COMPONENTS

VAX/VMS provides two programming environments:
- the native programming environment
- the compatibility mode programming environment

The native programming environment consists of the language processors that produce native object code and the program development tools that support native program development. The VAX-11 MACRO assembler, VAX-11 FORTRAN IV-PLUS compiler, VAX-11 COBOL-74 compiler, and BLISS-32 compiler all produce VAX-11 native mode code.

The compatibility mode programming environment consists of the language processors that produce PDP-11 compatibility mode object code and the program development tools that support compatibility program development. The PDP-11 BASIC-PLUS-2/VAX compiler and the PDP-11 RPG II/VAX compiler produce compatibility mode code.

From the programmer's point of view, there is very little difference between native and compatibility program development. The difference between the native programming environment and the compatibility mode programming environment is that:
- Native programs can be built from procedures written in any native language, and native programs share a common run-time procedure library.
- Compatibility mode programs can be built from procedures written in a given compatibility mode language, and compatibility mode programs each have their individual object time libraries.

Furthermore, native and compatibility mode programs can read and write the same files, with the exceptions that native programs can read and write sequential and relative files only, while compatibility mode programs can read and write sequential, relative, and indexed files. Refer to the Data Management Services section for more information on file accessing.

The compatibility mode programming environment can be extended to provide an RSX-11M program development environment. The option includes RSX-11M language processors and tools that can be used to create programs to be executed in PDP-11 compatibility mode on the VAX-11/780 system, or to be executed on PDP-11 systems running the RSX-11M or RSX-11S operating systems, and in some cases, the IAS operating system.

VAX-11 MACRO

The VAX-11 MACRO assembler accepts one or more source modules written in MACRO assembly language and produces a relocatable object module and optional assembly listing. VAX-11 MACRO is similar to PDP-11 MACRO, but its instruction mnemonics correspond to the VAX-11/780 native instructions. VAX-11 MACRO is characterized by:
- relocatable object modules
- global symbols for linking separately assembled object programs
- global arithmetic, global assignment operator, global label operator and default global declarations
- user-defined macros with keyword arguments
- multiple macro libraries with fast access structure
- program sectioning directives
- conditional assembly directives
- assembly and listing control functions
- alphabetized, formatted symbol table listing
- default error listing on command output device
- a Cross Reference Table (CREF) symbol listing

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. Permanent symbols consist of the VAX-11 instruction mnemonics and MACRO directives and do not have to be defined by the user. User-defined symbols are those used as labels or defined by direct assignment. Macro symbols are those symbols used as macro names.

MACRO maintains a symbol table for each type of symbol. The value of a symbol depends on its use in the program. To determine the value of a symbol in the operator field, the assembler searches the macro symbol table, user symbol table, and permanent symbol table in that order. To determine the value of the symbol used in the operand field, 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.

User-defined symbols are either internal to a source program module or globa (externally available). An internal symbol definition is limited to the module in which it appears. Internal symbols are local definitions which are resolved by the assembler.

A global symbol can be defined in one source program module and referenced within another. 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.

Directives

A program statement can contain one of three different operators: a macro call, a VAX-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

**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 tilting object modules and presenting other identification information in the listing output.

The listing control options can also be specified at assembly time through qualifiers in the command string issued to the MACRO assembler. The use of these qualifiers provides initial listing control options that may be overridden by the corresponding listing control directives in the source program.

**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 conditions within the body of the program. This capability allows several variations of a program to be generated from the same source module.

The user can define a conditional assembly block of code, and within that block, issue subconditional directives. Subconditional directives can indicate the conditional or unconditional assembly of an alternate or non-contiguous body of code within the conditional assembly block. Conditional assembly directives can be nested.

**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. MACRO automatically creates unique symbols where a label is required in an expanded macro to avoid duplicate label specifications. Macros can be nested; that is, the definition of one macro can include a call to another.

An indefinite repeat block is a structure that is similar to a macro definition, except it 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. This type of macro definition does not require calling the macro by name, as required in the expansion of conventional macros. 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**
A program can call macros that are not defined in that program. A user can create libraries of macro definitions, and MACRO will look up definitions in one or more given library files when the calls are encountered in the program. Each library file contains an index of the macro definitions it contains to enable MACRO to find definitions quickly.

**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 image.

The program sectioning directive allows the user to exercise complete control over the virtual memory allocation of a program, since any program attributes established through this directive are passed to the linker. For example, if a programmer is writing multi-user programs, the program sections containing only instructions can be declared separately from the sections containing only data. Furthermore, these program sections can be declared as read-only code, qualifying them for use as protected, reentrant programs.

**VAX-11 FORTRAN IV-PLUS**
VAX-11 FORTRAN IV-PLUS is an optional language processing system whose language specifications are based on the American National Standard FORTRAN X3.9-1966. VAX-11 FORTRAN IV-PLUS also includes many language features of the proposed American National Standard FORTRAN-77. The FORTRAN IV-PLUS compiler:

• produces highly optimized VAX-11 native object code
• makes use of the VAX-11 floating point and character string instructions
• produces sharable code

