informs any tasks that have requested power failure restart notification through the Asynchronous System Trap.
mechanism that a power failure has occurred. These tasks can then, if required, make restorations of state they deem necessary. (In RSX-11D systems, the device handler tasks use this mechanism so that they may reset their respective device control registers and retry any I/O that was in progress at power failure time.)

In addition, RSX-11M and RSX-11S systems add a fourth phase of power failure restart:

4. The Executive schedules all device drivers that were active at the time the power failure occurred at their powerfail entry point. Drivers have the option of always being scheduled on power recovery, or only when the driver has an outstanding I/O order.

These drivers can then, if required, make those restorations of state (for example, repeating I/O requests) they deem necessary.

The RSX-11 family's approach is quite efficient because the repeating of I/O is placed nearest the source most likely to know how to make the restoration.

**Disk-Based Operation (RSX-11D and RSX-11M)**

Except in dedicated applications, the total code in a system always exceeds the available main memory. A disk-based system uses random access peripherals both as an extension of executive main memory and as the principal data interchange medium. The use of disk as the system data storage medium provides the base for program development facilities, a common file system, checkpointing, and rapid initiation of tasks. The Task Builder makes it possible for the user to build overlaid tasks and call these overlays from disk. The total effect is to extend significantly the achievable peak load while still maintaining system response time requirements.

**Task Checkpointing (RSX-11D and RSX-11M)**

Effective multiprogramming is achieved when many tasks reside in memory simultaneously, spending some of their residency waiting for I/O completion, waiting for synchronization with other tasks, or in some way being unable to continue execution. While one or more tasks are waiting, another task can utilize the central processor's resources.

This multiprogramming scheme normally applies only to memory-resident tasks. Once a task is in memory, the Executive allows it to run to completion in a multiprogrammed fashion even if its memory is required for the execution of a higher priority, non-resident task. However, if it is desirable to free memory for execution of a higher priority task, a task can be declared checkpointable when it is task built.

A checkpointable task can be swapped out of memory when a higher priority task requests the partition in which it is active. Checkpointing is another method of making it possible to load the processor with as much work as it can possibly absorb, and still meet its real-time commitments.

In both RSX-11D and RSX-11M systems, task priority normally determines which tasks can checkpoint other tasks. A checkpointable task currently active in a partition, but of a lower priority than another task requesting the partition, can be pre-empted and rolled-out to disk. Later,
after the higher priority task has completed its execution, the lower priority task can be rolled-in and restored to active execution at the point where it was previously interrupted.

Both systems extend the checkpoint capability by disregarding the priority of a task in cases where the task currently active in a partition is waiting for terminal input. A task requesting a partition can checkpoint a task of higher priority if that task is waiting for terminal input.

The checkpointing capability is also used by the RSX-11D time-based scheduling algorithm which operates on those tasks which are competing for memory resources in a system controlled partition.

**Dynamic Memory Allocation (RSX-11D and RSX-11M)**
Dynamic memory allocation is an extension of the RSX-11 multiprogrammed partition structure. Dynamic memory allocation allows the system to respond rapidly to changing requirements for system resources.

In RSX-11D and mapped RSX-11M or RSX-11S systems, tasks are built with a base address of zero. This allows the user to load and execute a task in any partition large enough to contain it. When the task is loaded into memory, the systems sets the relocation registers to indicate where the task actually resides in memory.

RSX-11D and RSX-11M allow the user to load and execute more than one task in a system controlled partition. If a task loaded into a system controlled partition does not fill the entire partition, another task can be loaded into the space either above or below it, as long as the remaining contiguous physical space is large enough to contain it.

The Executive keeps an internal list of the available areas of memory in system controlled partitions, together with a list of all tasks requesting to run in those partitions. Tasks are brought in from the disk on a priority basis and are loaded into the first available memory area in the partition. The Executive continues to load tasks as long as there is sufficient contiguous physical memory available in the partition. When a task terminates, the memory it occupies becomes available again.

If the dynamic memory allocation option is included in an RSX-11M system, the user can also include the automatic memory compaction option. Normally, a task can not be loaded into a system controlled partition unless there is sufficient contiguous space for it between other tasks loaded in the partition. When a task terminates, it can leave a space which is insufficient alone to load another task, but considered together with other unused areas can be used to contain a task. If automatic memory compaction is included in the system, the tasks in a system controlled partition will be moved to obtain a large enough area in the partition to load another task.

**5.3 SYSTEM ORGANIZATION AND GENERATION**
The following sections discuss the basic design elements of the RSX-11D, RSX-11M and RSX-11S operating systems. In RSX-11 systems, total system structure is essentially dependent on the decisions that the user makes during system generation. The user defines the system organization and chooses the Executive services appropriate for the particular applications environment.

5-13
There are three basic functional uses for which memory is allocated. The amount of memory allocated to each function is specified by the user during system generation. The three functional memory spaces are for:

- the RSX-11 Executive
- partition space for tasks and shared global areas
- the system communication areas, including system lists and tables

In general, RSX-11D systems are designed to give the user the maximum flexibility needed during system operation to respond to a changing environment. As a consequence, RSX-11D is the easiest of the RSX-11 systems to generate. In addition, the device handlers generated into the system are provided as system tasks and do not affect the size of the Executive's memory area. The user does not have to generate a new system each time I/O handlers need to be added or removed; partitions only have to be allocated for the handler tasks.

For RSX-11D system generation, the user has only to define the system communication area parameters and specify the sizes of all but one of the system and user controlled partitions. The system can automatically place the Executive and partitions in memory, and will automatically calculate the size remaining for one partition if not already specified.

RSX-11D system generation is performed in two phases: the first phase defines the hardware and software configuration, the second phase performs the remaining functions which require the target hardware. When the user wants to generate subsequent systems, phase one can be quickly performed on-line without disrupting normal applications operations.

RSX-11M and RSX-11S systems are designed to provide the most efficient use of system resources during system operation. To be useful to a wide range of applications and still obtain maximum system performance for a given operating environment, therefore, RSX-11M/RSX-11S systems require the user to become more involved in system generation.

System generation for RSX-11M/RSX-11S systems provide the user with absolute control over system features and capabilities. Users concerned about size can eliminate the Executive services that are not essential to a particular application. In addition, I/O device drivers must be included in the Executive during system generation. Their inclusion in the Executive serves two purposes: it enhances overall system operation and it greatly decreases I/O driver size. System operation is enhanced because interrupt response, processing speed, and system throughput are increased. The size of the individual I/O drivers is decreased for two reasons. They do not have to contain code to initialize themselves, since they are initialized during system generation. Furthermore, the Executive can perform many operations common to all drivers; the drivers do not have to contain duplicate code required for independent operation.

For RSX-11M/RSX-11S system generation, the user has to define the system communication area parameters, specify the sizes and base addresses of the partitions, and select the Executive services needed for
the particular application. System generation is performed in two phases: the first phase defines the hardware configurations and software options; the second phase builds the complete system. Some system generation parameters can be changed on-line, for example, partition configuration. If Executive services are to be changed, however, the user must regenerate the system.

5.3.1 RSX-11D System Organization
During system generation, the user is primarily concerned with allocating partitions and defining system communication area parameters. The following paragraphs discuss these elements of the RSX-11D system.

Partitions
Partitions are areas of contiguous real memory that are used for task execution. There are two modes of partition usage: user controlled, where only one task at a time can occupy the partition; and system controlled, where the system controls allocation of memory within the partition for execution of one or more tasks. The name, base address, size, and mode of each partition is specified during system generation and cannot be changed on-line. Tasks are installed to run in a particular partition but, upon specific request, can run in any partition that is large enough. Figure 5-1 shows an example of how memory is divided into user-controlled and system controlled partitions.

![Figure 5-1 Typical RSX-11D Memory Layout](image)

Shared global areas (SGA’s) require space in partitions. Shared global areas include libraries, global common blocks and pure areas of multi-user tasks. Libraries are normally read-only and are used for code. Global common blocks can be addressed on a per-task basis as read/write or read-only. This is a characteristic of the task rather than the global area and is specified during task building. The global area includes FORTRAN COMMON space and, normally, is used for intertask exchange.
of large amounts of data. SEND and SEND AND REQUEST directives also can be used for large amounts of data.

In RSX-11D systems, some tasks are multiuser tasks, for example, the MACRO assembler, the FORTRAN compiler and the text editor. Multiuser tasks consist of a pure area and an impure area. The pure area of a multiuser task is the area that is not modified during task execution and can be shared among multiple versions of the task. The impure area changes during execution: one copy of the impure area exists for each simultaneous user of the task.

In RSX-11D systems, SGA's are normally loaded into memory when needed and removed automatically when the tasks using them exit. They can also be fixed in memory by fixing a task which forces the SGA to be loaded. (Such a task may, for example, consist only of an EXIT directive.)

**System Communication Area (SCOM)**
This memory space contains the tables, lists, system subroutines, and other information required by the Executive to perform its functions and maintain control of the system. It consists of a number of fixed tables or lists and code, with the remaining space being available in variable-length nodes. These nodes are used by the Executive and for the purpose of intertask communication. The major SCOM lists and tables are:

**PHYSICAL UNIT DIRECTORY**
The physical unit directory (PUD) is a table of entries for each physical unit specified during system generation. When a logical unit number is assigned to a physical unit, the physical unit is represented by the address of the corresponding PUD entry.

**TASK PARTITION DIRECTORY**
The task partition directory (TPD) is a table of entries for each task partition defined during system generation.

**GLOBAL COMMON DIRECTORY**
The global common directory (GCD) is a linked list of entries for each global common block and library installed in the system. The GCD entries are created for the pure area of multiuser tasks and for shareable global areas (libraries and global common), at the time they are installed.

**SYSTEM TASK DIRECTORY**
The system task directory (STD) establishes the maximum number of tasks that can be installed at one time. Normally the number of installed tasks is greater than the number of executing tasks. The number of simultaneously installed tasks is limited by the number of system task directory entries specified during system generation. The number of executing tasks is limited by the number of user-controlled partitions and the number of tasks that can fit into system controlled partitions. The number of STD entries for tasks should be greater than the number of available partitions so that a maximum number of tasks can execute simultaneously. Installed tasks can be removed as needed to free additional STD entries. Dynamic installation upon request and removal upon exit is also provided through a RUN command function.
ACTIVE TASK LIST
The active task list (ATL) is a priority-ordered list of active tasks. A task is considered active from the time its execution starts until the time it has exited. The ATL is used to determine which highest priority task to give control of the CPU when a significant event occurs.

MEMORY REQUIRED LIST
The memory required list is a priority-ordered list of active tasks that require memory in a partition. There is an MRL for each partition. Whenever a nonfixed task exits, the MRL associated with that partition is scanned, and an attempt is made to assign memory to the highest-priority task in the list. If the attempt is successful, the task is moved from the MRL to the active task list. It is on the MRL where the RSX-11D time-based scheduling algorithm operates. The checkpointable tasks are moved up and down the MRL as the time cycle proceeds.

CHECKPOINTABLE TASK LIST
For each partition, there is a priority-ordered list of checkpointable tasks that are active in that partition.

FIXED TASK LIST
The fixed task list (FTL) is a list of tasks that have been fixed in memory but are not active. When a fixed task is made active, its node is relinked from the FTL to the ATL. When the task exits it is relinked into the FTL.

MCR COMMAND BUFFER
The MCR command buffer holds the data for a requested MCR function task. The buffer is set up by the MCR dispatch task. The nodes required for the buffer are returned to the pool after the GET MCR COMMAND LINE directive passes the command line to the MCR function task.

BATCH COMMAND BUFFER
The batch command buffer holds data for the batch processor. It functions in the same manner as the MCR command buffer.

ASYNCHRONOUS SYSTEM TRAP QUEUE
The asynchronous system trap queue (ASQ) is a list that operates on a first-in/first-out basis. It consists of one node for each AST (Asynchronous System Trap) to be executed for the task as defined by the STD entry.

CLOCK QUEUE
The clock queue consists of one node for each operation to be performed at some time in the future, for example, starting the execution of a scheduled task. A ticks-till-due count in the first node of the clock queue is decremented at each clock tick until the node becomes due (i.e., until the count is zero). Then the indicated operation is performed.

NODE POOL
The node pool is a dynamic storage area in SCOM. It consists of a number of fixed-length empty blocks of words called nodes, and is initially allocated during system generation. When a node is needed to expand a system list, it is taken from the node pool. When a node is no longer needed, it is returned to the pool. Dynamic memory requirements
for a system are dependent on the configuration, application and amount of system loading. Some of the sources requiring dynamic memory are:

- task installation
- I/O requests
- partition definition
- task scheduling
- and intermediately buffered I/O (e.g., terminal I/O).

### 5.3.2 RSX-11M Executive and Memory Structures

As in the RSX-11D system generation, the RSX-11M system generation also requires the user to allocate partitions and define system communication area parameters. In RSX-11M systems, however, the user can define and delete partitions on-line.

Partitions in the RSX-11M system are essentially similar to RSX-11D partitions with the exceptions of subpartitioning in RSX-11M user controlled partitions and optional time-based swapping in RSX-11D system controlled partitions. RSX-11M systems have two kinds of Shared Global Areas: libraries and global common blocks. The shared global areas (SGA's) require space in partitions, but are not loaded automatically when tasks require them. In RSX-11M systems, SGA's are fixed and must be explicitly loaded with a task requiring them.

System communication areas in RSX-11M systems are also similar to those in RSX-11D systems, with some exceptions.

The RSX-11M basic Executive organization is illustrated in Figure 5-2. The individual regions are explained below; some of the regions are directly affected by system generation parameters.

#### Trap Vectors

This region contains the hardware trap and interrupt vectors and requires 128 words. This region is expandable during system generation to a maximum of 256 words.

#### System Stack

Used for nesting interrupts and internal calls made by the Executive. Forty words are required.

#### System Common Data

Contains pointers filled in during system generation.

#### System Tables

Contains the data used to control system operation. Included are Partition Descriptions, the System Task Directory, and Device Tables. The total size of the table region is established by system generation configuration selections.

#### Dynamic Storage Region

The Executive has continuing needs for temporary storage. Such storage is acquired, used, and returned to the available pool. If a given Executive service requests dynamic storage, and it is unavailable, the Executive will inform the user task, which usually waits for some storage to become available. The size of this region is important, for if it is too small, waiting periods will be induced; if it is too large, system effectiveness is lowered, since fewer tasks can fit in memory. The size of the region is a system generation parameter. The initial allocation can later be extended on-line by an MCR command from the console.

5-18
The Basic Executive

The basic Executive includes the code that controls the multiprogramming environment, performs task checkpointing and power fail restart, and handles system traps. During system generation, the user has the option of including or omitting the following services:

- task checkpointing
- task checkpointability during terminal input
- memory management unit support
- dynamic memory allocation
- automatic memory compaction
- I/O rundown (automatic system clean-up after a task aborts and leaves files in an indeterminate state)
- asynchronous system trap support
- external (user-written) MCR function support
- task termination and device not ready messages
- power failure recovery
• GET PARTITION PARAMETERS directive support
• GET SENSE SWITCHES directive support
• GET TASK PARAMETERS directive support
• SEND/RECEIVE directives support
• automatic install, request and remove on exit support (RUN command option)
• logical device assignment support
• setting upper/lower case conversion for terminal input
• transparent terminal READ/WRITE support
• Executive level round-robin scheduling
• user-written device driver support
• Executive debugging tool
• panic dump and system failure reporting

Executive Directive Services
This region contains the service routines which respond to the directives issued by users to request Executive services. These programs make use of the Basic Executive.

Device Drivers
Three drivers can be included in the basic 8K Executive:

• Disk
• Cassette, DECTape, magnetic tape, line printer or floppy disk
• terminal

These are multi-unit drivers that can service up to the maximum devices controlled by the respective hardware interfaces.

In general, Executives that grow beyond 8K will do so because of the presence of additional drivers. Drivers are included in the system during system generation.

Task Loader For Nonresident Tasks
This loader is a task and operates out of its own partition. Thus, it can run in parallel with system and user tasks. The loader, which is device independent:

1. Loads tasks on initial load requests;
2. Writes checkpointable tasks to disk when required, and
3. Returns previously checkpointed tasks to active competition for processor resources.

File System, Monitor Console Routine (MCR) and Task Termination (TKTN)

These three routines function as tasks. In the minimum system, they execute out of the same partition.

As distributed, the RSX-11M system generates a file system that runs in 2K words. The user has the option of building a 4K file system with greater processing speed for faster directory operations.
Panic Dump and Crash Modules
These two routines respond to system software failures, providing core dumps and selective analysis. They are not included (or shown) in the basic 8K system, but are mentioned because of their fundamental importance in error analysis. Most program development systems (as opposed to dedicated on-line systems) will likely include these routines.

In a 16K system with an 8K Executive, the remaining 8K words are available for user task partitions. In 16K word systems, partition definitions cannot be altered without regenerating the system. In systems with more than 16K words of memory, the user can re-define partitions on-line using an MCR console command. Figure 5-3 illustrates a typical memory organization for a large mapped RSX-11M system.

![Diagram of memory organization](image)

Figure 5-3 Mapped 124K-Word RSX-11M System

5.3.3 RSX-11S System Components
RSX-11S requires an RSX-11M system for system generation and program development. An RSX-11S system is generated from the RSX-11M system using the standard system generation process. The maximum hardware and software configuration is the same as that of an RSX-11M system with the exceptions of file system support, non-resident tasks, task checkpointing, and dynamic memory allocation.

Since it is based on RSX-11M, RSX-11S enjoys most of the inherent features and generation capability of that system. For example, RSX-11S

5-21
automatically supports all of the peripheral devices that RSX-11M supports including other hardware features such as floating point processors, parity memory and memory management. All are selectable at system generation and can be included in an RSX-11S system at the cost of memory.

The basic software building blocks for an RSX-11S system are:

1. The generatable features of the RSX-11M Executive (2.5 to 4K), including a special File Control Services (FCS) that contains no support for directory devices
2. All RSX-11M I/O device drivers
3. Subset MCR (2K)
4. Online Task Loader (2.5K)
5. System Image Preservation Program (1.5K)

The minimum software system is an Executive. The smallest Executive that can be generated requires 2.5K words of memory. Services that are omitted from the 2.5K Executive include:

- Address checking
- Asynchronous System Traps
- I/O rundown
- Task termination and device-not-ready notification
- External MCR functions (user-written functions)
- Install, Request, and Remove-on-exit support
- SEND, RECEIVE, GET TASK PARAMETERS, GET SENSE SWITCHES and GET PARTITION PARAMETERS directives
- Parity Memory support
- Network support
- All I/O drivers

Although omitted from the minimum Executive, these features can be generated into an RSX-11S system at the cost of memory.

The minimum RSX-11S software system must include the Executive and I/O device drivers. For example, two to four I/O device drivers could be added to the minimum Executive at the cost of an additional 1.5K words of memory. In an 8K word system, approximately 4K words would be available to application tasks.

If operator communication is required, subset MCR can be included in a system at a cost of 2K of memory. In an 8K system this still leaves approximately 2K for application tasks.

The On-line Task Loader (OTL) can be included in an RSX-11S system if on-line loading of tasks is desired. OTL is capable of loading tasks from the following media:

---

* These features must be included if FORTRAN tasks are desired.
• Tape Cassette (TA11)
• Paper Tape (PC11 or PR11)
• DECtape (TC11)
• Magtape (TM11/TMA11 or TJU16)
• Floppy Disk (RX11)

Tasks are created on a host RSX-11M system, transferred to the load medium using RSX-11M’s File Exchange Utility (FLX), and then loaded into a running RSX-11S system using OTL. The minimum size for OTL is 2.5K words. In 2.5K words, however, OTL supports only one load device. On-line task loading requires a 16K word system, since approximately 8.5K words will be required for system software (2.5K Executive, 2K MCR, 1.5K device drivers, and 2.5K OTL).

The System Image Preservation Program (SIP) is an on-line utility task that provides the capability to save the image of a running system onto a load device medium in bootstrappable format. Permissible load devices are the same as for OTL. The saved system can subsequently be restored by bootstrapping it from the load device medium. The minimum size for SIP is 1.5K words. In 1.5K words, it can support only one load device.

The standard RSX-11M File Control Services (FCS) record I/O package contains a large amount of code to support file-structured devices, RSX-11S contains no file support and therefore this code is unnecessary. The special version of FCS provided with RSX-11S is the standard FCS without the file support code. This provides a significant reduction in size while maintaining complete transparency.

5.4 SYSTEM CONVENTIONS
To simplify operations, RSX-11 systems observe certain conventions with respect to devices, file structures, file naming, operator commands, indirect files, and batch commands. The file conventions are relevant only for RSX-11M and RSX-11D systems. Batch processing is relevant only for RSX-11D systems. Indirect MCR command files are relevant only for RSX-11M systems.

5.4.1 Devices
The RSX-11 systems support a variety of peripheral devices. They are referred to by a 2-letter name and an optional 1- or 2-digit unit number followed by a colon. For example, TT12: represents user terminal number 12. Peripheral devices can be referred to by mnemonics, by pseudo-device names or, in task references, by logical unit numbers. In addition, RSX-11M systems support logical device name assignments.

Pseudo-device names are associated with normal device mnemonics assigned by the system manager. They permit the system manager to dynamically determine the physical devices that will send or receive information. Both RSX-11M and RSX-11D support the following pseudo devices:

SY: System device; indicates the device on which the system disk is mounted.

5-23
TI: Terminal interface; indicates the terminal with which a particular task is associated. Each terminal has a unique TI. The TI of each task is assigned to the requesting terminal.

CL: Console log; indicates the device normally used for the listing of files. The CL device is normally redirected to the line printer.

CO: Console output; indicates the device by which the system can communicate with the system manager. The CO device is normally redirected to the system console.

In addition, RSX-11D provides support for the following pseudo devices:

CI: Console input; indicates the device by which the system manager can communicate with the system.

MO: Message output; indicates that the system should provide detailed error messages when an error occurs. The advantage of the message output pseudo device is that the MO device handler needs to be resident in memory only when additional messages would be helpful, for example, during program development.

SP: Spooling output; redirected to the device to which all spooled output files will be written.

BP: Batch pseudo device; used in place of TI to execute batch jobs. The system uses BP to identify those output files generated by tasks initiated by the batch processor. These files are concatenated by the queue manager.

Logical unit numbers (LUN's) provide the mechanism for programs to maintain device independence. The logical unit numbers used in a program can be assigned by means of device mnemonics to any available peripheral device that performs the desired function. LUN's can be assigned by the programmer at task build time, or by the task itself at run-time. Because the system provides default LUN assignments, it is not always necessary to assign a LUN to a task. Furthermore, LUN's can be changed by an MCR function for any installed, inactive, non-fixed task.

RSX-11M has an additional facility for associating a logical name with a physical device, called logical device assignment. Logical device assignments are a convenient way to associate logical names with physical devices. There are two types of logical device assignments: local and global. Local assignments apply only to commands and tasks initiated from the terminal on which the assignment was made. Global assignments apply to all commands or tasks. If a logical name is defined as both global and local, the local assignment overrides the global assignment. Logical device names can be the same as physical device names or can be any character string using the syntax for device names.

5.4.2 File Structures
Both RSX-11M and RSX-11D support a common file structure for disk and DECTape called Files-11. In addition, RSX-11D supports ANSI Standard Level 3 format for single- or multi-volume magnetic tape files.

Files-11 is a general purpose file system that provides a facility for the dynamic creation, extension, and deletion of files on disk or DECTape.
Designed into Files-11 is a scheme for volume and file protection which allows the owner of a volume or file to deny all access or certain kinds of access to all users, groups of users, or particular users in the system. This scheme for volume and file protection provides the key to the system protection, in that only users with access privileges are allowed access.

A Files-11 volume is a collection of files which reside on a single disk or DECtape. The system can directly address each file on the volume by means of file pointers which reside in the volume’s directory files.

Each Files-11 volume has two kinds of directory files that are used for file management: the Master File Directory (MFD) file, and User File Directory (UFD) files.

The Master File Directory (MFD) file is automatically generated by the file system when a volume is initialized as a Files-11 volume, and is used to store pointers to all of the User File Directory (UFD) files on the volume.

User File Directory (UFD) files are created as needed. They are used to store pointers to all of the files belonging to, or associated with, the user whose account number (User Identification Code or UIC) corresponds to the UFD file name.

All Files-11 files, whether MFD, UFD, or user files, have the same basic format. All files have a file header area, and one or more data area(s). Figure 5-4 illustrates the Files-11 file format.

```
Figure 5-4  Sample Files-11 File Structure
```

\[5-25\]
The file header area contains all the pertinent information required by the file system to process the file. For the purposes of this introduction, the user need be familiar with only the following fields:

**File Owner Field**
The file owner field contains the account number (UIC) of the user who created the file.

**Filename Field**
The filename field contains the name assigned to the file when it was created. Filenames can be a maximum of nine alphanumeric characters long.

**File Type Field**
The file type field contains the mnemonic that identifies the file by its functionality; for example, FTN defines a FORTRAN source file.

**Version Number Field**
The version number field identifies the particular version or generation of the file.

**File Protection Field**
The file protection field contains a code that describes who is allowed to access the file: System, Owner, Group or World. It also describes the type of access allowed: read, write, extend or delete.

**Data Pointer Field**
The data pointer field describes the physical allocation of the file on the volume. Each data area pointer describes a physically contiguous portion of the file.

By establishing pointers to blocked data in the file's header area, as opposed to storing the data immediately following the file header, the system accomplishes two things: all files on the volume have the same structural format regardless of functionality; and all fragmented or non-contiguous areas of the volume can be put to use, that is, a file can be expanded by merely attaching another pointer to a blocked data area in its file header.

Users always address data in a file-relative manner. The translation of file-relative address into physical addresses is performed by the file system and is completely transparent to the user.

### 5.4.3 File Specifiers

Any system component that has a requirement to refer to files does so using a standard file command string with the following general format:

\[
\text{output file specifications} \equiv \text{input file specifications}
\]

There can be several file specifications on either side of the equal sign. Optional switches are used to indicate desired operations other than default operations. File specifiers have the following format:

\[
\text{dev:[uic]}\text{filename.type;version/switch}
\]

where:

- **dev:** is the physical device on which the volume containing the desired file is mounted (for example DTO: or DK1:).
is the User Identification Code that specifies the user file directory containing the desired file.

filename is the name of the file. Filenames can be up to nine alphanumeric characters in length. Filename and type are always separated by a period.

type is a designator distinguishing among various forms of files. For example, a FORTRAN source file might be named COMP.FTN, while the object file associated with that program might be named COMP.OBJ.

;version is a number used to differentiate among versions of a file. For example, when a file is first created using the text editor, it is assigned a version number of 1. If the file is subsequently opened for editing, the editor keeps the original file for back-up and creates a new file with the same filename and type designations, but with a version number of 2.

/switch is usually an optional qualifier. Switches are normally used either to direct the execution of a task, or to qualify an input parameter.

If any of the file specifier elements except the filename is omitted from the file specifier, the system uses a default value. A task can also establish defaults for a file. The system default for the device name is the system device. The default for the user file directory specification is the UFD that corresponds to the UIC under which the task is running. The default for the version specification is the latest version number. For both RSX-11M and RSX-11D systems, the defaults for the type specification vary according to the operation to be performed. The common set of file types are:

.CMD An indirect file containing a list of task or MCR commands for a task (in RSX-11M the commands can also be MCR commands)

.DAT A data file, as opposed to a program file

.DIR A directory file, for example, a UFD directory

.FTN A FORTRAN source program

.LST A listing file

.MAC A MACRO-11 source program

.MAP A task builder memory allocation listing file

.MLB A user macro library

.ODL An overlay description file

.OLB An object module library

.SML The system macro library

.STB A symbol table file

.TSK A task image file

.CRF A cross reference file that will be appended to the map file

RSX-11M systems also recognize the following type:

.SYS A system image file

RSX-11D systems also recognize the following types:

.BIS A batch input stream file

.EXE A batch execute file
.FOR  A batch FORTRAN file
.LIB  A batch library file
.LIS  A batch listing stream
.SAV  A system image file or file generated by the EDI editor
.SOB  A batch concatenated object file
.SPR  A spooled output file
.SRT  A SORT work file

5.4.4 MCR Operator Commands and Terminal Control
The Monitor Console Routine (MCR) is the terminal interface between the user and the RSX-11 operating system. In the system, terminals can have either of two functions: command or slave. The system does not accept any unsolicited input from a slave terminal; its I/O is normally under task control. A command terminal is used to activate MCR and interface with the system using MCR system commands. In both RSX-11D and RSX-11M, the SET command can be used to characterize a terminal as a slave or command terminal.

MCR's system commands enable the general user to perform the following functions:

- gain access to the system
- initiate and terminate execution of system or user programs

In addition, the privileged user can perform the following additional functions:

- adjust, modify and control the system environment

The privileged MCR user has complete control over the system's operation. The general user has no protection from the operations directed by the privileged user.

Privileged and non-privileged users are established in slightly different ways in the RSX-11D and RSX-11M systems. In RSX-11M, the privileged characteristic is associated with certain terminals, as determined by the system manager initially during system generation and subsequently by setting terminal characteristics using the SET command. Non-privileged commands can be invoked at any command terminal. Privileged commands can only be invoked at privileged terminals.

In RSX-11D, the system determines user privileges by the UIC under which the user logs on to the system. UICs are assigned by the system manager. Only users who have logged in under privileged UICs can issue privileged MCR commands. In RSX-11D as in RSX-11M, the SET command can be used to control privileged user characteristics.

In RSX-11D, MCR services are organized internally as a re-entrant dispatcher task and a set of independent system command tasks. The individual command tasks are called by the dispatcher task as needed and are released automatically after execution. In RSX-11M, the MCR task itself processes most of the standard MCR commands, but will call independent command tasks to process some commands. In both cases, the MCR organization makes it possible for users to add operator console services to meet their application needs.

5-28
The RSX-11 systems include four different kinds of MCR commands: initialization commands, informational commands, task control commands, and system maintenance commands. Table 5-4 lists the RSX-11 MCR commands, and indicates the systems in which they are available.

<table>
<thead>
<tr>
<th>Initialization Commands</th>
<th>Function</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOT</td>
<td>Bootstraps a system that exists as a task image file on a file-structured volume. It provides a convenient means for terminating one system and starting another. For example, BOOT can be used for terminating a real-time system and starting a program development system.</td>
<td>D, M</td>
</tr>
<tr>
<td>TIME</td>
<td>Lists the time and date maintained in the system clock calendar. A privileged user can change the time and date.</td>
<td>D, M and S</td>
</tr>
<tr>
<td>MOUNT</td>
<td>Declares that a volume is logically on-line for access by the system.</td>
<td>D, M</td>
</tr>
<tr>
<td>DISMOUNT</td>
<td>Declares that a volume is logically off-line and can not be accessed by the system.</td>
<td>D, M</td>
</tr>
<tr>
<td>INITVOLUME</td>
<td>Initializes a volume for use by the system.</td>
<td>D, M</td>
</tr>
<tr>
<td>INSTALL</td>
<td>Installs a task in the system by making an entry in the System Task Directory; this allows the operator to subsequently run the installed task. This function is performed by the On-line Task Loader utility in RSX-11S systems. In RSX-11D systems, this command is also used to install a built SGA in the system by making an entry in the Global Common Directory.</td>
<td>D, M</td>
</tr>
<tr>
<td>SET</td>
<td>Allows the user to establish or alter a variety of parameters, including terminal device characteristics, command or slave terminal characteristics, and default UIC for a terminal. In RSX-11D, the user can set the MCR terminal timeout parameter, state that all output to a particular terminal device is to be spooled, or state that all input from a particular device (e.g., card reader) is to be spooled. In RSX-11M, the user can create partitions and subpartitions or add space to the dynamic storage region.</td>
<td>D, M</td>
</tr>
<tr>
<td>UFD</td>
<td>Creates a User File Directory on a volume and enters its name in the Master File Directory.</td>
<td>D, M</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>Allows the user to create or change the password for a UIC.</td>
<td>D only</td>
</tr>
</tbody>
</table>
Table 5-4  RSX-11 MCR Commands (Cont.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELLO</td>
<td>Allows the user to log in to the system and be identified as a valid user.</td>
<td>D only</td>
</tr>
<tr>
<td>BYE</td>
<td>Logs a user off the system. The only valid command that MCR recognizes after BYE is HELLO.</td>
<td>D only</td>
</tr>
</tbody>
</table>

Informational Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE TASK LIST</td>
<td>Lists the active tasks in the system, indicating the tasks' current status, for example, task suspended, waiting for I/O, etc.</td>
<td>D, M, S</td>
</tr>
<tr>
<td>BAD</td>
<td>Locates any unusable blocks on a disk.</td>
<td>D, M</td>
</tr>
<tr>
<td>DEVICES</td>
<td>Prints the symbolic names of all device units known to the system. Indicates if a device handler is resident, a volume is mounted, or to what device a symbolic device name is assigned. (In RSX-11D this is an option of the SYSTEM command.)</td>
<td>M only</td>
</tr>
<tr>
<td>LOG</td>
<td>Allows the user to log comments about the system. (In RSX-11M systems, a command line starting with a semi-colon or an exclamation point is interpreted as a comment line.)</td>
<td>D only</td>
</tr>
<tr>
<td>LUNS</td>
<td>Prints a list of the physical device units and corresponding logical unit numbers for an indicated task. It is used to determine which physical devices a task requires.</td>
<td></td>
</tr>
<tr>
<td>PARTITIONS</td>
<td>Lists a description of each memory partition, including partition name, base address and use. It also lists a description of each memory-resident sharable library and global common block. (In RSX-11D, this is an option of the SYSTEM command.)</td>
<td>M only</td>
</tr>
<tr>
<td>POOL</td>
<td>Prints statistics about node pool usage. (In RSX-11M, this function is served by the SET command).</td>
<td></td>
</tr>
<tr>
<td>TASK LIST</td>
<td>Lists a description of each task installed in the system, including task name, version number, default partition name, priority and size. (RSX-11D, this is an option of the SYSTEM command.)</td>
<td>M and S</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Lists the names of the tasks that are using any one or all of the following system lists or tables: Active Task List, Memory Required List, Clock Queue, Global Common Directory, Checkpointable Task List, I/O Request Queue. Fixed Task List, System Task Direc-</td>
<td>D only</td>
</tr>
</tbody>
</table>
### Table 5-4  RSX-11 MCR Commands (Cont.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WHO</strong></td>
<td>Indicates which terminals are in use, provides the UIC of the user, and indicates whether the user is privileged.</td>
<td>D only</td>
</tr>
</tbody>
</table>

#### Task Control Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>Allows the user to change the priority of a task.</td>
<td>D, M</td>
</tr>
<tr>
<td>FIX</td>
<td>Allows the user to fix a task in its partition in memory. A fixed task gets faster response to requests for execution. (A function exists in OTL for RSX-11S systems to fix a task when it is loaded.)</td>
<td>D, M</td>
</tr>
<tr>
<td>UNFIX</td>
<td>Frees fixed tasks from memory.</td>
<td>D, M</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>Assigns a logical device name to a physical device.</td>
<td>M only</td>
</tr>
<tr>
<td>REASSIGN</td>
<td>Reassigns a Logical Unit Number (LUN) from one physical device and assigns it to another.</td>
<td>D, M only</td>
</tr>
<tr>
<td>REDIRECT</td>
<td>Redirects all I/O requests from one physical device to another.</td>
<td>D, M and S</td>
</tr>
<tr>
<td>LOAD</td>
<td>Makes a specified device handler resident in memory and ready to honor I/O requests.</td>
<td>D only</td>
</tr>
<tr>
<td>UNLOAD</td>
<td>Unloads a specified device handler from memory.</td>
<td>D only</td>
</tr>
<tr>
<td>QUEUE</td>
<td>Allows a user to queue files to a spooled line printer or terminal, list all the elements in the queue, kill an element in the queue, or change an element’s priority, special forms type request, or number of copies.</td>
<td>D only</td>
</tr>
<tr>
<td>OPERATE</td>
<td>Enables the user to direct the output processing of the despooler, for example, start processing, stop or restart processing. The privileged user can also set or change output forms type parameters.</td>
<td>D only</td>
</tr>
<tr>
<td>RUN</td>
<td>Initiates the execution of an installed task. An installed task can be started immediately, started a specified time from when the command is issued, started a specified time from the next time unit, or started at an absolute time of day. A special option of the RUN command allows the user to run a task not</td>
<td>D, M and S</td>
</tr>
<tr>
<td>Command</td>
<td>Function</td>
<td>Systems</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>CANCEL</td>
<td>Cancels any pending periodic rescheduling for a task.</td>
<td>D, M and S</td>
</tr>
<tr>
<td>ABORT</td>
<td>Terminates execution of a specified task.</td>
<td>D, M and S</td>
</tr>
<tr>
<td>RESUME</td>
<td>Continues execution of a previously suspended task.</td>
<td>D, M and S</td>
</tr>
<tr>
<td>REMOVE</td>
<td>Removes a task name from the System Task Directory (opposite of INSTALL).</td>
<td>D, M and S</td>
</tr>
<tr>
<td></td>
<td>Under RSX-11D, this command is also used to remove a Shareable Global Area (SGA) from the Global Common Directory.</td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Continues execution of a task that selected the suspend option of the pseudo device Message Output handler.</td>
<td>D only</td>
</tr>
<tr>
<td>DISABLE</td>
<td>Effectively deletes a task from the system without actually removing it. A disabled task can not be initiated by a RUN command or similar directive. This command allows the user to inhibit the use of a task for a period a time.</td>
<td>D only</td>
</tr>
<tr>
<td>ENABLE</td>
<td>Cancels the effect of the DISABLE command.</td>
<td>D only</td>
</tr>
</tbody>
</table>

**System Maintenance Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE</td>
<td>Copies the memory image of the system to the system disk so that a bootstrap can reload it and start up the system. (In RSX-11S systems, the System Image Preservation utility performs this function.)</td>
<td>D, M</td>
</tr>
<tr>
<td>OPEN</td>
<td>Allows the privileged user to examine or modify a word in memory.</td>
<td>D, M and S</td>
</tr>
<tr>
<td>BRK</td>
<td>Breaks to the Executive Debugging Tool (XDT).</td>
<td>M only</td>
</tr>
</tbody>
</table>

In addition to the MCR commands available to control system execution, RSX-11 system provide the following special terminal control characters:

- **CTRL/C**: Activates MCR at a terminal. The system types the prompt "MCR>". Note that, unlike most other PDP-11 systems, the
Table 5-4  RSX-11 MCR Commands (Cont.)