The VAX-11 FORTRAN IV-PLUS language is upward compatible with the PDP-11 FORTRAN IV and FORTRAN IV-PLUS languages. The VAX-11 FORTRAN IV-PLUS compiler supports the same enhancements to the language standard as PDP-11 FORTRAN IV and FORTRAN IV-PLUS, as well as providing additional enhancements. Table 7-1 lists most of the extensions to the ANSI FORTRAN specification provided by the FORTRAN language compilers. Some characteristics of VAX-11 FORTRAN IV-PLUS are described in the following paragraphs.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VAX-11 and PDP-11 FORTRAN IV-PLUS, PDP-11 FORTRAN IV</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Array Dimensions</strong></td>
<td>Arrays can have up to seven dimensions.</td>
</tr>
<tr>
<td><strong>Array Subscripts</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Character Literals</strong></td>
<td>Character strings bounded by apostrophes can be used in place of Hollerith constants.</td>
</tr>
<tr>
<td><strong>Mixed-mode Expressions</strong></td>
<td>Mixed-mode expressions can contain any data type, including complex and byte.</td>
</tr>
<tr>
<td><strong>End of Line Comments</strong></td>
<td>Any FORTRAN statement can be followed, in the same line, by a comment that begins with an exclamation point.</td>
</tr>
<tr>
<td><strong>Conditional Compilation of Debugging Statements</strong></td>
<td>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 option is set. They are treated as comments otherwise.</td>
</tr>
<tr>
<td><strong>General Expression DO and GO TO Parameters</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>DO Increment Parameter</strong></td>
<td>The value of the DO statement increment parameter can be negative.</td>
</tr>
<tr>
<td><strong>Optional Statement Label List</strong></td>
<td>The statement label list in an assigned GO TO is optional.</td>
</tr>
<tr>
<td><strong>General Expressions in I/O Lists</strong></td>
<td>General expressions are permitted in I/O lists of WRITE, TYPE, and PRINT statements.</td>
</tr>
<tr>
<td><strong>Default FORMAT Widths</strong></td>
<td>The programmer can specify input or output formatting by type and default width and precision values will be supplied.</td>
</tr>
<tr>
<td><strong>VAX-11 and PDP-11 FORTRAN IV-PLUS</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ENTRY statement</strong></td>
<td>ENTRY statements can be used in SUBROUTINE and FUNCTION subprograms to define multiple entry points in a single program unit.</td>
</tr>
<tr>
<td><strong>PARAMETER statement</strong></td>
<td>PARAMETER statements can be used to give symbolic names to constants.</td>
</tr>
<tr>
<td><strong>INCLUDE statement</strong></td>
<td>The INCLUDE statement incorporates FORTRAN source text from a separate file into a FORTRAN program.</td>
</tr>
<tr>
<td><strong>Generic function selection</strong></td>
<td>Function selection by argument data type is provided for many FORTRAN library functions.</td>
</tr>
<tr>
<td><strong>Array Dimension Bounds</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>List-Directed I/O Statements</strong></td>
<td>The READ (u,<em>), WRITE (u,</em>), TYPE*, ACCEPT*, and PRINT* statements provide list directed, or &quot;free format&quot;, I/O without requiring a FORMAT specification.</td>
</tr>
<tr>
<td><strong>Additional I/O Statements</strong></td>
<td>OPEN and CLOSE statements provide file control and attribute definition. ACCEPT, TYPE, and PRINT statements provide device-oriented I/O. ENCODE and DECODE statements provide memory-to-memory formatting. DEFINE FILE, READ (u'r), WRITE (u'r), and FIND (u'r) provide unformatted direct access I/O, which allows the FORTRAN programmer to read and write files written in any format.</td>
</tr>
<tr>
<td><strong>End-of-file or Error Condition Transfer</strong></td>
<td>The specifications END=n and ERR=n (where n is a statement label) 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.</td>
</tr>
<tr>
<td><strong>Additional Data Type</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Logical Operations</strong></td>
<td>The logical operators AND, OR, NOT, XOR, and logical .EQV, may be applied to integer data to perform bit masking and manipulation.</td>
</tr>
<tr>
<td><strong>IMPLICIT declaration</strong></td>
<td>The IMPLICIT statement has been added to redefine the implied data type of symbolic names.</td>
</tr>
<tr>
<td><strong>DO Loop Iteration Count</strong></td>
<td>The terminal and increment parameters can be modified within a DO loop without affecting the iteration count. The number of times a DO loop is executed 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.</td>
</tr>
<tr>
<td><strong>VAX-11 FORTRAN IV-PLUS</strong></td>
<td>Symbolic names used to identify programs, subprograms, external functions and subroutines, COMMON blocks, variables, arrays, symbolic constants, and statement functions can be longer than the standard 6 characters. Symbolic names can be from 1 to 15 characters long and can include letters, digits, dollar sign, and underscore. The first character in the name must be a letter.</td>
</tr>
</tbody>
</table>
Table 7-1 (cont)
Language Extensions to ANS X3.9-1966 FORTRAN

Additional Data Type
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.

INTEGER Data Type Defaults
A compiler command 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.

Additional I/O Statements
READ (u'd,fmt) and WRITE (u'r,fmt) provide input and output to direct access files.

DO Control Variable Data Types
The control variable of a DO statement can be a REAL or DOUBLE PRECISION variable, as well as an INTEGER*2 or INTEGER*4 variable. 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.

Hexadecimal Constants and Field Descriptor
Both octal and hexadecimal constants can be expressed in DATA statements. No conversion of the defined value (such as sign-extension) is performed. The Z field descriptor in FORMAT statements enables a program to read and write hexadecimal digits which are stored in an internal format in an I/O list element.

Additional Data Type
The data type CHARACTER permits manipulation of strings of ASCII characters expressed as constants, variables, arrays, substrings, symbolic names, or functions.

IF THEN ELSE Statements
The proposed FORTRAN-77 block-IF statements are provided: IF, ELSE IF, ELSE, and ENDIF. These structured programming statements provide more readable and reliable methods for expressing conditional statement execution.

Standard CALL Facility
Provides standard argument definitions for called procedures.

File Manipulation
The FORTRAN IV-PLUS OPEN and CLOSE statements extend the file manipulating characteristics of the FORTRAN language. The OPEN statement can contain specifications for file attributes that control file creation or subsequent processing. Attributes include: file organization (sequential, relative), method of access (sequential, direct), protection (read-only, read/write), record type (formatted, unformatted), record size, and file allocation or extension. The program can also specify whether the file can be shared, and whether the file is to be deleted or saved when closed. The OPEN statement can contain an ERR = keyword that specifies the statement to which control is transferred if an error is detected during OPEN.

Simplified I/O Formats
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, 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.

Character Data Type
A program can create fixed-length CHARACTER variables and arrays to store ASCII character strings. The VAX-11 FORTRAN IV-PLUS language provides a concatenation operator, substring notation, CHARACTER relational expressions, and CHARACTER-valued functions. CHARACTER constants, consisting of a string of printable ASCII characters enclosed in single quotes, can be assigned symbolic names using the PARAMETER statement.

Source Program Libraries
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 reference 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.

Calling External Functions and Procedures
FORTRAN programs can call assembly language subroutines, and can call the system services and record management services using the standard VAX-11 procedure calling statements. Special operators exist for passing argument values directly, by reference, or by descriptor. A special operator also exists for obtaining the location of argument values.

Sharable Programs
The FORTRAN IV-PLUS language can be used to created sharable programs because the compiler can produce reentrant code. FORTRAN IV-PLUS subprograms can also be placed in sharable image libraries created by the linker, which can be made available to any program written in a native programming language.

Compiler Operation and Optimizations
The VAX-11 FORTRAN IV-PLUS compiler accepts source programs written in the FORTRAN language and produces an object file which must be linked prior to execution. The compiler generates VAX-11 native machine language code. Figure 7-1 is an illustration of FORTRAN IV-PLUS code and the equivalent VAX-11 assembly code.
SUBROUTINE RELAX2(EPS)

PARAMETER M=40, N=60
DIMENSION X(M,M),XIN
COMMON X

LOGICAL DONE

DONE = .TRUE.

DO 10 J = 1,N+1
10   DO 10 I = 1,M+1

NNEW = ( X(I+1,J)+X(I+1,J)+X(I,J)+X(I,J+1) ) / 4

IF (ABS(NNEW-X(I,J)) .GT. EPS) DONE = .FALSE.

X(I,J) = NNEW

IF (.NOT. DONE) GO TO 1
RETURN
END

RELAX2 17-Oct-1977 12:04:17 VAX=11 FORTRAN IV-PLUS X=0.6=9
RELAX2,FOR.2

Figure 7-1
Sample VAX-11 FORTRAN IV-PLUS Program

Page 1 above shows, as a FORTRAN IV-PLUS subroutine, a relaxation function often found in engineering applications. This particular example is a planar (2-dimensional) function that can be used to obtain the values of a variable at coordinates on a surface, for example, temperatures distributed across a metal plate. The algorithm shown here finds the array element values relative to a given point in the plane.

Page 2 contains the equivalent VAX-11 MACRO assembly code for this FORTRAN subroutine. The line numbers in the comments just to the left refer to the lines in the FORTRAN subroutine listing above. Several FORTRAN IV-PLUS compiler optimizations are illustrated, including global and local register assignment, removal of invariant computations from the DO loop, recognition of common subexpressions, branch instruction optimizations, in-line ABS function, and peephole optimization.

The code for lines 7 and 8 contains the global register assignments for the function. The multiply statement just preceding the code for line 9 is an invariant computation (J*41) removed from the DO loop. DO loop control is provided by the Add One and Branch Less Than or Equal (OBLEQ) instructions in the code for line 11.

The code for line 9 evaluates the common subexpression for the computation. The code contains a local register assignment (R10), and uses 2- and 3-operand instructions and context indexing (R10) to calculate an array element value. The last instruction for line 9 is a peephole optimization that increases execution speed by using a multiply-by-25 in place of the FORTRAN statement's divide-by-4.
During compilation, the FORTRAN IV-PLUS compiler performs many code optimizations. The 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 constant conversion.**
- **Compile-time evaluation of constant subscript expressions in array calculations.**
- **Constant pooling.** Only a single copy of a constant is allocated storage in the compiled program. Constants that can be used as immediate mode operands are not allocated storage. For example, logical, integer, and small floating point constants are generated as immediate mode or short literal operands wherever possible.
- **Argument list merging.** If two function or subroutine references have the same arguments, a single copy of the argument list is generated.
- **Branch instruction optimizations for arithmetic or logical IF statements.**
- **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.**
- **Removal of invariant computations from DO loops.**
- **Local and global register assignment.** Frequently referenced variables and expressions are retained in registers (locally and across DO loops) to reduce the number of memory references.
- **Reordering expression evaluation to minimize the number of temporary registers required.**
- **Delaying negation/not to eliminate unary complement operations.**
- **Flow-Boolean optimizations.**
- **Jump/Branch instruction resolution.** The Branch instruction is used wherever possible to eliminate unnecessary Jump instructions.
- **Peephole optimization.** The code is examined on an operation-by-operation basis to replace sequences of operations with shorter and faster equivalent operations.

**DEBUGGING FACILITIES**

VAX-11 FORTRAN IV-PLUS debugging facilities include diagnostic messages, conditional compilation flags, and access to the VAX/VMS DEBUG program. The DEBUG program lets the programmer set breakpoints and trace points, and examine and modify the contents of locations dynamically when executing the program.

DEBUG understands FORTRAN data type representations and syntax. It can examine and deposit locations using floating point representation, and it can reference FORTRAN symbols, statement labels, and line numbers symbolically. It can also reference arrays symbolically. For example:

```
EXAMINE A(I,J+3)
```

If the programmer is going to use the DEBUG program, the programmer can request the FORTRAN IV-PLUS compiler to disable optimizations that would remove unreferenced statement labels, FORMAT statement labels, and immediately referenced labels. This ensures that all statement labels are available to the debugger.

**FORTRAN Symbolic Traceback**

VAX-11 FORTRAN-IV PLUS supports the Symbolic Traceback Facility. This is a run-time facility that aids programmers in finding errors by describing the call sequences that occurred prior to the error. The Traceback facility is automatic and does not require that any special qualifiers be included with the FORTRAN or LINK commands (but it can be suppressed by specifying NOTRACE with the LINK command).

When an error condition is detected, the error message is displayed by the run-time library indicating the nature of the error and the address at which the error occurred (user PC). This is followed by the traceback information, which is presented in inverse order to the calls. For each call frame, Traceback lists module name, routine name, source program line, and absolute and relative PC. Using this information, the programmer can usually locate the source of the error in a relatively short period of time. Figure 7-2 shows an example of a source program and traceback list.

(NoNote some of the entries in the list show relative and absolute PC but no corresponding values for module name and routine name; this indicates that the values refer to procedure calls internal to the run-time library.)

```
0001     I = 1
0002     CONTINUE
0003     J = 2
0004     CONTINUE
0005     K = 3
0006     CALL SUB1
0007     CONTINUE
0008     END

0001     SUBROUTINE SUB1
0002     I = 1
0003     J = 2
0004     CALL SUB2
0005     END

0001     SUBROUTINE SUB2
0002     COMP_EX W
0003     COMP_EX Z
0004     DATA/(0,0)/
0005     Z = LOG(W)
0006     END

%MTH-F-LOGZERNEG, logarithm of zero or negative value
user PC 00000449
%TRACE-F-TRACEBACK, symbolic stack dump follows
```

**Figure 7-2**

FORTRAN Symbolic Traceback

7-6
VAX-11 COBOL-74

VAX-11 COBOL-74 is an optional language processing system that provides data processing for commercial applications. It conforms in language elements, representation, symbology, and coding format to ANS-COBOL Spec. X.3-23-1974 and is highly compatible with PDP-11 COBOL-74. The VAX-11 COBOL-74 compiler produces native mode code and thus takes advantage of the system's virtual address space, internal instruction set and data types.

VAX-11 COBOL-74 includes the following language elements:
- Level 2 Nucleus module
- Level 2 Table Handling module
- Level 2 Sequential I/O module
- Level 2 Relative I/O module
- Level 2 Indexed I/O module
- Level 2 Segmentation module
- Level 1 Library Module, with partial Level 2 REPLACING facility
- Level 1 Interprogram Communication Module
- Cross Reference Compilation Listing
- DISPLAY verb WITH NO ADVANCING clause
- Conditional variables — Data Division level 88
- Nested conditionals

File Organizations

Both the Sequential I/O and Relative I/O modules meet the full ANS-74 Level 2 standards and include the following COBOL verbs:
- OPEN EXTEND — Add records to a previously created sequential file without recopying the file.
- DYNAMIC ACCESS — Process a relative file both randomly and sequentially in the same program.
- START — Select positioning within a relative file for subsequent 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 by the current program a second time.
- LINAGE — Specify logical page format.

The Level 2 Indexed I/O module statements enable COBOL programs to use the multikey indexed record management services to process indexed files. Indexed files can be accessed sequentially, randomly, or dynamically using one or more keys to select records. The Environment Division RESERVE clause enables the user to specify the number of buffer areas for fast multikey processing.

Data Types

VAX-11 COBOL-74 supports a variety of data types, including:
- Numeric DISPLAY Data
  - Trailing overpunch sign
  - Leading overpunch sign
  - Trailing separate sign
  - Leading separate sign
  - Unsigned
  - Numeric-edited

- Numeric COMPUTATIONAL Data
  - 1-word fixed binary
  - 2-word fixed binary
  - 4-word fixed binary

- Alphanumeric DISPLAY Data
  - Alphanumeric
  - Alphabetic
  - Alphanumeric-edited

- Packed Decimal Data

String Manipulation

COBOL has the capability to manipulate data strings. It offers the INSPECT, STRING, and UNSTRING verbs that search for embedded character strings, with tally and replace. In addition, it is possible to join together or break out separate strings with various delimiters.

Interactive COBOL Programs

The Procedure Division ACCEPT and DISPLAY statements allow terminal-oriented interaction between a COBOL program and a user. 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 DISPLAY statement transfers a message to a specified device, normally the user's console. The statement can be modified by a special WITH NO ADVANCING clause (without automatic appending of carriage return and line feed) that allows the COBOL program to control the format of the message sent. This is especially useful when typing prompting messages on the console.