RSX-11 family does not use CTRL/C to affect the execution of any currently running tasks other than MCR.

CTRL/Z  Logical end-of-file; when typed in response to a prompt from most utility programs, CTRL/Z causes the program to exit.

CTRL/I  Causes a horizontal tab.

CTRL/K  Causes a vertical tab of one line in RSX-11D, of four lines in RSX-11M.

CTRL/L  Causes a form feed to the top of the next page in RSX-11D, or eight line feeds in RSX-11M.

CTRL/U  Cancels the current input line.

CTRL/O  Enables or disables output to the terminal.

CTRL/S  Temporarily suspends output to the terminal. This feature enables users with high-speed terminals to fill the display screen, stop output with a CTRL/S and then continue with a CTRL/Q. (RSX-11M and RSX-11S systems only.)

CTRL/Q  Resumes printing of characters on the terminal from the point at which printing was interrupted using CTRL/S. (RSX-11M and RSX-11S systems only.)

CTRL/R  Causes the system to reprint the current line entered in the terminal buffer and allows the user to view exactly what has been entered so far. (RSX-11D systems only.)

CTRL/X  Causes a user-written terminal interface to run (RSX-11D systems only).

5.4.5 Indirect Files (Command Files)
An indirect file is a sequential file containing a list of commands. Rather than typing commonly-used sequences of commands, the user can type the sequence once and store it on a file using the Editor utility program. To execute the sequence, the user types an “at” sign (@) and then the command filename. The affected task locates the indirect file and executes the commands it contains.

There are two types of indirect files: indirect task command files and indirect MCR command files. Both RSX-11D and RSX-11M support indirect task command files. Only RSX-11M supports indirect MCR command files.

The commands contained in an indirect task command file are task specific. They can be interpreted only by a specific task such as the MACRO assembler, the Task Builder, or another utility program. The indirect file is specified in place of the command line normally given to the task when it is run. For example, to give an indirect file to the MACRO assembler to execute, the user types:

MCR > MAC @CMDFIL.CMD

5-33
which causes MACRO to read and execute the file CMDFIL.CMD for all of its commands.

RSX-11M supports an indirect command file processor for MCR command processing. In this case, the indirect file contains commands to the MCR console interface. To execute a series of MCR commands using the indirect MCR command file processor, the user types the “at sign” followed by an indirect file’s name in response to the MCR prompt. For example, to execute a series of MCR commands contained in the file named BEGIN.CMD, the user types:

```
MCR > @BEGIN.CMD
```

In addition to the standard MCR commands, the RSX-11M indirect command file processor can accept special commands that allow the user to control command file processing. These special commands provide the following capabilities:

**INITIATE PARALLEL TASK EXECUTION**
It is possible to request initiation of a task and not wait for the task to terminate before having the next command line processed. Normally, the indirect file processor passes a task initiation command line to MCR and then waits until the command is executed before continuing. In this case, however, the indirect file processor can initiate a task, pass a command string to it, and continue processing the indirect file command lines in parallel with the initiated task’s execution.

**WAIT FOR A TASK TO FINISH EXECUTION**
Indirect command file processing can be suspended until a particular task has terminated.

**TEST IF A TASK IS INSTALLED OR NOT INSTALLED**
A test can be made to determine whether a particular task is installed in the system or not. If the task is installed, the rest of the command line is processed. If the task is not installed, the remainder of the command line is ignored.

**TEST IF A TASK IS ACTIVE OR NOT ACTIVE**
A test can be made to determine whether a task is active or not. If the task is active, the rest of the command line is processed. If the task is not active, the rest of the command line is ignored.

**SUSPEND EXECUTION FOR A SPECIFIED TIME INTERVAL**
Indirect file processing can be suspended for a specified number of clock ticks, seconds, minutes or hours. When the interval is exhausted, indirect file processing continues at the point where it was interrupted.

**PROVIDE COMMENTARY**
Comments can be included in the command file. Comments are displayed on the entering terminal and are convenient to provide explanation or to give instructions to the user who issued the command file.

**PAUSE FOR OPERATOR ACTION**
It is possible to suspend indirect file processing until the user at the entering terminal performs some action. The file processor prints a message on the terminal to notify the user. To continue indirect command file processing, the user types a RESUME command.
ASK A QUESTION AND WAIT FOR A REPLY
It is possible to print a message on the entering terminal, suspend indirect command file processing until input is received, and then set a specified symbol true or false depending on the input contents. If the symbol is not already defined, an entry is made in the symbol table and its value set.

DEFINE A SYMBOL
A symbol can be defined or its value can be changed. A symbol can represent either a true or a false value. When a symbol is first defined, a symbol table entry is made and set to a specified value. A symbol can have any alphanumeric name up to six characters long.

TEST IF A SYMBOL IS TRUE OR FALSE
The value of a symbol can be tested at the beginning of a command line. If the test is true, the rest of the command line is processed. If the test is false, the remaining part of the command line is not processed.

TEST IF A SYMBOL IS DEFINED OR NOT DEFINED
A test can be made to determine whether a symbol has been defined or not. If it is defined, the rest of the command line is processed. If it is not defined, the rest of the command line is ignored.

DEFINE LABELS
A command line in the command file can be labelled.

BRANCH TO A LABELLED LINE
Control can be transferred from one line in an indirect file to another line in an indirect file by an unconditional branch to a labelled line. A branch can transfer processing to a labelled line before the branch command line or after the branch command line.

BRANCH TO A LABELLED LINE ON DETECTING AN ERROR
Control can be transferred from one line in an indirect file to another line if an error occurs. If the conditional branch on error line is processed, control is passed to a specified command line if one of the following errors is detected: undefined symbol reference, symbol table overflow, undefined label, or syntax error. This feature provides the user with a means of gaining control to clean up before aborting execution.

COMBINED LOGICAL TEST
Tests can be combined using Boolean AND and OR directives. In addition, an implied logical AND is effected if multiple tests are placed on the same command line; the command on the line is executed only if all tests are true.

MCR indirect files can reference other MCR indirect files. Up to four levels of indirect MCR command files can be specified. Each time a new level is entered, all symbols previously defined are masked out of the symbol table and only symbols defined in the current level are available. When control returns to a previous level, the symbols defined in that level are available again.

RSX-11M can execute multiple MCR indirect files simultaneously. Several users at MCR command terminals can initiate MCR indirect com-
mand file processing. This effectively provides multiple-stream "batch" processing in RSX-11M systems.

5.4.6 Batch Processing (RSX-11D systems only)
The RSX-11D batch capability allows the user to submit jobs to a system operator for subsequent processing. Each job contains a series of commands to the system and, optionally, associated data. Once the job has been submitted, its tasks can execute without operator intervention unless the job explicitly requests operator action. The following is a list of the batch commands that direct job processing:

$JOB Indicates the start of a job
$EOJ Indicates the end of a job
$MOUNT Mounts a device
$DISMOUNT Dismounts a device
$FORTRAN Compiles a FORTRAN source program
$MACRO Assembles a MACRO source program
$COBOL Compiles and optionally executes a COBOL source program
$LINK Links tasks for execution
$RUN Causes a task to execute
$DATA Indicates the start of data
$EOD Indicates the end of data
$MCR Indicates that the command to follow is to be directed to MCR
$MESSAGE Issues a message to the system operator
$COPY Copies files
$CREATE Creates new files from data in the batch stream
$SORT Sorts a data file
$DELETE Deletes files
$DIRECTORY Provides a directory of the default UFD
$PRINT Prints files

Jobs submitted to batch are executed as low-priority jobs. As such, batch jobs are prohibited from issuing certain real-time processing executive directives. Batch is, however, an excellent means of executing larger data processing jobs for which real-time requirements are not critical.

The RSX-11D batch processor operates on a single-stream of data; therefore only one batch job is executed at a time. Any user can submit batch jobs from record-oriented devices such as card reader or terminal, or submit a job file contained on a Files-11 device.

RSX-11D batch consists of two tasks. The first is an MCR function (BAT) that reads and interprets the input stream. Each job is queued on disk as a separate file. The second task is the batch processor, which reads and executes each job.

The first task, which requires 6K words of memory, reads batch input and creates at least two other files for each job: a job file and an error file. If no errors are detected during the input process, the error file is deleted and the job file is queued for execution. If errors are detected, the job file is deleted and the error file is printed on the batch log device.

When an error is detected in the input file, the entire job is interpreted even though it is not executed so that the user can be informed of all

5-36
errors in the job. All command sequence and syntax errors are detected and, therefore, can be corrected before the job is submitted again.

The second task, which requires 2K words of memory, is resident throughout job execution. It reads and executes each job in the stream. As each command in a job is executed, the command is printed on the batch log device.

Batch can be requested from any terminal by typing BAT in response to an MCR prompt or by using the EB command of the text editor program. Batch can also be requested automatically by reading a batch input deck from a spooled card reader.

The following are examples of requesting batch using the MCR function.

MCR>BAT CR: Indicates that the card reader is not spooled and the input stream is in the card reader.
MCR>BAT BAT>COMP.BIS Requests batch and specifies the file COMP.BIS as containing the input stream.
MCR>BAT PROG5 Requests batch to process the input file PROG.BIS.

A common procedure is to edit a batch input file and then execute the job contained in the file to test it. To make this process more convenient, the EDI text editor utility accepts the EB (exit to batch) command that closes the file that was being edited and automatically requests batch.

5.5 SYSTEM DIRECTIVES
System directives are instructions to RSX-11 to perform functions for an executing task. System directives allow tasks to perform the following:

- Schedule other tasks,
- Communicate with other tasks,
- Measure time intervals,
- Perform I/O functions,
- Suspend execution,
- Exit

Directives are generated by MACRO programs via macro calls and are supported for FORTRAN by library routines supplied by DIGITAL.

Directives are implemented solely through the PDP-11's EMT 377 instruction. By using only EMT 377, programs using EMT 0 through EMT 376 can be run via a non-RSX system trap. Any EMT other than EMT 377 traps to a task-contained service routine that can be written to simulate another environment to whatever degree is desired.

By using macro calls, instead of executing the directive, the programmer need only reassemble a program if changes are made in the directive specifications, rather than being required to edit the source code.

Listed below are the RSX-11 family's Executive directives. Those starred with a single asterisk cannot be issued by tasks written to run under RSX-11S systems.
**Task Execution Control Directives**

**REQUEST**
Instruct the executive to request immediate execution of an indicated task. Memory partition, priority, and UIC may be specified.

**RUN**
Instruct the executive to schedule an indicated task's execution at a time specified in terms of delta time from issuance. Periodic rescheduling, memory partition, priority, and UIC may be specified.

**CANCEL**
Instruct the executive to cancel scheduled requests for an indicated task's execution. Normally, all scheduling for the indicated task is cancelled. In RSX-11D systems, a task can also cancel only the scheduling it requested for the indicated task.

**SUSPEND**
Instruct the executive to suspend execution of the issuing task until explicitly resumed via a RESUME directive.

**EXIT**
Instruct the executive that the issuing task has completed its execution. Unless the exiting task is fixed, its memory is freed for use by other tasks.

**ABORT**
Instruct the executive to terminate the execution of an indicated task.

**Task Status Control Directives**

**DISABLE CHECKPOINTING**
Instruct the executive to make the issuing checkpointable (swappable) task no longer checkpointable.

**ENABLE CHECKPOINTING**
Instruct the executive to reenable the checkpointed-disabled checkpointable issuing task, i.e., to nullify a DISABLE CHECKPOINTING directive.

**Informational Directives**

**GET TASK PARAMETERS**
Instruct the executive to return parameters related to the issuing task.

**GET PARTITION PARAMETERS**
Instruct the executive to return parameters related to an indicated memory partition.

**GET LUN DATA**
Instruct the executive to return information regarding an indicated logical unit to the issuing task.

**GET TIME PARAMETERS**
Instruct the executive to return current time parameters (year, month, day, hour, minute, second, tick and ticks/second).

**GET SENSE SWITCHES**
Instruct the executive to return the polarities of the sixteen console switches.
### Event-associated Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARE SIGNIFICANT EVENT</td>
<td>Instruct the executive to test for a new “highest priority task capable of execution” and, if found, interrupt the currently executing task and start the execution of the new highest priority task. An event flag to be set may also be specified.</td>
</tr>
<tr>
<td>SET EVENT FLAG</td>
<td>Instruct the executive to set an indicated event flag and return the previous polarity of the indicated flag (without a declaration of a significant event).</td>
</tr>
<tr>
<td>CLEAR EVENT FLAG</td>
<td>Instruct the executive to clear an indicated event flag and return the previous polarity of the indicated flag.</td>
</tr>
<tr>
<td>READ ALL FLAGS</td>
<td>Instruct the executive to return the polarities of all event flags.</td>
</tr>
<tr>
<td>WAIT FOR SINGLE EVENT FLAG</td>
<td>Instruct the executive to suspend the execution of the issuing task until an event flag of an indicated set of flags is set.</td>
</tr>
<tr>
<td>WAIT FOR LOGICAL OR OF FLAGS</td>
<td>Suspend execution of the issuing task until any indicated event flag in one of five groups is set.</td>
</tr>
<tr>
<td>WAIT FOR ANY SIGNIFICANT EVENT</td>
<td>Instruct the executive to suspend the execution of the issuing task until the next significant event.</td>
</tr>
<tr>
<td>EXIT IF</td>
<td>Instruct the executive to cause the issuing task to exit, if, and only if, an indicated event flag is clear, i.e., to continue execution if the flag is set.</td>
</tr>
</tbody>
</table>

### Trap-associated Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARK TIME</td>
<td>Instruct the executive to set an indicated event flag after a specified period of time, and/or cause an AST (Asynchronous System Trap) after the specified time has elapsed.</td>
</tr>
<tr>
<td>CANCEL MARK TIME REQUESTS</td>
<td>Instruct the executive to cancel MARK TIME requests that have been made by the issuing task.</td>
</tr>
<tr>
<td>DISABLE AST RECOGNITION</td>
<td>Instruct the executive to inhibit AST (Asynchronous System Trap) recognition for the issuing task; the AST’s are queued, and only their recognition is inhibited.</td>
</tr>
<tr>
<td>ENABLE AST RECOGNITION</td>
<td>Instruct the executive to enable AST recognition for the issuing task.</td>
</tr>
<tr>
<td>SPECIFY POWER RECOVERY AST</td>
<td>Tell the system whether or not power recovery AST’s are desired for the issuing task. If desired, this directive indicates where control is to be transferred when the AST occurs.</td>
</tr>
</tbody>
</table>
SPECIFY FPP EXECUTION AST
Instruct the executive that the issuing task contains an AST service routine whose execution is to interrupt the task's execution whenever a floating point processor exception (error) occurs.

SPECIFY RECEIVE AST
Instruct the executive that the issuing task contains an AST service routine whose execution is to interrupt the task's execution whenever data is sent to the issuing task via the SEND directive.

AST EXIT
Instruct the executive that the issuing task is exiting from an AST service routine.

SPECIFY SST VECTORS
Instruct the executive that the issuing task contains a table of addresses of service routines to be executed upon task trap or fault conditions (viz., memory protect violation).

SPECIFY SST VECTOR TABLE FOR DEBUGGING AID
Specifies the virtual address of a table of synchronous system trap service routine entry points for use by ODT or other debugging aids.

I/O Related Directives

QUEUE I/O
Instruct the executive to queue an I/O request for the issuing task. This request is queued by priority for a logical unit which is assigned to a physical unit via various mechanisms. An event flag, an AST, and an in-task status block may be specified as an I/O completion indication.

ASSIGN LUN
Instruct the executive to assign a logical unit number (LUN) of the issuing task to an indicated physical unit.

SEND DATA
Instruct the executive to send a block of data to an indicated task.

RECEIVE DATA
Instruct the executive that the issuing task wishes to receive data that has been sent from another task via the SEND directive. In RSX-11D systems, a specific sender can be identified for restrictive receives.

RECEIVE DATA OR EXIT
Instruct the executive to attempt to receive data for the issuing task. If no data is received, the issuing task is allowed to exit.

GET MCR COMMAND
Instruct the executive to transfer an MCR (console) command line to the issuing task.

RSX-11D systems provide an additional set of directives, as listed below.

Task Execution Control Directives

EXECUTE
Instruct the executive to request immediate execution of an indicated task if, and only if, memory is available. Memory partition, priority, and UIC may be specified.
SCHEDULE
Instruct the executive to schedule an indicated task's execution at a time specified in terms of absolute time of day. Periodic rescheduling, memory partition, priority, and UIC can be specified.

SYNCHRONIZE
Instruct the executive to schedule an indicated task's execution at a time specified in terms of delta time from clock unit synchronization. Periodic rescheduling, memory partition, priority, and UIC can be specified.

RESUME
Instruct the executive to resume the execution of an indicated suspended task.

Task Status Control Directives

FIX
Instruct the executive to load an indicated task into memory, ready for execution, with guaranteed memory and no disk load latency.

UNFIX
Instruct the executive to unfix a fixed task, thus freeing previously dedicated memory.

DISABLE
Instruct the executive to cause further requests for execution of an indicated task to be rejected. Any current execution of the indicated task is not affected.

ENABLE
Instruct the executive to make a disabled task once again runnable.

ALTER PRIORITY
Instruct the executive to alter the priority of an indicated task.

Informational Directives

GET COMMON PARAMETERS
Instruct the executive to return parameters related to an indicated common memory area. (RSX-11M and RSX-11S return the same parameters on a GET PARTITIONS PARAMETERS directive.)

Event-Associated Directives

READ EVENT FLAG
Reads a specified event flag and indicates by the return code in the directive status word whether the flag is set or cleared.

I/O Related Directives

QUEUE I/O AND WAIT
Instruct the executive to queue an I/O request and suspend the execution of the issuing task until an indicated event flag is set (combination of QUEUE I/O and WAIT FOR SINGLE directives).

SEND DATA AND RESUME OR REQUEST RECEIVER
Instruct the executive to send data to an indicated task and trigger that task to issue a RECEIVE directive by: (1) requesting the task, if exited, or (2) resuming the task, if suspended.
5.6 FILE CONTROL SERVICES
RSX-11 file control services (FCS) enable the user to perform record-oriented and block-oriented I/O operations and to perform additional functions required for file control, such as open, close, wait, and delete operations. To invoke FCS functions, the user issues macro calls to specify desired file control operations. The FCS macros are called at assembly-time to generate code for specified functions and operations. The macro calls provide the system-level file control primitives with the necessary parameters to perform the file access operations requested by the user. Figure 5-5 illustrates the file access operation.

![Diagram](image)

Figure 5-5  File Access Operation

FCS is a set of routines that is linked with the user program at task-build time from a resident system library or a system object module library. These routines, consisting of pure, position-independent code, provide a user interface to the file system, enabling the user to read and write files on file-structured devices and to process files in terms of logical records.

Logical records are regarded by the user program as data units that are structured in accordance with application requirements, rather than existing merely as physical blocks of data on a particular storage medium.

FCS provides the capability to write a collection of data (consisting of distinct logical records) to a file in a way that enables the data to be retrieved at will. Data can be retrieved from the file without having to know the exact form in which it was written to the file.

FCS thus provides a sense of transparency to the user so that records can be read or written in logical units that are consistent with an applications requirement.

5.6.1 File Access Methods
RSX-11 supports both sequential and direct access to files. The sequential access method is device-independent, that is, it can be used for both record-oriented and file-structured devices (for example, card reader and
disk, respectively). The direct access method can be used only for file-structured devices.

5.6.2 Data Formats for File-structured Devices
Data is transferred between peripheral devices and memory in blocks. A data file consists of virtual blocks, each of which may contain one or more logical records. In FCS, a virtual block in a file consists of 512 bytes.

Records in a virtual block can be either fixed or variable in length. The first two bytes of a variable-length record contain a value defining the length of that record (in bytes), excluding the record length bytes.

Virtual blocks and logical records within a file are numbered sequentially, starting with one. A virtual block number is a file relative value, while a physical block number is a volume relative value. For example, the first virtual block in a file is always virtual block number 1, but at the same time it could also be physical block number 156.

5.6.3 Block I/O Operations
The READ and WRITE macro calls allow the user to read and write virtual blocks of data from and to a file without regard to logical records in a file. Block I/O operations provide a very efficient means of processing file data, since such operations do not involve the blocking and deblocking of records within the file. Also, in block I/O operations, the user can read or write files in an asynchronous manner; control can be returned to the user program before the requested I/O operation is completed.

When block I/O is used, the number of the virtual block to be processed is specified as a parameter in the appropriate READ or WRITE macro call. The virtual block so specified is processed directly in a buffer reserved by the program in its own memory space.

As implied above, the user is responsible for synchronizing all block I/O operations. Such asynchronous operations can be coordinated through an event flag specified in the READ or WRITE call. The event flag is used by the system to signal the completion of the I/O transfer, enabling the user to coordinate those block I/O operations which are dependent on each other.

5.6.4 Record I/O Operations
The GET and PUT macro calls are provided for processing record-oriented files. GET and PUT operations perform the necessary blocking and deblocking of the records within the virtual blocks of the file, allowing the user to read or write individual records.

In preparing for record I/O operations, the user program must specify the format of the records. For example, it must specify whether the records are fixed or variable in length, or whether records that are to be output to a carriage-control device are to contain carriage-control information, which can be either at the beginning of the record or imbedded within the record.

For sequential access files, I/O operations can be performed for both
fixed- and variable-length records. For direct access files, I/O operations can be performed only for fixed-length records.

In contrast to block I/O operations, all record I/O operations are synchronous; control is returned to the user program only after the requested I/O operation is performed.

Because GET and PUT operations process logical records within a virtual block, only a limited number of GET or PUT operations result in an actual I/O transfer, that is, when the end of a data block is encountered. Therefore, all GET and PUT I/O requests will not necessarily involve a physical transfer of data.

The File Storage Region
The file storage region (FSR) is an area allocated in the user program as the working storage area for record I/O operations. The FSR consists of two program sections which are always contiguous to each other. The first program section of the FSR contains the block buffers and the block buffer headers for record I/O processing. The user determines the size of this area at assembly-time. The number of block buffers and associated headers is based on the number of files that the user intends to open simultaneously for record I/O operations.

The second program section of the FSR contains impure data that is used and maintained by FCS in performing record I/O operations. Portions of this area are initialized at task-build time, and other portions are maintained by FCS. This program section is intentionally isolated from the user to preserve its integrity.

The size of the FSR can be changed, if desired, at task-build time.

The data flow during record I/O operations is depicted in Figure 5-6. Note that blocks of data are transferred directly between the FSR block buffer and the device containing the desired file. The blocking and de-blocking of records during input is accomplished in the FSR block buffer, and the building of records is likewise accomplished in the FSR block buffer during output. Note also that FCS serves as the user interface to the FSR block buffer pool. All record I/O operations initiated through GET and PUT calls are totally synchronized by FCS.

Data Transfer Modes
When record I/O is used, a program can gain access to a record in either of two ways after the virtual block has been transferred into the FSR from a file:

MOVE MODE Individual records are moved from the FSR buffer (as shown in Figure 5-6). Move mode simulates the reading of a record directly into a user record buffer, thereby making the blocking and de-blocking of records transparent to the user.

LOCATE MODE The user program accesses records directly in the FSR block buffer. Program overhead is reduced in locate mode, since records can be processed directly within the FSR block buffer.
Multiple Buffering for Record I/O (RSX-11D only)
By supporting multiple buffers for record I/O, FCS provides the capability for RSX-11D users to read data into buffers in anticipation of user program requirements and to write the contents of buffers while the user program is building records for output. The user can thus overlap the internal processing of data with file I/O operations.

5.6.5 Shared Access to Files
FCS permits shared access to files according to established conventions. Two macro calls, among several available in FCS for opening files, can be issued to invoke these functions. The OPNS macro call is used specifically to open a file for shared access. The OPEN call, on the other hand, invokes generalized open functions which have shared access implications only in relation to other I/O requests then issued.

OPNS allows several active read-access requests and one write-access request for the same file. OPEN allows multiple read-access requests for the same file, but does not permit concurrent write access. Note that shared access during reading does not necessarily imply the presence of read requests from several separate tasks. The same task can open the same file using different logical unit numbers.

5.6.6 Spooling Operations
FCS provides facilities at both the macro and subroutine level to queue files for subsequent printing. A task issues the PRINT macro call to queue a file for printing on a specified device. The device must be a unit-record, carriage-controlled device such as a line printer or terminal.

In RSX-11D systems, the user can control the priority of the spooled file, and specify a forms type and copy count.
5.6.7 FCS Macros and Macro Usage

FCS includes four basic kinds of macro that simplify the user’s interface to the system’s file control primitives. The four kinds are:

- initialization macros
- file-process macros
- command line processing macros
- the CALL macro

The initialization and file-processing macros are used to establish the database description and the necessary temporary storage areas needed to perform I/O operations. The command line processing macros are used to dynamically process I/O commands entered from a terminal. The CALL macro is used to invoke file control routines.

The initialization and file-processing macros set up the following structures to define the database:

- A file data block (FDB) that contains execution-time information necessary for file processing. It defines the basic characteristics of a file, i.e., record type, record size, access privileges, etc.
- A dataset descriptor that is accessed by FCS to obtain the file’s name, type, version number and location, which are necessary to open a specified file. The dataset descriptor is used when a program accesses a given set of known or pre-defined files.
- A default filename block that is accessed by FCS to obtain default file information required to open a file. This is accessed when complete file information is not specified in the dataset descriptor. It is used by programs written to access a general set of files.

There are two types of initialization macros: assembly-time macros and run-time macros. Data supplied during assembly of the source program establishes the initial values in the FDB. Data supplied at run-time can either initialize additional portions of the FDB or change values established at assembly-time. Furthermore, the data supplied through the file-processing macros can either initialize portions of the FDB or change previously-initialized values. The user not only has a broad range of control over defining the data base characteristics, but also has control over when the definitions are made.

File-processing macros also determine the way in which files are processed. These macro calls are invoked and expanded at assembly-time. The resulting code is then executed at run-time to perform the following operations:

- OPEN: Opens and prepares a file for processing.
- OPNS: Opens and prepares a file for processing; allows shared access to the file (depending on the mode of access).
- OPNT: Creates and opens a temporary file for processing.
- OFID: Opens an existing file using the file identification provided in the filename block.
- CLOSE: Terminates file processing in an orderly manner.
- GET: Reads logical records from a file.
GETR  Reads fixed-length records from a file in random-access mode.
GETS  Reads records from a file in sequential access mode.
PUT   Writes logical records to a file.
PUTR  Writes fixed-length records to a file in random mode.
PUTS  Writes records to a file in sequential mode.
READ  Reads virtual blocks from a file.
WRITE Writes virtual blocks to a file.
DELET Removes a named file from the associated volume directory and deallocates the space occupied by the file.
WAIT  Suspends program execution until a requested block I/O operation is performed.
PRINT Queues a file for printing on a specified terminal or line printer.

In summary, the file processing macros allow the user to specify random-access or sequential access to files, and to perform block-oriented or record-oriented file processing. In addition, the PRINT macro allows the user to spool files to a line printer or terminal device.

The command line processing macros allow the user to access special routines available in the system object library. The Get Command Line (GCML) routine accomplishes all the logical functions associated with the entry of a command line from a terminal, an indirect command file, or an on-line storage medium. The Command String Interpreter (CSI) routine takes command lines from the GCML input buffer and parses them into the appropriate dataset descriptors required by FCS for opening files.

The CALL macro allows the user to access a special set of file control routines. These routines allow a MACRO program to perform the following operations: find, insert, or delete a directory entry, rename a file, extend a file, mark a temporary file for deletion, and delete a file, among other operations.

5.7 SYSTEM UTILITY PROGRAMS
The RSX-11D and RSX-11M systems provide a wide variety of system utility programs. RSX-11S provides two utility programs, the On-line Task Loader and the System Image Preservation Program as discussed in section 5.3.3. This section describes the RSX-11D and RSX-11M system utilities.

There are two sets of system utilities: those primarily used for program development and debugging, and those used for general-purpose file manipulation. The common set of program development utilities are:

EDI Editor
EDI is used to enter source programs or data files into the system and to modify them as needed. A large set of easy-to-use commands makes the Editor an effective program development tool.
SLP Source Input Program
SLP is an editing program used to create and maintain source language files on disk.

TKB Task Builder
The task builder creates loadable memory images from assembled or compiled tasks. It links relocatable object modules and resolves any references to global symbols, common areas, and shared libraries. The task builder is used to specify a task’s attributes, such as checkpointability, priority, etc. The task builder is also used to create Shareable Global Areas. The task builder provides an overlay descriptor language to construct task overlays. The overlay descriptor language simplifies the process of dividing a task into overlaid segments and specifying load methods. Finally, if requested by the task build command, the user can obtain a cross reference of all global symbols defined or referenced in the task or Shareable Global Area.

LBR Librarian
LBR provides the user with the capability to create and maintain disk-resident libraries of object modules and user-defined macros.

ODT On-line Debugger
ODT aids the user in debugging programs that have been assembled or compiled and task built. From the keyboard, the user interacts with ODT to:

• print or change the contents of a location in the task
• run the program using the breakpoint features to halt the program at specified points
• search the program for a specific bit pattern
• calculate offsets for relative addresses

A Trace capability is also provided to aid in the debugging of FORTRAN programs.

ZAP Task Patch
ZAP provides a facility for examining and modifying task image files and data files. With ZAP, permanent patches can be made to task image or data files without having to re-create the file.

RSX-11D provides two additional program development and debugging aids.

SQZ File Compressor
SQZ removes unnecessary spaces and tabs from MACRO source program files. Additionally, the SQZ program removes comment lines. The resultant file requires less storage space. (The RSX-11M Librarian can perform this function on request.)

CDA Core Dump Analyzer
CDA analyzes the state of the system when a system crash occurs. A module that can be linked to the Executive produces the output that CDA analyses. It provides a comprehensive listing of all system tables, lists and queues, CPU registers, and node pool at the time of the crash. The contents of particular real memory areas and of particular task
memory images can also be listed. CDA also performs validations of many of the elements of the system data base. The dump can be taken on and analysed from an RK05 disk, TU10 magnetic tape, TU16 magnetic tape or TU56 DECTape device.

RSX-11M also includes additional program development and debugging aids:

**XDT Executive Debugging Tool**
XDT, a subset of ODT, is an interactive debugging tool for Executive modules, I/O drivers and interrupt service routines.

**PMD Post-mortem Dump**
PMD is a debugging tool that can generate a memory dump for a task that terminates abnormally (called a post-mortem dump) or an edited memory dump for a running task (called a snapshot dump). Snapshot dumps can be requested any number of times during task execution. In general, both post-mortem and snapshot dumps provide the following information: contents of the CPU registers, including the stack pointer, program counter, processor status word, and floating-point registers (if the program used the latter), the task status code, event flags set, overlays that were in memory at the time, outstanding I/O requests, LUN assignments, etc.

The common set of general-purpose file manipulation programs include:

**PIP Peripheral Interchange**
PIP is used to copy files from one device to another, for example, from disk to printer, to rename files, to list files, and to delete files.

**FLX File Exchange**
FLX is a special-purpose file copy utility. It can copy a file in Files-11 format and convert it to either RT-11 format or DOS/BATCH format on another device, or copy a file in either RT-11 or DOS/BATCH format and convert it to Files-11 format. It can also copy files in Files-11, RT-11 or DOS/BATCH format to another device in the same format. Valid RT-11 devices for copy operations are RK11 disk, cassette and DECTape. Valid DOS/BATCH devices for copy operations are RK11 disk, DECTape, magnetic tape and cassette. RSX-11M’s also supports RT-11 format on floppy disks and both RT-11 and DOS/BATCH formats on paper tapes. (Note that RSTS/E can use the DOS/BATCH format for DECTapes. It is therefore possible to use FLX to transfer files to and from RSTS/E systems as well.) Furthermore, FLX can initialize a cassette in RT-11 or DOS/BATCH format, list RT-11 or DOS/BATCH cassette directories, and delete files on RT-11 or DOS/BATCH volumes.

**PRE Media Preservation**
PRE creates copies of volumes. PRE copies volumes to or from the following devices: disks, magnetic tapes, DECTapes, or cassette tapes. Under RSX-11D, PRE can run on-line. PRESERV is a stand-alone version available with RSX-11M and RSX-11S.

**DMP File Dump**
DMP enables the user to obtain a listing of Files-11 files or volumes in either ASCII or octal format.

5-49
VFY File Verification
VFY checks the consistency and accuracy of the Files-11 file structure on a Files-11 device, for example, a disk. It also prints the number of available blocks in a volume, locates files that could not otherwise be accessed, and lists the names of files on the volume.

5.8 LANGUAGES
RSX-11 programs can be written in MACRO, FORTRAN IV, or FORTRAN IV-PLUS. In addition, RSX-11D supports a fourth source language, COBOL. Programs written in MACRO or FORTRAN are translated into object code by the MACRO assembler, FORTRAN IV compiler or FORTRAN IV-PLUS compiler, and then are task built into executable images by the TKB Task Builder. Programs written in COBOL can be executed immediately after translation by the COBOL compiler, because they produce intermediate code which is run by an interpreter.

5.8.1 MACRO
RSX-11 MACRO assembler allows the user to take full advantage of the PDP-11 instruction set, the RSX-11 system directives and File Control Services. During assembly, MACRO detects errors and produces output indicating the types of errors found, thus simplifying the process of locating program errors.

In addition to supporting the same MACRO assembler provided in RSX-11D, RSX-11M also supports a subset MACRO assembler. The full MACRO assembler requires a 14K word partition to run. The subset MACRO assembler requires only an 8K word partition.

5.8.2 FORTRAN IV
RSX-11 FORTRAN is an ANSI standard FORTRAN IV language accompanied by an object time system (OTS). FORTRAN programs can be compiled into object code which can be linked, optionally with other object programs (either FORTRAN or MACRO), into executable task images. The OTS is an object library of commonly used FORTRAN routines, including math, error handling, process I/O, laboratory peripheral routines, and Executive directives.

5.8.3 FORTRAN IV-PLUS
FORTRAN IV-PLUS is an option available on both RSX-11D and RSX-11M systems. FORTRAN IV-PLUS programs can also be written to execute on RSX-11S systems. The FORTRAN IV-PLUS compiler and OTS make use of the Floating Point Processor and produce highly optimized code. The FORTRAN IV-PLUS language is upward compatible with FORTRAN IV.

5.8.4 COBOL
RSX-11D systems support the PDP-11 COBOL compiler and object time system as an option. The PDP-11 COBOL compiler translates ANS-74 standard COBOL source programs into an object form that guides the execution of the COBOL object time system. The compiler runs under the supervision of the RSX-11D operating system, and requires a 23K word partition to run.

Several utility programs are provided with COBOL, including a report-generating program, the PDP-11 SORT utility, and a reformatting program. The SORT utility is also generally available for use on RSX-11M systems as an option.
CHAPTER 6

INTERACTIVE APPLICATION SYSTEM
IAS

6.1 OPERATING SYSTEM FUNCTIONS AND FEATURES
IAS is a large, general-purpose operating system that runs on a PDP-11/70 or PDP-11/45 Processor. It is a multi-user timesharing system that supports concurrent interactive, batch and real-time applications. It includes the MACRO assembler. As options, FORTRAN IV, FORTRAN IV-PLUS, COBOL and BASIC language processors can be added. IAS features:

- multiple programming languages: FORTRAN, BASIC, COBOL and MACRO
- a single, easy to learn and use interactive command language
- priority scheduling for real-time tasks
- submission of batch jobs from interactive terminals
- timesharing services for development of interactive applications programs
- a simple internal software interface for the development and use of special-purpose, multi-user interactive applications
- a sophisticated file system providing device independence, file protection, sequential and random file access
- dynamic allocation of system resources
- use of shared, reentrant code to minimize memory requirements
- system management facilities for system configuration, generation and control
- facilities to account for and restrict the use of system resources

IAS requires, in addition to the central processor, a system operator's console terminal, at least one user terminal, 64K words of parity memory, and a disk system with a TM11 or TU16 magnetic tape system. The disk system can be an RP03 or RP04 disk system. Memory can be expanded up to 124K words on a PDP-11/45 or up to 1024K words on a PDP-11/70. IAS can support up to 10 interactive timesharing terminals on a PDP-11/45 or up to 20 interactive timesharing terminals on a PDP-11/70 performing BASIC programming and editing functions. In addition, IAS supports a variety of peripherals useful in batch and real-time applications, including line printers, card readers and laboratory peripherals.

As an interactive timesharing system, IAS presents an easy-to-use system interface. The Program Development System, PDS, provides a
computing environment that supports most application processing requirements of IAS users. As such, it presents to IAS terminal users a standard interface which requests and processes valid passwords and user names before making system facilities available to the user. The interface allows the user to create programs, submit jobs to the batch stream and issue commands to create and manipulate program and data files.

As a batch system, IAS services a single-stream queue of batch jobs. FORTRAN, MACRO and COBOL jobs can be submitted to the batch stream. The user interface for batch processing is the same as the PDS interactive interface. Therefore, programs can be developed in interactive mode and run in production in batch mode. The system manager can control the amount of service that batch jobs receive from the processor. In particular, the system manager can guarantee a minimum processor time for batch processing.

As a generalized, flexible base for executing interactive applications, IAS provides support for application-specific user interfaces for applications such as data entry, bank teller terminals or engineering computation, where it is necessary or desirable to present a customized interface to terminal users (operators, for example).

Further, IAS supports the concurrent execution of multiple interactive applications. Thus, a data processing application and the program development system can execute concurrently and be serviced jointly by the timesharing facilities of the system.

The interactive application facility is further enhanced by the capability of the FORTRAN IV-PLUS compiler and IAS to develop and support shareable programs. For the user, this means that system overheads (memory occupancy and swapping time) are minimized. Also, the user can allocate specific application interfaces, and deallocate them, as required. This facility is flexible and extendable. The system is easily modified and additional applications are easily added.