While the ACCEPT and DISPLAY statements are intended primarily for use with keyboard devices, COBOL allows the ACCEPT statement to accept input from a card reader or the batch input stream, and the DISPLAY statement to display data on a line printer.

SAMPLE COBOL-74 CODE

The sample VAX-11 COBOL-74 code illustrates some of the powerful language elements of COBOL-74. This coding illustrates an interactive COBOL program which will generate various types of reports depending upon user-specified options. The program operates upon an indexed information file via the dynamic access mode. Illustrated are three major COBOL-74 verbs: ACCEPT, DISPLAY and INSPECT.

In Figure 7-3, the program describes the file organization and the access mode. Also described are the primary and alternate keys used for accessing the file randomly.

```plaintext
00064  INPUT-OUTPUT SECTION.
00065  FILE-CONTROL.
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080

SELECT CUSTOMER-FILE
ASSIGN TO "CUSTOM.DAT"
ORGANIZATION IS Indexed
ACCESS MODE IS Dynamic
RECORD KEY IS CUST-CUST-Number
ALTERNATE RECORD KEY IS
CUSTOM-CUSTOM-NAME
FILE STATUS IS CUSTOMER-FILE-STATUS.

SELECT STATEMENT-REPORT
ASSIGN TO "STATEM.REP"
FILE STATUS IS STATEMENT-REPORT-STATUS.
```

Figure 7-3
File Description
Figure 7-4 describes an options area in the working storage section. The programmer uses "88" level numbers allowing reference to coded information via symbolic names.

```
00174 01 OPTIONS-AREA.
00175 03 OPTIONS-AREA-CHAR OCCURS 30 PIC X(1).
00176
00177 01 A-COUNT PIC 9(2).
00178
00179 01 OPTION-STORAGE.
00180 03 OPTION-ENTRY OCCURS 8 PIC 9(1).
00181 01 OPTION-VALUES REDEFINES OPTION-STORAGE.
00182 03 FILLER
00183 88 WANT-STATEMENTS VALUE 1 THRU 9.
00184 03 FILLER
00185 88 WANT-INVOICES VALUE 1 THRU 9.
00186 03 FILLER
00187 88 WANT-ALL-CATALOGS VALUE 1 THRU 9.
00188 03 FILLER
00189 88 WANT-SOME-CATALOGS VALUE 1 THRU 9.
00190 03 FILLER
00191 88 WANT-CREDIT-LIMIT-LETTERS VALUE 1 THRU 9.
00192 03 FILLER
00193
00194 01 RECORD-COUNT PIC 9(5) VALUE 0.
00195 01 STATEMENT-COUNT PIC 9(5) VALUE 0.
00196 01 INVOICE-COUNT PIC 9(5) VALUE 0.
00197 01 CREDIT-LIMIT-COUNT PIC 9(5) VALUE 0.
00198 01 CATALOG-COUNT PIC 9(5) VALUE 0.
00199 01
```

**Figure 7-4**

**Working Storage Area**

In Figure 7-5, using the DISPLAY verb, the interactive COBOL program requests the user to specify an options selection. The user response is then transmitted to the program via the ACCEPT verb. The program uses the INSPECT verb to check that a valid response has been received.

```
00250 DISPLAY " ENTER OPTIONS: ".
00251 DISPLAY " S = Print statements ".
00252 DISPLAY " I = Print invoices ".
00253 DISPLAY " CA = Mail all catalogs ".
00254 DISPLAY " CO = Mail selective catalogs ".
00255 DISPLAY " CL = Credit limit letters ".
00256 ACCEPT OPTIONS-AREA.
00257 MOVE ALL ZERO TO OPTION-STORAGE.
00258 IF OPTIONS-AREA = SPACES
00259     DISPLAY "Discrepancy Report Only"
00260     GO TO CONFIRM-OPTIONS.
00261 MOVE 0 TO A-COUNT.
00262 INSPECT OPTIONS-AREA TALLYING
00263     OPTION-ENTRY (1) FOR ALL "S"
00264     OPTION-ENTRY (2) FOR ALL "I"
00265     OPTION-ENTRY (3) FOR ALL "CA"
00266     OPTION-ENTRY (4) FOR ALL "CO"
00267     OPTION-ENTRY (5) FOR ALL "CL".
00268
00269 IF OPTION-STORAGE = ALL ZERO
00270     DISPLAY "No options recognized"
00271     STOP RUN.
00272
00273 DISPLAY "Selected options: ".
00274 IF WANT-STATEMENTS
00275     DISPLAY " Statements ".
00276 IF WANT-INVOICES
00277     DISPLAY " Invoices ".
00278 IF WANT-ALL-CATALOGS

**Figure 7-5**

**Procedure Division Utilizing Interactive COBOL Verbs**

Figure 7-6 illustrates the dynamic access method, i.e., shift from random to sequential access. The user moves zero to the primary record key, searches the file randomly, and commences sequential processing at the first non-zero number.

```
00302 OPEN INPUT CUSTOMER-FILE.
00303 MOVE "000000" TO CUST-CUST-NUMBER.
00304 START CUSTOMER-FILE
00305     KEY IS CUST-CUST-NUMBER.
00306 OPEN OUTPUT STATEMENT-REPORT.
00307
00308 00309 MAINLINE SECTION.
00310 SBEGIN.
00311 READ CUSTOMER-FILE NEXT
00312     AT END
00313     GO TO END-PROCESS.
00314 ADD 1 TO RECORD-COUNT.
00315 00316 * Print statement if required.
00317 *
00318 *

**Figure 7-6**

**Random to Sequential Access**

Source Library Facility

With COBOL, the user has a full ANS-74 Level 1 Library facility, plus high-level extensions. All frequently used data descriptions and program text sections can be stored in library files available to all programs. These files can then be copied into source programs to reduce program preparation time and eliminate a common source of errors.
CALL Facility
VAX-11 COBOL-74 supports the CALL statement, allowing COBOL programs to invoke separately compiled subprograms and to pass arguments between them. These subprograms may be written in COBOL or in another VAX-11 supported language and may include system service routines written in MACRO-11. To facilitate calls between programs written in different languages, VAX-11 COBOL-74 provides an extended call facility which allows arguments to be passed by value, reference, or descriptor; this is in contrast to standard COBOL in which arguments are passed by reference only. Figure 7-7 shows a sample program which utilizes all three types of argument passing mechanisms.

00001 IDENTIFICATION DIVISION.
00002 PROGRAM-ID. CALLTST2.
00003 ENVIRONMENT DIVISION.
00004 CONFIGURATION SECTION.
00005 SOURCE-PROCESSOR. VAX-11/780.
00006 OBJECT-PROCESSOR. VAX-11/780.
00007 DATA DIVISION.
00008 WORKING-STORAGE SECTION.
00009 01 TIMLEN PIC 9(4) USAGE IS COMP VALUE IS 0.
00010 01 D-TIMLEN PIC 9(4) VALUE IS 9999.
00011 01 TIMB buf PIC X(24) VALUE IS SPACES.
00012 01 DUMMY PIC 9(5) USAGE IS COMP VALUE IS 0.
00013 01 RETURN-VALUE PIC 9(9) USAGE IS COMP
00014 VALUE IS 999999999.
00015 01 D-RETURN-VALUE PIC 9(9) VALUE IS 999999999.
00016 PROCEDURE DIVISION.
00017 DISPLAY "CALL SYS$ASC TIM".
00018 CALL "SYS$ASC TIM" USING BY REFERENCE TIMLEN
00019    BY DESCRIPTOR TIMB Uf
00020    BY VALUE DUMMY
00021    BY VALUE DUMMY
00022    GIVING RETURN VALUE.
00023 DISPLAY "DATE/TIME = " TIMBUF.
00024 MOVE TIMLEN TO D-TIMLEN.
00025 DISPLAY "LENGTH OF RETURNED = " D-TIMLEN.
00026 MOVE RETURN-VALUE TO D-RETURN-VALUE.
00027 DISPLAY "RETURN-VALUE = " D-RETURN-VALUE.
00028 STOP RUN.

Figure 7-7
System Services Call

In this program, the system service routine $ASC TIM is called, which converts binary time to an ASCII string representation. In this example, the buffer length as specified in "timbuf" plus the value of the item "dummy" determine the type of information which the service routine will return to the COBOL program (e.g., specifying a length of 24 plus values of 0 in the following two arguments will cause both current date and time to be returned; if a length of 11 had been specified, then only the date would be returned).

External Subprograms
The CALL statement enables a COBOL program to execute routines that are external to the object module in which the CALL statement appears. The COBOL-74 compiler produces an object module from a single source module supplied as input. The object module file can be linked with other COBOL object modules, or with object modules created by other VAX-11/780 native mode compilers to produce an executable image.

Debugging COBOL Programs
To make program debugging easier, the COBOL compiler produces source language listings with embedded diagnostics. Fully descriptive diagnostic messages are listed at the point of error. Over 500 different error conditions are checked, varying from simple warnings to fatal error detections.

When the compiler detects an error in the source program, the compiler attempts to recover from an error and to 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. The compiler will not produce object code if fatal errors are detected at compile time.

Debugging large source programs is made still easier by the use of the optional allocation maps for the Data and Procedure Divisions, external program references, and object library references. The compiler can also produce a cross-reference listing.

When a fatal error occurs at run time, an error message identifying the cause of the error is displayed to the user. In addition, VAX-11 COBOL-74, like VAX-11 FORTRAN IV-PLUS, supports the Symbolic Traceback Facility, a runtime facility which describes the call sequences which occurred prior to the error. For each call frame, traceback displays the module name, routine name, source line number, and program-counter information. This information identifies in source language terms the module, routine, and source line in which the fatal error occurred.

Figure 7-8 illustrates the printing of an error message and the subsequent traceback for a COBOL module in which a subscript violation occurs at run time. The "module name" and "routine name" fields identify the entry point, SUBERTST, into the COBOL module. The subscript violation occurs on line number 15 of the source module. The "relative PC" field specifies that the subscript violation correspondingly occurs at “3C” hexadecimal bytes into the object code relative to the entry point SUBERTST. The "absolute PC" field also specifies that the violation occurs at absolute location “3074” in the executable image containing SUBERTST. Thus, the issuance of a specific, English-like error message coupled with the traceback facility offers the user a powerful debugging tool in identifying programming errors.

%C74-F-SUBOUTRAN, subscript out of range
%TRACE-F-TRACEBACK, symbolic stack dump follow
module name routine name line relative PC absolute PC
SUBERTST SUBERTST 15 0000003C 00003074

Figure 7-8
Example of Traceback Facility

Source Program Formats
The COBOL compiler accepts source programs that are coded using either the conventional 80-column card refer-
ence format or a shorter, easy-to-enter terminal format. Terminal format is designed for use with the interactive text editors. It eliminates the line-number and identification fields and allows the user to enter horizontal tab characters and short lines.

The RFRMT (Reformat) utility program reads COBOL source programs that were coded using terminal format and converts the source statements to the 80-column source line format accepted by other COBOL compilers throughout the industry. The programmer can enter source programs in the simpler terminal format and then, if compatibility is ever required with other systems, run the RFRMT utility to convert the source to conventional format.

- load in a source program for editing
- compile a source program, produce an executable load module, and execute it ("load and go")
- compile a source program and produce an object module which can be linked with previously compiled object modules

The object time system is a collection of library modules used during program execution. The library routines include math and floating point functions, input/output operations, error handling, and dynamic string storage functions. Since the OTS is a library, the linker can select only those functions needed at run-time to be included in a program. Unnecessary routines are omitted from the program and memory usage is reduced.

**Program Format**

The BASIC source program unit is a line. In its simplest form, it consists of a line number, a keyword and statement, and a line terminator. In BASIC-PLUS-2, one or several spaces or tabs can be used to separate line numbers, keywords, and variable names. Line number determines the order in which the program is processed; the programmer can write BASIC-PLUS-2 program lines in any order.

BASIC-PLUS-2 programs can be one or several lines long. The programmer can place one statement on each line, place several statements on any one line, or spread one statement over several lines. Program comments can be placed anywhere within a line using the REM (Remark) statement or using comment field delimiters. These facilities enable the programmer to freely format a program to make it more readable.

**Long Variable and Function Names**

Most BASIC languages limit the length of a variable or user-defined function name to one character. BASIC-PLUS-2 recognizes variable names and function names as long as thirty characters. The programmer can fully identify variables and functions.

**Dynamic String Handling**

The BASIC-PLUS-2 language enables the programmer to manipulate strings of alphanumeric characters easily. The BASIC-PLUS-2 relational operators enable programmers to concatenate and compare strings; string operators enable the programmer to convert strings and numerics; and string functions add the ability to analyze the composition of strings. The BASIC-PLUS-2 language includes string functions that:

- create a string of a given length
- search for the position of a set of characters within a string
- insert spaces within a string
- trim trailing blanks from a string
- determine the length of a string

Unlike many BASIC languages, BASIC-PLUS-2 imposes no limit on the size of string scalars or string elements of arrays manipulated in memory other than the amount of memory available to the program.

**Virtual Arrays**

Virtual arrays are random access disk-resident files. A program can create and access virtual arrays in the same
way memory-resident arrays are accessed: using element names. Explicit read/write programming is not required. The last element in the array can be accessed as quickly as the first. Because the arrays are stored on disk, however, the programmer can manipulate large amounts of data without affecting program size.

**PRINT USING Output Formats**
The PRINT USING statement allows the programmer to control the appearance and location of data on an output line to create complex lists, tables, reports, and forms. In addition to numeric field definitions that allow the programmer to generate floating dollar sign, aligned decimal point, trailing minus, asterisk fill, and exponential format fields, BASIC-PLUS-2 provides string field definitions which allow the programmer to generate left-justified, right-justified, centered, and extended string fields.

**Subprograms and the CALL Statement**
The BASIC-PLUS-2 CALL statement enables a program to access external subprograms. A programmer can, therefore, write a program in several modular segments, each of which can be compiled separately to speed program development. BASIC-PLUS-2 provides a complete traceback on errors occurring in subroutines.