The special-purpose interfaces can be written and checked out using the IAS Program Development System and then installed by the system manager for use on specific terminals. IAS provides a number of system services, which can be called from the application program, to enhance the function of these special purpose interfaces.

IAS is built on the RSX-11D operating system. It provides, therefore, the RSX-11D real-time processing facilities of multiprogramming, priority scheduling, power-fail restart, contingency exits, disk-based operation and task checkpointing of real-time tasks. Real-time, interactive, and batch operations can occur concurrently and, normally, in that priority order.

IAS system operations are managed by two executives. The real-time executive schedules real-time activities according to their priorities and manages the system resources not allocated to the timesharing activities. The timesharing executive schedules timesharing users on the basis of a time-slicing algorithm when real-time activities do not take precedence. Batch processing normally uses processor time available after interactive
users are serviced. Both batch tasks and interactive tasks run under control of the timesharing scheduler.

Table 6-1 provides a summary of the IAS system’s features.

<table>
<thead>
<tr>
<th>Table 6-1 IAS System Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System type</strong></td>
</tr>
<tr>
<td><strong>CPU’s supported</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Memory size range</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Additional CPU hardware</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Minimum peripherals</strong></td>
</tr>
<tr>
<td><strong>Additional peripherals</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Interactive terminals</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Batch processing</strong></td>
</tr>
<tr>
<td><strong>Languages</strong></td>
</tr>
<tr>
<td><strong>System tasks and special utilities</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>File system</strong></td>
</tr>
</tbody>
</table>

6.2 IAS EXECUTIVE ORGANIZATION AND SERVICES

To provide system flexibility, the IAS operating system is controlled by a system monitor consisting of a real-time executive kernel and timesharing executive. The primary functions of the kernel include memory and disk management, supervision of privileged tasks (including real-time tasks and device handlers), file management, and maintenance of the
general integrity of the system. The kernel maintains the Active Task List (ATL) to control task dispatching.

The timesharing executive controls both interactive and batch processing. It controls the execution of timesharing tasks by time slicing and by swapping tasks in and out of memory.

6.2.1 The Active Task List
The kernel coordinates the dispatching of all tasks on the system by scanning the entries in the Active Task List (ATL). The ATL is a priority-ordered list of all resident active tasks in the system. Because of their requirement for immediate service, the I/O device handler tasks are put at the top of the ATL. For the same reason, any user-designated real-time tasks are assigned to high-priority levels. The timesharing executive, which runs at a lower priority than I/O and real-time tasks, controls the scheduling of user timesharing tasks by inserting tasks in the ATL. Figure 6-1 illustrates the priority structure of the ATL.

The timesharing scheduler uses two tasks, TSS1 and TSSNUL, to control the dispatching of tasks. The timesharing scheduler task TSS1 selects a task for executing by placing its entry in the ATL at a priority of one less than itself. The scheduler task then gives up control (e.g., waits for an event flag such as time slice complete) to allow the kernel to dispatch the user task. TSSNUL is the null job and runs continuously in a loop so that tasks below it on the ATL can never execute. When a timesharing task is not executing, TSS1 places the ATL entry for that task below that of TSSNUL. TSSNUL always executes at priority 1.

6.2.2 The Timesharing Scheduler
The prime objective of the scheduler is to reduce as far as possible the average response time to all user demands. In order to do so, the scheduler distinguishes between various levels of user importance and urgency of service. The scheduler maintains a number of round-robin queues, or levels, of tasks to be scheduled. The scheduler scans each level (high to low) in a round-robin fashion until it finds a memory-resident runnable task. A non-resident ready-to-run task will cause the swapping system to be activated.

A task which uses a full time quantum is transferred to the next lower level unless it is already at the lowest level. Tasks at lower levels are not scheduled as often as tasks at a higher level, so a time factor is associated with each level. Tasks allocated to a lower priority level are given a longer time quantum when next activated. Thus, large jobs are run and swapped less frequently, but in compensation, receive more processor time once activated.

To prevent tasks from being starved of processor time because the scheduler is continuously scheduling higher priority tasks, a means of promoting tasks from one level to the level above is provided. If, over a given period of time, no scheduling has been performed at a given level, then a task at that level is moved to the bottom of the level above.

If the scheduler finds a runnable task which is not resident, then the task must be loaded into memory to receive its quantum of CPU time. Space will be created in memory by moving resident tasks to create the
required contiguous space, and, if necessary, by writing inactive tasks to the swap area on disk(s).

Two time factors are associated with every task. The quantum determines the amount of CPU time a job may have before it is swapped out of main memory. The time slice is the maximum CPU time a task is allowed to use before a rescheduling operation is performed.

The time quantum for a particular task is determined by:

\[ Q = At + c \]

where

A is a factor (in clock ticks) assigned to a task when it is loaded: for example, 1 tick per 1K words in the task.
t is a time factor associated with the scheduling level; t increases as the level number increases.

c is the minimum guaranteed quantum for the system.

The quantum for a task at a low scheduling level may be quite large. In order not to block other higher priority tasks awaiting service, the scheduler calculates the quantum of the task, and then allocates the task a number of time slices. At the end of each time slice, the scheduler will try to run higher priority tasks. However, the task will not be swapped until its quantum has expired. If the task enters a wait state, however, the quantum will be set to zero. In this case, therefore, it will be made available for swapping.

The time slice parameter can be adjusted to achieve the desired compromise between responsiveness and system throughput. If the time slice is set to its maximum value, all tasks will execute without interruption for their entire quantum. The time slice should never be smaller than the maximum quantum for a level one task. All the parameters of the scheduling algorithm can be adjusted by the system manager to tailor IAS scheduling to the needs of the local installation.

6.2.3 Batch Processing

Batch runs as if it were another timesharing terminal. The batch command language is the same as the general-purpose interactive program development command language, and it is processed by the same command language interpreter (see section 6.3).

The batch processor obtains its command input from a queue of commands. The batch queue is maintained independently, thus enabling jobs to be submitted to the queue at any time. The processor can service two types of queues. The system can maintain a spooled queue which consists of: 1) batch job files submitted from interactive terminals, and 2) command input from the card reader (if the card reader is designated as a spooled device). The batch processor can also service a queue of commands directly from the card reader if it is designated as an unspooled device.

Batch processing is initiated and terminated by the system manager. The batch processor executes at the batch scheduling level where it is serviced by the timesharing scheduler. Batch processing shares CPU time with interactive tasks, but its priority for service is always below that of the active tasks.

To assure that batch processing receives adequate service, the system manager can specify the percentage of CPU time to be made available to it, and the length of time (quantum) batch should run when it does receive service. For example, the system manager could direct IAS to devote ten percent of the available time to batch jobs in 2-second quanta. Tasks in the batch level are not subject to the promotion/demotion mechanism of the timesharing scheduler; that is, tasks remain in the batch level for as long as they are executing.

Batch user tasks share space with the interactive tasks (if any) currently executing. While any space not currently in use by batch is used for interactive processing, batch can be guaranteed space so that re-
requirements up to that maximum will always be satisfied by swapping interactive tasks out of memory if necessary.

6.2.4 Executive Data Structures
The IAS system maintains a number of common areas in which the various executive tasks store information and communicate with each other. SCOM contains system tables, the kernel node pool, lists, and some servicing routines. SYSRES, the System Resident Library, contains common routines which will be used by most tasks. IASCOM is a library containing timesharing nodes, lists, tables, and common routines for manipulating the timesharing data structures. IASBUF is a buffer area used for communication between the Timesharing Control Primitives, IASCOM, and the timesharing executive.

6.2.5 I/O Services and Device Independence
Input and output comprise a significant part of all programmed activity. Thus, IAS provides a variety of services to perform these operations.

The IAS File System is a collection of system services that permits the user to view I/O as a transaction between a program and a named, protected collection of records known as a file. The File System manages all data transfers and provides the mechanism whereby a file intended for a record-oriented device, such as a line printer, can be dynamically directed to an area on magnetic storage.

Access to a user's files stored on a disk, DECTape or labelled magnetic tape is controlled by a protection specification on each file. When creating a file, a user can specify whether other users may have access to the file and, if so, whether they may modify the file or merely read it.

One of the goals of any file system is to make the user program independent of the I/O hardware. Thus, while the storage characteristics of a medium are organized around physical records, the user deals only with logical records.

To provide greater device independence, the IAS user will, in general, use logical units instead of referring directly to physical devices. IAS provides a set of logical unit numbers (LUN's) which are not associated with specific physical devices or files until run time. In the source program, all device and file references use LUN's. These LUN's may be assigned to particular devices by a command issued before the program is executed.

6.2.6 Sharing of Common Routines
In a system designed to support many users, there is a high probability that many tasks will use the same code sequences, such as mathematical routines and specialized I/O routines.

The common code could be built directly into each task requiring it, but this might result in several copies of the same code occupying memory space at the same time. The alternatives employed by IAS is to put the common code where all users can share it, so that only one copy of the code is required. The IAS system uses shared code heavily.

Under IAS, shared areas may be data areas (global common), sets of common routines (libraries), or the pure (read-only) areas of complete
tasks (shared tasks). Global common areas allow simultaneously-active
tasks to share data. A shareable library consists of routines which may
be interrupted to service another request, then resume execution later
at the point of interruption. Users who write reentrant routines can in-
clude their own shareable libraries in the IAS system. Shared code does
not need to be permanently resident; it can be loaded at the time a task
which uses it is run. Programs written in either FORTRAN IV-PLUS or
MACRO can be shared.

6.2.7 System Generation and Initialization
System generation is the process by which a collection of system ser-
vice is tailored to meet local physical constraints and performance
requirements.

IAS consists of a set of independent program segments which can be
linked selectively to eliminate services not required at a given installa-
tion. For example, a system manager might eliminate the device handlers
for devices not included in the hardware configuration.

During system generation, the system manager also defines and names
the partitions in which programs will execute. This normally includes de-
fining a timesharing partition and any special partitions dedicated to the
execution of real-time applications.

After generating the system, the system manager runs a special start-up
task to initialize the timesharing system. The start-up task prompts for
a series of parameters that specify the system configuration for the cur-
rent session. The parameters can be entered at the terminal, or read
from a predefined file. The values specified for the start-up parameters
override the defaults specified at system generation.

The start-up parameters include:

- Terminals to be allocated for timesharing use
- Devices to be made available for timesharing users
- Devices to be used for swapping
- System control parameters
- Partition names for timesharing
- Partition names for executive tasks

Once the start-up parameters are established, the system operator en-
ables general timesharing users to access the system by allocating time-
sharing terminals to the system’s command language interpreters.

6.3 COMMAND LANGUAGE INTERPRETERS
A command language interpreter (CLI) is a task which interfaces with a
person who uses the IAS system. PDS, the Program Development System
CLI supplied with IAS, allows the general user to access all non-privileged
facilities of the system. Another CLI called the System Control Interface
(SCI) allows the system operator to alter the state of the system, to
designate user interfaces (CLI’s), and to allocate facilities to each user.

Normally, PDS is the standard CLI to which a general terminal in the
IAS system is allocated. Using the SCI interface, the system operator can
designate a specific task other than PDS as the CLI for a terminal. For example, the system operator might set aside one terminal to be used solely for program editing. When EDIT is designated as the only CLI for that terminal, EDIT will be invoked when CTRL/C is typed, and a user at that terminal will not be able to issue commands to anything except the Editor.

Users can write their own CLI tasks. The CLI tasks can be installed and allocated timesharing terminals. This means that the system can present a number of different terminal interfaces. A user-written CLI task can define its own command language, which can be as simple and understandable as required. It can be specifically designed for a particular application operation. Application terminal users do not, therefore, have to learn a generalized command language such as PDS to perform their subset of daily activities.

A CLI is written as a normal, non-privileged user task which can use, in addition to the standard system directives and file system facilities, the IAS system's Timesharing Control Primitives (see section 6.4). A CLI can be written in any language which provides the facilities it requires; for example, a CLI that wishes to use the system QIO directive must be written in FORTRAN, MACRO or BASIC (with user-defined functions).

After a task has been installed as a CLI, IAS automatically provides certain task execution controlling functions. For example, when CTRL/C is typed on a terminal allocated to a CLI, a copy of the non-shared part of the CLI is activated. If the task specifically requests the information, IAS will inform the task of any events happening at its terminal or terminals.

The following two sections describe the two standard CLI tasks provided with the IAS system: PDS, the Program Development System, and SCI, the System Control Interface.

6.3.1 Program Development System (PDS)
A typical timesharing user interfaces with IAS through the Program Development System (PDS) command language interpreter. Under PDS, users can create, compile, link, load, and run programs. They can submit jobs to the batch stream, use various peripheral devices, and obtain system information.

PDS is a prompt-oriented system. After PDS is activated at a terminal, either by the autostart mechanism or by typing a CTRL/C, PDS invites the input of a command by issuing the prompt "PDS >". The user replies by typing a command name and its parameters, if any, followed by a carriage return. If a user does not supply all the parameters required in a command, the system will prompt the user for them. Additionally, the user can issue the HELP command to display the commands available.

As an example, the user can login to the system, using the LOGIN command, in two ways. If the user desires the prompts, the user can simply type the command LOGIN in response to the PDS prompt.

PDS > LOGIN The user issues the command.
USERID? SMITH The User ID is a 1- to 12-character username which identifies a person to the system. PDS requests a User ID.
PASSWORD? PDS requests a password. The password is not displayed.


PDS validates the user name and password and accepts the user to the system by printing information relevant to the user’s job.

If the user types a user name after issuing the LOGIN command, PDS does not prompt for a user name, it only prompts for the password.

PDS > LOGIN SMITH
PASSWORD?
PDS >

As another example, the user can issue a command to rename a file in any of three ways. If the user simply types the command name RENAME, PDS prompts for the old file specification and the new file specification parameters.

PDS > RENAME
OLD? MATRIX.FTN
NEW? BACKUP.TMP
PDS >

If the user does not want the prompts, the user can enter the entire command on one line.

PDS > RENAME MATRIX.FTN BACKUP.TMP
PDS >

If the user issues the command name followed by a carriage return, but does not need the second prompt, it is also acceptable to enter the command parameters on the line with the first prompt.

PDS > RENAME
OLD? MATRIX.FTN BACKUP.TMP.
PDS >

The user can supply PDS commands in a file rather than typing them in one at a time on the terminal. The user creates a file containing the commands PDS is to execute, called an indirect file. To execute the commands in the file, the user replies "@ filename" to a PDS prompt, where "filename" is the name of the indirect file. PDS processes the file in the same manner it processes commands typed individually on the console. The commands, as well as any error messages that occur during the execution of the commands, will be displayed on the user’s output device.

For example, suppose the user creates an indirect file named PERF.CMD containing the PDS commands to compile, link and run the source program PERF.FTN. By typing the command "PERF.CMD" in response to a PDS prompt, PDS will execute the command file.

PDS > @PERF.CMD

The user issues the indirect file command.
The first command requests the FORTRAN compiler to compile the source program named PERF.FTN (the .FTN extension is assumed by default) and produce an object program (named PERF.OBJ by default) and a listing file named PTEST.LST (the .LST extension is assumed by default).

22:34:17 TASK TERMINATION
CORE SIZE 20K   CPU TIME 01.05

PDS prints a message when compilation is complete.

LINK PERF

The second command requests the Linker to link the object program named PERF.OBJ (the .OBJ extension is assumed by default).

22:35:49 TASK TERMINATION
CORE SIZE 15K   CPU TIME 14.41

PDS prints a message when linking is complete.

RUN PERF

The third command requests PDS to execute the program PERF.TSK (the extension .TSK is assumed by default).

22:35:58
22:37:12 TASK TERMINATION
CORE SIZE 10K   CPU TIME 00.13

PDS prints messages regarding program execution.

PDS >

After processing the command file, PDS indicates that it is ready to accept another command.

There are several types of PDS command: commands that provide access or system information, commands that allocate resources, commands that manipulate files, and commands that control task execution. The system manager can designate certain PDS commands as privileged or non-privileged for any particular user. That is, when defining the user accounts, the system manager specifies which PDS commands each user can issue. For example, some PDS commands control real-time task execution. Only those users who have been given real-time execution privileges can issue the real-time execution control commands.

Except for the LOGIN, LOGOUT, JOB, EOD and EOJ commands, all non-privileged commands can be issued in either interactive or batch mode. When a command is issued in batch mode, it requires a dollar sign ($) preceding the first character of the command name.
Table 6-2 lists the general PDS commands. Commands that contain the term "real-time" in their description are available only to the users with real-time execution privileges. All other commands listed are non-privileged.

Table 6-2  
PDS Commands Summary

<table>
<thead>
<tr>
<th>SYSTEM INITIALIZATION COMMANDS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>Used to change system and device defaults for a particular user.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>Changes password.</td>
</tr>
<tr>
<td>LOGIN</td>
<td>Initiates an interactive session at a terminal.</td>
</tr>
<tr>
<td>LOGOUT</td>
<td>Ends an interactive session at a terminal.</td>
</tr>
<tr>
<td>JOB</td>
<td>Denotes start of a batch job.</td>
</tr>
<tr>
<td>EOD</td>
<td>Denotes end of data in a batch job.</td>
</tr>
<tr>
<td>EOJ</td>
<td>Denotes end of a batch job.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYSTEM INFORMATIONAL COMMANDS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HELP</td>
<td>Displays a list of all available commands.</td>
</tr>
<tr>
<td>SHOW</td>
<td>Displays system and device defaults and various types of system status information.</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Sends a message to the system operator.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JOB CONTROL COMMANDS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOCATE</td>
<td>Reserves a device for single user access.</td>
</tr>
<tr>
<td>DEALLOCATE</td>
<td>Releases a device.</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>Associates a user-specified logical name or a physical device with a logical unit number.</td>
</tr>
<tr>
<td>DEASSIGN</td>
<td>Disassociates a logical name or physical device with a logical unit number.</td>
</tr>
<tr>
<td>MOUNT</td>
<td>Requests mounting of a volume or volume set.</td>
</tr>
<tr>
<td>DISMOUNT</td>
<td>Requests operator to dismount a volume.</td>
</tr>
<tr>
<td>INSTALL</td>
<td>Installs a real-time task.</td>
</tr>
<tr>
<td>RUN</td>
<td>Initiate execution of a user program (used also for real-time task execution).</td>
</tr>
<tr>
<td>ABORT</td>
<td>Kills a suspended user program or command.</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Restarts a previously-suspended program or command.</td>
</tr>
<tr>
<td>QUEUE</td>
<td>Queues a file for printing or queues a batch job.</td>
</tr>
<tr>
<td>SUBMIT</td>
<td>Submits a job for batch execution.</td>
</tr>
<tr>
<td>ALTERPRIORITY</td>
<td>Changes priority of a real-time user task.</td>
</tr>
</tbody>
</table>
Table 6-2 (Cont.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNFIX</td>
<td>Allows a real-time task to be swapped.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>Cancels a periodic rescheduling of a real-time task.</td>
</tr>
<tr>
<td>REMOVE</td>
<td>Removes an installed real-time task.</td>
</tr>
</tbody>
</table>

**FILE MANIPULATION COMMANDS**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td>Appends one or more file(s) to another.</td>
</tr>
<tr>
<td>COPY</td>
<td>Copies one or more files.</td>
</tr>
<tr>
<td>CREATE</td>
<td>Creates a disk file from card or terminal input or creates a directory file.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes one or more files.</td>
</tr>
<tr>
<td>DIRECTORY</td>
<td>Displays the names of files in the indicated directory.</td>
</tr>
<tr>
<td>PRINT</td>
<td>Prints a file or files on the system’s printer.</td>
</tr>
<tr>
<td>RENAME</td>
<td>Changes the name of a file.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Types a file at the user’s terminal.</td>
</tr>
</tbody>
</table>

**PDS PROGRAM DEVELOPMENT COMMANDS**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT</td>
<td>Invokes the interactive editor or, optionally the line editor.</td>
</tr>
<tr>
<td>MACRO</td>
<td>Invokes the MACRO assembler.</td>
</tr>
<tr>
<td>LINK</td>
<td>Invokes the Linker to link together FORTRAN and/or MACRO object modules.</td>
</tr>
<tr>
<td>LIBRARIAN</td>
<td>Invokes the Librarian to create object program libraries.</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>Invokes either the FORTRAN IV or FORTRAN IV-PLUS compiler.</td>
</tr>
<tr>
<td>COBOL</td>
<td>Invokes the COBOL language processor.</td>
</tr>
<tr>
<td>BASIC</td>
<td>Invokes the BASIC subsystem.</td>
</tr>
</tbody>
</table>

In addition to the PDS commands, IAS supports special terminal control commands issued from the terminal. These control commands are:

CTRL/C  Return control to PDS (suspends a running program).
CTRL/U  Delete current line.
CTRL/I  Skip to next tab position.
CTRL/K  Vertical tab.
CTRL/L  Form feed.
CTRL/O  Enable/disable terminal output.
CTRL/R  Retype the current input line.
CTRL/S  Suspend current output until CTRL/Q is typed.
CTRL/Q  Resume current output.
CTRL/Z  Generates an end-of-file.

In addition to the general PDS commands, IAS includes special PDS commands available only to the system manager. The system manager
must be logged in under the system management account to gain access to these privileged PDS commands. There are three types of privileged PDS system management commands:

- accounting commands to authorize users and report system use
- real-time system control commands
- volume and file control commands

6.3.2 System Control Interface (SCI)
The system operator communicates with IAS through the System Control Interface (SCI) command language interpreter. The SCI command language uses the same syntax and conventions as the PDS command language, including prompting for missing parameters. Indirect SCI command files are also supported.

SCI commands enable the operator to monitor the system in four different areas:

- command language interpreter control
- overall system and task control
- peripheral device control
- system information

The Command Language Interpreter (CLI) commands allow the operator to install and remove CLI tasks, allocate and deallocate resources (e.g., terminals) to a CLI task, and abort a CLI task at a particular terminal. These commands are used both to initialize a timesharing system and to modify the system's characteristics during system operation.

The system and task control commands enable the operator to: load and unload device handlers which are not permanently resident; mount and dismount volumes; set the system parameters to suit the current workload; and shut down the system. These commands also enable the operator to have ultimate task execution control. For example, the operator can terminate any task in the system. This can be useful, for example, when a batch task, due to internal errors, loops indefinitely.

Peripheral device control commands provide the operator with the facility to service user requests for access to disk packs, magnetic tapes or other removable media. Additionally, the operator can control the output spooling mechanism and the type of printer forms being used.

The system information commands allow the system operator to display system information such as the active task list, CLI allocations, partition names and sizes, date and time, and device status.

6.4 TIMESHRARING CONTROL PRIMITIVES
IAS provides an installation with a convenient mechanism for implementing special-purpose interactive applications systems such as inventory control, order entry, on-line file update, etc. Programs written in either FORTRAN IV-PLUS or MACRO are ideally suited for this purpose since the program, if reentrant, can be shared by multiple users. The programmer writes a program as if it were responding to only one terminal, thus eliminating many of the problems associated with interactive multi-user
applications. If the application requires special system services or interlocks between users, it can use the system directives, the file system, and the IAS timesharing coordination facilities to perform these functions. The timesharing facilities are provided through a set of routines called Timesharing Control Primitives (TCP).

A task designated as a CLI obtains service from TCP by issuing calls to specify desired operations. TCP runs at a higher priority level than the timesharing scheduler to provide a high service level to CLI tasks and to ensure that up-to-date system information is always available to the timesharing executive. TCP presents a kernel handler interface to the system; the basic method is through a QIO (Queue I/O) directive.

There are eight kinds of TCP routines available to the user writing a CLI. For protection purposes, the system operator can control the privileges of a CLI task. When the operator installs the CLI task, the operator specifies which TCP facilities the CLI can use. The following sections describe the TCP routines.

6.4.1 CLI Control Primitives
These primitives provide the necessary CLI authorization and allocation to terminals required to establish the timesharing environment. This group provides the facilities for dynamically controlling the CLI population of the system and the allocation of terminals to those CLI's. The CLI control primitives are:

INITIALIZE A CLI
A task is initialized either at system start-up by the IAS executive or at any subsequent time by the system manager. The CLI must already be installed. It can be initialized as a batch subsystem (the default is interactive).

ALLOCATE TERMINALS TO A CLI
Allocates or starts up a CLI task for specified terminals or processing spooled input.

RELINQUISH TERMINALS FOR A CLI
Releases the terminals that the calling CLI is servicing.

6.4.2 Task Initiation and Control
Task initiation and control primitives enable a CLI to submit tasks to timesharing either on behalf of the terminal user or for the CLI's own purposes. They also provide the facilities for the CLI to subsequently control the execution of those tasks by suspending, continuing, or aborting the tasks. They are:

TASK STRING PARSE
 Parses a command string, identifying a task name. It also identifies whether or not a parameter string is present in the command.

SET UP A JOB NODE
Sets up the information in a job to enable successful scheduling of a task on submission of the job node to the scheduling queue.

QUEUE JOB NODE
Queues a specified job node to be run under IAS. The scheduling level at which the task is to initially run can be specified.
SET TASK TERMINATION
Enables a CLI to abort a task previously submitted for scheduling.

SET TASK CONTINUE
Enables a task to be continued after being suspended, either as a result of a suspend request or a CTRL/C issued from the terminal.

SET TASK EVENT FLAG
Allows a CLI to communicate with a task which it has previously initiated by setting a local event flag for that task.

6.4.3 Buffer Management
A CLI can claim and relinquish buffers by using this group of primitives. They are:

CLAIM IAS BUFFER
Allows a single buffer to be picked from the IAS buffer pool.

RELINQUISH IAS BUFFER
Releases a buffer by returning it to the buffer pool.

6.4.4 Terminal Event Control Primitives
The terminal event control primitives enable TCP to communicate to the CLI events that occur asynchronously with the current CLI activity. For example, TCP can notify the CLI of task termination for the CLI's terminal user, of a CLI exit requested, or that a CTRL/C was received on terminal. This mechanism allows the CLI to make decisions as to its subsequent actions. The primitives are:

DECLARE TERMINAL EVENT
Declares a terminal event for the CLI to service.

SERVICE TERMINAL EVENT
Invoked by the .CLI task to service asynchronous events occurring for its terminal user.

6.4.5 Job Node Management
Every task submitted for scheduling under IAS timesharing must have an associated job node. The job node management primitives enable a calling CLI task to control the allocation of job nodes to its user and therefore the user's ability to run tasks.

CLAIM JOB NODE
Enables a job node to be picked from the job node pool for the requestor. Normally the job node is claimed on behalf of a terminal belonging to the requesting CLI task, but a CLI can claim a job node for its own use.

ASSIGN A UIC
Assigns a user identification code to a job node. The assigned job node usually belongs to a terminal node but it could also be a floating node claimed by the CLI for some other purpose.

RELINQUISH A JOB NODE
Releases a job node currently allocated to the terminal serviced by the calling CLI or releases a specified job node previously claimed by the calling CLI.

6-16
6.4.6 System Management

The system management primitives enable the system manager to obtain information about the system and reset the tuning parameters. Access to the system tuning parameters (batch and interactive quanta, maximum number of interactive jobs, etc.) is only available to the system manager.

SET OR REPORT THE TIMESHARING TASK PROMOTIONS PERIOD
Reports and optionally gives new values to the time between the timesharing scheduler's promotion of tasks between scheduling levels. During timesharing tasks execution, the system allocates tasks among the scheduling levels according to their activity. A task that uses a full time quantum in a high level is transferred to the next lowest level, where the quantum size is greater. The goal is to move highly interactive tasks to high levels, while CPU-bound tasks move to low levels. To avoid having tasks in low levels becoming starved for CPU time, tasks are periodically promoted. If, during the promotion period, no task in a level has been scheduled, the task at the top of that level is promoted.

SET OR REPORT THE BATCH QUANTUM
Reports the current values of the batch time quantum, the time between batch schedules, and optionally gives new values to one or both of the parameters.

SET OR REPORT THE TIMESHARING QUANTUM
Reports the current value of and optionally changes the time quantum allocated to a specific scheduling level. The IAS scheduler relies on the quantum values increasing with level number.

SET OR REPORT THE TIMESHARING QUANTUM CONSTANTS
Reports and optionally gives new values to the constants used in calculating the time quantum given to each task in each scheduling level. The quantum given to a task is determined by the relationship:

\[ Q = At + C \]

C represents the minimum quantum given to any task and t represents the timesharing quantum allocated to a specific scheduling level. A, the allocation factor, is a function of task size, where \( A = N/M \). N is the number of clock ticks allocated for M number of 1000 byte blocks of task size. Using this TCP, the task can modify the values for C, N and M.

REPORT TASK CHARACTERISTICs
Reports the limits associated with user tasks which run under the timesharing system.

- maximum allowed number of timesharing tasks
- maximum task size
- maximum number of user LUNs assigned
- maximum number of currently assigned devices

REPORT TIME STATISTICS
Reports the following system statistics:

- elapsed time since start-up
- elapsed time since start of statistics collection

6-17
- total time given to timesharing
- total time given to executing timesharing tasks
- total time when no execution occurred
- total time given to batch

REPORT TASK INFORMATION
Reports information about the active timesharing user tasks in the system. Any one of the following can be requested: report all user tasks in the system; report all tasks initiated from a specific terminal.

6.4.7 Device Management Primitives
This group of primitives controls the allocation of devices to timesharing users. It enables the control of multiple users of the system who wish to make use of peripheral devices. It allows any number of interactive users (and optionally one batch user) to have simultaneous access to Files-11 volumes or directory devices and exclusive use of foreign volumes and non-directory devices. The primitives also allow the assignment and deassignment of LUN's to devices which are effective for all subsequent timesharing user tasks run or the CLI.

ASSIGN LUN TO DEVICE
Assigns a terminal user's LUN to: 1) a specified volume mounted on a given device unit; 2) a specified device; 3) a specified volume mounted on any of the devices of the specified device type. In all cases the device must be one which is allocated as available to timesharing.

DEASSIGN LUN
Deassigns the device assigned to a given LUN for a terminal user or deassigns all LUNs for a given device for a user.

CHECK DEVICE ALLOCATION
Checks whether a device is in the timesharing user's device map and is on-line.

RECORD ON-LINE VOLUME
Records the information about volumes mounted for the timesharing users. The number of timesharing users using a device is incremented and a device table entry is made for the requestor's terminal, signifying that the volume has been mounted.

CHECK DEVICE FOR MOUNT
Called by the MOUNT program when the MOUNT utility is being run on behalf of a timesharing user. It ensures that the device can be successfully mounted for the user by checking for a free device map entry and that no other user has the device's exclusive use.

RELINQUISH VOLUME
Makes a device available to other users if the current user has its exclusive control. If the user does not have exclusive control of the device, the system is notified that the device is no longer needed by the user (that it can be dismounted as far as the current user is concerned).

6.4.8 CLI Service
These primitives provide service functions to CLI tasks. Information about the user task currently running for the CLI (name, size, CPU time

6-18
used so far) and devices currently assigned to the CLI’s terminal user are provided. Additionally, the CLI can control the “terminal context” for its user terminal—the CLI can inhibit or allow the action of CTRL/C on the terminal via this mechanism, as well as using it to record the CLI’s own context information.

SET OR REPORT TERMINAL CONTEXT
Reports and optionally changes the context of the terminal. Control context governs whether a CTRL/C is recognized at a terminal.

GIVE JOB STATISTICS
Returns the task and CPU time used for the task currently running for the terminal being serviced by the calling CLI task. It also returns the device and LUN information for the terminal user.

REPORT TERMINALS FOR A CLI
Reports the terminals in the system for the requesting CLI task.

6.5 SYSTEM TASKS AND SPECIAL UTILITIES
IAS provides a common command language for all standard system program development utilities such as the Editor, Linker and Librarian. These utilities are the same as the utilities provided in the RSX-11D system.

In addition to the standard program development utilities, IAS also provides two special system tasks called VERIFY and BAD BLOCKS. These tasks are available only to the system manager. VERIFY is used verify the consistency and validity of the files on a Files-11 volume. BAD BLOCKS is used to locate any unusable blocks on a disk and is normally run prior to disk volume initialization.

The system manager or operator also has available a special utility called CDA. CDA (Core Dump Analyzer) is a task that executes on-line with other tasks to capture system information at the time of crash. It provides the capability to later analyze the state of the system at the time the crash occurred.

General users have access to a special utility called PRESERVE. PRESERVE is a multi-user task that creates copies of disk, magnetic tape or DECtape volumes. PRESERVE can also be booted into memory as a stand-alone program.

6.6 LANGUAGES
IAS supports five source languages: MACRO, FORTRAN IV, FORTRAN IV-PLUS, BASIC and COBOL. Programs written in these languages can be initiated in either interactive or batch mode.

Programs in BASIC and COBOL can be executed immediately after translation because they produce intermediate code which is run by an interpreter. Like the MACRO assembler, the FORTRAN compilers produce machine-language code and therefore require the additional step of linking.

6.6.1 MACRO
The MACRO assembler provided with IAS is the same as the RSX-11D MACRO.
6.6.2 FORTRAN IV
PDP-11 FORTRAN conforms to the specifications of American National Standard FORTRAN (X3.9-1966), with substantial extensions to that standard. Extensions include:

- General expressions are permitted as array subscripts, in DO statements and in I/O statements.
- Random access I/O statements, end-of-file and error transfers
- I/O statements for terminal input and output
- Character literals and LOGICAL*1 variables
- IMPLICIT statement

The FORTRAN system consists of a compiler and an object time system (OTS). The compiler produces object code from the source program. The OTS consists of routines which are selectively loaded with the user's program to perform certain arithmetic, I/O, and system-dependent service operations. The OTS also detects and reports run-time error conditions.

6.6.3 FORTRAN IV-PLUS
There are two FORTRAN systems supported on IAS: FORTRAN IV and FORTRAN IV-PLUS. FORTRAN IV-PLUS, which requires the Floating Point Processor, produces highly optimized code.

In addition to the enhancements FORTRAN IV supplies, FORTRAN IV-PLUS includes:

- ENTRY statement in SUBROUTINE and FUNCTION subprograms
- Specification of lower bounds on arrays
- INTEGER*4 variables, sign plus 31 bits of precision
- OPEN, CLOSE and direct access formatted I/O statements
- Generic selection for many FORTRAN supplied functions

6.6.4 COBOL
PDP-11 COBOL conforms to the ANS-74 COBOL low-level standard, with many high-level extensions.

For situations where the terminal is the only input device, PDP-11 COBOL provides a simple, terminal-oriented line format. Several utility programs are provided with COBOL, including a report-generating program, a sort utility, and a reformatting program.

The PDP-11 COBOL compiler is a compile-and-go system. Programs are compiled, the COBOL object time system is loaded and the program is executed when the user enters the single command "COBOL". The compiler supports the following modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nucleus</td>
<td>All language elements necessary for internal processing</td>
</tr>
<tr>
<td>Table Handling</td>
<td>Defining and manipulating tabular data</td>
</tr>
<tr>
<td>Sequential I/O</td>
<td>Defining and accessing sequential files</td>
</tr>
<tr>
<td>Relative I/O</td>
<td>Defining and accessing relative files including dynamic access</td>
</tr>
</tbody>
</table>

6.20
Segmentation: Specifying overlay of the Procedure Division at object time

Library: Copying predefined COBOL text into the source program

6.6.5 BASIC
BASIC is an incremental compiler that provides immediate translation and storage of the user program while it is being entered. BASIC's "immediate" mode allows each statement to be executed as it is typed in; the computer can be used like a desk calculator. Although BASIC is designed primarily for interactive use, IAS also supports editing and execution of BASIC programs in batch mode. The IAS BASIC language is an extension of Dartmouth BASIC.
section3
pdp11
language
processors

CHAPTER 1  MACRO
CHAPTER 2  FORTRAN IV
CHAPTER 3  FORTRAN IV-PLUS
CHAPTER 4  COBOL
CHAPTER 5  BASIC
1.1 FUNCTIONS AND FEATURES
PDP-11 MACRO processes source programs written in the MACRO assembly language and produces a relocatable object module and optional assembly listing. MACRO is included with the RT-11, RSX-11D, RSX-11M and IAS operating systems. In addition, MACRO is optionally available on the RSTS/E operating system as part of the FORTRAN IV package.

MACRO provides the following features:
- relocatable object modules
- global symbols for linking separately-assembled object programs
- device and filename specifications for input and output files
- user-defined macros
- comprehensive system macro library
- program sectioning directives
- conditional assembly directives
- assembly and listing control functions at program and command string levels
- alphabetized, formatted symbol table listing
- default error listing on command output device

The MACRO assembler included in the RT-11 system (and optionally in the RSTS/E system) also features:
- a Cross Reference Table (CREF) symbol listing

The MACRO assembler included in the RSX-11 and IAS systems also features:
- global arithmetic, global assignment operator, global label operator and default global declarations
- multiple macro libraries with fast access structure
- predefined (default) register definitions
- an indirect command file facility for controlling the assembly process

1.2 LANGUAGE
A MACRO source program is composed of a sequence of source coding lines. Each source line contains a single assembly language statement followed by a statement terminator, such as a carriage return. The assembler processes source statements sequentially, generating binary ma-
chine instructions and data words or performing assembly-time opera-
tions (such as macro expansions) for each statement.

A statement can contain up to four fields which are identified by order
of appearance and by specified terminating characters. The general for-
mat of a MACRO assembly language statement is:

    label: operator operand(s) ; comments

The label and comment fields are optional. The operator and operand
fields are interdependent; either can be omitted depending upon the
contents of the other. Some statements have one operand, for example:

    CLR R0

while others have two:

    MOV #344,R2

A label is a unique user-defined symbol which is assigned the current
location counter and entered into the user-defined symbol table. A label
is a symbolic means of referring to a specific location within a program.
The value of the label can be either absolute (fixed in memory inde-
dependently of the position of the program) or relocatable (not fixed in
memory), depending on whether the location counter value is currently
absolute or relocatable.

An operator field follows the label field, if present, and can contain a
macro call, a PDP-11 instruction mnemonic, or an assembler directive.
When the operator is a macro call, the assembler inserts the appropriate
code during assembly to expand the macro. When the operator is an
instruction mnemonic, it specifies the instruction to be generated and
the action to be performed on any operands which follow. When the
operator is an assembler directive, it specifies a certain function or ac-
tion to be performed during assembly.

An operand is that part of the statement manipulated by the operator.
Operands can be expressions, numbers, symbolic arguments, or macro
arguments.

The comment field can contain any ASCII text characters. Comments do
not affect assembly processing or program execution, but are useful in
source listings for later analysis, documentation or debugging purposes.

1.2.1 Symbols and Symbol Definitions

Three types of symbols can be defined for use within MACRO source
programs: permanent symbols, user-defined symbols and macro symbols.
Correspondingly, MACRO maintains three types of symbol tables: the
Permanent Symbol Table (PST), the User Symbol Table (UST) and the
Macro Symbol Table (MST).

Permanent symbols consist of the PDP-11 instruction mnemonics and
MACRO directives. The PST contains all the permanent symbols auto-
matically recognized by MACRO and is part of the assembler itself. Since
these symbols are permanent, they do not have to be defined by the user
in the source program.
User-defined symbols are those used as labels or defined by direct assignment. Macro symbols are those symbols used as macro names. The UST and MST are constructed during assembly by adding the symbols to the UST or MST as they are encountered.

The value of a symbol depends on its use in the program. A symbol in the operand field can be a macro symbol, a user-defined symbol or a permanent symbol. To determine the value of the symbol, the assembler searches the three symbol tables in the following order: MST, UST and PST.

A symbol used in the operand field can be either a user-defined symbol or a permanent symbol. To determine the value of the symbol, the assembler searches the User Symbol Table and the Permanent Symbol Table in that order.

These search orders allow redefinition of Permanent Symbol Table entries as user-defined or macro symbols. The same name can be assigned to both a macro and a label.

User-defined symbols are either internal or external (global) to a source program module. An internal symbol definition is limited to the module in which it appears. A global symbol can be defined in one source program module and referenced within another.

Internal symbols are temporary definitions which are resolved by the assembler. Global symbols are preserved in the object module and are not resolved until the object modules are linked into an executable program. With some exceptions, all user-defined symbols are internal unless explicitly defined as being global.

When a label is given to a program statement, a symbol table entry is made and the value of the current location counter is assigned to it. In the RSX-11/IAS MACRO, a label can be defined as a global symbol by ending the label name with two colons instead of just one.

A direct assignment statement associates a symbol with a value. When a direct assignment is first used to define a symbol, that symbol is entered into the User-defined Symbol Table, and the specified value is associated with it. The general format for a direct assignment is:

\[
\text{symbol} = \text{expression}
\]

In RSX-11/IAS MACRO, a symbol can be defined as a global symbol by separating the symbol from the expression with two equal signs instead of just one. That is, the statement:

\[
\text{symbol} = = \text{expression}
\]

automatically declares the symbol as a global symbol.

Expressions are combinations of terms that are joined together by binary operators and that reduce to a 16-bit value. Binary operators are, for example, addition, subtraction, multiplication, division, logical AND, and logical inclusive OR.
The expression in a direct assignment statement can itself be a reference to another symbol. In this way, a symbol can be redefined in a subsequent direct assignment statement if the symbol definition contains a reference to a subsequently-defined symbol. Only one level of forward referencing is allowed. The following example illustrates an illegal forward reference:

\[
X = Y \quad \text{(Legal forward reference)} \\
Y = Z \quad \text{(Illegal forward reference)} \\
Z = 1
\]

Although one level of forward referencing is allowed for local symbols, a global symbol defined in a direct assignment statement must not contain a forward reference. The global assignment expression \((= =)\) must not itself contain an undefined reference to another symbol.

Local symbols are specially-formatted internal symbols used as labels within a given range of source code, called a local symbol block. Local symbols are of the form \(n\$\), where \(n\) is a decimal integer between 1 and 65535, inclusive. Examples of local symbols are: \(1\$, 27\$, 59\$, 104\$.

A local symbol block can be delimited in one of three ways:

- The range of a local symbol block usually consists of those statements between two normally-defined labels.
- The range of a local symbol block is normally terminated upon entering a new program section, as defined by a program section directive.
- The range of a local symbol block can be explicitly defined by the use of the \(.ENABL\) and \(.DSABL\) directives.

Local symbols provide a convenient means of generating labels to be referenced only within a local symbol block. The use of local symbols reduces the possibility of entry point symbols with multiple definitions appearing within a program. A local symbol, then, is not referenced from other source program modules or even from outside its local symbol block. Thus, local symbols of the same name can appear in other local symbol blocks without conflict.

### 1.2.2 Directives

A program statement can contain one of three different operators: a macro call, a PDP-11 instruction mnemonic, or an assembler directive. MACRO includes directives for:

- listing control
- function specification
- data storage
- radix and numeric usage declarations
- location counter control
- program termination
- program boundaries information
- program sectioning
• global symbol definition
• conditional assembly
• macro definition
• macro attributes
• macro message control
• repeat block definition
• macro libraries

Table 1-1 lists the MACRO directives.

### Table 1-1 Assembly and Macro Directives

<table>
<thead>
<tr>
<th>Listing Control Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.LIST</strong></td>
</tr>
<tr>
<td><strong>.NLIST</strong></td>
</tr>
<tr>
<td><strong>.TITLE</strong></td>
</tr>
<tr>
<td><strong>.SBTTL</strong></td>
</tr>
<tr>
<td><strong>.IDENT</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.ENABL</strong></td>
</tr>
<tr>
<td><strong>.DSABL</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Storage Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.BLKB</strong></td>
</tr>
<tr>
<td><strong>.BLKW</strong></td>
</tr>
<tr>
<td><strong>.BYTE</strong></td>
</tr>
<tr>
<td><strong>.WORD</strong></td>
</tr>
</tbody>
</table>
Table 1.1  Assembly and Macro Directives (Cont.)

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ASCII</td>
<td>Translates a character string into its equivalent 7-bit ASCII values and stores them in the object module.</td>
</tr>
<tr>
<td>.ASCIZ</td>
<td>Translates a character string into its equivalent 7-bit ASCII values and stores them in the object module appending a zero byte to the string. This enables the program to identify the end of the string by searching for a null character (zero byte).</td>
</tr>
<tr>
<td>.RAD50</td>
<td>Allows three ASCII characters to be packed into one word (Radix-50 format); using this directive, any 6-character symbol can be stored in two consecutive words.</td>
</tr>
<tr>
<td>.FLT2</td>
<td>Stores a floating-point number in 2-word floating-point format.</td>
</tr>
<tr>
<td>.FLT4</td>
<td>Stores a floating-point number in 4-word floating-point format.</td>
</tr>
</tbody>
</table>

Radix and Numeric Control Operators

- **.RADIX** Declare any one of the following radices to apply to succeeding numbers in the source program: 2, 4, 8 or 10.
- **↑D** Declare a (temporary) decimal, octal or binary radix for the number following the control operator.
- **↑B** Declare that the number following the control operator is to be one's complemented as it is evaluated during assembly.
- **↑C** Declare that the number following the control operator is to be interpreted as a 1-word floating point argument.

Location Counter Control Directives

- **.EVEN** Ensures that the current location counter contains an even value by adding one if the current value is odd.
- **.ODD** Ensures that the current location counter contains an odd value by adding one if the current value is even.

Terminating Directives

- **.END** Indicates the logical end of source input and, optionally, specifies the entry point, starting or transfer address.
- **.EOT** End of input tape. Ignored by the Assembler (included for compatibility with PAL-11 assemblers).

Program Boundaries Directive

- **.LIMIT** Reserves two words in the object module which, during linking, are used to store the address of the bottom of the
Table 1-1 Assembly and Macro Directives (Cont.)

program and the address of the first free word following the program image. This enables the program to determine its upper and lower address boundaries during execution.

<table>
<thead>
<tr>
<th>Program Sectioning Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.PSECT</strong></td>
</tr>
<tr>
<td><strong>.ASECT</strong></td>
</tr>
<tr>
<td><strong>.CSECT</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol Control Directive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.GLOBL</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conditional Assembly Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.IF</strong></td>
</tr>
<tr>
<td><strong>.ENDC</strong></td>
</tr>
<tr>
<td><strong>.IFF</strong></td>
</tr>
</tbody>
</table>
Table 1-1  Assembly and Macro Directives (Cont.)

or to the end of the conditional assembly block, is to be included in the program, providing that the condition tested upon entering the conditional assembly block is false.

**.IFT**
The code following this subconditional directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program, providing that the condition tested upon entering the conditional assembly block is true.

**.IFTF**
The code following this subconditional directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program, regardless of the result of the condition tested upon entering the conditional assembly block.

**.IIF**
Assemble this line of code if the condition specified on the line is met.

**Macro Definition Directives**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO</td>
<td>Identifies the beginning of a macro definition.</td>
</tr>
<tr>
<td>ENDM</td>
<td>Identifies the end of a macro definition.</td>
</tr>
<tr>
<td>MEXIT</td>
<td>Terminates a macro expansion before the end of the macro is encountered.</td>
</tr>
</tbody>
</table>

**Macro Attribute Directives**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NARG</td>
<td>Determines the number of arguments in the macro call currently being expanded.</td>
</tr>
<tr>
<td>NCHR</td>
<td>Determines the number of characters in a specified character string. It is useful in calculating the length of macro arguments.</td>
</tr>
<tr>
<td>NTYPE</td>
<td>Determines the addressing mode of a specified macro argument.</td>
</tr>
</tbody>
</table>

**Macro Message Control Directives**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>Sends a message to the listing file during assembly pass 2. A common use of this directive is to provide a diagnostic announcement of a rejected or erroneous macro call or to alert the user to the existence of an illegal set of conditions specified in a conditional assembly.</td>
</tr>
<tr>
<td>PRINT</td>
<td>Identical to the .ERROR directive, except that it is not flagged in the assembly with an error code.</td>
</tr>
</tbody>
</table>

**Macro Repeat Block Directives**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRP</td>
<td>Replaces a dummy argument with successive real arguments specified in an argument string.</td>
</tr>
</tbody>
</table>
Table 1-1  Assembly and Macro Directives (Cont.)

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.IRPC</td>
<td>Replaces a dummy argument with each successive character of the specified string.</td>
</tr>
<tr>
<td>.REPT</td>
<td>Duplicates a block of code a number of times in-line with other source code.</td>
</tr>
</tbody>
</table>

**Macro Library Directive**

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.MCALL</td>
<td>Includes in the assembly macro definitions which are taken from system or user macro library files.</td>
</tr>
</tbody>
</table>

**LISTING CONTROL DIRECTIVES**
Several listing control directives are provided in MACRO to control the content, format and pagination of all listing output generated during assembly. Facilities also exist for creating object module names and other identification information in the listing output.

The listing control options can also be specified at assembly-time through switch options included in the listing file specification in the command string issued to the MACRO assembler. The use of these switch options overrides all corresponding listing control directives in the source program.

When no listing file is specified, any errors encountered in the source program are printed on the terminal from which MACRO was initiated.

**FUNCTION DIRECTIVES**
Several function control options are provided by MACRO through the .ENABL and .DSABL directives. These directives are included in a source program to invoke or inhibit certain MACRO functions and operations incident to the assembly process itself. They include the ability to:

- Produce absolute binary output.
- Assemble all relative addresses as absolute addresses. This function is useful during the debugging phase of program development.
- Cause source columns 73 and greater (to the end of the line) to be treated as comment. The most common use of this feature is to permit sequence numbers in card columns 73-80.
- Truncate or round floating point literals.
- Accept lower case ASCII input instead of converting it to upper case.
- Enable a local symbol block to cross program section boundaries. A local symbol block is normally established by encountering a new symbolic label or a program section directive in the source program. By enabling a local symbol block to cross program section boundaries, a local symbol block can be established which is not terminated until another symbolic label or program section directive is encountered following a disable local symbol block function directive. Local symbols cannot, however, be defined in a program section other than that which was in effect when the block was entered. The basic function of this directive in regard to program sections is limited to those in-
stances were it is desirable to leave a program section temporarily to store data, followed by a return to the original program section.

- Inhibit binary output.
- Inhibit the normal default register definitions (RSX-11/IAS MACRO only).
- Treat all undefined symbol references as default global references (RSX-11/IAS MACRO only).

CONDITIONAL ASSEMBLY DIRECTIVES
Conditional assembly directives enable the programmer to include or exclude blocks of source code during the assembly process, based on the evaluation of stated condition tests within the body of the program. This capability allows several variations of a program to be generated from the same source.

The user can define a conditional assembly block of code, and within that block, issue subconditional directives. Subconditional directives within conditional assembly blocks are used to indicate:

- The assembly of an alternate body of code when the condition of the block tests false.
- The assembly of a non-contiguous body of code within the conditional assembly block, depending on the result of the conditional test on entering the block.
- The unconditional assembly of a body of code within a conditional assembly block.

Conditional assembly directives can be nested. MACRO permits a nesting depth of 16 conditional assembly levels.

MACRO DEFINITIONS AND REPEAT BLOCKS
In assembly-language programming, it is often convenient and desirable to generate a recurring coding sequence by invoking a single statement within the program. In order to do this, the desired coding sequence is first established with dummy arguments as a macro definition. Once a macro has been defined, a single statement calling the macro by name with a list of real arguments (replacing the corresponding dummy arguments in the macro definition) generates the desired coding sequence or macro expansion.

Macros can be nested; that is, the definition of one macro can include a call to another. The depth of nesting allowed is dependent on the amount of memory used by the source program being assembled.

A label is often required in an expanded macro. Normally, a label can be explicitly specified as an argument with each macro call. Care must be taken, however, in issuing subsequent calls to the same macro in order to avoid specifying a duplicate label as a real argument. This concern is eliminated through a feature of MACRO which creates a unique symbol where a label is required in an expanded macro.

MACRO can automatically create unique local symbols. This automatic facility is invoked on each call of a macro whose definition contains a
dummy argument preceded by the question mark (?) character, if a real argument of the macro call is either null or missing. If the real argument is specified in the macro call, however, MACRO does not generate a local symbol and normal argument replacement occurs.

An indefinite repeat block is a structure that is very similar to a macro definition. Such a structure is essentially a macro definition that has only one dummy argument. At each expansion of the indefinite repeat range, this dummy argument is replaced with successive elements of a specified real argument list. An indefinite repeat block directive and its associated repeat range are coded in-line within the source program. This type of macro definition does not require calling the macro by name, as required in the expansion of conventional macros described above.

An indefinite repeat block can appear within or outside of another macro definition, indefinite repeat block, or repeat block.

MACRO CALLS AND STRUCTURED MACRO LIBRARIES
All macro definitions must occur prior to their references within the user program. MACRO provides a selection mechanism for the programmer to indicate in advance those system macro definitions required in the program. (System macros include the monitor programmed requests or executive directives available with each operating system.)

The .MCALL directive is used to specify the names of all the macro definitions not defined in the current program but which are used in the program. When this directive is encountered, MACRO searches the system macro library file to find the requested definition.

RSX-11/IAS MACRO extends this macro call facility by enabling the programmer to retrieve macros from libraries of user-defined macros. The .MCALL directive provides the means to access both user-defined and system macro libraries during assembly.

The RSX-11/IAS MACRO assembler assumes that the system macro library and user-defined macro libraries are constructed in a special direct access format to retrieve macro definitions quickly. These structured macro libraries are created by the RSX-11/IAS Librarian utility program.

Each library file contains an index of the macro definitions it contains. When an .MCALL directive is encountered in the source program, MACRO searches the user macro library for the named macro definitions and, if necessary, continues the search with the system macro library. Because each macro library contains an index of all of its entries, MACRO searches only the index in each library to find where the macro definition is stored.

1.3 ASSEMBLER OPERATION
The MACRO Assembler assembles one or more ASCII sources containing MACRO statements into a single relocatable binary object program. MACRO can accept source data from any input device, such as a disk or card reader. The sources to be included in a single assembly are listed in the command string from left to right in the order in which
they are to be assembled. The last statement in the last source specified is normally the .END statement. In RSX-11/IAS MACRO, if the .END statement is not provided, it is assumed.

Assembler output consists of the binary object file and an optional assembly listing followed by the symbol table listing. Using the MACRO available under RT-11 or RSTS/E, cross reference (CREF) listings can also be produced.

MACRO is a two-pass assembler. During assembly pass one, MACRO locates and reads all required macros from libraries, builds symbol tables and program section tables for the program, and performs a rudimentary assembly of each source statement. During assembly pass two, MACRO completes the assembly, writes out an object file, and generates an assembly and symbol table listing for the program.

At the end of assembly pass one, MACRO determines whether a given global symbol is defined in the current program modules or whether it is to be treated as an external symbol. In general, all undefined global symbols appearing in a given program must be defined by the end of assembly pass one. In RSX-11/IAS MACRO, however, all symbols remaining undefined at the end of assembly pass one are assumed to be default global references.

The object module MACRO produces must be processed by the operating system's linker utility program (called the Linker or Task Builder) to create an executable program. The linker joins separately-assembled object modules into a single load module (task image). The linker fixes (makes absolute) the values of the external or relocatable symbols in the object module.

To enable the linker to fix the value of an expression, MACRO passes it certain directives and parameters. In the case of the relocatable expressions in the object module, the linker adds the base of the associated relocatable program section to the value of the relocatable expression provided by MACRO. In the case of external expression values, the linker determines the value of the external term in the expression (since the external expression must be defined in at least one of the other object modules being linked together) and then adds it to the absolute portion of the external expression, as provided by MACRO.

In summary, an executable program image can be constructed from one or more source modules, which can be assembled either separately or together. The resultant object module(s) must be linked together using the linker utility. Figure 1-1 illustrates the processing steps required to produce an executable program from several sources stored as files.

PROGRAM SECTIONING
The MACRO program sectioning directives are used to declare names for program sections and to establish certain program section attributes. These program section attributes are used when the program is linked into an executable load module or task.

A program can consist of an absolute program section, an unnamed relocatable program section, and up to 254 named relocatable program
sections. The absolute program section serves to "link" the program with fixed memory locations such as interrupt vectors and the peripheral device register addresses.

The relocatable program sections are also called control sections, since they normally contain instructions. The unnamed control section is internal to each object module. That is, every object module can have an unnamed control section but the linker treats each control section independently. Each is assigned an absolute address such that it occupies an exclusive area of memory. Named control sections, on the other hand, are treated globally, in the same manner as FORTRAN COMMON.* If different object modules have control sections with the same name, they are all assigned the same absolute load address and the size of the area reserved for loading of the section is the size of the largest. Thus, named control sections allow for the sharing of data and/or instructions among object modules.

The assembler maintains separate location counters for each section. The first occurrence of a program section directive assumes that the current location counter is set at relocatable zero. The scope of this directive then extends until a directive declaring a different program section is specified. For example:

*If declared with the .PSECT directive (see below), they must have the attributes global and overlaid.
.CSECT ;start the unnamed relocatable section
A: 0 ;assembled at relocatable 0,
B: 0 ;relocatable 2 and
C: 0 ;relocatable 4
ST: CLR A ;assemble code at
CLR B ;relocatable address
CLR C ;6 through 21
.ASECT ;start the absolute section
..=4 ;assemble code at
.WORD .+2, HALT ;absolute locations 4 through 7
.CSECT ;resume the unnamed relocatable section
INC A ;assemble code at
BR ST ;relocatable 22 through 27
.END

By maintaining separate location counters for each program section, MACRO allows the user to write statements which are not physically contiguous within the program, but which can be loaded contiguously following assembly.

MACRO under RT-11 and RSTS/E includes two program sectioning directives: .CSECT and .ASECT. The .CSECT directive is used to define the named and unnamed relocatable program sections. The .ASECT directive is used to identify the portions of the absolute program section.

RSX-11/IAS MACRO includes an additional program sectioning directive, .PSECT, due to the unique nature of the RSX-11/IAS linker utility program, the Task Builder. RSX-11/IAS assembly language programs can use the .PSECT directive exclusively, since it affords all the capabilities of the .CSECT and .ASECT directives. RSX-11/IAS MACRO will accept .ASECT and .CSECT directives, but assembles them as if they were .PSECT directives with implicit default attributes which make them identical to their meaning in RT-11 and RSTS/E MACRO.

The .PSECT directive allows the user to exercise absolute control over the memory allocation of a program at task-build time, since any program attributes established through this directive are passed to the Task Builder. For example, if a programmer is writing programs for a multi-user environment, a program section containing pure code (instructions only) or a program section containing impure code (data only) can be explicitly declared through the .PSECT directive. Furthermore, these program sections can be explicitly declared as read-only code, qualifying them for use as protected, reentrant programs. In addition, program sections exhibiting the global attribute can be explicitly allocated in a task’s overlay structure by the user at task-build time. The advantages gained through sectioning programs in this manner therefore relate primarily to control of memory allocation, program modularity, and more effective partitioning of memory in RSX-11 and IAS systems.

The .PSECT directive allows the user to define the following program section attributes:

Access
Two types of access can be permitted to the program section: read-only
or read/write. RSX-11D and IAS support read-only access by setting hardware protection for the program section.

Contents
A program section can contain either instructions or data. This attribute allows the Task Builder to differentiate global symbols that are program entry-point instructions from those that are data values.

Scope
The scope of the program section can be global or local. In building single-segment programs, the scope of the program has no meaning, because the total memory allocation for the program will go into the root segment of the task. The global or local attribute is significant only in the case of overlays. If an object module contains a local program section, then the storage allocation for that module will occur within the segment in which the module resides. Many modules can reference this same program section, and the memory allocation for each module is either concatenated or overlaid within the segment, depending on the argument of the program section defining its allocation requirements (see below). If an object module contains a global program section, the memory area allocations to this program section are collected across segment boundaries, and the allocation of memory for that section will go into the segment nearest the root in which the first memory allocation to this program section appeared.

Relocatability
A program section can be absolute or relocatable. When a program section is declared to be absolute, the program section requires no relocation. The program section is assembled and loaded, starting at absolute virtual address 0. When the program section is declared to be relocatable, the Task Builder calculates a relocation bias and adds to it all references within the program section.

Allocation Requirements
The program section can be concatenated or overlaid. When concatenated, all memory allocations to the program section are to be concatenated with other references to this same program section in order to determine the total memory allocation requirements for this program section. When overlaid, all memory allocations to the program section are to be overlaid. Thus, the total allocation requirement for the program section is equal to the largest individual allocation request for this program section.

1.4 ASSEMBLER ENVIRONMENTS
MACRO is the assembler included in the RT-11, RSX-11, and IAS operating systems. IAS MACRO is identical to the MACRO assembler used in the RSX-11 systems. In addition, MACRO is included in the FORTRAN IV package option available for the RSTS/E system. The RSTS/E version of MACRO is essentially identical to the MACRO assembler included in the RT-11 operating system.

The following sections summarize the operating environments of the RT-11 and RSX-11 MACRO products.
Under RT-11
MACRO requires an RT-11 system configuration (or background partition) of 12K words or more. If more than 12K words are available, MACRO will use the additional memory to increase the amount of symbol table space possible.

RT-11 MACRO provides a system macro library containing the expanded code for all the RT-11 Monitor's programmed requests. Refer to the RT-11 chapter in Section II of this handbook for a list of the RT-11 programmed requests.

Under the RT-11 operating system, a smaller version of MACRO, called ASEM BL, is available for users with minimum system configurations. ASEM BL has the same features as MACRO with the following exceptions:

- macro directives (.MACRO, .MCALL, .EN DM, .IRP, etc.) are not recognized
- DATE is not printed in listings
- wide line-printer output is not available
- there is no lower case mode
- there is no enable/disable punch directive
- there are no floating point directives
- there are no local symbols or local symbol blocks
- CREF is not available

Most of the macro definition features not available in ASEM BL are supported by EXPAND. EXPAND is an RT-11 system program which processes the macro references in a macro assembly language source file.

EXPAND simply copies its input files to its output file unless it encounters any of the following directives: .MCALL, .MACRO, .name, and .ENDM. The .MCALL directive instructs EXPAND to search the system macro library to find the macro names listed in the directive, and store their definition in internal tables. The .MACRO directive instructs EXPAND to copy a macro definition from the user's input file to its internal tables. The .name directive, if "name" is the name of a macro, instructs EXPAND to replace the macro call with the definition stored in its internal tables. The .EN DM directive terminates the macro definition when encountered while EXPAND stores a macro definition.

Under RSX-11
MACRO requires a minimum of 14K words of partition space to execute. The system macro library includes executive directives and file system calls. Refer to the RSX-11 chapter of Section II for a description of the executive directives and file system calls.

Under the RSX-11M system, an 8K word version is available for users who have limited memory space. The 8K version differs from the 14K version in the following ways:

- it does not search the Permanent Symbol Table for a symbol appearing in the operand field of a statement
• It does not recognize the following .ENABL/.DSABL directive function control options: produce absolute binary output, ignore card column sequence numbers, truncate floating point values, accept lower case input, inhibit binary output

• It does not recognize or accept the PAL-11R conditional assembly directives and .EOT directive

• It does not flag in the assembly listing the instructions which are not common among all members of the PDP-11 family

• It does not accept floating point directives or control operators (.FLT2, .FLT4, ↑F), etc.), or PDP-11/45 or PDP-11/70 floating point opcodes.
CHAPTER 2

FORTRAN IV

2.1 FUNCTIONS AND FEATURES
The FORTRAN IV compiler and Object Time System is included with the RSX-11D and RSX-11M operating systems. It is available as an optional language processing system for the RT-11, RSTS/E and IAS operating systems. The FORTRAN compiler accepts source programs written in the FORTRAN IV language and produces an object file which must be linked prior to execution. The PDP-11 FORTRAN IV language conforms to the specifications for the American National Standard FORTRAN X3.9-1966. The following are enhancements to the American National Standard:

- Array Subscripts—any arithmetic expression can be used as an array subscript. If the value of the expression is not an integer, it is converted to integer format.
- Array Dimensions—arrays can have up to seven dimensions.
- Character Literals—character strings bounded by apostrophes can be used in place of Hollerith constants.
- Mixed-mode Expressions—mixed-mode expressions can contain any data type, including complex and byte.
- End of line comments—any FORTRAN statement can be followed, in the same line, by a comment that begins with an exclamation point.
- Debugging Statements—statements that are included in a program for debugging purposes can be so designated by the letter D in column 1. Those statements are compiled only when the associated compiler command string option switch is set. They are treated as comments otherwise.
- Read/Write End-of-file or Error Condition Transfer—the specifications END = n and ERR = n (where n is a statement number) can be included in any READ or WRITE statement to transfer control to the specified statement upon detection of an end-of-file or error condition. The ERR = n option is also permitted in the ENCODE and DECODE statements, allowing program control of data format errors.
- General Expressions in I/O lists—general expressions are permitted in I/O lists of WRITE, TYPE, and PRINT statements.
- General Expression DO and GO TO Parameters—general expressions are permitted for the initial value, increment, and limit parameters in the DO statement, and as the control parameter in the computed GO TO statement.
- DO Increment Parameter—the value of the DO statement increment parameter can be negative.
• Optional Statement Label List—the statement label list in an assigned
GO TO is optional.
• Override Field Width Specifications—undersized input data fields can
contain external separators to override the FORMAT field width speci-
cifications for those fields (called "short field termination"), permitting
free-format input from terminals.
• Default FORMAT Widths—the FORTRAN IV programmer may specify
input or output formatting by type and default width, and precision val-
ues will be supplied.
• Additional I/O Statements:
  Device-oriented I/O
      ACCEPT
      TYPE
      PRINT
  Memory-to-memory formatting
      ENCODE
      DECODE
  Unformatted direct access I/O
      DEFINE FILE
      READ (u'r)  u = logical unit number
      WRITE (u'r)  r = record number
      FIND (u'r)

The unformatted direct access I/O facility allows the FORTRAN pro-
grammer to read and write files written in any format.
• Logical Operations on INTEGER Data—the logical operators .AND., .OR.,
.NOT., .XOR., and .EQV. may be applied to integer data to perform bit
masking and manipulation.
• Additional Data Type—the byte data type (keyword LOGICAL*1 or BYTE)
is useful for storing small integer values as well as for storing and
manipulating character information. It enables the programmer to
save space when manipulating small integer values without affecting
their characteristics, because they are treated internally as integer
values. Since the arithmetic and masking operations are legal for the
byte data type, it is also possible to manipulate and modify character
data.
• IMPLICIT Declaration—the IMPLICIT statement has been added to
redefine the implied data type of symbolic names.
• Fewer Syntactic Restrictions—FORTRAN IV has no statement ordering
requirements, allowing declarations to appear anywhere within the
source program. Terminal format input (using the tab character to de-
limit fields) eases program preparation.

The FORTRAN IV compiler is characterized by extremely rapid compila-
tion rates. Typical 300 line programs compile in less than nine seconds
on PDP-11/45 RT-11 systems (less than 25 seconds on a PDP-11/10).
Large 2200 line programs compile in just one minute on a PDP-11/45.

The FORTRAN IV compiler performs well in small environments. On an
RT-11 system with as little as 8K words of memory, FORTRAN IV can
compile programs containing as many as 450 lines. On an RT-11 system
with 28K words, FORTRAN IV can compile programs containing as many
as 2200 lines. On a RSTS/E system with a 28K word user area, FORTRAN can compile programs with as many as 2400-2500 lines.

Despite its small size requirements and high compilation rate, FORTRAN IV provides a high level of automatic object program optimization. The compiler performs redundant expression elimination, constant expression folding, branch structure optimization, and several types of subscripting optimizations.

2.2 LANGUAGE
A FORTRAN program consists of FORTRAN statements and optional comments. There are two kinds of statements: executable and non-executable. Executable statements describe the action of the program. Non-executable statements describe the data arrangement and characteristics, and provide editing and data conversion information.

There are assignment statements, control statements, I/O statements, FORMAT statements and specification statements. FORMAT and specification statements are non-executable. Table 2-1 summarizes the FORTRAN IV language components.

<table>
<thead>
<tr>
<th>Expression Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>** TYPE ** OPERATOR OPERATES ON</td>
</tr>
<tr>
<td>Arithmetic ** ** exponentiation arithmetic or logical constants, variables and expressions</td>
</tr>
<tr>
<td>Arithmetic * multiplication</td>
</tr>
<tr>
<td>Arithmetic / division</td>
</tr>
<tr>
<td>Arithmetic +, — addition, subtraction, unary plus and minus</td>
</tr>
<tr>
<td>Relational .GT. greater than arithmetic or logical constants, variables and expressions (all relational operators have equal priority)</td>
</tr>
<tr>
<td>Relational .GE. greater than or equal to</td>
</tr>
<tr>
<td>Relational .LT. less than</td>
</tr>
<tr>
<td>Relational .LE. less than or equal to</td>
</tr>
<tr>
<td>Relational .EQ. equal to</td>
</tr>
<tr>
<td>Relational .NQ. not equal to</td>
</tr>
<tr>
<td>Logical .NOT. .NOT.A is true if and only if A is false</td>
</tr>
<tr>
<td>Logical .AND. A.AND.B is true if and only if A and B are both true</td>
</tr>
<tr>
<td>Logical .OR. A.OR.B is true if and only if A or B or both are true</td>
</tr>
<tr>
<td>Logical .EQV. A.EQV.B is true if and only if either A and B are both true or A and B are both false</td>
</tr>
</tbody>
</table>

2-3
<table>
<thead>
<tr>
<th>TYPE</th>
<th>OPERATOR</th>
<th>OPERATES ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>.XOR.</td>
<td>A.XOR.B is true if and only if A is true and B is false or B is true and A is false</td>
</tr>
</tbody>
</table>

**Assignment Statements**

- variable=expression
  - Arithmetic/Logical Assignment:
    - The value of the arithmetic or logical expression is assigned to the variable.
- ASSIGN-TO
  - The ASSIGN statement is used to associate a statement label with an integer variable. The variable can then be used as a transfer destination in a subsequent assigned GO TO statement in the same program unit.

**Control Statements**

- GO TO
  - Unconditional
    - Transfers control to the same statement every time it is executed.
  - Computed
    - Permits a choice of transfer destinations, based on a value of an expression within the statement.
  - Assigned
    - Transfers control to a statement label that is represented by a variable. Because the relationship between the variable and a specific statement label must be established by an ASSIGN statement, the transfer destination can be changed, depending upon which ASSIGN statement was most recently executed.
- IF
  - Arithmetic
    - Transfers control to a statement depending on the value of an arithmetic expression. Used for conditional control transfers.
  - Logical
    - Executes a statement if the test of a logical expression is true.
- DO
  - Causes the statements in its range to be repeatedly executed a specified number of times. The range of the DO begins with the statement following the DO and ends with a specified terminal statement. The number of iterations is determined by the values for the initial, terminal, and increment parameters.
- CONTINUE
  - Causes no processing. Passes control to the next executable statement. Used primarily as the terminal statement of
Table 2-1  FORTRAN IV Language Summary (Cont.)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>Calls a SUBROUTINE subprogram and passes its actual arguments to replace</td>
</tr>
<tr>
<td></td>
<td>the dummy arguments in the subprogram.</td>
</tr>
<tr>
<td>RETURN</td>
<td>Returns control from a subprogram to the calling program unit.</td>
</tr>
<tr>
<td>PAUSE</td>
<td>Prints a message (if specified) on the terminal and suspends execution until</td>
</tr>
<tr>
<td></td>
<td>the user responds.</td>
</tr>
<tr>
<td>STOP</td>
<td>Terminates program execution and prints a message (if specified) on the</td>
</tr>
<tr>
<td></td>
<td>terminal.</td>
</tr>
<tr>
<td>END</td>
<td>Marks the end of a program unit. In a main program, if control reaches the</td>
</tr>
<tr>
<td></td>
<td>END statement, a CALL EXIT is implicitly executed. In a subprogram, a</td>
</tr>
<tr>
<td></td>
<td>RETURN statement is implicitly executed.</td>
</tr>
</tbody>
</table>

Input/Output Statements

- **READ**
  - Formatted: Reads at least one logical record from the specified device according to the given format specifications, and assigns values to the elements in a list.
  - Unformatted: Reads one logical record from the specified device, assigning the input values to the variables in a list.
  - Direct Access: Reads from the specified logical record and assigns the input values to the variables in a list.
  - Error Control: Optional elements in the READ statement allow control transfer on error conditions. If an end-of-file condition is detected and the END option is specified, execution continues at a given statement. If a recoverable I/O error occurs and the ERR option is specified, execution continues at a given statement.

- **WRITE**
  - Formatted: Writes one or more logical records containing the values of the variables in a list onto the specified device in the given format.
  - Unformatted: Writes one logical record containing the values of the variables in the list onto the specified device.
<table>
<thead>
<tr>
<th><strong>Table 2-1</strong> FORTRAN IV Language Summary (Cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Direct Access</strong></td>
</tr>
<tr>
<td><strong>Error Control</strong></td>
</tr>
<tr>
<td><strong>ACCEPT</strong></td>
</tr>
<tr>
<td><strong>TYPE</strong></td>
</tr>
<tr>
<td><strong>PRINT</strong></td>
</tr>
<tr>
<td><strong>DEFINE FILE</strong></td>
</tr>
<tr>
<td><strong>REWIND</strong></td>
</tr>
<tr>
<td><strong>BACKSPACE</strong></td>
</tr>
<tr>
<td><strong>END FILE</strong></td>
</tr>
<tr>
<td><strong>FIND</strong></td>
</tr>
<tr>
<td><strong>ENCODEx</strong></td>
</tr>
<tr>
<td><strong>DECODEx</strong></td>
</tr>
</tbody>
</table>
Table 2-1 FORTRAN IV Language Summary (Cont.)

Format Statements

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT</td>
<td>Describes the format in which one or more records are to be transmitted. The format descriptors include integer and octal, logical, real, double precision, complex, literal and editing. Real, double precision and complex formats can be scaled.</td>
</tr>
</tbody>
</table>

Specification Statements

<table>
<thead>
<tr>
<th>Implicit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPLICIT_</td>
<td>Overrides the implied data type of symbolic names, in which all names that begin with the letters I, J, K, L, M, or N are presumed to be INTEGER values, and all names beginning with any other letter are assumed to be REAL values, unless otherwise specified. IMPLICIT allows the programmer to define the initial letters for implied data types. If a variable is not given an explicit type, and its name begins with a letter defined in an IMPLICIT statement, its default type is that defined by the IMPLICIT statement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type var1, var2, ..., varn</td>
<td>Type Declaration: The given variable names are assigned the specified data type in the program unit. Type is one of INTEGER<em>2, INTEGER</em>4, REAL<em>4, REAL</em>8, DOUBLE PRECISION, COMPLEX<em>8, LOGICAL</em>4, LOGICAL*1 or BYTE.</td>
</tr>
</tbody>
</table>

| DIMENSION  | Reserves storage space for the specified array(s).                         |
| COMMON     | Reserves one or more blocks of storage space under the specified name to contain the variables associated with the block name. |
| EQUIVALENCE| Declares two or more variable names in the same program unit to be associated with the same storage location. |
| EXTERNAL   | Permits the use of external procedures (functions, subroutines and FORTRAN library functions) as arguments to other subprograms. It informs the system that the names specified are those of routines not contained in the current program unit. |
| DATA       | Permits the assignment of initial values to variables and array elements prior to program execution. |
| PROGRAM    | Assigns a name to a main program unit. If present, it is the first statement in the main program. |


### Table 2.1 FORTRAN IV Language Summary (Cont.)

**User-Written Subprograms**

- **name(var1, var2, ...) = expression**  
  Arithmetic Statement Function: Creates a user-defined function having the variables as dummy arguments. When referenced, the expression is evaluated using the actual arguments in the function call.

- **FUNCTION**  
  Begins a FUNCTION subprogram, indicating the program name and any dummy variable names. An optional type specification can be included.

- **SUBROUTINE**  
  Begins a SUBROUTINE subprogram, indicating the program name and any dummy variable names.

- **BLOCK DATA**  
  Specifies the subprogram which follows as a BLOCK DATA subprogram. An optional name for the program unit may be given.