**COMMON Statement**
The COMMON statement enables a program to pass data to another program or subprogram. The BASIC-PLUS-2 COMMON statement format is similar to FORTRAN COMMON. Strings passed in COMMON are fixed length, which reduces string handling overhead.

**Debugging Statements**
BASIC-PLUS-2 provides an interactive debugging mode similar to the "immediate mode" facilities found in most BASIC interpreters. During program development, the programmer can use the compiler to create, save, edit, and test the source program. The compiler checks syntax immediately on input from a terminal so that many errors can be found prior to compilation. The debugging statements can be used when executing and testing the program. The BREAK, LET, PRINT, UNBREAK, CONTINUE, STEP, and STOP statements enable the programmer to control and observe program execution interactively.

To set breakpoints, the programmer uses the BREAK command just prior to running the program, or while it is stopped. As many as 10 breakpoints can be set during the course of program execution. On reaching a breakpoint, the program halts to allow the programmer to examine or modify variables or set other breakpoints.

To examine variables while a program is stopped, the programmer uses the PRINT statement. The LET statement allows the programmer to modify the value stored in the variable.

Typing the CONTINUE command resumes execution until the next breakpoint is reached. Before typing CONTINUE, the programmer can issue an UNBREAK command to selectively disable one or all of the breakpoints set, and execution continues until a STOP statement is encountered in the program or the program completes.

When a program halts because a STOP statement is included in the program or because a BREAK command was issued interactively, the programmer can type the STEP command on the terminal to let program execution continue on a line-by-line basis. Typing a STOP command in interactive debugging mode terminates program execution, just as if an END statement was encountered in the program.

**RPG II**

PDP-11 RPG II/VAX is an optional language processing system that includes a compiler and an object time library system. It is a language particularly suited to producing printed output. The PDP-11 RPG II/VAX compiler produces code that executes in PDP-11 compatibility mode.

**Language Elements**
RPG II programming involves coding an ordered set of source specifications. Specifications have seven possible formats:

- the Control specifications (H Format), which supply information pertaining to the compilation as a whole
- the File Description specifications (F Format), which describe files to be used by the program
- the Extension specifications (E Format), which describe the tables and arrays to be used by the program and provide for additional file information
- the Line Counter specifications (L Format), which give special information about print output
- the Input specifications (I Format), which describe input records and fields
- the Calculation specifications (C Format), which describe the operations to be performed on previously specified data and define the data fields which are not previously defined
- the Output specifications (O Format), which describe the format of output records and the types of data fields

Figure 7-9 shows several examples of specification statements.

**Special Features**

RPG II provides a set of 31 instructions. These include standard arithmetic instructions, move instructions, compare and GOTO for looping and procedure branching, and instructions for defining and transferring to subroutines. In addition, RPG II provides a number of special features. These include:

- Table and Array Handling Facilities. These include special lookup operations allowing matches on low, high, or equal values; ability to load tables at compile time or as input at execution time, and use of the special arithmetic operation, XFOOT, to sum elements of an array.
- Ability to specify up to nine matching fields to control multiple file processing.
- Ability to access records out of normal sequence via the FORCE and READ operations.
- Control of random record access via the CHAIN operation.
- The ability to specify multiple edit operations via a simple edit code.
Figure 7-9
Sample RPG-II Specifications
7-12
File Organizations and Access Methods
RPG II supports the Sequential, Direct, and Indexed file organizations. Records may be accessed consecutively, sequentially by key, sequentially within limits, randomly by key, randomly by relative record number, and randomly using ADDROUTING files.
ADDROUT files are produced by the SORT utility program. They contain relative address pointers which may be used as indices to access the data in the original file. Since more than one ADDROUT file may be constructed vis-a-vis a single data file, the ADDROUT construct affords a method for accessing that file using several different sequences.

RPG II Utilities
In order to facilitate the coding of RPG II source specifications on the terminal, PDP-11 RPG II/VAX provides a special editor (RPGESEP) which displays form templates, column numbers, and program statements. RPGESEP allows the programmer to "fill in" columns on the display screen as if he were filling in a coding sheet. The programmer selects the appropriate source specification form by specifying the letter format designator (e.g., H for control, I for input, E for extensions, etc.) in the column following the sequence number. RPGESEP performs syntax checking as information is entered at the terminal and displays a message if an error has been committed.
The RPGEPC utility program converts IBM System/3 RPG II source programs into a form which can be processed by the PDP-11 RPG II/VAX compiler. RPGEPC processes an input file of IBM System/3 RPG II statements. The utility automatically changes those fields containing language features supported by the PDP-11 RPG II/VAX compiler with or without a warning message. If an unsupported feature is encountered, the field is left unchanged and a "fatal error" message is issued. If no fatal error has been detected, then RPGEPC creates three files — the converted source file, the original input source file as backup, and a listing file containing source statements and error messages. In the fatal error case, the program will opt to create the three files but the default will be to produce the listing file only.
The RPGEPC utility program converts an IBM System/3 standard labeled or unlabeled magnetic tape, fixed length, EBCDIC format data file with or without packed decimal fields into ASCII format. The input tape must have a density of either 800 or 1600 bpi. The converted output is stored on disk for further processing.

Debugging
RPG II features a special operation code, DEBUG, which will cause the system to generate information helpful in solving program errors. Information listed includes:
- data contained in a specific field
- a list of indicators which are on at the time indicated
The DEBUG operation is designed to be used at critical points in the RPG II source program and is coded on the H (Control) and C (Calculation) specification forms. The information requested will appear on an output file specified on the C form.

BLISS-32
BLISS-32 is a high-level systems implementation language for VAX-11. It is specifically designed for building software such as compilers, real-time processors, and operating systems modules. It contains many of the features associated with high level languages (e.g., DO loops, IF-THEN-ELSE statements, automatic stack, and mechanisms for defining and calling routines), but also provides the flexibility and access to hardware which one would expect from an assembly language. The BLISS-32 compiler runs in native mode under the VAX/VMS operating system.

Features of BLISS-32
BLISS-32 has several characteristics which set it apart from other higher-level languages:
- Data — All BLISS-32 calculations are performed on 32-bit values; the interpretation of any data item is provided by the instruction operator. BLISS-32 has the power to access and assign values to "variable bit" fields (i.e., contiguous fields of from 1 to 32 bits located anywhere in the virtual address space).
- Value Assignments — All names in BLISS-32 represent addresses. Contents of storage locations are accessed by means of a fetch operator (.). Hence, the expression X := Y + 3 is interpreted as adding 3 to the contents of location Y, then assigning the result to the storage location beginning at X.
- Expressions — BLISS-32 permits construction of complex expressions in which several different kinds of operations can be performed in a single program statement. For example, the expression 2*(B = C + 1) calculates 2*(C + 1) and simultaneously assigns the value of C + 1 to B.
- Structures — BLISS-32 defines data structures such as vectors, blocks, bitvectors, and blockvectors. In addition, the programmer can define arbitrary data structures specifically designed for a given application.