### Fortran Library Functions

- **ABS(X)**  
  Real absolute value

- **IABS(X)**  
  Integer absolute value

- **DABS(X)**  
  Double Precision absolute value

- **CABS(Z)**  
  Complex to Real, absolute value

- **FLOAT(I)**  
  Integer to Real conversion

- **IFIX(X)**  
  Real to Integer conversion

- **SNGL(X)**  
  Double to Real conversion

- **DBLE(X)**  
  Real to Double conversion

- **REAL(Z)**  
  Complex to Real conversion

- **AIMAG(Z)**  
  Complex to Real conversion

- **CMPLX(X,Y)**  
  Real to Complex conversion

- **AINT(X)**  
  Real to Real truncation

- **INT(X)**  
  Real to Integer conversion

- **IDINT(X)**  
  Double to Integer conversion

- **AMOD(X,Y)**  
  Real remainder

- **MOD(I,J)**  
  Integer remainder

- **DMOD(I,J)**  
  Double Precision remainder

- **AMAX0(I,J,...)**  
  Real maximum from Integer list

- **AMAXI(I,J,...)**  
  Real maximum from Real list

- **MAX0(I,J,...)**  
  Integer maximum from Integer list

- **MAXI(X,Y,...)**  
  Integer maximum from Real list

- **DMAXI(X,Y,...)**  
  Double maximum from Double list

- **AMIN0(I,J,...)**  
  Real minimum of Integer list

- **AMINI(X,Y,...)**  
  Real minimum of Real list

- **MIN0(I,J,...)**  
  Integer minimum of Integer list

- **MINI(X,Y,...)**  
  Integer minimum of Real list

- **DMINI(X,Y,...)**  
  Double minimum from Double list

- **SIGN(X,Y)**  
  Real transfer of sign

- **ISIGN(I,J)**  
  Integer transfer of sign

- **DSIGN(X,Y)**  
  Double Precision transfer of sign

- **DIM(X,Y)**  
  Real positive difference

- **IDIM(I,J)**  
  Integer positive difference
| EXP(X)     | e raised to the X power (X is Real) |
| DEXP(X)    | e raised to the X power (X is Double) |
| CEXP(Z)    | e raised to the Z power (Z is Complex) |
| ALOG(X)    | Returns the natural log of X (X is Real) |
| ALOG10(X)  | Returns the log base 10 of X (X is Real) |
| DLOG(X)    | Returns the natural log of X (X is Double) |
| DLOG10(X)  | Returns the log base 10 of X (X is Double) |
| CLOG(Z)    | Returns the natural log of Z (Z is Complex) |
| SQRT(X)    | Square root of Real argument |
| DSQRT(X)   | Square root of Double Precision argument |
| CSQRT(Z)   | Square root of Complex argument |
| SIN(X)     | Real sine |
| DSIN(X)    | Double Precision sine |
| CSIN(Z)    | Complex sine |
| COS(X)     | Real cosine |
| DCOS(X)    | Double Precision cosine |
| CCOS(Z)    | Complex cosine |
| TANH(X)    | Hyperbolic tangent |
| ATAN(X)    | Real arctangent |
| DATAN(X)   | Double Precision arctangent |
| ATAN2(X,Y) | Real arctangent of (X/Y) |
| DATAN2(X,Y)| Double Precision arctangent of (X/Y) |
| CONJG(Z)   | Complex conjugate |
| RAN(I,J)   | Returns a random number between 0 and 1 |

2.3 COMPILER OPERATION
The FORTRAN IV compiler accepts a source written in the FORTRAN language as input and produces an object file and a listing file as output. The object file must be subsequently processed by the operating system’s linker program, for example, the Linker or Task Builder, to produce an executable program.

Command String Specification Options
In the input/output file specification command string issued to the FORTRAN IV compiler to request program compilation, the user can specify a number of switch parameter options. Some of the parameters are:

SPECIFY LISTING OPTIONS
The user can request a number of listing options. By default, the user is supplied with diagnostics (if any), a source program listing, and the storage map. In addition; the user can request a generated code listing, or can combine any of the listing options in a single listing. The generated code listing contains a symbolic representation of the object code generated by the compiler, including a location offset from the base of the program unit, the symbolic Object Time System (OTS) routine names, and routine arguments. The code generated for each statement is labelled with the same internal sequence number that appears in the source program listing, for easy cross reference.

COMPILE DEBUGGING STATEMENT LINES
The user can request the compiler to include in the compilation those lines with a “D” in column one. These statements allow the inclusion of programmer-selected debugging aids (see below).
ENABLE/DISABLE THE COMMON SUBEXPRESSION OPTIMIZER
In general, the optimizer will make the program run faster. Disabling the optimizer can reduce program storage requirements, but will increase execution time.

INCLUDE OR SUPPRESS INTERNAL SEQUENCE NUMBERS
Suppressing internal sequence number accounting reduces program storage requirements for generated code and slightly increases execution time, but disables line number information during traceback.

ALLOCATE TWO WORDS FOR DEFAULT LENGTH OF INTEGER VARIABLES
Normally, single storage words will be the default allocation for integer variables not given an explicit length specification (i.e., INTEGER*2 or INTEGER*4). Only one word is used for computation. The user can request that the default allocation be two storage words.

ENABLE/DISABLE VECTORING OF ARRAYS
Array vectoring is a process which decreases the time necessary to reference elements of a multidimensional array by using some additional memory to store array accessing information. If array vectoring is enabled, the compiler decides whether to vector a multidimensional array based on the ratio of the amount of space required to vector the array over the total space required by the array. If this ratio is greater than 25\%, the array is not vectored, and standard mapping is used instead. If size is a more critical factor than speed, the user can disable the vectoring of all arrays. If arrays are vectorized, it is so noted in the storage map listing.

ENABLE/DISABLE COMPILER WARNING DIAGNOSTICS
Warning diagnostics report conditions which are not fatal error conditions, but which can be potentially dangerous at execution time, or which may present compatibility problems with other FORTRAN compilers running on PDP-11 operating systems. For example, a warning message is generated if a variable name exceeds six characters in length. This is potentially dangerous if another variable name has the same first six characters. Another example is that statement ordering restrictions are not imposed by the FORTRAN IV compiler, but are imposed by the FORTRAN IV-PLUS compiler. A program written for the FORTRAN IV compiler which does not conform to the FORTRAN IV-PLUS convention could not be compiled by both FORTRAN IV and FORTRAN IV-PLUS compilers. The warning diagnostics are normally enabled, but the user can suppress their inclusion in the diagnostics listing.

Internal Operation and Structure
Instead of using temporary files to process source programs, the FORTRAN IV compiler performs all its activities in main memory. It reads the entire source program once, stores it in memory in a compacted format, and processes the compacted code in memory. The advantage to this method is that the FORTRAN IV compiler does not require a disk device for its operation. In addition, since a disk device is not used for temporary file operations, compilation speed is significantly increased.

To reduce the memory requirements of such a compilation system, the FORTRAN IV compiler employs a multi-phase overlaid structure. The com-
The compiler consists of a large number of overlays, each of which occupies no more than 1K to 1.25K words of memory. Most of the space allocated to the compiler is occupied by the compressed source code. Figure 2-1 illustrates the compile-time memory map.

![Figure 2-1 Compile-Time Memory Map](image)

The compiler goes through a series of processing phases, one for each of its 18-20 overlays. Each program segment is processed separately, generally using the entire sequence of overlay phases. The basic processing phases are:

1. Source program compaction and listing
2. Syntax analysis and error reporting
3. Non-executable statement processing
4. Code generation
5. Code optimization

The compiler begins by reading in as much of the source subprogram as it can fit in memory. It then compresses the source code in memory by removing blanks and other unnecessary data. It continues to read in more source code, compressing it as it goes, until the entire program segment fits in memory.

Once the source code is compacted into memory, the compiler begins processing the internal form of the source code as a whole. Because the entire program segment is available to the compiler at a glance, FORTRAN IV does not require statement ordering restrictions.

During the first stage of code generation, the compiler immediately writes as much information as possible to the object file. This step is necessary to further compress the internal source code to enable the symbol table to grow in the later stages of processing.
The non-executable statements are eligible for immediate processing, since the information they provide is not needed until run-time. Therefore, the compiler searches for all the occurrences of non-executable statements, such as FORMAT and DATA statements, produces the beginning of the object module, and compacts the internal source code further.

To begin the next stage of code generation, the compiler enters all variables and constants not yet processed into the symbol table, and performs the syntax scan of the executable statements. The program is translated into an internal format in preparation for final code generation.

Object Code Generation
A few executable FORTRAN statements can be translated directly into machine instructions. Typical FORTRAN operations, however, require long sequences of PDP-11 machine instructions. For example, standard sequences are needed to locate an element of a multidimensional array, initialize an I/O operation, or simulate a floating-point operation not supported by the hardware configuration.

The common sequences of PDP-11 machine instructions are contained in a library known as the FORTRAN Object Time System (OTS). The FORTRAN IV compiler does not actually generate pure machine instructions for the FORTRAN source code statements. It simply determines which combination of appropriate OTS routines are needed to implement a FORTRAN program. During the linking process for an object program, the linker utility includes the needed OTS routines into the load module. During program execution, these routines are chained together to effect the desired result.

The compiler references a library instruction sequence by generating a word containing the address of the first instruction in the OTS routine, followed by the information upon which the routine is to operate (the operands). For example, an OTS routine used to perform the end-of-DO-loop sequence must be passed the location of the index variable, the limit value, and the address of the beginning of the loop.

The compiler and OTS make use of the PDP-11 general register and indirect addressing facility to have the OTS routines executed at run-time. Register 4 (R4) is used to chain together the selected OTS routines. The last instruction executed in each library routine is a JMP @(R4)+, which transfers control to the next library instruction sequence.

Optimizations
The FORTRAN IV compiler performs the following optimizations during code generation:

1. Compiled FORMAT Statements
   The compiler interprets the FORMAT statements at compile-time, translating the format into an internal form. This not only increases the execution speed of the program, it decreases its size, because less run-time code is needed.

2. Array Vectoring
   Array vectoring decreases the time necessary to reference elements of a multidimensional array by using additional memory to store the array. If an array is vectored, a particular element in the array can be
located by a simplified mapping function, without the need for multiplication operations. A table lookup is performed to determine the location of a particular element.

3. Constant Folding
   Integer constant expressions are evaluated at compile-time.

4. Compile-time Evaluation of Constant Subscript Expressions
   Constant subscript expressions in array calculations are evaluated at compile-time.

5. Elimination of Unreachable Code
   Statements that are never reached by flow of control are eliminated from the object code.

6. Common Subexpression Elimination
   Redundant subexpressions whose operands are not changed between computations are replaced by a temporary value calculated only once.

7. Peephole Optimizations
   The compiler examines the internal form of the object code on an operation-by-operation basis to replace sequences of operations with shorter and faster equivalent operations. For example, the compiler replaces a divide-by-two operation with a multiply-by-one-half operation. There are a large set of these kinds of operations.

8. Branch Optimizations for Arithmetic and Logical IF
   Branch structure optimizations improve program speed and decrease its size. For example, an arithmetic IF statement can often be improved:

   IF(A-7.0)100, 200, 100 !goto 200 if A is equal to 7.0
   100 CONTINUE

   The compiler will optimize this statement to:

   IF(A .EQ. 7.0) GOTO 200

Libraries
The FORTRAN programmer can create a library of commonly-used assembly language and FORTRAN functions and subroutines. The operating system's librarian utility provides a library creation and modification capability. Library files may be included in the command string to the linker utility. The linker recognizes the file as a library file and links only those routines in the library that are required in the executable program. By default, the linker also automatically searches the FORTRAN system library for any other required routines.

Debugging a FORTRAN Program
Two debugging facilities are available to the FORTRAN programmer. The FORTRAN Object Time System provides the traceback feature for fatal run-time errors. This feature locates the actual program unit and line number of a run-time error. Immediately following the error message, the error handler will list the line number and program unit name in which the error occurred. If the program unit is a subroutine or function subroutine, the error handler will trace back to the calling program unit and display the name of that program unit and the line number where the call occurred. This process will continue until the calling sequence has been traced back to a specific line number in the main program. This
allows the exact determination of the location of an error even if the error occurs in a deeply nested subroutine.

In addition to the FORTRAN OTS error diagnostics which include the traceback feature, there is another debugging tool available. A "D" in column one of a FORTRAN statement allows that statement to be conditionally compiled. These statements are considered comment lines by the compiler unless the appropriate debugging lines switch is issued in the compiler command string. In this case, the lines are compiled as regular FORTRAN statements. Liberal use of the PAUSE statement and selective variable printing can provide the programmer with a method of monitoring program execution. This feature allows the inclusion of debugging aids that can be compiled in the early program testing stages and later eliminated without source program modification.

2.4 FORTRAN IV OPERATING ENVIRONMENTS
The FORTRAN IV compiler and OTS is available as an optional language processor for the RT-11, RSTS/E, and IAS operating systems. The FORTRAN IV compiler is included with the RSX-11M and RSX-11D operating systems. The compiler operation and facilities under each of these systems are essentially identical.

Each operating system provides additional features particular to the environment. For example, the monitor programmed requests or executive directives are usually available as a library of FORTRAN-callable routines.

Under RT-11
The RT-11 System Subroutine Library (SYSLIB) is a collection of FORTRAN-callable routines which allow a FORTRAN user to utilize various features of the RT-11 Foreground/Background (F/B) and Single-Job monitors. SYSLIB also provides various utility functions, a complete character string manipulation package, and 2-word integer support. SYSLIB is provided as a library of object modules to be combined with FORTRAN programs at link-time.

SUMMARY OF SYSLIB CAPABILITIES

- Complete RT-11 I/O facilities, including synchronous, asynchronous, and event-driven modes of operation. FORTRAN subroutines may be activated on the completion of an input/output operation.
- Timed scheduling of asynchronous subjobs (completion routines). (Under F/B operation only.)
- Complete facilities for interjob communication. (Under F/B operation only.)
- FORTRAN interrupt service routines.
- Complete timer-support facilities, including timed suspension of execution (F/B only), conversion of different time formats, and time-of-day information. These timer facilities support either 50 or 60-cycle clocks.
- All auxiliary input/output functions provided by RT-11, including the capabilities of opening, closing, renaming, creating, and deleting files from any device.
- All monitor level informational functions, such as job partition parameters, device statistics, and input/output channel statistics.

2-14
• Access to the RT-11 Command String Interpreter (CSI) for acceptance and parsing of standard RT-11 command strings.

• A character string manipulation package supporting variable-length character strings.

• INTEGER*4 support routines that allow two-word integer computations.

SYSLIB allows the RT-11 FORTRAN user to write almost all application programs in FORTRAN with no assembly-language coding.

Also available under RT-11 are:

• A library of FORTRAN-callable graphics routines supporting the VT11, GT40, GT42, and GT44 graphics hardware systems.

• Plotting support for the LV11 electrostatic printer/plotter.

• Laboratory data acquisition and manipulation routines used in conjunction with the LPS-11 and AR11 laboratory peripheral hardware.

• The Scientific Subroutine Library, providing FORTRAN-language routines for mathematical and statistical applications.

**Under RSTS/E**

RSTS/E FORTRAN IV operates in interactive or batch mode under the RSTS/E monitor. The FORTRAN IV system includes the FORTRAN IV compiler, the Object Time System (OTS), and several utility programs.

The entire system (including compiler and optimization components) is completely functional in an 8K word user area. A system interface occupying 4K words of memory is shareable among all FORTRAN IV users on the system. In addition, the FORTRAN IV system provides overlay support for programs and data, allowing extremely large programs to be run in a small region of memory.

FORTRAN IV supports all processor arithmetic options available on RSTS/E systems, including Floating Point Processor option and the Floating Instruction Set option.

RSTS/E FORTRAN IV provides assembly language subprogram support, using the MACRO assembler. Although the assembly language subprogram can not issue any monitor calls, MACRO provides the experienced user with a tool to further enhance computational performance.

**Under RSX-11 and IAS**

In RSX-11M, the FORTRAN IV compiler runs in a minimum partition of 7K words. If run in a larger partition it uses the extra space for program and symbol table storage. In RSX-11D and IAS, the compiler task requires 8K words minimally and can be extended when it is installed. As with RSX-11M systems, the additional space allows the processing of larger FORTRAN programs.

An RSX-11/IAS library consists of object modules. Two types of libraries exist, shared and relocatable.

Relocatable libraries are stored in files. Object modules from relocatable libraries are built into the task image of each task referencing the module. The Task Builder is used to include modules from relocatable libraries in
a task image. When a library specification is encountered in the command string, those modules in the library which contain definitions of any currently undefined global symbols are included in the task image. The user can construct relocatable libraries of assembly language and FORTRAN routines using the Librarian utility.

Shared libraries are located in main memory and a single copy of each library is used by all referencing tasks. Access to a shared library is gained by specifying the name of the library in an option at task build time. Shared libraries are built using the Task Builder. They must contain shareable (reentrant) code.

Each RSX-11/IAS system has a system relocatable library. The system relocatable library is automatically searched by the Task Builder if any undefined global references are left after processing all user-specified input files. The FORTRAN OTS may be included in the system library and hence is loaded automatically with FORTRAN programs.

The RSX-11/IAS system library provides FORTRAN-callable forms of most executive directives. The FORTRAN programmer can schedule the execution of tasks, communicate with concurrently executing tasks, and manipulate system resources through these calls.

Industrial Society of America (ISA) extensions for process I/O control are available in FORTRAN-callable format under RSX-11. Support for laboratory and process control peripherals is also included.
CHAPTER 3

FORTRAN IV-PLUS

3.1 FUNCTIONS AND FEATURES
The FORTRAN IV-PLUS compiler and Object Time System is an optional language processing system for the RSX-11D, RSX-11M, and IAS operating systems. The FORTRAN IV-PLUS compiler accepts source programs written in the FORTRAN IV language and produces an object file which must be linked prior to execution. The FORTRAN IV-PLUS language conforms to the specifications for the American National Standard FORTRAN X3.9-1966.

Both the FORTRAN IV and the FORTRAN IV-PLUS compilers can be used in the same environment. If both FORTRAN compilers are to be used on the same system, two separate FORTRAN library files are maintained. One compiler must be selected as the "default" compiler. The one selected as the default is the one that can be used in batch processing.

The primary differences between the FORTRAN IV compiler and the FORTRAN IV-PLUS compiler are that the FORTRAN IV-PLUS compiler:

- produces highly optimized PDP-11 machine language code
- requires the FPP Floating Point Processor option
- can produce shareable code

On typical compute-bound programs, a FORTRAN IV-PLUS compiled program executes two to three times faster than the same program compiled by the FORTRAN IV compiler on identical configurations.

The FORTRAN IV-PLUS language is upward compatible with the PDP-11 FORTRAN IV language. The FORTRAN IV-PLUS system supports the same enhancements to the language standard as FORTRAN IV. In addition, FORTRAN IV-PLUS also includes the following extensions:

- ENTRY statements can be used in SUBROUTINE and FUNCTION subprograms to define multiple entry points in a single program unit.
- PARAMETER statements can be used to give symbolic names to constants.
- Lower bounds as well as upper bounds of the array dimension can be specified in array declarators. The value of the lower bound dimension declarator can be negative, zero, or positive.
- The data type INTEGER*4 provides a sign plus 31 bits of precision. INTEGER*4 allows a greater range of values to be represented than INTEGER*2. Both data types can be used in the same program.
• A compiler command line specification allows all INTEGER and LOGICAL declarations without explicit length specifications to be considered as INTEGER*2 and LOGICAL*2, or INTEGER*4 and LOGICAL*4, respectively.

• The following I/O statements have been added:
  OPEN } File control and attribute definition
  CLOSE
  READ(u'r,fmt) } Direct access using formatted records
  WRITE(u'r,fmt)

The latter two I/O statements provide formatted direct access I/O operations, since the READ and WRITE statements contain references to FORMAT statements or format specifications in arrays.

• Formatted I/O can be performed without a format specification using list-directed I/O.

• Generic function selection by argument data type is provided for many FORTRAN library functions.

• The control variable of a DO statement can be a DOUBLE PRECISION data type as well as an INTEGER*2, INTEGER*4, or REAL data type. The initial, terminal, and increment parameters can be of any data type and are converted before use to the type of the control variable if necessary.

• The INCLUDE statement incorporates FORTRAN source text from a separate file into a FORTRAN program.

• The number of times a DO loop is executed (called the iteration count) is determined at the initialization of the DO statement and is not re-evaluated during successive executions of the loop. Consequently, the number of times the loop is executed will not be affected by changing the variables used in the DO statement. That is, the terminal and increment parameters can be modified within the loop without affecting the iteration count.

3.2 LANGUAGE EXTENSIONS

The FORTRAN IV-PLUS language is upward compatible from the PDP-11 FORTRAN IV language. The following paragraphs discuss some of the additional language components that FORTRAN IV-PLUS provides from the FORTRAN IV language. Table 3-1 at the end of this chapter compares the implementation of the FORTRAN IV and FORTRAN IV-PLUS languages.

I/O Statements

OPEN
Associates an existing file with a logical unit, or creates a new file and associates it with a logical unit. In addition, the statement can contain specifications for file attributes that direct the creation or subsequent processing. The attributes include specifying: the file name, the method of access (direct, sequential or append), protection (read only or read/write), form (formatted, unformatted), record size, block allocation or extension, whether the file can be shared, and whether the file is to be deleted or saved when closed (disposition). In addition, the OPEN statement can be modified by an ERR keyword which specifies the statement to which control is transferred if an error is detected.
CLOSE
Disassociates a file from a logical unit. Disposition attributes specified in the OPEN statement can be modified. For example, a file opened as a file to be deleted can be saved, or a file opened to be saved can be deleted.

FORMATTED DIRECT ACCESS INPUT/OUTPUT
Formated direct access READ and WRITE statements are used to perform direct access I/O of character data with a file on a direct access device. The OPEN statement is used to establish the attributes of the file. Each READ or WRITE contains an expression that specifies the number of the record to be accessed.

The formatted direct access READ statement causes the specified record to be read from the direct access file currently associated with the given logical unit. The characters in the record are scanned and converted as indicated by the given format specification. The resulting values are assigned to the elements in a list.

The formatted direct access WRITE statement writes the specified record in the direct access file currently associated with the given logical unit. A list specifies a sequence of values which are converted to characters and positioned as specified by a format specification.

LIST DIRECTED INPUT/OUTPUT
List-directed input and output statements provide a method for obtaining simple sequential formatted input or output without the need for FORMAT statements. On input, values are read from the unit, converted to internal format, and assigned to the elements of the I/O list. On output, values in the I/O list are converted to characters and written in a fixed format according to the data type of the value. The I/O list must be present.

The list-directed READ statement transfers data from the specified unit, converts it into an internal format, and assigns the input values to the elements of the I/O list in the order in which they appear.

The list-directed WRITE statement transmits the elements in the I/O list to the specified unit, translating and editing each value according to the data type of the value.

The list-directed ACCEPT statement functions identically to the list-directed READ statement, except that input is from a logical unit normally connected to the terminal keyboard.

The list-directed TYPE statement functions identically to the list-directed WRITE statement, except that output is directed to a logical unit normally connected to the terminal printer.

The list-directed PRINT statement functions identically to the list-directed WRITE statement, except that output is directed to a logical unit normally connected to the line printer.

Specification Statements
INCLUDE
Specifies that the contents of a designated file are to be incorporated in the FORTRAN compilation directly following the INCLUDE statement. An
INCLUDE statement can appear anywhere a comment line can appear. When the compiler encounters an INCLUDE statement, it stops reading statements from the current file and starts reading statements from the included file. When the end of the included file is reached, compilation resumes with the statement following the INCLUDE statement. An INCLUDE statement can be contained in an included file.

The INCLUDE statement provides a mechanism for writing modular, reliable and maintainable programs by eliminating duplication of source code. A section of program text that is used by several program units, such as a COMMON block specification, can be created and maintained as a separate source file. All program units which referenced the COMMON block then merely INCLUDE this common file. Any changes to the COMMON block will be reflected automatically in all program units after compilation.

EXTERNAL *name
Specifies that a name refers to a user-defined external FUNCTION or SUBROUTINE subprogram, in order to differentiate it from a FORTRAN library processor-defined function.

PARAMETER
Allows a constant to be given a symbolic name. The symbolic name of a constant assumes the type implied in the form of its corresponding constant. The initial letter of the name has no effect on its type.

User-Written Subprograms
ENTRY
Provides multiple entry points within a subprogram. It is not executable and can appear within a function or subroutine program after the FUNCTION or SUBROUTINE statement. Execution begins with the first executable statement following the ENTRY statement.

Library Functions
The following additional FORTRAN library functions are provided:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIN(X)</td>
<td>Real arcsine</td>
</tr>
<tr>
<td>DASIN(X)</td>
<td>Double precision arcsine</td>
</tr>
<tr>
<td>ACOS (X)</td>
<td>Real arccosine</td>
</tr>
<tr>
<td>DACOS(X)</td>
<td>Double precision arccosine</td>
</tr>
<tr>
<td>SINH(X)</td>
<td>Real hyperbolic sine</td>
</tr>
<tr>
<td>DSINH(X)</td>
<td>Double precision hyperbolic sine</td>
</tr>
<tr>
<td>COSH(X)</td>
<td>Real hyperbolic cosine</td>
</tr>
<tr>
<td>DCOSH(X)</td>
<td>Double precision hyperbolic cosine</td>
</tr>
<tr>
<td>TANH (X)</td>
<td>Real hyperbolic tangent</td>
</tr>
<tr>
<td>DTANH (X)</td>
<td>Double precision hyperbolic tangent</td>
</tr>
<tr>
<td>TAN(X)</td>
<td>Real tangent</td>
</tr>
<tr>
<td>DTAN(X)</td>
<td>Double precision tangent</td>
</tr>
<tr>
<td>NINT(X)</td>
<td>Real to Integer nearest Integer</td>
</tr>
<tr>
<td>ANINT(X)</td>
<td>Real nearest Integer</td>
</tr>
<tr>
<td>IDNINT(X)</td>
<td>Double to Integer nearest Integer</td>
</tr>
<tr>
<td>IAND(I,J)</td>
<td>Integer bitwise AND</td>
</tr>
<tr>
<td>IOR(I,J)</td>
<td>Integer bitwise OR</td>
</tr>
<tr>
<td>IEOR(I,J)</td>
<td>Integer bitwise Exclusive OR</td>
</tr>
</tbody>
</table>
NOT(I)  Integer NOT
ISHFT(I,J)  Integer bitwise Shift

**Generic Function References**

Generic function names provide a means by which some of the FORTRAN mathematical functions can be called with selection of the actual library routine used, based on the type of the argument that occurs in the function reference. For example, if X is a real variable, then SIN(X) will reference the real-valued sine function. If D is a double precision variable then SIN(D) will reference the double precision sine function. It is not necessary to write DSIN(D).

Generic function selection is performed independently for each function reference. Given the above example, both SIN(X) and SIN(D) can be used in the same program unit.

The set of functions for which generic name selection is performed are shown below. Generic function selection can only be used with the argument types shown.

<table>
<thead>
<tr>
<th>SYMBOLIC NAME</th>
<th>TYPE OF ARGUMENT</th>
<th>TYPE OF RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>AINT, ANINT</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>INT, NINT</td>
<td>Real</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Integer</td>
</tr>
<tr>
<td>SNGL</td>
<td>Integer</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Real</td>
</tr>
<tr>
<td>DBLE</td>
<td>Integer</td>
<td>Double</td>
</tr>
<tr>
<td></td>
<td>Real</td>
<td>Double</td>
</tr>
<tr>
<td>MOD, MAX, MIN, SIGN, and DIM</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>EXP, LOG, SIN, COS, and SQRT</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>LOG10, TAN, ATAN, ATAN2, ASIN, ACOS, SINH, COSH, and TANH</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>Double</td>
</tr>
</tbody>
</table>

**3.3 Compiler Operation and Optimizations**

The FORTRAN IV-PLUS compiler accepts a source written in the FORTRAN language and produces an object file which must be linked by the RSX-11/IAS Task Builder prior to execution. The compiler uses a work file system similar to that used by the PDP-11 COBOL compiler to produce the object file. This work file system allows very large FORTRAN programs to be compiled in a limited amount of memory.
The compiler generates PDP-11 machine language code for the object program which includes FPP instructions. During compilation, the FORTRAN IV-PLUS compiler performs many code optimizations.

The FORTRAN IV-PLUS optimizations are designed to produce an object program that executes in less time than an equivalent non-optimized program. The optimizations are also designed to reduce the size of the object program.

The FORTRAN IV-PLUS compiler performs the following optimizations:

- Constant folding. Integer constant expressions are evaluated at compile-time.
- Compile-time evaluation of constant subscript expressions in array calculations.
- Elimination of unreachable code. An optional warning message is issued to mark unreachable statements in the source program listing.
- Recognition and replacement of common subexpressions.
- Peephole optimization. The code is examined on an operation-by-operation basis to replace sequences of operations with shorter and faster equivalent operations.
- Branch instruction optimizations for arithmetic or logical IF statements.
- Compile-time constant conversion.
- Argument-list merging. If two function or subroutine references have the same arguments, a single copy of the argument list is generated.
- Removal of invariant computations from DO loops.
- Local register assignment. Frequently referenced variables are retained (if possible) in registers to reduce the number of load and store instructions.
- Assignment of frequently used variables and expressions to registers across DO loops.
- JMP/BRANCH instruction resolution. The BRANCH instruction is used wherever possible to eliminate unnecessary JMP instructions.

A FORTRAN IV-PLUS program is computationallly equivalent to an unoptimized program according to the definition of the FORTRAN language. Thus identical numerical results are obtained and equivalent run-time diagnostics are produced. Messages may not, however, occur at exactly the same statements in the source programs. In addition, fewer messages are produced. For example, if common subexpression elimination takes place for a redundant calculation which produces a error, only one message is produced for the subexpression.

Compile-Time Operations on Constants
The FORTRAN IV-PLUS compiler performs the following compile-time computations on expressions involving constants, including PARAMETER constants.

- Negation of Constants. For example,
  \[ X = -10.0 \]
  is compiled as a single MOV instruction.
• Type Conversion of Constants. For example,
  \[ X = 10 \times Y \]
  is compiled as
  \[ X = 10.0 \times Y \]

• Integer Arithmetic on Constants. For example,
  \[ \text{PARAMETER NN=27} \]
  \[ I = 2 \times NN + J \]
  is compiled as
  \[ I = 54 + J \]

In addition, array subscripts involving constants are simplified at compile-time where possible. For example,

\[
\begin{align*}
\text{DIMENSION I(10,10)} \\
I(1,2) &= I(4,5)
\end{align*}
\]

is compiled as a single MOV instruction:

\[
\text{MOV I+130,I+26}
\]

This not only significantly increases the speed of the program, it reduces its size.

Elimination of Common Subexpressions

Often the same subexpression appears in more than one computation. If the values of the operands of a common subexpression are not changed between computations, the value of the subexpression can be computed once and its result can be substituted where the subexpression appears. For example, the sequence:

\[
\begin{align*}
A &= B \times C + E \times F \\
H &= A + G - B \times C \\
\text{IF}((B \times C - H)10,20,30)
\end{align*}
\]

contains the common subexpression \( B \times C \). The sequence is compiled as:

\[
\begin{align*}
t &= B \times C \\
A &= t + E \times F \\
H &= A + G - t \\
\text{IF}((t - H)10,20,30)
\end{align*}
\]

where \( t \) is a temporary variable created by the compiler. Two computations of the subexpression \( B \times C \) are eliminated from the sequence.

A more subtle application of common subexpression elimination occurs in the following example. The statements:

\[
\begin{align*}
\text{DIMENSION A(25,25), B(25,25)} \\
A(I,J) &= B(I,J)
\end{align*}
\]
are compiled, without optimization, as the sequence of instructions in the following form:

\[
\begin{align*}
t1 &= J*25+1 \\
t2 &= J*25+1 \\
MOVE B(t2) &\ TO\ A(t1)
\end{align*}
\]

The variables \( t1 \) and \( t2 \) represent equivalent expressions. The redundancy is recognized and the following shorter, faster sequence is generated:

\[
\begin{align*}
t &= J*25+1 \\
MOVE B(t) &\ TO\ A(t)
\end{align*}
\]

**Removal of Invariant Computations from DO Loops**

The speed with which a given algorithm can be executed is increased if computations are moved from frequently executed program sequences to less frequently executed program sequences. In particular, computations within a loop involving only constants and variables whose values are not changed within the loop can be moved outside the loop.

For example, in the sequence:

\[
\begin{align*}
DO 10, I=1,100 \\
10 & \quad F = 2.0*Q*A(I) + F
\end{align*}
\]

the value of the subexpressions \( 2.0*Q \) is the same during each iteration of the loop. Transformation of the sequence to:

\[
\begin{align*}
t &= 2.0*Q \\
DO 10, I=1,100 \\
10 & \quad F = t*A(I) + F
\end{align*}
\]

moves the calculation \( 2.0*Q \) outside the loop and eliminates 99 multiply operations.

**Generated Code Example**

The FORTRAN routine:

```
0001       DIMENSION A1(25)
```

```
0007       AMIN= A1(1)
0008       AMAX= A1(1)
0009       DO 40 I=2,N
0010       IF (A1 (I) .LT. AMIN) AMIN= A1(I)
0011       IF (A1 (I) .GT. AMAX) AMAX= A1(I)
0012       40 CONTINUE
```

is compiled into the following code:

```
SETF
LDF   A1,F0
```

;Statement 0007.

;AMAX is bound to FPP register
;F0 and initialized to A1(1).
LDF F0,F1 ;AMIN is bound to FPP register
      ;F1 and initialized to A1(1).
      ;Statement 0009.
MOV #2,R0 ;The DO loop control variable I
      ;is bound to register R0 and
      ;initialized to 2.

L$GACD:
      ;Statement 0010.
MOV R0,R1
ASL R1
ASL R1
LDF A1–4(R1),F2 ;For each iteration of the
CMPF F1,F2 ;loop, A1(1) is bound to FPP
CFCC ;register F2.
BLE L$GAPE
LDF F2,F1

L$GAPE:
      ;Statement 0011.
CMPF F0,F2
CFCC
BGE L$GAFF
LDF F2,F0

L$GAFF:
      ;Statement 0012.
INC R0
CMP R0,N
BLE L$GACD
STF F1,AMIN ;At the end of the loop, the
STF F0,AMAX ;values of AMIN, AMAX and I
MOV R0,I ;are stored in memory.

3.4 ENVIRONMENTS
The FORTRAN IV-PLUS compiler is available as an optional language
processor for the RSX-11M, RSX-11D and IAS operating systems. The
compiler's operation and facilities under each of these operating sys-
tems are essentially identical. One of the special features is that FOR-
TRAN IV-PLUS can produce shareable code.

In all operating systems, the hardware configuration must include the
FPP Floating Point Processor. The FORTRAN IV-PLUS compiler requires
a minimum partition size of 17K words to execute in an RSX-11D or
IAS system. Under RSX-11M, the compiler requires a minimum partition
size of 18K words.
Table 3-1 FORTRAN Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>ANS</th>
<th>FOR</th>
<th>FOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X3.99</td>
<td>TRA</td>
<td>TRA</td>
</tr>
<tr>
<td></td>
<td>1966</td>
<td>IV</td>
<td>IV-PLUS</td>
</tr>
<tr>
<td>PROGRAMMING CONSIDERATIONS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of array dimensions</td>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Mixed mode expressions</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double exponentiation (right to left evaluation)</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Generalized subscripts</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Adjustable dimensions</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>No statement ordering restrictions</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Generic function selection</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Maximum number of continuation lines</td>
<td>19</td>
<td>∞</td>
<td>99</td>
</tr>
<tr>
<td>Embedded blanks in key words</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Comment line starts with</td>
<td>C</td>
<td>C,D</td>
<td>C,D</td>
</tr>
<tr>
<td>End of line comments</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SUBPROGRAM STATEMENTS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLOCK DATA</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ENTRY</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Arithmetic functions</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Logical functions</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SUBROUTINE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CONSTANT FORMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double-precision</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Logical</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Literal (preceded by H)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Literal (enclosed in single quotes)</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hexadecimal (DATA statements only)</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Octal</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Radix 50 (DATA statements only)</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATA TYPES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating Point</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REAL*4</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REAL*8</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COMPLEX*8</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Logical</td>
<td>X</td>
<td>X(^1)</td>
<td>X(^2)</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>X</td>
<td>X(^1)</td>
<td>X(^2)</td>
</tr>
<tr>
<td>LOGICAL*1 (or BYTE)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LOGICAL*2</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>LOGICAL*4</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Integer</td>
<td>X</td>
<td>X(^2)</td>
<td>X(^2)</td>
</tr>
</tbody>
</table>

\(^1\) Implemented as *4.

\(^2\) Defaults to *2 or *4 at compile-time.

3-10
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>ANS X3.99 1966</th>
<th>FORTRAN IV</th>
<th>FORTRAN IV-PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER*2</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>SPECIFICATION STATEMENTS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAM name</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PARAMETER</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPLICIT type</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COMMON name</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DIMENSION</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EQUIVALENCE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EXTERNAL*</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>DATA INITIALIZATION STATEMENTS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA name</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Data values converted to type of name</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RELATIONAL OPERATORS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.GT.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.GE.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.LT.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.LE.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.EQ.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.NE.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>LOGICAL OPERATORS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.NOT.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.AND.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.OR.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.XOR.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>.EQV.</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>ASSIGNMENT STATEMENTS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Logical</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>CONTROL STATEMENTS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DO</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>END</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GO TO</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Arithmetic IF</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Logical IF</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PAUSE (1-6 octal digits)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PAUSE (1-5 decimal digits)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PAUSE 'message'</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RETURN</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>STOP (1-6 octal digits)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>STOP (1-5 decimal digits)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>STOP 'message'</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>SEQUENTIAL I/O STATEMENTS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCEPT</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>BACKSPACE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

¹Four byte allocation; two byte precision.
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>ANS X3.99 1966</th>
<th>FORTRAN IV</th>
<th>FORTRAN IV-PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENDFILE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PRINT</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ formatted sequential (default unit)</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ formatted sequential</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ unformatted sequential</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ list-directed</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Read END, ERR keywords</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REWIND</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TYPE</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WRITE formatted sequential</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WRITE unformatted sequential</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WRITE unformatted sequential (null record)</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WRITE list-directed</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Write END, ERR keywords</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ENCODE/DECODE</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>FILE CONTROL STATEMENTS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>-</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CLOSE</td>
<td>-</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>DIRECT ACCESS I/O STATEMENTS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEFINE FILE</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FIND</td>
<td>-</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>READ formatted direct access</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ unformatted direct access</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WRITE formatted direct access</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WRITE unformatted direct access</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>FORMAT statement</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A (Alphanumeric)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D (Real D decimal exponent)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>E (Real E decimal exponent)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>F (Real, no exponent)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>G (Real)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H (Literal)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'...' (Literal)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I (Integer)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>L (Logical)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>O (Octal)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>P (Scale factor)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>T (Record position indicator)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X (Skipped data or blank)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Format specification in arrays</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Carriage control</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>/ (record separator)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>: (format scan terminator)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$ (format separator and carriage control)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1Unformatted only.
Table 3-1 FORTRAN Language Implementations (cont)

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>ANS X3.99</th>
<th>FORTRAN 1966</th>
<th>FORTRAN IV</th>
<th>FORTRAN IV-PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORTRAN LIBRARY FUNCTIONS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Absolute value:</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Integer</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double Precision</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex to Real</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Conversion:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer to Real</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Integer to Double</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real to Integer</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double to Real</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real to Double</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex to Real (return real part)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex to Real (return imaginary part)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real to Complex</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cosine:</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real (radians)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double (radians)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hyperbolic: (Real and Double)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sine</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Cosine</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Tangent</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Arcsine: (Real and Double)</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Arccosine: (Real and Double)</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Arctangent:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Quotient of two arguments</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sine:</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real (radians)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double (radians)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Tangent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical functions:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IAND (or .AND. for integer values)</td>
<td></td>
<td>-</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>IOR (or .OR. for integer values)</td>
<td></td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IEOR (or .XOR. for integer values)</td>
<td></td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>NOT (or .NOT. for integer values)</td>
<td></td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Switch register</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex conjugate</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Positive difference</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Exponential:</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

1 Available as implied conversion only.
2 RSX-11/IAS systems only.
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>ANS X3.99 1966</th>
<th>FOR TRAN IV</th>
<th>FOR TRAN IV-PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logarithm:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Square root:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complex</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Truncation:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real to Real</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Real to Integer</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double to Integer</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Remaindering:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Integer</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double precision</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Transfer of sign:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Integer</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Double precision</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Miscellaneous:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test sense switch</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Random number</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
4.1 FUNCTIONS AND FEATURES
PDP-11 COBOL provides fast direct access data processing for commercial applications. PDP-11 COBOL is available as an optional language processor for the RSTS/E, RSX-11D and IAS operating systems. Included in the COBOL package are the PDP-11 SORT file sorting program, RFRMT source reformatting program, and COBRG report generator program utilities.

PDP-11 COBOL is a fully implemented low-level compiler conforming in language element, representation, symbology, and coding format to ANS-74 COBOL, specification X.3.23-1974. ANS-74 COBOL includes the following:

- full high-level Nucleus module
- full high-level Table handling module
- full high-level Sequential I/O module
- full high-level Relative I/O module
- low-level segmentation module
- full low-level Library function, with partial high-level REPLACING facility
- Conditional variables—DATA DIVISION level 88
- Nested conditionals

The hardware configuration supporting COBOL is any valid RSTS/E, RSX-11D or IAS operating system configuration with a line printer and enough memory to support a 23K word COBOL task size. The recommended minimum disk storage is either two RK05 disk drives or an RP03 or RP04 disk drive. Optional hardware supported includes a card reader.

The disk-resident compiler can accept source program input from cards, console terminals, and disks—including the capability of accepting input from source text library files stored on disks. In addition, COBOL programs can also create ANSI standard format magnetic tape files if magnetic tape systems are included in the system's hardware configuration.

Compilation and execution of COBOL programs by the DIGITAL PDP-11 systems are characterized by a high rate of throughput and efficient memory utilization. Depending on the size of the compiler generated in a particular system, COBOL source programs can contain up to 6,000 statements. Running on a PDP-11/45 processor, COBOL can compile up
to 400 lines a minute. On a PDP-11/70 processor, COBOL can compile at a rate between 1000 and 1200 lines per minute.

4.2 LANGUAGE FEATURES
PDP-11 COBOL is a fully implemented low-level compiler conforming in language element, representation, symbology and coding format to ANS-74 COBOL.

The PDP-11 COBOL processing modules are:

- **Nucleus**: All language elements necessary for internal processing.
- **Table Handling**: Defining and manipulating tabular data
- **Sequential I/O**: Defining and accessing sequential files
- **Relative I/O**: Defining and accessing relative files, including dynamic access
- **Segmentation**: Specifying overlay of the Procedure Division at object time
- **Library**: Copying predefined COBOL text into the source program; changing text while copying

The Nucleus, Table Handling, Sequential I/O and Relative I/O modules of PDP-11 COBOL meet full ANS-74 high-level standards. RERUN, ENTER and ALTERNATE are not, however, included in the PDP-11 COBOL Nucleus Level 2 code set. PDP-11 COBOL offers high-level extensions in the Segmentation and Library modules. Figure 4-1 illustrates the ANS-74 implementation level of PDP-11 COBOL. Table 4-1 at the end of this chapter compares the language elements implemented in PDP-11 COBOL with the ANS-74 COBOL language elements.

**String Manipulation**
PDP-11 COBOL has the capability to manipulate data strings. It offers INSPECT, STRING, and UNSTRING—powerful COBOL verbs for character string handling—to search for embedded character strings, with tally and replace. Plus, they have the ability to join together or break out separate strings with various delimiters.

**On-Line Interactive COBOL Program Execution**
The Procedure Division ACCEPT and DISPLAY statements allow terminal-oriented interaction between a COBOL program and a user. Using these statements, a COBOL program can exercise interactive operation with a user running the program. This is useful, for example, in an order-entry application.

The ACCEPT statement allows the terminal user to enter input lines which the COBOL program can interpret. The ACCEPT statement also has a second format that allows it to retrieve the current date or time from the system.

The DISPLAY statement transfers data from a specified literal or data item to a specified device, normally the user's console. The statement can be modified by a special WITH NO ADVANCING phrase (without automatic appending of carriage return and line feed) that allows the COBOL program to control the format of the message sent. If the device
<table>
<thead>
<tr>
<th>NUCLEUS</th>
<th>TABLE HANDLING</th>
<th>FILE TYPES</th>
<th>SORT MERGE</th>
<th>INTER PROGRAM CALL &amp; CANCEL</th>
<th>DEBUG</th>
<th>COMMUNICATIONS</th>
<th>SEGMENT</th>
<th>LIBRARY</th>
<th>REPORT WRITER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SEQ</td>
<td>REL</td>
<td>INDEX'D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4-1** PDP-11 COBOL Implementation Levels
handler allows it, the WITH NO ADVANCING phrase will have the device
remain positioned on the same line and the same character position
following the last character displayed. This is especially useful when
typing prompting messages on the console.

While the ACCEPT and DISPLAY statements are primarily intended for
use with keyboard devices, PDP-11 COBOL allows the ACCEPT statement
to accept cards from a card reader, and the DISPLAY statement to dis-
play data on a line printer.

File Organization
Both the Sequential I/O and Relative I/O modules meet the full ANSI-74
high-level standards and include the following COBOL verbs:

- OPEN EXTEND
  Add records to a previously-created sequential file
  without recopying the file.

- DYNAMIC ACCESS
  Process relative file both randomly and sequentially
  in the same program.

- START
  Logical positioning within a relative file for subse-
  quent record retrieval.

- REWRITE/WRITE
  Logically replace a record in a mass storage file
  and generate a new record.

- CLOSE LOCK
  Protect a file from being opened second time by
  the currently processing program.

- LINAGE
  Specify logical page format.

Library Facility
With PDP-11 COBOL the user has a full ANSI-74 low-level Library facility,
plus high-level extensions. All frequently-used data descriptions and pro-
gram text sections can be held in library files available to all programs.
These files can then be copied into source programs to simultaneously
save unnecessary repetitions during program preparation and prevent
a common source of errors.

Debugging Features
With PDP-11 COBOL, programming need not be complicated by device
specifications because device assignments can be made at execution
time—a benefit for test and later production runs.

The VALUE OF ID clause gives an explicit file specification to the COBOL
OTS. The clause accepts either a literal entry or an identifier entry. The
entry (either literal or identifier) can be a complete or partial file speci-
fication. The following identifies a sample literal form of the clause,
using a complete RSX-11 file specification:

    VALUE OF ID IS "DP1:[200,200]PAYROL.MAS;2"

The identifier form of the VALUE OF ID clause is especially useful when
different runs of a program process different files. An ACCEPT statement
in the Procedure Division can request a file specification from the user
at a console or from a batch input stream. The following example illus-
trates how a COBOL program could request a file specification from an
interactive terminal:
DATA DIVISION

FD FILEIN
   VALUE OF ID IS GETNAM.

WORKING-STORAGE SECTION.
   77 GETNAM PIC X(40).
PROCEDURE DIVISION.

DISPLAY "ENTER INPUT FILE SPECIFICATION".
ACCEPT GETNAM.

OPEN INPUT FILEIN.

This sample causes the following interaction between the program and the user:

ENTER INPUT FILE SPECIFICATION
DK1:PAYROL.MAS

The OPEN INPUT statement in this case will open the file PAYROL.MAS on RK11 disk drive unit number one.

If the VALUE OF ID clause does not specify an explicit device name, the ASSIGN clause can assign a default file specification to the file. The ASSIGN clause is written as part of the SELECT statement, as shown below:

SELECT PAYROL.MAS ASSIGN TO "DK1:"

The example assigns the default device name DK1: for the location of the file PAYROL.MAS. Another device name specification in the VALUE OF ID clause can override the ASSIGN clause later in the source program.

To ease program debugging, the COBOL complier produces source language listings with embedded diagnostics. Fully descriptive diagnostic messages are listed at the point of error. Over 400 different error conditions are checked—varying from simple warnings to major error detections.

When the compiler detects an error in the source program, the compiler attempts to recover from an error and continue to compile the program. The kind of error message, whether informational, warning or fatal, indicates the likelihood that the assumption made to recover from the error will produce an object program that will run as the programmer intended. Normally, the COBOL compiler will not generate an object
program if major fatal errors are detected. The user can however, force
the compiler to produce an object program by requesting in the com-
mand string that it accept fatal errors. This facility is provided as an
extra debugging option. It can be useful in shortening the compile-debug
cycle, particularly if applied to large COBOL programs which take con-
siderable compilation time, but should be used with caution.

Debugging large source programs is made still easier by the use of the
of the optional Data Division allocation map and modular programming
techniques offered by the segmentation facility.

4.3 COMPILER IMPLEMENTATION AND OPERATION

PDP-11 COBOL is a compile-and-go system. This means that the normal
processing flow is from source program input to object program crea-
tion to object program execution. There is no intermediate step of link-
ing. The COBOL system can, of course, be requested to compile only,
or to execute a previously compiled object program.

Figure 4-2 illustrates the steps the system follows, from source program
creation to object program execution. The source program can be: 1)
typed on a terminal for immediate execution, 2) punched on cards in
the conventional format, or 3) typed on a terminal and created as a disk
file using the operating system's Editor. The user can then, at either an
interactive terminal or through a batch processor, run the COBOL com-
piler and compile the source program. The compiler reads the source
program, translates it into the intermediate code acceptable to the OTS,
produces all requested listings, and places an object program (if re-
quested) on disk. If the compilation is successful and the compiler was
not instructed otherwise, it loads and executes the object program.

The PDP-11 COBOL system can be envisioned as two separate com-
ponents: the compiler and the Object Time System (OTS). The PDP-11
COBOL compiler is an interpretive compiler. The object program the
compiler produces can be regarded as a data file that is processed by
the interpreter, called the Object Time System. In actual operation,
therefore, the COBOL system functions as a single task which has two
stages: compilation and run-time execution.

When COBOL is invoked, it is loaded into memory to occupy a task
image area that is between 23K words and .28K words long. The perma-
nently-resident portion of the task loads the compiler code if com-
ilation is requested and starts the compilation process. During the
compilation stage, the COBOL task creates an object program file on
disk and a temporary work file on disk, which can have a maximum size
of 256 (512-byte) blocks.

When compilation is complete, the permanently-resident portion of the
COBOL task loads object code into memory to start its execution (un-
less requested otherwise). The COBOL task overlays the compiler code
in its task area with: the I/O buffers required for I/O processing, the
Data Division file descriptors and internal working storage definitions,
and the executable object code. All available space in the task area pre-
viously occupied by the COBOL compiler code is overlaid with this in-
formation. The I/O buffers and Data Division information areas have a
fixed size based on the requirements of the program itself. Whatever
space remains after the I/O buffers and Data Division information requirements have been satisfied is designated as the area used for loading executable object code.

Figure 4-3 illustrates the task image area during the compile-time and run-time stages of the COBOL task’s execution. The work file created during compilation is deleted after compilation if the compiler is requested to compile only. If the COBOL task immediately executes the object program after it is created, the work file is not deleted; it is used during run-time. If the COBOL task is requested to execute an object program that was compiled earlier, it re-creates the work file before starting program execution.

Virtual Memory
The work file is used as a virtual extension of processor memory. In particular, the work file is used to extend that portion of the COBOL task area needed for handling data tables created and accessed during compilation and execution. An example of a table created during compilation is the recursive stack used to process Record Definitions.
The management of the data in the task area in memory and the work file on disk is handled by a special table management system called the Work File System. The Work File System allows the COBOL compiler to compile and execute much larger programs than would normally be expected of a minicomputer with limited available memory. The system makes an efficient use of processor memory by allowing working data to overflow to disk. In addition, it monitors the use of data kept in memory to minimize accesses made to disk, based on the activity of data in memory and on disk.

One important use of the Work File System is to enable the Object Time System to execute large COBOL programs. Since the COBOL task area used to contain executable statements varies on the size of I/O buffer and Data Division areas, it may often be small in relation to the size of the whole COBOL task area. Because of COBOL's interpretive nature and its use of the Work File System, object programs reside in main memory as strings of command blocks which are called into memory only as needed, so the user has the illusion of memory much larger than physical memory.

The actual movement of this information from main memory back and forth to disk is completely automatic. Areas of programs are transferred
through main memory. Only the currently active areas need to be in main memory. Using the Work File System, COBOL can easily fetch additional areas as required. For example, in order to execute a GO TO command it would be necessary to perform a random read. In most cases, the new procedure will already be in memory. If the procedure is not in memory, it will automatically be read in as if it had been there all along. Thus there is no need for the programmer to be concerned whether the object program fits in available memory or not. In fact, a very large program will run on a machine with little memory, but will be slowed in execution due to overhead from disk. Using a more conventional system, it would not be possible to run the program at all.

Flexibility
The PDP-11 COBOL compiler and run-time system has the flexibility to allow it to be built for various operating system configurations. Specifically, the size of the COBOL task can be increased or decreased depending on the amount of memory available on a particular configuration.

The size of the COBOL task image is a general determinant for the maximum size of a COBOL program, the speed at which it can compile, and the speed at which it can execute. In general, the larger the COBOL task, the faster it will run and the greater its capacity to execute very large COBOL programs.

When the compiler code or run-time code needs to use the disk work file, it obviously takes a longer time to run. By expanding the size of the internal work areas, the user can eliminate the need to access the work file so often. This means faster compilation and execution.

There are two internal work areas used by the COBOL task. The first area, WFBBLK, is used during compilation. The second area, WFEND, is used during object program execution. Changing the size of WFBBLK causes a variation in the speed of compilation; increasing the size significantly will speed up compilation.

Changing the size of WFEND will also cause variations in the speed of execution, though to a lesser degree. The speed of execution is largely dependent on the operations the COBOL program performs and less on the WFEND work space. A significant exception to this is when the COBOL program itself is very large—requiring not only a large amount of executable statements but also very large I/O buffers and Data Division areas. In this case, the object program when loaded by the COBOL run-time system may not fit in the space available in WFEND. The work area WFEND can be increased to allow the large program to execute.

The COBOL task size can vary from approximately 23K words to 28K words. The size variation is dependent on the size of the two work areas, WFBBLK and WFEND, selected by the system manager when building the COBOL task for the particular configuration on which it will execute.

The following table shows how the COBOL task size can be varied for different sizes of WFBBLK and WFEND. Size values for WFBBLK and WFEND are given in number of octal bytes. Note that the values for WFBBLK and WFEND are not intrinsically related. For example, WFBBLK can be built to its maximum size while WFEND remains at its smallest size. The table
simply shows the relative sizes if WFBLK and WFEND sizes are increased at the same rate.

<table>
<thead>
<tr>
<th>WFBLK</th>
<th>WFEND</th>
<th>COBOL Task Size (minimum)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>26000</td>
<td>23K</td>
</tr>
<tr>
<td>12000</td>
<td>32000</td>
<td>24K</td>
</tr>
<tr>
<td>16000</td>
<td>36000</td>
<td>25K</td>
</tr>
<tr>
<td>22000</td>
<td>42000</td>
<td>26K</td>
</tr>
<tr>
<td>26000</td>
<td>46000</td>
<td>27K</td>
</tr>
<tr>
<td>32000</td>
<td>52000</td>
<td>28K</td>
</tr>
<tr>
<td>32000</td>
<td>56000</td>
<td>28K</td>
</tr>
<tr>
<td>32000</td>
<td>62000</td>
<td>28K</td>
</tr>
<tr>
<td>32000</td>
<td>66000</td>
<td>28K</td>
</tr>
<tr>
<td>32000</td>
<td>72000</td>
<td>28K</td>
</tr>
</tbody>
</table>

4.4 COBOL OPERATING SYSTEM ENVIRONMENTS
PDP-11 COBOL is available under three operating systems: RSTS/E, RSX-11D and IAS. In general, PDP-11 COBOL is completely independent of the operating systems on which it is available. It uses the standard file system available under the operating system on which it runs. It does not use operating system directives or function calls. The commands used to invoke the compiler and run-time system are the same from system to system. The following paragraphs briefly state the implementation characteristics under each of the operating systems.

Under RSTS/E
Under RSTS/E, the COBOL task can run in either interactive or batch mode. The files that COBOL programs create can be read by BASIC-PLUS programs using ASCII sequential file processing techniques (INPUT, READ and PRINT statements).

Under RSX-11D and IAS
The PDP-11 COBOL compiler requires at least 23K words of memory to compile all elements of the COBOL language. The COBOL task runs, by default, in the general system controlled GEN partition. This partition should be at least 27K words on a 48K system. This allows other tasks to reside in the partition when COBOL is running.

If the partition is built large enough to hold only one copy of the COBOL task, only one COBOL program can be compiled at a time. The partition can be built large enough to hold as many copies of the COBOL task as needed to invoke the compiler more than once. The COBOL task can run shared when executing COBOL programs, but only if it is not running under control of the batch processor.

COBOL operates as any other language processor under RSX-11D or IAS with the exceptions that there is no directive usage and there can be no inter-task communication. COBOL does use the FCS (File Control System) and, therefore, files created using COBOL are compatible with all other RSX-11 or IAS files.

4.5 UTILITY PROGRAMS
PDP-11 COBOL offers three utility programs to aid the user with data processing.
SORT     Accepts simple parameters as descriptions of files to process.
RFRMT    Converts PDP-11 terminal format COBOL programs into conventional format ANS COBOL programs.
COBRG    A high-level language with efficient, commercially-oriented problem solving capabilities. With COBRG, vital reports can be developed quickly by saving tedious, time-consuming format coding.

The SORT program allows the user to:
• rearrange, delete and reformat records in a mass-storage file
• select a sorting process and input device that best suits the processing environment
• create new file indices to access a large data base
• batch stream several sorting jobs
• define sort specification files in a standard fixed format

The variety of file access methods allows construction of a general data base. Several access methods facilitate recognition of the individual records:
• consecutive records
• sequential by key
• sequential within key limits
• random by key
• random with ADDress ROUTing files
• random by relative record number

SORT enhances these file management facilities. The user arranges the data base for consecutive processing using the SORT program. SORT can also create several different index files for sequential access of a large data base. The user can then access data according to these indices or index files without reordering the data itself.

SORT provides four sorting techniques:

RECORD SORT—produces a reordered data file by manipulating all records in their entirety. The data can be available on any acceptable input device: cards, magnetic tape, DECtape or disk. The records can be variable or fixed length.

TAG SORT—produces a reordered data file by manipulating only the key fields used to order the records.

ADDRROUT SORT—produces an ordered address file that enables the user to maintain a single data file which can be accessed sequentially in the order of the address file.

INDEX SORT—produces an ordered key file that can be used to sequentially or randomly access a data file.

SORT accepts two kinds of command formats: a simple keyboard-oriented command string and a conventional specification file format. The
latter format features input record selection techniques based on user-defined record characteristics. In addition, the input records can vary in format and the output records can be restructured.

The RFRMT (Reformat) utility program reads COBOL source programs that were coded using terminal format and converts the source statements to 80-column conventional format. PDP-11 COBOL accepts source programs that are coded using either the conventional 80-column card reference format or the shorter, easy-to-enter terminal format.

- Terminal Format is designed for ease of use with context editors controlled from an on-line terminal keyboard, and is, therefore, compatible for use with PDP-11 systems. It eliminates the line-number and identification fields. It allows horizontal tab characters and short lines.
- Conventional Format produces source programs that are compatible with the reference format of other COBOL compilers throughout the industry.

RFRMT allows the programmer to enter source programs in the simpler terminal format and then, if compatibility is ever required for those programs, provides a simple method for conversion to conventional format.

The COBRG (COBOL Report Generator) utility program provides a fast, simple mechanism for producing printed reports from data files, COBRG recognizes input specification lines which enable the user to:

- define a report's page headings
- describe the format of input and output files
- set the rules for moving information from input records to detail output lines
- set the rules for adding values in accumulators
- set the rules for monitoring the sort keys for value changes
- set the rules for constructing and printing the accumulated values

COBRG uses these specification lines to produce a tailored COBOL source program. This program, when compiled and executed, generates the actual report.
Table 4-1 COBOL Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Words</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,1,...9 A,B,...Z</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Punctuation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>. ( ) space or blank</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>=</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>;</td>
<td></td>
<td>EXT</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ - * / **</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt; &gt; =</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Editing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 0 + - CR DB Z * $ .</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>/</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Separators</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>; and , not permitted</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>; and , are permitted</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COBOL WORDS (max 30 chars)</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>User-defined Words</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cd-name</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>condition name</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>data-name (1st char alpha)</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>data-name</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>file-name</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>index-name</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>level-number</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>library-name</td>
<td>2</td>
<td>LIB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>mnemonic-name</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>paragraph-name</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>program-name</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>record-name</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>report-name</td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>routine-name (optional)</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>section-name</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>segment-number</td>
<td>1</td>
<td>SEG</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>text-name</td>
<td>1</td>
<td>LIB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>System Names</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>computer-name</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>implementor-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>language-name (optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved Words</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>key words</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>optional words</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>qualifier connectives: OF IN</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>series connectives: , ;</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>logical connectives: AND, OR, AND NOT, OR NOT</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LINE-, PAGE-COUNTER registers</td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

4-13
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINAGE-COUNTER register</td>
<td>2</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>DEBUG-ITEM register</td>
<td>1</td>
<td>DEB</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TALLY register</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ZERO constant</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ZEROS, ZEROES constants</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SPACE constants</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SPACES contant</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HIGH-VALUE, LOW-VALUE</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HIGH-VALUES, LOW-VALUES</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>QUOTE constant</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>QUOTES constant</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ALL literal</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>arithmetic special chars</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>relational special chars</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>non-numeric literals (1-120 chars)</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>quote within non-numeric literals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>numeric literals (1-18 chars)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PICTURE strings</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>comment entries</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>No Qualification Permitted</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Qualification Permitted</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Subscripting to 3 Levels</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Indexing to 3 Levels</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**IDENTIFICATION DIVISION**

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM-ID</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INSTALLATION</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATE-WRITTEN</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATE-COMPILED</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SECURITY</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REMARKS</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**ENVIRONMENT DIVISION**

**Configuration Section**

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can be omitted</td>
<td></td>
<td>EXT</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SOURCE-COMPUTER</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WITH DEBUGGING MODE</td>
<td>1</td>
<td>DEB</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OBJECT-COMPUTER</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MEMORY SIZE</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COLLATING SEQUENCE</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>SEGMENT-LIMIT</td>
<td>2</td>
<td>SEG</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SPECIAL-NAMES</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ON STATUS</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OFF STATUS</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SPECIAL-NAMES series</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
## Table 4-1 (cont) COBOL Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANDARD-1 alphabet-name</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>NATIVE alphabet-name</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Implementor-name alphabet-name</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>literal alphabet-name</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CURRENCY SIGN</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DECIMAL-POINT</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### Input-Output Section

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE-CONTROL SELECT</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OPTIONAL</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ASSIGN TO implementor-name</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MULTIPLE REEL/UNIT</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RESERVE integer AREA(S)</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FILE-LIMIT literal THRU literal</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FILE-LIMIT literal series</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>data-name THRU data-name</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FILE-LIMIT data-name series</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SEQUENTIAL ORGANIZATION</td>
<td>1</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>RELATIVE ORGANIZATION</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>INDEXED ORGANIZATION</td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SEQUENTIAL ACCESS MODE</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RANDOM ACCESS MODE</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DYNAMIC ACCESS MODE</td>
<td>2</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PROCESSING MODE SEQ</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RECORDING MODE</td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RELATIVE KEY</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>RECORD KEY</td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
### Table 4-1 (cont) COBOL Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL</th>
<th>PDP-11 COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYMBOLIC KEY</td>
<td>EXT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>ALTERNATE RECORD KEY</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FILE STATUS</td>
<td>1</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>I-O CONTROL: RERUN</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SAME AREA</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SAME RECORD AREA</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SAME SORT-MERGE AREA</td>
<td>2</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SAME series</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MULTIPLE FILE TAPES</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>APPLY</td>
<td>EXT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

### DATA DIVISION

| Communication Section | 1     | COM    | -      | X      | X            | -            |
| File Section          | 1     | SEQ    | X      | X      | X            | X            |
|                       | 1     | REL    | -      | X      | -            | X            |
|                       | 1     | RAC    | X      | -      | X            | -            |
|                       | 1     | INX    | -      | X      | -            | -            |
|                       | 1     | SRT    | X      | X      | X            | -            |
|                       | 1     | RPW    | X      | X      | X            | -            |

| linkage section      | 1     | IPC    | -      | X      | X            | -            |
| working storage section | 1   | NUC    | X      | X      | X            | X            |
| report section        | 1     | RPW    | X      | X      | X            | -            |
| communications description entry | 1 | COM   | -      | X      | X            | -            |
| data description entry | 1   | NUC    | X      | X      | X            | X            |
| file description entry | 1   | SEQ    | X      | X      | X            | X            |
|                       | 1     | REL    | -      | X      | -            | X            |
|                       | 1     | RAC    | X      | -      | X            | -            |
|                       | 1     | INX    | -      | X      | X            | X            |
|                       | 1     | RPW    | X      | X      | X            | -            |
| record description entry | 1   | SEQ    | X      | X      | X            | X            |
|                       | 1     | REL    | -      | X      | X            | X            |
|                       | 1     | RAC    | X      | -      | X            | -            |
|                       | 1     | INX    | -      | X      | X            | -            |

4-16
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10</th>
<th>PDP-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINAGE clause</td>
<td>2</td>
<td>SEQ</td>
<td></td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>LINE NUMBER clause</td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>NEXT GROUP clause</td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>OCCURS clause</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer times</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ASCENDING/DESCENDING</td>
<td>2</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>data-name series</td>
<td>2</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INDEXED BY index-name</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>i1 TO i2 DEPENDING ON</td>
<td>2</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>PAGE clause</td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>PICTURE clause</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character string max 30 chars</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Data characters: A X 9</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Operational symbols: S V P</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fixed insertion characters:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 B .. $ + - CR DB</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fixed insertion character: /</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Replacement chars: $ + - Z *</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Currency sign substitution</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Decimal point substitution</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RECORD CONTAINS clause</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>REDEFINES clause (no nesting)</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>REDEFINES clause (nesting)</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RENAMES clause</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REPORT clause</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGN clause</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>SOURCE clause</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SUM clause</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SYNCHRONIZED clause</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TYPE clause</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>USAGE clause</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTATIONAL (means binary)</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COMP-1 (floating point)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DISPLAY-6 (SIXBIT)</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DISPLAY-7 (ASCII)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATABASE-KEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUE clause</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>literal</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>literal series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>literal THRU literal</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>.</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

4-17
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>literal range series VALUE OF clause</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>implementor-name IS literal</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>implementor-name IS data-name</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

**PROCEDURE DIVISION**

Using phrase                                          | 1     | IPC    | -      | X      | X              | -              |
Declaratives                                           | 1     | SEQ    | X      | X      | -              | X              |
|                                                       | 1     | REL    | -      | X      | -              | X              |
|                                                       | 1     | RAC    | X      | -      | X              | -              |
|                                                       | 1     | INX    | -      | X      | -              | -              |
|                                                       | 1     | RPW    | X      | X      | -              | -              |
|                                                       | 1     | DEB    | -      | X      | -              | -              |
Arithmetic expressions                                  | 2     | NUC    | X      | X      | X              | X              |
Conditional expressions                                | 1     | NUC    | X      | X      | X              | X              |
Simple conditions                                       | 1     | NUC    | X      | X      | X              | X              |
Relation conditions                                     | 1     | NUC    | X      | X      | X              | X              |
[NOT] GREATER THAN                                      | 1     | NUC    | X      | X      | X              | X              |
[NOT] >                                                  | 2     | NUC    | X      | X      | X              | X              |
[NOT] LESS THAN                                         | 1     | NUC    | X      | X      | X              | X              |
[NOT] <                                                  | 2     | NUC    | X      | X      | X              | X              |
[NOT] EQUAL TO                                          | 1     | NUC    | X      | X      | X              | X              |
[NOT] =                                                  | 2     | NUC    | X      | X      | X              | X              |
EQUALS                                                 | EXT   | -      | X      | -      | -              | -              |
numeric operands                                        | 1     | NUC    | X      | X      | X              | X              |
nonnumeric operands (equal size)                        | 1     | NUC    | X      | X      | -              | -              |
nonnumeric (may be unequal)                             | 2     | NUC    | X      | X      | X              | X              |
Class conditions                                        | 1     | NUC    | X      | X      | X              | X              |
NOT option                                              | 1     | NUC    | X      | X      | X              | X              |
Switch-status condition                                 | 1     | NUC    | X      | X      | X              | X              |
NOT option                                              | EXT   | -      | X      | -      | -              | -              |
Condition-name condition                                | 2     | NUC    | X      | X      | X              | X              |
NOT option                                              | EXT   | -      | X      | -      | -              | -              |
Sign condition                                          | 2     | NUC    | X      | X      | X              | X              |
NOT option                                              | 2     | NUC    | X      | X      | X              | X              |
Logical AND OR and NOT                                   | 2     | NUC    | X      | X      | X              | X              |
Negated simple conditions                               | 2     | NUC    | X      | X      | X              | X              |
Combined and negated combined                           | 2     | NUC    | X      | X      | X              | X              |
Abbreviated combined relation                           | 2     | NUC    | X      | X      | X              | X              |
### Table 4-1 (cont) COBOL Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL V6</th>
<th>PDP-11 COBOL V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic operands</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Overlapping operands</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multiple arithmetic results</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ACCEPT statement</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Only one transfer of data</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>No restrictions on transfers</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FROM</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FROM DATE</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>*</td>
<td>X</td>
</tr>
<tr>
<td>FROM DAY</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FROM TIME</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>*</td>
<td>X</td>
</tr>
<tr>
<td>MESSAGE COUNT</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>ADD statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>identifier literal series</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TO identifier</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TO identifier series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ROUNDED</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SIZE ERROR</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CORRESPONDING</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ALTER procedure-name</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ALTER procedure-name series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CALL literal</td>
<td>1</td>
<td>IPC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>CALL identifier</td>
<td>2</td>
<td>IPC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>CALL USING data-name</td>
<td>1</td>
<td>IPC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CALL ON OVERFLOW</td>
<td>2</td>
<td>IPC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CANCEL statement</td>
<td>2</td>
<td>IPC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CLOSE single file-name</td>
<td>1</td>
<td>SEQ</td>
<td>(_)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CLOSE file-name series</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REEL</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UNIT</td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>NO REWIND</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LOCK</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REEL</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UNIT</td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>NO REWIND                                    2</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>LOCK</td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>FOR REMOVAL</td>
<td>2</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>WITH DELETE</td>
<td></td>
<td>EXT</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>COMPUTE identifier series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>DELETE statement</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBM</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Language Elements</td>
<td>Level</td>
<td>Module</td>
<td>ANS-68</td>
<td>ANS-74</td>
<td>TOPS-10 COBOL V6</td>
<td>PDP-11 COBOL V2</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>DISABLE statement</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>INPUT</td>
<td>2 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TERMINAL INPUT</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>KEY identifier/literal</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DISPLAY statement</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>only one transfer of data</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>no restriction</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UPON</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WITH NO ADVANCING</td>
<td>EXT</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DIVIDE statement</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INTO identifier</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INTO identifier series</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>BY identifier</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier series</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ROUNDED</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SIZE ERROR</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ENABLE statement</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>INPUT</td>
<td>2 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TERMINAL INPUT</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>KEY identifier/literal</td>
<td>1 COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ENTER statement (optional)</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ENTRY statement</td>
<td>EXT</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EXAMINE statement</td>
<td>1 NUC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EXIT statement</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EXIT PROGRAM statement</td>
<td>1 IPC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FIND statement</td>
<td>1 IPC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GENERATE statement</td>
<td>1 RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GET statement</td>
<td>1 DBM</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GOBACK statement</td>
<td>1 EXT</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TO optional</td>
<td>1 NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>procedure-name required</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>procedure-name optional</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DEPENDING ON phrase</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IF statement</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>statements must be imperative</td>
<td>2 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>nested statements</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ELSE</td>
<td>DBM</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>INITIATE statement</td>
<td>1 RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>INSERT statement</td>
<td>1 NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

4-20
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL</th>
<th>PDP-11 COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSPECT statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>single character data item</td>
<td>1</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>multi-character data item</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>INVOKE statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MERGE statement</td>
<td>2</td>
<td>SRT</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MODIFY statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TO identifier</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TO identifier series</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CORRESPONDING</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MULTIPY statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BY identifier</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>BY identifier series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier series</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>ROUNDED</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SIZE ERROR</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>NOTE sentence</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OPEN statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT single file-name</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INPUT file-name series</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INPUT REVERSED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT NO REWIND</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OUTPUT single file-name</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OUTPUT file-name series</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OUTPUT NO REWIND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I-O single file-name</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>I-O file-name series</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EXTEND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT, OUTPUT, I-O, EXTEND</td>
<td>2</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>INPUT, OUTPUT and I-O series</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

4-21
## Table 4-1 (cont) COBOL Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 V6</th>
<th>PDP-11 V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERFORM statement</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>procedure name</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>THRU</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TIMES</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UNTIL</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>VARYING</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ statement</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>file-name</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>INVALID KEY</td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>INTO identifier</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AT END</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>NEXT</td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>KEY IS</td>
<td>2</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RECEIVE statement</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>2</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>INTO identifier</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>NO DATA phrase</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>RELEASE statement</td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>record name</td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>FROM</td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>REMOVE statement</td>
<td>1</td>
<td>DBM</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>RETURN statement</td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>file-name</td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>INTO</td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>AT END</td>
<td>1</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>REWRITE statement</td>
<td>1</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>FROM identifier</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>INVALID KEY phrase</td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>SEARCH statement</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SEEK statement</td>
<td>2</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>
## Table 4-1 (cont) COBOL Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>Level</th>
<th>Module</th>
<th>ANS-68</th>
<th>ANS-74</th>
<th>TOPS-10 COBOL</th>
<th>PDP-11 COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEND statement</td>
<td>2</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>FROM identifier</td>
<td>2</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>FROM identifier WITH</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>WITH identifier</td>
<td>2</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>WITH EGI</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>WITH EMI</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>BEFORE/AFTER ADVANCING</td>
<td>1</td>
<td>COM</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>SET statement</td>
<td>1</td>
<td>TBL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SORT statement</td>
<td>2</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>max 1 SORT STOP, I-O procedure</td>
<td>2</td>
<td>SRT</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>not limited to 1 SORT</td>
<td>2</td>
<td>SRT</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>COLLATING SEQUENCE phrase</td>
<td>2</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>START statement</td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>STOP statement</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>STORE statement</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>STRING statement</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SUBTRACT statement</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>identifier/literal series</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FROM</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FROM series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GIVING identifier series</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ROUNDED</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SIZE ERROR</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CORRESPONDING</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SUPPRESS statement</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TERMINATE statement</td>
<td>1</td>
<td>RPW</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TRACE statement</td>
<td>1</td>
<td>RPW</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UNSTRING statement</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>USE statement</td>
<td>2</td>
<td>NUC</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EXCEPTION/ERROR PROCEDURE</td>
<td>1</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ON fil-name/INPUT/OUTPUT/I-O</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>On file-name series</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>REL</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RAC</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>INX</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>ON EXTEND</td>
<td>2</td>
<td>SEQ</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>LABEL PROCEDURE</td>
<td>2</td>
<td>SEQ</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>BEFORE REPORTING</td>
<td>1</td>
<td>RPW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>USE FOR DEBUGGING statement</td>
<td>1</td>
<td>DEB</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>procedure-name</td>
<td>1</td>
<td>DEB</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>procedure-name series</td>
<td>1</td>
<td>DEB</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ALL PROCEDURES</td>
<td>1</td>
<td>DEB</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Language Elements</td>
<td>Level</td>
<td>Module</td>
<td>ANS-68</td>
<td>ANS-74</td>
<td>TOPS-10 COBOL V6</td>
<td>PDP-11 COBOL V2</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ALL REFERENCES OF identifier</td>
<td>2</td>
<td>DEB</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>file-name series</td>
<td>2</td>
<td>DEB</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cd-name series</td>
<td>2</td>
<td>DEB</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WRITE statement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>record-name</td>
<td>1</td>
<td>SEQ</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>FROM identifier</td>
<td>1</td>
<td>SEQ</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>BEFORE/AFTER ADVANCING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer LINES</td>
<td>1</td>
<td>SEQ</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>identifier LINES</td>
<td>2</td>
<td>SEQ</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>mnemonic-name</td>
<td>2</td>
<td>SEQ</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>PAGE</td>
<td>1</td>
<td>SEQ</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>AT END-OF-PAGE</td>
<td>2</td>
<td>SEQ</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>INVALID KEY</td>
<td>1</td>
<td>REL</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RAC</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>INX</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>SEGMENTATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>segment-number (priority-number)</td>
<td>1</td>
<td>SEG</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Fixed Memory Range 0-49</td>
<td>1</td>
<td>SEG</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Non-fixed Memory Range 50-99</td>
<td>1</td>
<td>SEG</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>SEGMENT-LIMIT</td>
<td>2</td>
<td>SEG</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>LIBRARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COPY</td>
<td>1</td>
<td>LIB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>text-name</td>
<td>1</td>
<td>LIB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>literal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>literal OF/IN LIBRARY</td>
<td>2</td>
<td>LIB</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>literal REPLACING</td>
<td>2</td>
<td>LIB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>used like a COBOL word</td>
<td>1</td>
<td>LIB</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>pseudo-text may be replaced</td>
<td>2</td>
<td>LIB</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>REFERENCE FORMAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence numbers</td>
<td>1</td>
<td>NUC</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>may be omitted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Area A</td>
<td>1</td>
<td>NUC</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Division header</td>
<td>1</td>
<td>NUC</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Section header</td>
<td>1</td>
<td>NUC</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Paragraph header</td>
<td>1</td>
<td>NUC</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Data Division entries</td>
<td>1</td>
<td>NUC</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Language Elements</td>
<td>Level</td>
<td>Module</td>
<td>ANS-68</td>
<td>ANS-74</td>
<td>TOPS-10 COBOL V6</td>
<td>PDP-11 COBOL V2</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Area B</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Paragraphs</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Data Division Entries</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Continuation Lines</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Nonnumeric literals</td>
<td>2</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Words and numeric literals</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Comments with *</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Comments with /</td>
<td>1</td>
<td>NUC</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

**Abbreviations:**

- **X** = feature implemented according to standard
- **-** = feature not implemented
- *** = feature available through a non-standard method**

<table>
<thead>
<tr>
<th>NUC</th>
<th>= Nucleus</th>
<th>RPW</th>
<th>= Report Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBL</td>
<td>= Table Handling</td>
<td>SEG</td>
<td>= Segmentation</td>
</tr>
<tr>
<td>SEQ</td>
<td>= Sequential I/O</td>
<td>LIB</td>
<td>= Library</td>
</tr>
<tr>
<td>REL</td>
<td>= Relative I/O</td>
<td>DEB</td>
<td>= Debugging</td>
</tr>
<tr>
<td>INX</td>
<td>= Indexed I/O</td>
<td>IPC</td>
<td>= Inter-program comm</td>
</tr>
<tr>
<td>SRT</td>
<td>= Sort/Merge</td>
<td>COM</td>
<td>= Communications</td>
</tr>
</tbody>
</table>
CHAPTER 5

BASIC

5.1 FUNCTION AND FEATURES
BASIC is an incremental compiler which provides immediate translation and storage of a program written in the BASIC source language while it is being entered. A single-user BASIC system is available as an option on the CAPS-11, RT-11, and IAS operating systems (running shared under IAS). A multi-user BASIC system is available as an option for the RT-11 operating system. *

BASIC provides the following features:

- incremental compiler for immediate source translation
- immediate mode for ease in debugging and use as a desk calculator
- ASCII sequential files compatible with FORTRAN
- integer, string and floating point virtual array files for random access
- dynamic allocation of string storage
- PRINT-USING statement for output formatting
- complete set of string manipulation functions
- user-defined functions
- programs chained together can pass data through common
- CALL statement for assembly language subroutines
- graphics and laboratory peripherals support

5.2 LANGUAGE
The BASIC language is a conversational programming language which uses simple English-type statements and familiar mathematical notation to perform operations. BASIC is one of the simplest computer languages to learn, and once learned, provides advanced techniques to perform intricate data manipulations and efficient problem expression.