Other BLISS-32 features include:
- IF, CASE, SELECT, SELECTONE, and IF-THEN-ELSE sequences — providing for sequencing of instructions based upon evaluation of expressions at run-time.
- DO, WHILE, and UNTIL — providing for looping until a particular condition is satisfied.
- GLOBAL and EXTERNAL data declarations — allowing code to be shared between several modules
- LOCAL, STACKLOCAL, and REGISTER declarations — allowing dynamic stack-like allocation using either the execution stack of the VAX-11 or the VAX-11 general registers.
- REQUIRE and LIBRARY declarations — allowing outside files to be included in the module at compile time.
- LEXICAL functions — allowing a variety of compile-time operations such as concatenation of strings, construction of names, testing properties of macro parameters, testing compiler switch options, generating compiler diagnostic messages, and controlling macro expansion.
BLISS-32 also provides a number of features specifically designed to utilize or interact with the VAX-11 machine or VAX-VMS. These include the following:

- **LINKAGES declarations** allow the programmer to make full use of the VAX-11 call facilities; in particular, he may specify either the CALLS/CALLG/RET or JSB/BSB/RSB call and return sequences, pass parameters in general registers or parameter blocks, and control the use of registers by a routine or across a set of routines.

- **PSELECT declarations** provide information to the linker regarding storage requirements for various sections of a program. For example, a particular data segment may be designated as READ or NOREAD, SHARE or NO-SHARE, LOCAL or GLOBAL, and so on.

- **BUILTIN declarations** allow use of VAX-11 machine-specific functions for access to VAX-11 features not otherwise provided in the BLISS-32 language. The compilation of a machine-specific function results in the generation of internal code, often a single instruction, rather than a call to an external routine. Machine specific functions generally have the same name as their corresponding VAX-11 instructions (e.g., ADAWI, BISP4SW, CRC, HALT, INDEX, MTPR, PROBER, REMQUE, MOV5, etc.). Over 50 such functions are provided. (The complete list is shown in Table 7-2.)

---

### Table 7-2

**VAX-11 Machine-Specific Functions**

<table>
<thead>
<tr>
<th>Processor Register Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MTPR</td>
<td>Move to a Processor Register</td>
</tr>
<tr>
<td>MFPR</td>
<td>Move from a Processor Register</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Validation Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBER</td>
<td>Probe Read accessibility</td>
</tr>
<tr>
<td>PROBEW</td>
<td>Probe Write accessibility</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Status Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVPSL</td>
<td>Move from PSL</td>
</tr>
<tr>
<td>BISP4SW</td>
<td>Bit set PSW</td>
</tr>
<tr>
<td>BICPSW</td>
<td>Bit clear PSW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INSQUE</td>
<td>Insert entry in Queue</td>
</tr>
<tr>
<td>REMQUE</td>
<td>Remove entry from Queue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TESTBITSS</td>
<td>Test for Bit Set, then Set bit</td>
</tr>
<tr>
<td>TESTBITSC</td>
<td>Test for Bit Set, then Clear bit</td>
</tr>
<tr>
<td>TESTBITCS</td>
<td>Test for Bit Clear, then Set bit</td>
</tr>
<tr>
<td>TESTBITCC</td>
<td>Test for Bit Clear, then Clear bit</td>
</tr>
<tr>
<td>TESTBITSSI</td>
<td>Test for Bit Set, then Set bit Interlocked</td>
</tr>
<tr>
<td>TESTBITCCI</td>
<td>Test for Bit Clear, then Clear bit Interlocked</td>
</tr>
<tr>
<td>FFS</td>
<td>Find First Set bit</td>
</tr>
<tr>
<td>FFC</td>
<td>Find First Clear bit</td>
</tr>
</tbody>
</table>

### Extended Arithmetic Operations

- **ASHQ**: Arithmetic Shift Quad
- **EDIV**: Extended Divide
- **EMUL**: Extended Multiply
- **INDEX**: Index (Subscript) Calculation
- **CRC**: Cyclic Redundancy Calculation

### Floating Point Conversion Operations

- **CVTFL**: Convert Long to Floating
- **CVTLD**: Convert Long to Double
- **CVTFL**: Convert Floating to Long
- **CVTDL**: Convert Double to Long
- **CVTFD**: Convert Floating to Double
- **CVTDF**: Convert Double to Floating
- **CVTRDL**: Convert Rounded Double to Long
- **CVTRFL**: Convert Rounded Floating to Long

### String Operations

- **MOVUC**: Move Translated Until Character
- **SCANC**: Scan Characters
- **SPANC**: Span Characters

### Decimal String Operations

- **MOVP**: Move Packed
- **CMPP**: Compare Packed
- **CVTLP**: Convert Long to Packed
- **CVTPL**: Convert Packed to Long
- **CVTPT**: Convert Packed to Trailing Numeric
- **CVTTP**: Convert Trailing Numeric to Packed
- **CVTPS**: Convert Packed to Leading Separate Numeric
- **CVTSP**: Convert Leading Separate Numeric to Packed
- **EDITPC**: Edit Packed to Character

### Miscellaneous Operations

- **HALT**: Halt Processor
- **ROT**: Rotate
- **ADAWI**: Add Aligned Word Interlocked
- **BPT**: Breakpoint
- **CHMx**: Change Mode
- **CALLG**: Call with General Argument List
- **NOP**: No Operation

---

**The BLISS-32 Compiler**

The BLISS-32 compiler performs a number of optimizations. These include: common subexpression elimination, removing loop invariants, constant folding, block register allocation, peephole replacement, test instruction elimination, jump vs. branch instruction, branch chaining, and cross-jumping.

The BLISS-32 compiler optionally produces source text and generated code in a format closely resembling a VAX-11 assembly listing. Other options allow the programmer to control the degree of optimization, suppress production
of object code, determine types and formats of output list-
ings, generate traceback information, and specify the
types of information to be listed at the terminal.

Library and Require Files
BLISS-32 provides two methods for including commonly
used text in BLISS-32 programs at compile time. These in-
volve use of either Require files or Library files:
- Require Files — These are source (text) files which are
  invoked via the REQUIRE declaration in the BLISS-32
  source program.
- Library Files — These are special files created by the
  compiler in a previous library compilation and are in-
voked by the LIBRARY declaration in the BLISS-32
  source program.

Since Library files are "pre-compiled," lexical processing
and declaration parsing and checking need not be repeat-
ed each time these files are included in a compilation;
 hence their usage implies a considerable reduction in total
compilation time.

The contents of Require files must be fully processed each
time the file is used in a compilation. Hence, using Require
files will, in general, be less efficient than using Library
files. However, since these files operate under a somewhat
less stringent set of syntactical rules, their usage may be
warranted in situations where a higher level of flexibility is
desired.

Macros
BLISS-32 provides an extensive macro building facility al-
lowing frequently used groups of declaratives or expres-
sions to be expressed in an abbreviated way. Macros are
defined via MACRO and KEYWORDMACRO declaratives
and are accessed by simple call statements. They are fully
expanded at compile time. BLISS-32 allows parameters to
be specified in the macro definition, thus allowing each
block of text to be specialized by the actual parameters
passed to it. Macros may be simple, iterative, or condition-
al expressions.

Debugging
The BLISS-32 compiler produces a list of error messages
showing the source program line on which the error oc-
curred followed by a description of the error. If the error is
recoverable, then the compiler will generate a "warning"
diagnostic and continue with the compilation process. If
the error is serious enough to invalidate the compiler’s in-
ternal representation of the module, then an "error" diag-
nostic is generated, and processing ceases following the
syntax checking — no object module is produced.