A BASIC program is composed of lines of statements containing instructions to the BASIC compiler. Each line of the program begins with a number that identifies that line as a statement and indicates the order of statement execution relative to other lines in the program. Each statement starts with an English word specifying the type of operation to be performed.

* The RSTS/E system provides another version of BASIC called BASIC-PLUS. Refer to the RSTS/E chapter in Section II of this handbook for a description of the BASIC-PLUS system.
All BASIC statements and computations must be written on a single line. Statements cannot be continued on a following line. More than one statement, however, can be written on a single line when each statement after the first is preceded by a backslash. For example,

```
  10  INPUT A,B,C
```

is a single statement line, while

```
  20  LET X = 11  \  PRINT X,Y,Z  \  IF X = A THEN 10
```

is a multiple statement line containing three statements: LET, PRINT, and IF.

**Constants and Variables**

BASIC treats all numbers (real and integer) as decimal numbers. The advantage of treating all numbers as decimal numbers is that any number or symbol can be used in any mathematical expression without regard to its type. Numbers used must be in the approximate range $10^{-38}$ to $10^{+38}$.

In addition to real and integer formats, BASIC accepts exponential notation. Numeric data can be input in any one or all of these formats. BASIC automatically uses the most efficient format for printing a number, according to its size. It automatically suppresses leading and trailing zeros in integer and decimal numbers and formats all exponential numbers.

Both floating point and integer formats are used when storing and calculating numbers. If a number can be stored as an integer, it is handled as an integer unless the operation requires that it be stored as floating point.

BASIC also processes information in the form of strings. A string is a sequence of alphabetic, numeric or special characters treated as a unit. A string constant is a list of characters enclosed in quotes. A string constant can be used in the PRINT, CALL, and CHAIN statements. These usages of string constants are allowed in versions of BASIC that do not support strings.

In BASIC with string support, string constants can also be used to assign a value to a string variable, for example, in the LET and INPUT statements.

BASIC recognizes four types of variables: numeric and subscripted numeric, string and subscripted string. A numeric variable is an algebraic symbol representing a number and is formed by a single letter or letter optionally followed by a single digit. For example: I, B3, or X.

Subscripted variables provide additional computing capabilities for dealing with lists, tables, matrices, or any set of related variables. In BASIC, variables are allowed one or two subscripts. For example, a list might be described as A(I) where I goes from 0 to 5:

```
A(0), A(1), A(2), A(3), A(4), A(5).
```
This allows reference to each of the six elements in the list, and can be considered a one-dimensional algebraic matrix. Two-dimensional matrices are also allowed.

Any variable name followed by a dollar sign ($) character indicates a string variable. For example: A$, C7$. Any list or matrix variable name followed by the dollar sign character denotes the string form of that variable. For example: V$(n)$, M2$(n)$, C$(m,n)$, G1$(m,n)$.

The user can assign values to variables by indicating the values in a LET statement, by entering the value as data in an INPUT statement, or by a READ statement. The value assigned to variable does not change until the next time a statement is encountered that contains a new value for that variable.

**Operators**

BASIC performs addition, subtraction, multiplication, division and exponentiation. The five operators used in writing most familiar formulas are:

\[
\begin{align*}
+ & \quad A + B \quad \text{Add to } A \text{ and } A \\
- & \quad A - B \quad \text{Subtract } B \text{ from } A \\
\times & \quad A \times B \quad \text{Multiply } A \text{ by } B \\
/ & \quad A / B \quad \text{Divide } A \text{ by } B \\
\uparrow & \quad A \uparrow B \quad \text{Raise } A \text{ to the } B^{th} \text{ power}
\end{align*}
\]

In addition, BASIC allows unary plus and minus arithmetic operators. For strings, the concatenation operator (+ or &) puts one string after another without any intervening characters.

Relational operators allow comparison of two values and are used to compare arithmetic expressions or strings in an IF-THEN statement. The relational operators are:

\[
\begin{align*}
= & \quad \text{Equals (alphabetically equal)} \\
< & \quad \text{Less than (alphabetically precedes)} \\
\leq & \quad \text{Less than or equals (precedes or equals)} \\
> & \quad \text{Greater than (alphabetically follows)} \\
\geq & \quad \text{Greater than or equals (follows or equals)} \\
\ne & \quad \text{Not equals (not alphabetically equal)}
\end{align*}
\]

**Statements**

The following summary of BASIC statements gives a brief explanation of each statement's use.

**REM**

Contains explanatory comments in a BASIC program.

**LET =**

Assigns the value of an expression to the specified variable. Variable and expression must be of the same type.

**DIM**

Reserves space in memory for arrays according to the subscripts specified.

**DATA**

Used in conjunction with READ to input listed data into an executing program. Can contain any mixture of strings and numbers.
READ Assigns values listed in DATA statements to the specified values. Variables can be numeric or string.

OPEN FOR INPUT [OUTPUT] AS FILE Opens a file for input (or output) and associates the file with the specified logical unit number.

INPUT Reads data from the file associated with the logical unit specified or from the user's terminal. Variables can be arithmetic or string.

IF END Tests for an end-of-file condition on input sequential file associated with logical unit expression.

PRINT Prints the values of the specified expressions on the terminal or, when specified, to the file associated with the logical unit expression. The TAB function can also be included.

PRINT- USING Prints the values of the specified expression on the terminal or, when specified, to the file associated with logical unit expression in the format determined by the given string. Both numeric and string expressions can be used.

RESTORE Resets either the data pointer or, when specified, the input file associated with the given logical unit number to the beginning.

RESET Equivalent to RESTORE.

CLOSE Closes the file(s) associated with the logical unit number(s) and virtual file logical unit number(s) specified. If no logical unit number is specified, closes all open files.

NAME-TO Renames the specified file.

KILL Deletes the specified file.

RANDOMIZE Causes the random number generator (RND function) to produce different random numbers every time the program is run.

DEF FN Defines a user function.

CALL Used to call assembly language subroutines from a BASIC program.

FOR-TO Sets up a loop to be executed the specified number of times.

NEXT Placed at the end of the FOR loop to return control to the FOR statement.

IF Conditionally executes the specified statement or transfers control to the specified line number. When the condition is not satisfied, execution continues at the next sequential line. The expressions and the relational operator must all be string or all be numeric.

GOSUB Unconditionally transfers control to specified line of subroutine.
RETURN
Terminates a subroutine and returns control to the statement following the last executed GOSUB statement.

GO TO
Unconditionally transfers control to specified line number.

ON-GOSUB Conditionally transfers control to the subroutine at one line number specified in the list. The value of the expression determines the line number to which control is transferred.

ON-GO TO Conditionally transfers control to one line number in the specified list. The value of the expression determines the line number to which control is transferred.

CHAIN Terminates execution of the program, loads the program specified, and begins execution of the lowest line number or, when a line number is present in the statement, at the specified line number.

COMMON Preserves values and names of specified variables and arrays when the CHAIN statement is executed. Both string and arithmetic variables and arrays can be passed. The statement also dimensions the specified arrays.

END Placed at the end of the physical end of the program to terminate execution (optional).

STOP Terminates execution of the program. Placed at the logical end of the program.

Functions
BASIC provides a variety of functions to perform mathematical and string operations.

ARITHMETIC FUNCTIONS
ABS Returns the absolute value of an expression.
ATN Returns the arctangent as an angle in radians.
COS Returns the cosine of an expression in radians.
EXP Returns the value of the constant e (approx. 2.71828) raised to a given power (expression).
INT Returns the greatest integer less than or equal to a given expression.
LOG Returns the natural logarithm of an expression.
LOG10 Returns the base 10 logarithm of an expression.
PI Returns the value of pi (3.141593 approx.)
RND Returns a random number between 0 and 1.
SGN Returns value indicating the sign of an expression.
SIN Returns the sine of an expression in radians.
SQR Returns the square root of an expression.
TAB Causes the terminal print head to tab to column number given by an expression (valid only in PRINT).
SYS Special system function calls; control terminal I/O and perform special functions.
STRING FUNCTIONS
ASC      Returns the ASCII code in decimal for the 1-character string expression.
BIN      Converts a string expression containing a binary number to a decimal value.
CHR$     Generates a 1-character string whose ASCII value is the low-order 8 bits of the integer value of the given expression.
DAT$     Returns the date as a string.
LEN      Returns the number of characters in the given string.
OCT      Converts a string expression containing an octal number to a decimal value.
POS      Searches for and returns the position of the first occurrence of a substring in a string.
SEG$     Returns the string of characters in the given positions in the string.
STR$     Returns the string which represents the numeric value of the given expression.
TRM$     Returns the given string without trailing blanks.
VAL      Returns the value of the decimal number contained in the given string expression.

USER-DEFINED FUNCTIONS
In some programs it may be necessary to execute the same sequence of statements in several different places. BASIC allows definition of unique operations or expressions and the calling of these functions in the same way as, for example, the square root or trigonometric functions. Each function is defined once and can appear anywhere in the program.

A function definition consists of the function name, a dummy variable list (up to five), and an expression.

When the user-defined function is used in the program, the expressions in the argument list passed to the function will replace the dummy variables in the defining expression. Any variable in the defining expression that is not in the dummy variable list will have the value that the variable is currently assigned.

Programming Example
The POS function is used to find the position of a substring in a string. The POS function can be used to map a string of characters to a corresponding integer value which can be used for subsequent processing. This technique is called a table look-up. The table string is the first argument of the POS function and the string to be mapped is the second argument. For example:

```
LISTNH
10   REM PROGRAM TO TRANSLATE MONTH NAMES
15   REM TO NUMBERS
20   T$ = "JANFEBMARAPRMAJUNJULAUGSEPOTNOVDEC"
100  PRINT "TYPE THE FIRST 3 LETTERS OF A MONTH";
110  INPUT M$
120  IF LEN(M$) <> 3 GO TO 200
130  M = (POS(T$,M$,1) +2)/3
```

5-6
140 REM CHECK IF MONTH IS SPELLED CORRECTLY
150 IF M <> INT(M) GO TO 200
160 PRINT M$" IS MONTH NUMBER "M
170 GO TO 100
200 PRINT "INVALID ENTRY—TRY AGAIN" \ GO TO 100
READY
RUNNH
TYPE THE FIRST 3 LETTERS OF A MONTH? NOV
NOV IS MONTH NUMBER 11
TYPE THE FIRST 3 LETTERS OF A MONTH? DEC
DEC IS MONTH NUMBER 12
TYPE THE FIRST 3 LETTERS OF A MONTH? JUN
INVALID ENTRY—TRY AGAIN
TYPE THE FIRST 3 LETTERS OF A MONTH?↑C
STOP AT LINE 110
READY

Graphics and Laboratory Peripherals Support
BASIC provides graphics support for the VT11, GT42 and GT44 graphics
display systems. The support consists of a collection of routines ac-
cessible by the CALL statement.

Points, vector, text, and graph data can all be combined though simple
CALL statements. The screen can be scaled to any coordinates. Portions
of the display can be controlled independently through the use of the
subpicture feature. Special graphic routines allow the display of an
entire array of data by one call statement. The area of memory that is
allocated to the display buffer can be dynamically controlled.

Laboratory peripheral support for BASIC allows the user to use the com-
plete set of LPS11 hardware. LPS support enables the user to sample
and display in a real-time environment a variety of data from analog to
digital converters, digital I/O, and external events. Sampling is con-
trolled by crystal clocks and/or Schmitt triggers. It is possible to specify
such parameters as sampling rate and response time thus allowing
maximum flexibility.

Since BASIC is a higher-level language, even the novice programmer can
solve complex data acquisition problems with a minimum amount of
effort. All LPS routines are issued by the BASIC CALL statement allowing
the PDP-11 assembly language programmer to easily include or modify
the routines to meet particular or special requirements.

5.2.1 BASIC Files
Data is stored either in sequential files or in random access, virtual array
files. Sequential files are treated in the same way as terminal I/O; data
is read by an INPUT statement and written by a PRINT statement. Se-
quential files are useful for storing data that is processed serially.

Virtual array files are similar to arrays stored in memory. An element of
data in a virtual array can be part of any BASIC expression just the same
as an element of a normal array. An element of a virtual array file can
be assigned a value by a special form of the LET statement. Virtual array
files allow data to be accessed in a random, non-serial manner and are the only BASIC files in which existing data can be updated without re-writing the entire file.

There are three data types for virtual array files: integer, floating point, and string. A file can contain only one data type.

5.2.2 Creating, Modifying and Executing BASIC Programs
A BASIC program is entered in the system using the editing commands. Once a program has been entered, it can be saved, retrieved, listed or executed using the editing commands. When the BASIC system is running, it prints the message READY on the terminal to indicate that it is ready to accept an editing command.

The BASIC system's editor is a replacement editor. That is, an incorrect line is changed by entering a new line with the same line number as the incorrect line. The editor replaces the old line with the new line entered. A line can be deleted by typing its line number and a carriage return. Both the line number and the line are removed from the program. The following provides a summary of the BASIC editing commands.

**NEW**
- Clears the user area in memory and assigns a specified name to the current program. Used to create a new program.

**LIST**
- Types on the terminal the program currently in memory.

**LISTNH**
- A range of line numbers can be specified. If the command does not have the "NH" suffix, the program header is printed.

**RUN**
- If issued with no file specification, executes the program currently in memory. If a file specification is issued, clears the user area, reads a program in from the file, and executes the program. If the command does not have the "NH" suffix, the program header is printed.

**RUNNH**
- Clears the contents of the user array and string buffers. This command is used when a program has been executed and then edited. Before rerunning the program, the array and string buffers are cleared to provide more memory space.

**CLEAR**
- Changes the current program name to a specified name.

**SAVE**
- Copies the contents of the user area to a file, lists the contents on the line printer, or punches the contents on paper tape.

**OLD**
- Clears the user area and reads a program from a specified file into the user area in memory.

**REPLACE**
- Replaces the specified file with the program currently in memory.

**APPEND**
- Merges the program currently in memory with a program stored in a file. All lines in the program in memory that have duplicate line numbers with the program in the file are replaced by the lines from the program in the file.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH</td>
<td>Displays on the terminal the amount of storage required by the BASIC program currently in memory. This information is useful in determining the minimum user area in which a specific program can run.</td>
</tr>
<tr>
<td>SCR</td>
<td>Erases the user area in memory.</td>
</tr>
<tr>
<td>UNSAVE</td>
<td>Deletes the specified file.</td>
</tr>
<tr>
<td>BYE</td>
<td>If the BASIC system supports multiple users, terminates the session at the terminal.</td>
</tr>
</tbody>
</table>

In addition to the editing commands, the BASIC system recognizes the following special control characters:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL/C</td>
<td>Interrupts program execution and prints the READY message.</td>
</tr>
<tr>
<td>CTRL/O</td>
<td>Enables/disables console output.</td>
</tr>
<tr>
<td>CTRL/U</td>
<td>Deletes the current line being entered.</td>
</tr>
<tr>
<td>RUBOUT</td>
<td>Deletes the last character typed.</td>
</tr>
</tbody>
</table>

### 5.3 COMPILER OPERATION

When the user enters a BASIC program, the BASIC system does not store the program exactly as it is typed or read from the input file. Instead, it translates the program into an intermediate form which can be used in two different ways. The intermediate code can be returned to its initial form by the LIST or SAVE commands to produce an ASCII program which looks like the input program, or the translated code can be quickly interpreted by the RUN command to execute a program under the operating system.

**Immediate Mode of Execution**

It is not necessary to write a complete program to use BASIC. Almost any BASIC statement can be executed in immediate mode. This latter facility makes BASIC an extremely powerful calculator.

BASIC distinguishes between those lines entered for immediate execution and those entered for later execution by the presence or absence of a line number. Statements which begin with line numbers are stored; those without line numbers are executed immediately.

Immediate mode operation is especially useful for program debugging and desk calculation problems.

To facilitate debugging a program, STOP statements can be placed throughout the program. When the program is run, each STOP statement causes the program to halt. The data values can then be examined and modified in immediate mode. The immediate mode statement

\[
\text{GO TO line number}
\]

is used to continue program execution. The values assigned to variables when the RUN command was issued remain intact until a SCRatch, CLEAR or another RUN command is issued.

If the STOP statement occurs in the middle of a FOR loop, modifications can not be made to the section of the program which precedes the FOR statement.
If CTRL/C is used to halt program execution, the GO TO command can be used to continue execution at the line where execution stopped.

When using immediate mode, nearly all the standard statements can be used to generate or print immediate mode results. Multiple statements can be used on a single line in immediate mode. For example:

\[ \text{A=1 \ PRINT A} \]
\[ 1 \]

Program loops in immediate mode are allowed in multiple statement lines. Thus a table of square roots can be produced as follows:

\[ \text{FOR I=1 TO 10 \ PRINT I,SQR(I) \ NEXT I} \]
\[ 1 \ 1 \]
\[ 2 \ 1.41421 \]
\[ 3 \ 1.73205 \]
\[ 4 \ 2 \]
\[ 5 \ 2.23607 \]
\[ 6 \ 2.44949 \]
\[ 7 \ 2.64575 \]
\[ 8 \ 2.82843 \]
\[ 9 \ 3 \]
\[ 10 \ 3.16228 \]

Certain statements, while not illegal, make no sense when used in immediate mode, such as COMMON, DEF, DIM, DATA and RANDOMIZE. The INPUT statement is illegal in immediate mode. Also, function references in immediate mode are illegal unless the program containing the definition was previously executed.

User Area Allocation and Program Size

The BASIC system stores each user's program in memory in the following format:

<table>
<thead>
<tr>
<th>Arrays</th>
<th>Buffers</th>
<th>Strings</th>
<th>Symbol Table</th>
<th>User Code</th>
</tr>
</thead>
</table>

The symbol table and user code area are created when the program is entered. When the RUN command is issued, the user program is scanned and arrays are set up. The string area is created during program execution.

The SCRatch command clears all the user code, symbol table, strings, and arrays from memory. The CLEAR command clears the arrays and strings but does not affect the user code or symbol table.

A symbol table entry is created for each distinct line number or variable name referenced in the program. These entries are not deleted, how-
ever, even when all references in the program to a particular line number or variable are removed. Thus, if the program in memory is heavily modified, it may be desirable to save it with the SAVE command and then restore the program with the OLD command to obtain the largest possible user area.

The LENGTH command displays on the terminal the amount of storage required by the BASIC program in memory. This information is useful in determining the minimum user area in which a specific program can run.

LENGTH prints the number of words used and the number of words remaining free in the user’s area. The LENGTH command always returns the current memory requirements; they may differ depending on when the command is issued. The number of words in use includes memory currently needed by the BASIC program itself, arrays, string variables, and file buffers in the user area. To determine the size of the program alone, issue the LENGTH command immediately after an OLD or CLEAR command. Arrays are created after the RUN command is issued and file buffers are created when the OPEN statement is executed. The memory required for string variables and string arrays varies with the current values of the strings.

5.4 BASIC ENVIRONMENTS

BASIC is available on the CAPS-11, RT-11 and IAS systems. Two versions are available for the RT-11 operating system: the standard single-user version available on CAPS-11 and IAS, and a special multi-user version.

The single-user versions of BASIC available on CAPS-11 and RT-11 provide graphics and laboratory peripherals support. BASIC with Laboratory Peripheral System support requires at least 16K words of memory. The hardware required for use of the BASIC graphics support includes a GT44 system (RT-11) or a VT11 display screen, 16K or more words of memory, and a user’s terminal (CAPS-11). In addition to the peripheral I/O device needed to support the BASIC system (disk, DECTape, cassette or paper tape), the use of the timer routines require a real-time clock. The memory required for the graphics support itself is approximately 2.5K words.

The following paragraphs discuss the features and capabilities of the BASIC versions available. Table 5-1 compares the features and language elements of each version.

Under CAPS-11

BASIC/CAPS runs on any PDP-11 system with 8K or more words of memory, a dual cassette drive and console terminal. BASIC/CAPS also supports the following optional hardware: a line printer, high-speed paper tape reader/punch, a low-speed paper tape reader/punch, special arithmetic hardware (EAE, FPU, 11/40 extended processor and 11/45 processor), and up to 28K words of memory. For systems with more than 8K words of memory, support is also provided for the Laboratory Peripheral System and the GT42 Display Processor (VT11).

The BASIC/CAPS system is provided in two versions. One version is for systems with more than 8K words of memory, and the other version is

5-11
for systems with only 8K words of memory. The 8K version employs overlaying of the BASIC system from cassette and does not support string variables or the PRINT USING statement. The greater than 8K version is a non-overlaying BASIC and supports string variables and the PRINT USING statement.

**Under RT-11: Single-user Version**

BASIC/RT-11 interfaces with the RT-11 monitor. BASIC is loaded under control of the monitor by typing the "R BASIC" command. Users can access any RT-11 supported device, including disk, DECTape, cassette, magnetic tape, card reader, paper tape reader/punch and floppy disk. BASIC/RT-11 files can be processed by FORTRAN IV/RT-11. At least 8K words of memory are required to run BASIC. In systems with more than 8K words of memory, BASIC/RT-11 provides alphanumeric character string I/O and string variable support.

**Under RT-11: Multi-user version**

MU BASIC/RT-11 is a multi-user BASIC system, capable of accommodating up to eight users simultaneously. Each user independently creates and executes BASIC programs. All of the features of MU BASIC, including statements, commands, functions and immediate mode execution are available to all users.

MU BASIC runs under the RT-11 monitor. Users can access the following devices supported by RT-11: disk, DECTape, magtape, cassette, card reader, paper tape reader/punch, and line printer.

Up to eight users can be supported on Single-job systems with at least 24K words of memory. Up to four users can be supported on Single-job systems with at least 16K words of memory, or on Foreground/Background systems with at least 28K words of memory.

To accommodate multiple users, MU BASIC provides a scheduling supervisor and terminal handler. In addition, the system provides a log on procedure and file protection as options.

The log on procedure requires that users obtain a user ID and password from the system manager to gain access to the system. The HELLO and BYE commands are used in this case to log on and log off the system.

The file protection system provides several degrees of file access. There are three classes of files: public, group, and private. Public library files are available to all users. Group library files are accessible to all users have the same first character in their user ID. A private file is accessible only to the user who created it.

If file protection is desired, a file can be given any of the following access characteristics:

- **Run** Allows access by the RUN command or CHAIN statement.
- **Read** Allows access by the OLD or APPEND command or the OPEN FOR INPUT or OVERLAY statement or use of the value of an element of a virtual array.
- **Update** Allows a virtual array file to be updated.
Complete Allows access by all of the above and by the SAVE, REPLACE or UNSAVE command or the OPEN FOR OUTPUT, NAME TO or KILL statement.

A nonprivileged user is allowed complete access only to the user's own private files. A nonprivileged user can have Run and Read access to files in the public library and the user's own group library. Nonprivileged users are not allowed access to other user's private library files or other group's files. The access allowed a nonprivileged user to all files other than the user's own files can be modified by the inclusion of a protection code in the filename.

A privileged user has complete access to all files. Group library and public library files can only be created by a privileged user.

In addition to the log on procedure and file protection, MU BASIC includes the following commands:

HELLO Allows the user to log on to the system (optional).
ASSIGN Assigns a specified device to a user if it is available.
DEASSIGN Deassigns a specified device.
TAPE Disables echoing for the low-speed paper tape reader.
KEY Enables echoing after the TAPE command (or a disable echoing system function call).

SET TTY Sets the system to allow different terminal characteristics.

MU BASIC provides a comprehensive set of system functions. Certain system functions are available to all users. These functions enable the programmer to cancel CTRL/O typed at a terminal, disable/enable echoing, enter single character input mode, scratch the user area in memory and return to the READY message, and return the current user's ID. Certain other system function calls can be executed only by a privileged user. These functions include the ability to disable the CTRL/C interrupt, set the user ID, terminate the privileged user status, and cause BASIC to exit and return control to the RT-11 monitor.

The single character input mode system function call is useful for special read operations. It returns the decimal ASCII value of the next character input from the terminal or a file. It is the only method for BASIC programs to process terminal input without waiting for a carriage return to be typed. This allows interactive programs to use single character response and not require a carriage return.

Single character input mode allows data in any file to be read with no need for separating commas or carriage returns. Binary files can be copied exactly.

Under IAS

BASIC under IAS is treated as a shared single-user system. That is, a single copy of the pure area of BASIC can be shared by several users, but each user has his own copy of the impure BASIC code.

The minimum requirement for the BASIC system is a 12K word partition. The partition can be expanded up to 28K words. BASIC/IAS can access any device supported by the IAS file system.
Table 5-1 BASIC Language Implementations

<table>
<thead>
<tr>
<th>Language Elements</th>
<th>BASIC CAPS</th>
<th>BASIC RT</th>
<th>MU BASIC RT</th>
<th>IAS BASIC</th>
<th>BASIC PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FUNCTIONS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mathematical (ABS, SIN, COS, SGN, LOG, EXP, ATN, INT, RND, SQR)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>String LEFT, RIGHT, MIDdle LENght</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>String LEN, SEG</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>String search for substring</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>SPACE$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TRMS$ (delete trailing spaces)</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>ASCII code of character</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Character equivalent to ASCII code (CHR$)</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Date</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Time</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Numeric string conversion</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>Octal and binary functions (OCT and BIN)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Common log (LOG10)</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TAB for print positioning</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X&lt;sup&gt;2&lt;/sup&gt;</td>
<td>X&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>Assembly language routines</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X&lt;sup&gt;2&lt;/sup&gt;</td>
<td>-</td>
</tr>
<tr>
<td><strong>PROGRAM LINES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple statement lines</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multiple line statements</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Assignment</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LET</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Word LET optional</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multiple assignment</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IF-THEN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>IF-GOTO</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>IF-THEN-ELSE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FOR, NEXT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>FOR-WHILE, FOR-UNTIL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ON-GOTO</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CHAIN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Subroutines and Functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOSUB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RETURN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ON-GOSUB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DEF, single line</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DEF, multiple line</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Error trapping</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ON ERROR GOTO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>RESUME</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

<sup>1</sup> Not in 8K word systems.
<sup>2</sup> Not in version 1.
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>BASIC CAPS</th>
<th>BASIC RT</th>
<th>MU BASIC RT</th>
<th>IAS BASIC</th>
<th>BASIC PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAT READ</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>MAT INPUT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>MAT PRINT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>MAT let</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Modifiers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF, UNLESS, FOR, WHILE, UNTIL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Record I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>PUT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>FIELD</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>LSET, RSET</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RESTORE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INPUT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INPUT LINE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>LINPUT</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PRINT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PRINT USING</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Files</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPEN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CLOSE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>NAME-AS</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>NAME-TO</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>KILL</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>IF END</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Virtual files</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DIM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RANDOMIZE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COMMON</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHANGE string to/from array</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>STOP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>END</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>END statement optional</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SLEEP</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>WAIT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

\(^1\) Not in version 1.
<table>
<thead>
<tr>
<th>Language Elements</th>
<th>BASIC CAPS</th>
<th>BASIC RT</th>
<th>MU BASIC RT</th>
<th>IAS BASIC</th>
<th>BASIC PLUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>RENAME</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LIST</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LENGTH</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Program storage/retrieval</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OLD</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SAVE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REPLACE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UNSAVE</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>APPEND</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>COMPILE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>CATALOG</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td>-</td>
<td>-</td>
<td>*</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>TAPE</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>RUN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HELLO</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>BYE</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SCALE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Immediate Mode</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

5-16
appendices
APPENDIX A

GLOSSARY

Absolute address
A binary number that is assigned as the address of a physical memory storage location.

Absolute loader
A stand-alone program which, when in memory, enables the user to load into memory data in absolute binary format.

Account number
A discrete code used to identify a system user. It normally consists of two numbers, separated by a comma, called the project number and programmer number or the group number and member number. See also User Identification Code.

Active Task List
A priority-ordered list of active tasks used normally in an event-driven multiprogrammed system to determine the order in which tasks receive control of the CPU.

Address
A label, name or number that designates a location where information is stored.

Alphanumeric
Referring either to the entire set of 128 ASCII characters or the subset of ASCII characters which includes the 26 alphabetic characters and the ten numeric characters.

Ancillary peripherals
In the MUMPS-11 system, peripherals not under control of the data base supervisor.

Append
To add information to the end of an existing file.

Application program
A program that performs a task specific to a particular end-user's needs. Generally, an application program is any program written on a program development operating system that is not of the basic operating system.

Array
An ordered arrangement of subscripted variables.

ASCII
The American Standard Code for Information Interchange, consisting of
128 7-bit binary codes for upper and lower case letters, numbers, punctuation and special communication control characters.

**Assembler**
A program that translates symbolic source code into machine instructions by replacing symbolic operation codes with binary operation codes and symbolic addresses with absolute or relocatable addresses.

**Assembler directives**
The mnemonics used in an assembly language source program that are recognized by the assembler as commands to control and direct the assembly process.

**Assembly language**
A symbolic programming language that can normally be translated directly into machine language instructions and is, therefore, specific to a given computing system.

**Assembly listing**
A listing produced by an assembler that shows the symbolic code written by a programmer next to a representation of the actual machine instructions generated.

**Assigning a device**
Putting an I/O device under control of a particular user's job either for the duration of the job or until the user relinquishes control. See also Attach.

**Asynchronous**
A mode of operation in which an operation is started by a signal that the operation on which it depends is completed. When referring to hardware devices, it is the method in which each character is sent with its own synchronizing information. The hardware operations are scheduled by ready and done signals rather than by time intervals. In addition, it implies that a second operation can begin before the first operation is completed.

**Asynchronous System Trap**
A system condition which occurs as the result of an external event such as completion of an I/O request. On occurrence of the significant event, control passes to an AST service routine.

**Attach**
To dedicate a physical device unit for exclusive use by the task requesting attachment. See also Assigning a device.

**Backup file**
A copy of a file created for protection in case the primary file is unintentionally destroyed.

**Bad block**
A defective block on a storage medium that produces a hardware error when attempting to read or write data in that block.

**Base address**
An address used as the basis for computing the value of some other relative address.
Base segment
The portion of a program using overlays that is always memory-resident. See also Root segment.

Batch processing
A processing method in which programs are run consecutively without operator intervention.

Batch stream
The collection of commands and data interpreted by a batch processor that direct batch processing.

Binary
The number system with a radix of two.

Binary code
A code that uses two distinct characters, usually the numbers 0 and 1.

Binary loader
See Absolute loader.

Bit
A binary digit.

Bit map
A table describing the state of each member of a related set. A bit map is most often used to describe the allocation of storage space. Each bit in the table indicates whether a particular block in the storage medium is occupied or free.

Block
A group of physically adjacent words or bytes of a specified size which is particular to a device. The smallest system-addressable segment on a mass-storage device in reference to I/O.

Boolean Valued Expression
An expression which, when evaluated, produces either a True or False result.

Bootstrap
A technique or device designed to bring itself into a desired state by its own action.

Bootstrap loader
A routine whose first instructions are sufficient to load the remainder of itself into memory from an input device and normally start a complex system of programs.

Bottom address
The lowest memory address in which a program is loaded.

Breakpoint
A location at which program operation is suspended in order to examine partial results. A preset point in a program where control passes to a debugging routine.

Buffer
A storage area used to temporarily hold information being transferred
between two devices or between a device and memory. A buffer is often a special register or a designated area of memory.

**Bug**
An instruction or sequence of instructions in a program that causes unexpected and undesired results.

**Byte**
The smallest memory-addressable unit of information in a PDP-11 system. A byte is equivalent to eight bits.

**Carriage return key**
The key on a terminal keyboard most often used in PDP-11 systems to terminate input lines.

**Central Processing Unit or Central Processor**
That part of a computing system containing the arithmetic and logical units, instruction control unit, timing generators and memory and I/O interfaces.

**Character**
A single letter, numeral, or symbol used to represent information.

**Checksum**
A number used for checking the validity of data transfers.

**Clock**
A time-keeping or frequency-measuring device within a computing system.

**Code**
A system of symbols and rules used for representing information.

**Coding**
To write instructions for a computer using symbols meaningful to the computer itself, or to an assembler, compiler or other language processor.

**Collate**
To combine items from two or more ordered sets into one set having an order not necessarily the same as any of the original sets.

**Command or Command Name**
A word, mnemonic or character, which, by virtue of its syntax in a line of input, causes a predefined operation to be performed by a computer system.

**Command language**
The vocabulary used by a program or set of programs that directs the computer system to perform predefined operations.

**Command Language Interpreter**
The program that translates a predefined set of commands into instructions that a computer system can interpret.

**Command string**
A line of input to a computer system that generally includes a command, one or more file specifications, and optional qualifiers.
Command String Interpreter
A special program or routine that accepts a line of ASCII string input and interprets the string as input and output file specifications with recognized qualifiers.

Common
A section in memory which is set aside for common use by many separate programs or modules.

Compile
To produce binary code from symbolic instructions written in a high-level source language.

Compiler
A program which translates a high-level source language into a language suitable for a particular machine.

Completion routine
A routine that is called at the completion of an operation.

Compute-bound
A state of program execution in which all operations are dependent on the activity of the central processor; for example, when a large number of calculations are being performed. See also I/O bound.

Computer operator
A person who performs standard system operations such as adjusting system operation parameters at the system console, loading a tape transport, placing cards in a card reader, and removing listings from the line printer.

Concatenate
To combine several files into one file, or several strings of characters into one string, by appending each file or string one after the other.

Conditional assembly
The assembly of certain parts of a symbolic program only when certain conditions are met.

Configuration
A particular selection of hardware devices or software routines or programs that function together.

Consecutive access
The method of data access characterized by the sequential nature of the I/O device involved. For example, a card reader is an example of a consecutive access device; each card must be read one after another, and no distinction is made between logical sets of data in or among the cards in the input hopper.

Console
The console of a central processor is the set of switches and display lights used by an operator or programmer to determine the status and control the operation of the computer.

Console terminal
A keyboard terminal which acts as the primary interface between the
computer operator and the computer system and is used to initiate and
direct overall system operation through software running on the com-
puter.

**Constant**
A value which remains the same throughout a distinct operation. Com-
pare with Variable.

**Context switching**
The switching between one mode of execution and other, involving the
saving of key registers and other memory areas prior to switching be-
tween jobs, and restoring them when switching back. A common example
of context switching is the temporary suspension of a user program so
that the monitor or executive can execute an operation.

**Contiguous file**
A file consisting of physically adjacent blocks on a mass-storage device.

**Control character**
A character whose purpose is to control an action rather than to pass
data to a program. An ASCII control character has an octal code between
0 and 37. It is typed by holding down the CTRL key on a terminal key-
board while striking a character key.

**Control section**
A named, contiguous unit of code (instructions or data) that is con-
sidered an entity and that can be relocated separately without destroying
the logic of the program.

**Core memory**
The most common form of main memory storage used by the central
processing unit, in which binary data is represented by the switching
polarity of magnetic cores.

**Core common**
See Common.

**Crash**
A hardware crash is the complete failure of a particular device, some-
times affecting the operation of an entire computer system. A software
-crash is the complete failure of an operating system characterized by
some failure in the system's protection mechanisms. In actual occur-
rence, it is a system-level trap, e.g., trap to location 4 or 10 (attempt to
execute an illegal instruction, parity error, etc.) when the system's trap
routines have been destroyed.

**Create**
To open, write data to, and close a file for the first time.

**Cross reference listing or table**
A printed listing that identifies all references in a program to each spe-
cific label in a program. A list of all or a subset of symbols is used in a
source program and statements where they are defined or used.

**CTRL/C**
The control character issued from a terminal which is most commonly
used to return the operator to communication with the system-level program. In most PDP-11 systems, it is typed on the terminal keyboard to gain the attention of the operating system before commencing the login procedure, or to terminate the currently-executing program and return to communication with the monitor. In some cases, it simply issues a call to the console listener or console service routine without interrupting current program execution.

CTRL/U
The control character issued from a terminal that tells the program currently accepting input to ignore the characters entered on the line up to the point where CTRL/U was typed.

CTRL/Z
Used in RSX-11 systems to terminate the system program currently waiting for input from the terminal. It is essentially an end-of-file character.

Data base
A collection of interrelated data items organized by a consistent scheme that allows one or more applications to process the items without regard to physical storage locations.

Data base management system
A scheme used to create, maintain and reference a data base.

Debug
To detect, locate, and correct coding or logic errors in a computer program.

DECTape
A convenient pocket-sized reel of magnetic tape developed by DIGITAL for extremely reliable data storage and random access.

Default
The value of an argument, operand or field assumed by a program if a specific assignment is not supplied by the user.

Delimiter
A character that separates, terminates or organizes elements of a character string, statement or program.

Detach a device
Free an attached physical device unit for use by tasks other than the one that attached it.

Device
A hardware unit such as an I/O peripheral, e.g., magnetic tape drive, card reader, etc. Also often used synonymously with volume.

Device controller
A hardware unit which electronically supervises one or more of the same type of devices. It acts as the link between the CPU and the I/O devices.

Device driver
A program that controls the physical hardware activities on a peripheral
device. The device driver is generally the device-dependent interface be-
tween a device and the common, device-independent I/O code in an
operating system.

Device handler
A program that drives or services an I/O device. A device handler is
similar to a device driver, but provides more control and interfacing func-
tions than a device driver.

Device name
A unique name that identifies each device unit on a system. It usually
consists of a 2-character device mnemonic followed by an optional device
unit number and a colon. For example, the common device name for
DECTape drive unit one is “DT1:”.

Device unit
One of a set of similar peripheral devices, e.g., disk unit 0, DECTape unit
1, etc. Also used synonymously with volume.

Direct access
See Random access.

Direct Mode
The mode of MUMPS-11 system operation which enables the programmer
to: enter commands and or functions for immediate execution, and create
or modify steps of a user’s program.

Directive
A type of executive request issued by a program that provides a facility
inherent in the hardware which is controlled and organized by the oper-
ating system. See also Programmed request.

Directory
A table that contains the names of and pointers to files on a mass-
storage device.

Directory device
A mass-storage retrieval device, such as disk or DECTape, that contains
a directory of the files stored on the device.

Double-buffered I/O
An input or output operation which uses two buffers to transfer data.
While one buffer is being used by the program, the other buffer is being
read from or written to by an I/O device.

Executive
The controlling program or set of routines in an operating system. The
executive coordinates all activities in the system including I/O supervi-
sion, resource allocation, program execution, and operator communica-
tion. See also Monitor.

Executive mode
A central processor mode characterized by the lack of memory protection
and relocation and by the normal execution of all defined instruction codes.
Exponentiation
A mathematical operation denoting increases in the base number by a factor previously selected.

Expression
A combination of operands and operators which can be evaluated to a distinct result by a computing system.

External storage
A storage medium other than main memory.

File
A logical collection of data treated as a unit which occupies one or more blocks on a mass-storage device such as disk, DE Ctape, or mag tape. A file can be referenced by a logical name.

File gap
A fixed length of blank tape separating files on a magnetic tape volume.

Filename
The alphanumer  ic character string assigned by a user to identify a file, and which can be read by both an operating system and a user. A filename identifies a unique member of a group of files which: 1) has the same filename extension and version number (if any), 2) is located on the same volume, and 3) belongs in the same User File Directory (if any). A filename has a fixed maximum length which is system dependent ( generally six or nine characters).

Filename extension
The alphanumer  ic character string assigned to a file either by an operating system or a user, and which can be read by both the operating system and the user. System-recognizable filename extensions are used to identify files having the same format or type. If present in a file specification, a filename extension follows the filename in a file specification, separated from the filename by a period. A filename extension has a fixed maximum length which is system dependent ( generally three characters, excluding the preceding period).

File specification
A name that uniquely identifies a file maintained in any operating system. A file specification generally consists of at least three components: a device name identifying the volume on which the file is stored, a filename, and a filename extension. In addition, depending on the system, a file specification can include a User File Directory name or UIC, and a version number.

File structure
A method of recording and cataloging files on mass-storage media.

File-structured device
A device on which data is organized into files. The device usually contains a directory of the files stored on the device.

File type
See Filename extension.
Floating point numeric
A floating point number which, if stored in four words, is approximately in the range $10^{-38}$ to $10^{+38}$.

Foreground
The area in memory designated for use by a high-priority program. The program, set of programs, or functions that gain the use of machine facilities immediately upon request.

Format
The arrangement of the elements comprising any field, record, file or volume.

Formatted ASCII
Refers to a mode in which data is transferred. A file containing formatted ASCII data is generally transferred as strings of 7-bit ASCII characters (bit eight is zero) terminated by a line feed, form feed or vertical tab. Special characters, such as null, RUBOUT and tab may be interpreted specially.

Formatted binary
Refers to a mode in which data is transferred. Formatted binary is used to transfer checksummed binary data (8-bit characters) in blocks. Formatting characters are start of block indicators, byte count and checksum values.

Formatted device
A volume which has been prepared for use on a system under program control.

Function
An algorithm accessible by name and contained in the system software which performs commonly-used operations. For example, the square root calculation function.

Generation number
See Version number.

Global
A value defined in one program module and used in others. Globals are often referred to as entry points in the module in which they are defined, and externals in the other modules which use them. Also, in the MUMPS-11 system, a global array.

Global Array
A data file stored in the common MUMPS-11 data base. Global arrays comprise an external system of symbolically referenced arrays.

Global Variable
A global variable in the MUMPS-11 system is a subscripted variable which forms a part (or node) of a global array.

Handler
See device handler.

Hardware
The physical equipment components of a computer system.
High-level language
A programming language whose statements are translated into more than one machine language instruction. Examples are BASIC, FORTRAN and COBOL.

I/O page
That portion of memory in which specific storage locations are associated directly with I/O devices.

I/O rundown
A process which delays the availability of a partition until all transfers to and from that partition have been stopped or have been allowed to complete. I/O rundown is invoked when a task is terminated and has outstanding transfers pending to or from its partition.

Idle time
That part of uptime in which no job could run because all jobs are halted or waiting for some external action such as I/O.

Image mode
Refers to a mode of data transfer in which each byte of data is transferred without any interpretation or data changes.

Impure code
The code which is modified during the course of a program’s execution, e.g., data tables.

Incremental compiler
A compiler that immediately translates each source statement into an internal format, ready for execution.

Indirect file
A file containing commands that are processed sequentially, yet which could have been entered interactively at a terminal.

Indirect Mode
The mode of MUMPS-11 system operation in which steps of a stored program can be executed. In this mode, neither commands nor functions can be entered at the terminal, nor can programs be created or modified.

Indirect reference
A feature of the MUMPS language which permits the symbolic representation of an argument or argument list in a command by a string variable. In operation, the string value of the variable is taken as the argument or argument list for the command. The indirection symbol, a back-arrow or underscore, must precede the variable reference.

Initialize
To set counters, switches, or addresses to starting values at prescribed points in the execution of a program, particularly in preparation for re-execution of a sequence of code. To format a volume in a particular file-structured format in preparation for use by an operating system.

Interactive
A technique of user/system communication in which the operating system immediately acknowledges and acts upon requests entered by the user at a terminal. Compare with batch.
Interpreter  
A computer program that translates and executes each source language statement before translating and executing the next statement.

Interrupt  
A signal which, when activated, causes a transfer of control to a specific location in memory, thereby breaking the normal flow of control of the routine being executed. An interrupt is normally caused by an external event such as a done condition in a peripheral. It is distinguished from a trap which is caused by the execution of a processor instruction.

Interrupt service routine  
The routine entered when an external interrupt occurs.

Interrupt vector address  
A unique address which points to two consecutive memory locations containing the start address of the interrupt service routine and priority at which the interrupt is to be serviced.

I/O bound  
A state of program execution in which all operations are dependent on the activity of an I/O device. For example, when a program is waiting for input from a terminal. See also Compute-bound.

Job  
A group of data and control statements which does a unit of work, e.g., a program and all its related subroutines, data and control statements; also, a batch control file.

Keyboard monitor  
A program that provides and supervises communication between the user at the system console and an operating system.

Latency  
The time from initiation of a transfer operation to the beginning of actual transfer; i.e., verification plus search time. The delay while waiting for a rotating memory to reach a given location.

Leader  
A blank section of tape at the beginning of a reel of magnetic tape or at the beginning of paper tape.

Library  
A file containing one or more relocatable binary modules which are routines that can be incorporated into other programs.

Library Programs  
A class of MUMPS programs which are listed in the system program directory and which are available to all users of the system to run.

Line  
A string of characters terminated with a vertical tab, form feed or line feed.

Linked file  
A file whose blocks are joined together by references (a link word or pointer imbedded in the block) rather than consecutive location.
Linker
A program that combines many relocatable object modules into an executable module. It satisfies global references and combines control sections.

Linking loader
A program that provides automatic loading, relocation and linking of compiler and assembler generated object modules.

Listing
The hard copy generated by a line printer.

Literal
An element of a programming language which permits the explicit representation of character strings in expressions and command and function elements. In most languages, a literal is enclosed in either single or double quotes to denote that the enclosed string is to be taken "literally" and not evaluated.

Load
To store a program or data in memory. To mount a tape on a device such that the read point is at the beginning of the tape. To place a removable disk in a disk drive and start the drive.

Load image file
A program that can be executed in a stand-alone environment without the aid of relocation.

Load map
A table produced by a linker that provides information about a load module's characteristics, e.g., the transfer address and the low and high limits of the relocatable code.

Load module
A program in a format ready for loading and executing.

Local variable
In the MUMPS-11 system, a local variable is a variable which is stored only in the partition in which a program is executed (as opposed to a global variable).

Location
An address in storage or memory where a unit of data or an instruction can be stored.

Log in
To identify oneself to an operating system as a legitimate user of the system and gain access to its services.

Log out or log off
To sign off a system.

Logical block
An arbitrarily-defined fixed number of contiguous bytes which is used as the standard I/O transfer unit throughout an operating system. For example, the commonly-used logical block in PDP-11 systems is 512 bytes.
long. An I/O device is treated as if its block length is 512 bytes, although the device may have an actual (physical) block length which is not 512 bytes. Logical blocks on a device are numbered from block 0 consecutively up to the last block on the volume. A logical block is synonymous with a physical block on any device that has 512-byte physical blocks. See also Virtual Block, Physical Block, Logical Record, and Physical Record.

Logical device name
An alphanumeric name assigned by the user to represent a physical device. The name can then be used synonymously with the physical device name in all references to the device. Logical device names are used in device independent systems to enable a program to refer to a logical device name which can be assigned to a physical device at run-time.

Logical record
A logical unit of data within a file whose length is defined by the user and whose contents have significance to the user. A group of related fields treated as a unit.

Logical Unit Number
A number associated with a physical device unit during a task's I/O operations. Each task in the system can establish its own correspondence between Logical Unit Numbers and physical device units.

Macro
Directions for expanding abbreviated text. A boilerplate that generates a known set of instructions, data or symbols. A macro is used to eliminate the need to write a set of instructions which are used repeatedly. For example, an assembly language macro instruction enables the programmer to request the assembler to generate a predefined set of machine instructions.

Main memory
The set of storage locations connected directly to the Central Processing Unit. Also called (generically) core memory.

Main program
The module of a program that contains the instructions at which program execution begins. Normally, the main program exercises primary control over the operations performed and calls subroutines or subprograms to perform specific functions.

Mapped System
A system which uses the hardware memory management unit to relocate virtual memory addresses.

Mass storage
Pertaining to a device which can store large amounts of data readily accessible to the Central Processing Unit; for example, disk, DECtape, magnetic tape, etc.

Master File Directory
The system-maintained file on a volume that contains the names and addresses of all the files stored on the volume.
Memory
Any form of data storage, including main memory and mass storage, in which data can be read and written. In the strict sense, memory refers to main memory.

Memory image
A replication of the contents of a portion of memory.

Memory mapping
A mode of computer operation in which the high-order bits of a virtual address are replaced by an alternate value, providing dynamic relocatability of programs.

Memory protection
A scheme for preventing read and/or write access to certain areas of memory.

Modulo
A mathematical operation that yields the remainder function of division. Thus 39 modulo 6 equals 3.

Monitor
The master control program that observes, supervises, controls or verifies the operation of a computer system. The collection of routines that controls the operation of user and system programs, schedules operations, allocates resources, performs I/O, etc.

Monitor command
An instruction issued directly to a monitor from a user.

Monitor console
The system control terminal.

Monitor Console Routine (MCR)
The executive routine that allows the user to communicate with the system using an on-line terminal device. MCR accepts and interprets commands typed on the terminal keyboard and calls appropriate routines to execute the specified requests.

Mount a device or volume
To logically associate a physical mass storage media with a physical device unit. To place a volume on a physical mass storage drive unit, for example, place a DECTape on a DECTape drive and put the drive on-line.

Multiprocessing
Simultaneous execution of two or more programs by two or more processors.

Multiprogramming
A processing method in which more than one task is in an executable state at any one time.

Naked syntax
A feature of the MUMPS language which provides an abbreviated method for accessing global variables which controls the disk access time. The
node reference includes only subscript(s) for the element; the global variable name is assumed from the last global reference in which a name was explicitly stated.

**Node**
A dynamically allocated set of bytes from a node pool used for system communication and control in an RSX-11/IAS system. An element of a global array in a MUMPS-11 system (also called a global variable).

**Non-contiguous file**
A file whose blocks are not physically contiguous on the volume.

**Non-file-structured device**
A device, such as paper tape, line printer or terminal, in which data cannot be referenced as a file.

**Object code**
Relocatable machine language code.

**Object module**
The primary output of an assembler or compiler, which can be linked with other object modules and loaded into memory as a runnable program. The object module is composed of the relocatable machine language code, relocation information, and the corresponding symbol table defining the use of symbols within the module.

**Object program**
The relocatable binary program which is the output of a compiler or assembler.

**Object Time System**
The collection of modules that is called by compiled code in order to perform various utility or supervisory operations. For example, an Object Time System usually includes I/O and trap handling routines.

**Off-line**
Pertaining to equipment or devices not under direct control of the Central Processing Unit.

**Offset**
The difference between a base location and the location of an element related to the base location. The number of locations relative to the base of an array, string or block.

**On-line**
Pertaining to equipment or devices directly connected and under control of the Central Processing Unit.

**Operating system**
The collection of programs, including a monitor or executive and system programs, that organizes a central processor and peripheral devices into a working unit for the development and execution of application programs.

**Overlay Description Language**
The set of instructions interpreted by a linker that defines the overlay structure of a task.
Overlay segment
A section of code treated as a unit which can overlay code already in memory and be overlaid by other overlay segments.

Overlay structure
A task overlay system consisting of a root segment and optionally one or more overlay segments.

P-section
A section of memory that is a unit of the total task allocation. A source program is translated into object modules that consist of p-sections (program sections) with attributes describing access, allocation, relocatability, etc.

Pack
To compress data in storage by using an algorithm for its storage and retrieval. A removable disk.

Parity bit
A binary digit appended to a group of bits to make the sum of all the bits always odd (odd parity) or always even (even parity). Used to verify data storage.

Parse
To break a command string into its elemental components for the purpose of interpretation.

Part number
In the MUMPS language, the integer portion of a program step which is used to refer collectively to all steps having a common integer base.

Partition
A contiguous area of memory within which tasks are loaded and executed.

Patch
To modify a program by changing the binary code rather than the source code.

Peripheral
Any device distinct from the central processor which can provide input or accept output from the computer.

Physical address space
The set of memory locations where information can actually be stored for program execution. Virtual memory addresses can be mapped, relocated or translated to produce a final memory address which is sent to hardware memory units. The final memory address is the physical address.

Physical block
A physical record on a mass storage device.

Physical device
An I/O or peripheral storage device connected to or associated with a central processor.
Physical record
The largest unit of data that the read/write hardware of an I/O device can transmit or receive in a single I/O operation. The length of a physical record is device dependent. For example, a punched card can be considered the physical record for a card reader; it is 80 bytes long. The physical record for an RK11 disk is a block, it is 512 bytes long.

Position independent code
Code which can execute properly wherever it is loaded in memory, without modification or relinking. Generally, this code uses addressing modes which form an effective memory address relative to the central processor's Program Counter (PC).

Priority
A number associated with a task that determines the preference its requests for service receives from the executive, relative to other tasks requesting service.

Privilege
A characteristic of a user or program which determines what kinds of operations that user or program can perform. In general, a privileged user or program is allowed to perform operations normally considered to be the domain of the monitor or executive, or which can affect system operation as a whole.

Program development
The process of writing, entering, translating, and debugging source programs.

Programmed requests
An instruction available only to programs that is used to invoke a monitor service.

Programmer Access Code
The system identification code that enables a user to gain access to a MUMPS-11 system in Direct Mode to create, modify and execute programs.

Project-programmer number
See account number.

Pseudo-device
A logical entity treated as an I/O device by the user or the system, but which is not actually any particular physical device.

Public disk structure
The disk volume or set of volumes which are used as a general storage pool available to any users having quotas on the public structure.

Pure code
Code that is never modified during execution. It is possible to let many users share the same copy of a program that is written as pure code.

Qualifier
A parameter specified in a command string that modifies some other parameter. See Switch.
Queue
Any list of items; for example of items waiting to be scheduled or processed according to system or user assigned priorities.

Radix-50
A storage format in which three ASCII characters are packed into a 16-bit word.

Random access
Access to data in which the next location from which data are to be obtained is not dependent on the location of the previously obtained data.

Real-time processing
Computation performed while a related or controlled physical activity is occurring so that the results of the computation can be used in guiding the process.

Record
A collection of adjacent data items treated as a unit. See Logical Record and Physical Record.

Record gap
An area between two consecutive records.

Recursive
Pertaining to a process that is inherently repetitive. The result of each repetition of the process is usually dependent on the result of the previous repetition.

Reentrant
The property of a program that enables it to be interrupted at any point by another program, and then resumed from the point where it was interrupted.

Resident
Pertaining to data or instructions that are normally permanently located in main memory.

Restart address
The address at which a program can be restarted. It is normally the address of the code required to initialize variables, counters, etc.

Root segment
The segment of an overlay tree that, loaded, remains resident in memory during the execution of a task.

RUNOFF
A program that is used to prepare printed documents by performing formatting, case conversion, line justification, page numbering, titling, and indexing.

Secondary storage
Mass storage other than main memory.
Sentinel file
The last file on a cassette tape which represents the logical end-of-tape.

Sequential access
A data access method in which records or files are read one after another in the order in which they appear in the file or volume.

Shareable program
A (reentrant) program that can be used by several users at the same time.

Significant event
An event or condition which indicates a change in system status in an event-driven system. A significant event is declared, for example, when an I/O operation completes. A declaration of a significant event indicates that the executive should review the eligibility of task execution, since the event might unblock the execution of a higher priority task. The following are considered to be significant events: I/O queuing, I/O request completion, a task request, a scheduled task execution, a mark time expiration, a task exit.

Single user access
The status of a volume that allows only one user to access the file structure of a volume.

Single-stream batch
A method of batch processing in which only one stream of batch commands is processed.

Source language
The system of symbols and syntax easily understood by people which is used to describe a procedure that a computer can execute.

Sparse array
Refers to the method of storage allocation used in MUMPS-11 for local and global arrays in which space is allocated only as variables are explicitly defined (unlike other languages which require dimension or size statements for preallocation of storage).

Spooling
The technique by which output to low-speed devices is placed into queues on faster devices to await transmission to the slower devices.

Step number
A number in the range 0.01 to 327.67 used to identify each line of a MUMPS program.

Subscript
A numeric valued expression or expression element which is appended to a variable name to uniquely identify specific elements of an array. Subscripts are enclosed in parentheses. Multiple subscripts must be separated by commas. For example, a two-level subscript might be (2,5).

Swapping
The process of copying areas of memory to mass storage and back in
order to use the memory for two or more purposes. Data are swapped out when a copy of the data in memory is placed on a mass storage device; data are swapped in when a copy on a mass storage device is loaded in memory.

**Swapping device**
A mass storage device especially suited for swapping because of its fast transfer rate.

**Switch**
An element of a command or command string that enables the user to choose among several options associated with the command. In PDP-11 software systems, a switch element consists of a slash character (/) followed by the switch name and, optionally, a colon and a parameter. For example, a command used to print three copies of a file on the line printer could be: “PRINT filename/COPIES: 3”.

**Synchronous**
The performance of a sequence of operations controlled by an external clocking device. Implies that no operation can take place until the previous operation is complete.

**Synchronous System Trap**
A system condition which occurs as a result of an error or fault within the executing task.

**System device**
The device on which the operating system is stored.

**System generation**
The process of building an operating system on or for a particular hardware configuration with software configuration modifications.

**System manager**
The person at a computer installation responsible for the overall nature of its operation.

**System operator**
See operator.

**System program**
A program that performs system-level functions. Any program that is part of the basic operating system. A system utility program.

**System programmer**
A person who designs and codes the programs that control the basic operations of a computer system, as opposed to an application program.

**System UCI**
The User Class Identifier (UCI) code in a MUMPS-11 system which is assigned to the first entry in the system’s UCI table. The program and global directories associated with the System UCI are used to contain both system and library programs and globals.

**Task**
In RSX-11 terminology, a load module with special characteristics. In general, any discrete operation performed by a program.
Terminal
An I/O device, such as an LA36 terminal, which includes a keyboard and a display mechanism. In PDP-11 systems, a terminal is used as the primary communication device between a computer system and a person.

Time slice
The period of time allocated by the operating system to process a particular program.

Unformatted ASCII
A mode of data transfer in which the low-order seven bits of each byte are transferred. No special formatting of the data occurs or is recognized.

Unformatted binary
A mode of data transfer in which all bits of a byte are transferred without regard to their contents.

Unmapped system
An RSX-11M or RSX-11S system that does not have a hardware memory management unit available for virtual address relocation.

User Class Identifier
An identification code that enables a user to gain access to a MUMPS-11 system to execute programs.

User Identification Code
The number or set of numbers that serves to distinguish a particular user or collection of files in a multi-user system. The common format for a User Identification Code is two numbers separated by a comma, enclosed in brackets.

User program
An application program.

Utility
Any general-purpose program included in an operating system to perform common functions.

Variable
The symbolic representation of a logical storage location which can contain a value that changes during a discrete processing operation.

Virtual address space
A set of memory addresses that are mapped into physical memory addresses by the paging or relocation hardware when a program is executed.

Virtual array
A RSTS/E file structure that is logically organized as a dimensioned array.

Virtual block
One of a collection of blocks comprising a file (or the memory image of that file). The block is virtual only in that its block number refers to its position relative to other blocks within the file, instead of to its position relative to other blocks on the volume. That is, the virtual blocks of
a file are numbered sequentially beginning with one, while their corresponding logical block numbers can be any random list of valid volume-relative block numbers.

**Volume**
A mass storage media that can be treated as file-structured data storage.

**Word**
Sixteen binary digits treated as a unit in PDP-11 processor memory.

**Zero a device**
To erase all the data stored on a volume and re-initialize the format of the volume.
COMMONLY-USED MNEMONICS AND ABBREVIATIONS

A  Amps
A/D  Analog/Digital
ACP  Ancillary Control Processor
ANSI  American National Standards Institute
AR11  Analog Real-time subsystem
ASCII  American Standard Code for Information Interchange
AST  Asynchronous system trap
ATL  Active Task List
BAC  BASIC compiled program file
BAK  Back-up file
BAS  BASIC source program file
BASIC  Beginner’s all-purpose symbolic instruction code
BAT  Batch file
BAUD  Bits per second
BI  Batch input device
BM873  Restart bootstrap loader
BP  Batch pseudo-device
CAI  Computer Assisted Instruction
CAPS-11  Cassette Programming System
CBL  COBOL source program
CCITT  Comite Consultatif Internationale de Telegraphie et Telephonie
CIL  Core image library
CL  Console log device
CLI  Command Language Interpreter
CM11  Mark sense or punched card reader
CMD  Command file
CO  Console output device
COB  COBOL source program
COBOL  Common Business Oriented Language
COMTEX  Communications Oriented Multiple Terminal Executive
CPU  Central processing unit
CR  Card reader
CR11  Punched card reader
CRC  Cyclic redundancy check
CREF  Cross-reference
CRT  Cathode ray tube
CSECT  Control section
CSI  Command String Interpreter
CT  Cassette tape
CTRL  Control key
CUSPS  Commonly-used system programs
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAT</td>
<td>Data file</td>
</tr>
<tr>
<td>DB</td>
<td>RP04 disk</td>
</tr>
<tr>
<td>DBMS</td>
<td>Data base management system</td>
</tr>
<tr>
<td>DEC</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>DDCMP</td>
<td>Digital Data Communications Message Protocol</td>
</tr>
<tr>
<td>DF</td>
<td>RF11 disk</td>
</tr>
<tr>
<td>DH11</td>
<td>Asynchronous programmable multiplexor</td>
</tr>
<tr>
<td>DIR</td>
<td>Directory file</td>
</tr>
<tr>
<td>DJ11</td>
<td>Asynchronous fixed multiplexor</td>
</tr>
<tr>
<td>DK</td>
<td>RK11 disk</td>
</tr>
<tr>
<td>DL11</td>
<td>Single-line asynchronous interface</td>
</tr>
<tr>
<td>DM11</td>
<td>Line adapter or modem control</td>
</tr>
<tr>
<td>DMP</td>
<td>Dump file</td>
</tr>
<tr>
<td>DMS</td>
<td>Data Management System</td>
</tr>
<tr>
<td>DOS</td>
<td>Disk Operating System</td>
</tr>
<tr>
<td>DP</td>
<td>RP02, RP03 or RP04 disk</td>
</tr>
<tr>
<td>DPB</td>
<td>Directive parameter block</td>
</tr>
<tr>
<td>DQ11</td>
<td>Full/half duplex synchronous NPR interface</td>
</tr>
<tr>
<td>DS</td>
<td>RS03 or RS04 disk</td>
</tr>
<tr>
<td>DSW</td>
<td>Directive status word</td>
</tr>
<tr>
<td>DT</td>
<td>DECTape</td>
</tr>
<tr>
<td>DU11</td>
<td>Full/half duplex synchronous interface</td>
</tr>
<tr>
<td>DV11</td>
<td>Synchronous multiple line communications preprocessor</td>
</tr>
<tr>
<td>EAE</td>
<td>Extended Arithmetic Element</td>
</tr>
<tr>
<td>EDI</td>
<td>Editor utility</td>
</tr>
<tr>
<td>EIA</td>
<td>Electronics Industry Association</td>
</tr>
<tr>
<td>EIS</td>
<td>Extended Instruction Set</td>
</tr>
<tr>
<td>EMT</td>
<td>Emulator trap</td>
</tr>
<tr>
<td>EOD</td>
<td>End of data</td>
</tr>
<tr>
<td>EOF</td>
<td>End of file</td>
</tr>
<tr>
<td>EOJ</td>
<td>End of job</td>
</tr>
<tr>
<td>EOL</td>
<td>End of line</td>
</tr>
<tr>
<td>EOM</td>
<td>End of medium</td>
</tr>
<tr>
<td>FA</td>
<td>Formatted ASCII</td>
</tr>
<tr>
<td>FB</td>
<td>Formatted binary</td>
</tr>
<tr>
<td>F/B</td>
<td>Foreground/background</td>
</tr>
<tr>
<td>FCP</td>
<td>File Control Primitives</td>
</tr>
<tr>
<td>FCS</td>
<td>File Control Services</td>
</tr>
<tr>
<td>FDB</td>
<td>File data block</td>
</tr>
<tr>
<td>FILEX</td>
<td>File Exchange Utility</td>
</tr>
<tr>
<td>FIS</td>
<td>Floating Instruction Set</td>
</tr>
<tr>
<td>FLX</td>
<td>File Exchange Utility</td>
</tr>
<tr>
<td>FNB</td>
<td>Filename block</td>
</tr>
<tr>
<td>FOCAL</td>
<td>Formula Calculator</td>
</tr>
<tr>
<td>FOR</td>
<td>FORTRAN source program</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>Formula Translator</td>
</tr>
<tr>
<td>FP11</td>
<td>Floating point processor</td>
</tr>
<tr>
<td>FPP</td>
<td>Floating point processor</td>
</tr>
<tr>
<td>FSR</td>
<td>File storage region</td>
</tr>
<tr>
<td>FTN</td>
<td>FORTRAN source program</td>
</tr>
<tr>
<td>F4P</td>
<td>FORTRAN IV-PLUS source program</td>
</tr>
<tr>
<td>GT</td>
<td>Graphics terminal</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>hz</td>
<td>Hertz</td>
</tr>
<tr>
<td>IAS</td>
<td>Interactive Application System</td>
</tr>
<tr>
<td>ID</td>
<td>Identification code</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/output</td>
</tr>
<tr>
<td>IOT</td>
<td>Input/output trap</td>
</tr>
<tr>
<td>IOX</td>
<td>I/O Executive</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt service routine</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to subroutine</td>
</tr>
<tr>
<td>K</td>
<td>1024 decimal (from &quot;kilo&quot;)</td>
</tr>
<tr>
<td>KB</td>
<td>Keyboard</td>
</tr>
<tr>
<td>KBL</td>
<td>Keyboard listener</td>
</tr>
<tr>
<td>KCT</td>
<td>Kilo-core tick</td>
</tr>
<tr>
<td>KE11-A</td>
<td>Extended Arithmetic Element option</td>
</tr>
<tr>
<td>KE11-E</td>
<td>Extended Instruction Set option</td>
</tr>
<tr>
<td>KE11-F</td>
<td>Floating Point option</td>
</tr>
<tr>
<td>KT11-D</td>
<td>Memory Management Unit</td>
</tr>
<tr>
<td>KW11-L</td>
<td>Line frequency clock</td>
</tr>
<tr>
<td>KW11-P</td>
<td>Programmable clock</td>
</tr>
<tr>
<td>KW11-W</td>
<td>Watchdog timer clock</td>
</tr>
<tr>
<td>LA36</td>
<td>DECwriter hard copy terminal</td>
</tr>
<tr>
<td>LBR</td>
<td>Librarian</td>
</tr>
<tr>
<td>LDA</td>
<td>Load module</td>
</tr>
<tr>
<td>LED</td>
<td>Light emitting diode</td>
</tr>
<tr>
<td>LIB</td>
<td>Library file</td>
</tr>
<tr>
<td>LIBR</td>
<td>Librarian</td>
</tr>
<tr>
<td>LICIL</td>
<td>Linked core image library</td>
</tr>
<tr>
<td>LIS</td>
<td>Listing file</td>
</tr>
<tr>
<td>LP</td>
<td>Line printer</td>
</tr>
<tr>
<td>LP11</td>
<td>132 column high-speed line printer</td>
</tr>
<tr>
<td>LPS11</td>
<td>Laboratory Peripheral System</td>
</tr>
<tr>
<td>LS11</td>
<td>132 column low-speed line printer</td>
</tr>
<tr>
<td>LST</td>
<td>Listing file</td>
</tr>
<tr>
<td>LUN</td>
<td>Logical unit number</td>
</tr>
<tr>
<td>LV11</td>
<td>Printer/plotter</td>
</tr>
<tr>
<td>μ</td>
<td>micro (mu—one millionth)</td>
</tr>
<tr>
<td>m</td>
<td>milli (one thousandth), or meters</td>
</tr>
<tr>
<td>M</td>
<td>mega (one million)</td>
</tr>
<tr>
<td>MAC</td>
<td>MACRO source program</td>
</tr>
<tr>
<td>MAP</td>
<td>Load map</td>
</tr>
<tr>
<td>MCR</td>
<td>Monitor Console Routine</td>
</tr>
<tr>
<td>MFD</td>
<td>Master File Directory</td>
</tr>
<tr>
<td>MM</td>
<td>TU16 magnetic tape</td>
</tr>
<tr>
<td>MO</td>
<td>Message output device</td>
</tr>
<tr>
<td>MR11-DB</td>
<td>Bootstrap loader for mass-storage devices</td>
</tr>
<tr>
<td>MST</td>
<td>Macro Symbol Table</td>
</tr>
<tr>
<td>MT</td>
<td>Magnetic tape</td>
</tr>
<tr>
<td>MTBF</td>
<td>Mean time between failures</td>
</tr>
<tr>
<td>n</td>
<td>nano (one billionth)</td>
</tr>
<tr>
<td>NCP</td>
<td>Network control processor</td>
</tr>
<tr>
<td>NPR</td>
<td>Non-processor request</td>
</tr>
<tr>
<td>OBJ</td>
<td>Object module</td>
</tr>
</tbody>
</table>
ODT  On-line debugging techniques
OEM  Original equipment manufacturer
OTL  On-line Task Loader
OTS  Object Time System
PC   Program counter
PC11 Paper tape reader/punch
PDF  Processor-defined function
PDS  Program Development System
PDP  Programmed Data Processor
PIC  Position independent code
PIP  Peripheral Interchange Program
PP   Paper tape punch
PR   Paper tape reader
PSECT Program section
PST  Permanent Symbol Table
PTS-11 Paper Tape System
PUD  Physical unit directory
QIO  Queue I/O
RJP04 RP04 disk drive and controller
RJS03 RS03 disk drive and controller
RJS04 RS04 disk drive and controller
RK05 1.2M word disk cartridge drive, or disk cartridge
RK11 RK05 disk cartridge drive and controller
ROM  Read-Only Memory
RP02 Disk for RP03 disk drive
RP03 20M word moving head disk pack drive
RP04 44M word moving head disk pack drive, or disk pack
RS03 256K word fixed head disk drive
RS04 512K word fixed head disk drive
RSTS/E Resource-sharing Timesharing System/Extended
RSX-11 Real-time Resource Sharing Executive
RT-11 Real-time Foreground/Background System
RTS  Run Time System
RWED Read, write, extend and delete
RPW04 RP04 disk drive and controller for PDP-11/70
RWS03 RS03 disk drive and controller for PDP-11/70
RWS04 RS04 disk drive and controller for PDP-11/70
SAV  Saved file or system image file
SCI  System Control Interface
SCOM System Communication Area
SGA  Shareable Global Area
SIP  System Image Preservation
SP   Stack pointer
SPC  Small Peripheral Controller
SPR  Software Performance Report
SST  Synchronous system trap
SY   System device
SYS  System file
SYSGEN System generation
TA11 Dual cassette transport and controller
TC11 TU56 dual DECTape transport and controller
TCP  Timesharing Control Primitives
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI</td>
<td>Terminal interface</td>
</tr>
<tr>
<td>TJU16</td>
<td>TU16 magnetic tape transport and controller</td>
</tr>
<tr>
<td>TKB</td>
<td>Task Builder</td>
</tr>
<tr>
<td>TKTN</td>
<td>Task Termination Notice</td>
</tr>
<tr>
<td>TM11</td>
<td>TU10 magnetic tape transport and controller</td>
</tr>
<tr>
<td>TMP</td>
<td>Temporary file</td>
</tr>
<tr>
<td>TSK</td>
<td>Task image file</td>
</tr>
<tr>
<td>TT</td>
<td>Terminal device</td>
</tr>
<tr>
<td>TTY</td>
<td>Terminal device</td>
</tr>
<tr>
<td>TU10</td>
<td>800 bits per inch magnetic tape transport</td>
</tr>
<tr>
<td>TU16</td>
<td>800 or 1600 bits per inch magnetic tape transport</td>
</tr>
<tr>
<td>TU56</td>
<td>Dual DECtape transport</td>
</tr>
<tr>
<td>TWU16</td>
<td>Magnetic tape transport and controller for PDP-11/70</td>
</tr>
<tr>
<td>UA</td>
<td>Unformatted ASCII</td>
</tr>
<tr>
<td>UB</td>
<td>Unformatted binary</td>
</tr>
<tr>
<td>UFD</td>
<td>User File Directory</td>
</tr>
<tr>
<td>UIC</td>
<td>User Identification Code</td>
</tr>
<tr>
<td>USR</td>
<td>User Service Routine</td>
</tr>
<tr>
<td>UST</td>
<td>User Symbol Table</td>
</tr>
<tr>
<td>V</td>
<td>Volts</td>
</tr>
<tr>
<td>VDT</td>
<td>Video display terminal</td>
</tr>
<tr>
<td>VT11</td>
<td>Display processor with light pen</td>
</tr>
<tr>
<td>VT50</td>
<td>DECscope video display terminal</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive OR</td>
</tr>
</tbody>
</table>