If an error occurs at execution time, then BLISS-32 can ac-
cess the VAX-11 Symbolic Debugger program. This pro-
gram may be accessed when the object module is linked
with the Debug option. The Debug program allows the
programmer to examine and deposit values in storage, set
breakpoints, call routines, trace through a program as it
executes, and perform other operations useful in checking
out a program. (See section on Symbolic Debugger for a
further description of VAX-11 debugging facilities.)

Transportability Features
The BLISS-32 language is designed to facilitate transpor-
tability; that is, the writing of programs that can be execut-
ed on architecturally different machines with little or no
modification. Several language features enhance trans-
portability:
- Higher-level language elements such as IF-THEN-ELSE,
  DO, SELECT, etc., may be transferred from one ma-
  chine to another with little alteration.
- Machine-specific functions can be separated from the
  mainline code via modularization, macros, and Library
  and Require files.
- Parameterization allows machine-specific information
to be passed to BLISS-32 data structures.

BLISS’s transportability makes it an ideal language for
system programming applications — and a viable alterna-
tive to assembly language coding in those applications
in which extreme machine dependence is not involved.

VAX-11 SORT
VAX-11 SORT is a utility and subroutine package designed
to take advantage of the large addressing of the VAX/VMS
operating system. It allows the user to reorder data from
an input file into a new file in a sequence based upon con-
rol or key fields within the input records. SORT can be
used as a separate utility invoked from a terminal, a batch
stream, or as a set of subroutines callable from the VAX-11
languages.

SORT provides four different sorting techniques:
- RECORD — Record sort produces a reordered file sort-
ed by specified key fields (that is, entire records are re-
ordered). A record sort uses any VAX/VMS input device
can process any valid VAX-11 RMS format.
- TAG — Tag sort produces a reordered file by sorting
  only the record keys SORT then randomly reaccesses
  the input file to create a resequenced output file accord-
ing to those record keys. The tag sort method conserves
  temporary storage, but can accept only input files resid-
ing on disk.
- ADDRESS — Address sort produces an address file
  without reordering the input file. The address file, sorted
  by record keys, can later be used as an index to read the
  data base in the desired sequence. Any number of ad-
dress files may be created for the same data base. A
  customer master file, for instance, may be referenced
  by customer-number index or sales territory index for
different reports. This is the fastest of the four sorting
  methods.
- INDEX — Index sort produces an address file containing
  the key field of each data record and a pointer to its lo-
  cation in the input file. The address file can be used by
  programs to randomly access data from the original file
  in the desired sequence. Like address sort, this is a high-
speed process.

VAX-11 SORT Performance Features
VAX-11 SORT can:
- produce reordered data files in ascending or descend-
ing order using up to 10 key fields
- sort sequential, relative, and indexed files
- sort fixed, variable, and VFC (variable with fixed control)
  records
• sort Character (in ASCII collating sequence), Decimal, and Binary data types
• sort input files from any VAX/VMS input device
• write sorted data files on any VAX/VMS output device
• determine its own work file requirements based on input file characteristics
• be called from COBOL, FORTRAN, and MACRO

Command String and Specification File
VAX-11 SORT is invoked via the VAX/VMS Digital Command Language (DCL) command SORT. The command statement contains a string of qualifiers which control the sort operation. These qualifiers indicate the type of sort to be performed (RECORD is the default), define sort keys, specify the number of work files to be used, and describe format, organization, and other characteristics of input and output files.

Use of the /SPECIFICATION: qualifier allows the the SORT utility to be controlled by sort specification statements contained in a separately defined specification file. This option allows dynamic program control of specification file statements, and therefore dynamic control of subsequent sorts using the same specification file modified. Also, specification file libraries can be maintained for often used sorts.

Sort As a Callable Routine
VAX-11 SORT can be called by COBOL, FORTRAN, and MACRO. Two interfaces may be used, File I/O Interface and Record I/O Interface. The File I/O Interface allows the user to specify an input file and an output file to SORT. SORT then reads the data from the input file and sorts it into the output file. The Record I/O Interface allows the user to pass each individual record to SORT. SORT then orders them and passes each record back in the correct order, individually, to the calling program. Both interfaces share a common set of routines, and the same calls are used for all languages. These calls and their functions are shown in Figure 7-10.

Routine Name  Function
1. SOR$INIT.Sort  Initialize scratch files, work area, sorting parameters.
2. SOR$PASS_FILES  Pass a file name to SORT.
3. SOR$RELEASE_REC  Pass a record to SORT.
4. SOR$SORT_MERGE  Initiate sorting and intermediate merging of records.
5. SOR$RETURN_REC  Initiate final merge pass and receive output record from SORT.
6. SOR$END_SORT  Allow clean-up of files and work area to complete the sort operation.

Sort Statistics
A set of statistics is printed automatically at the completion of each sort operation. These consist of elapsed execution time, the number of records read, sorted and output, the longest record length, the multiblock count used and the multibuffer count used for input and output, the merge order, the number of merge passes, the working set size used, the number of initial runs and the virtual memory used for the sort tree.

In addition, SORT statistics include statistics kept by VAX/VMS for the number of buffered and direct I/O operations, CPU time, and the number of page faults. Figure 7-11 illustrates a typical sort statistics printout.

These statistics can be used to tune the parameters used in a particular sort, such as working set size, number of work files, and other variables.

SORT STATISTICS
RECORDS READ: 10000
RECORDS SORTED: 10000
RECORDS OUTPUT: 10000
LONGEST RECORD LENGTH: 80
INPUT MULTI BLOCK COUNT: 11
INPUT MULTI BUFFER COUNT: 2
OUTPUT MULTI BLOCK COUNT: 20
OUTPUT MULTI BUFFER COUNT: 2
ORDER OF MERGE: 7
NUMBER OF MERGE PASSES: 2
NUMBER OF INITIAL RUNS: 38
MAXIMUM WORKING SET USED: 128
DYNAMIC VIRTUAL MEMORY ALLOCATED: 236032
ELAPSED TIME: 00:01:26:40

BUFFERED IO COUNT: 23
DIRECT IO COUNT: 27
CPU TIME: 5055
PAGE FAULTS: 15596

Figure 7-11
Sort Statistics Listing

PDP-11 DATATRIEVE/VAX
PDP-11 DATATRIEVE/VAX is a software product that provides users with the ability rapidly to extract and display data from RMS-11 files; it operates in PDP-11 compatibility mode under VAX/VMS. DATATRIEVE's facilities include the following:
• Inquiry and Update capability — allowing interactive data retrieval, sort, and file modification
• Report generation — producing summary and detailed reports, headings, footnotes, group totals, and report totals
• Data Definition — allowing creation, maintenance, and access of a data dictionary that defines RMS records and DATATRIEVE command procedures

DATATRIEVE operates on sequential, indexed, or relative files, and will perform selective retrieval on keyed or non-keyed fields.

Data Structures
PDP-11 DATATRIEVE/VAX uses a set of data structures called "record structures," "domains," and "collections." Record structures describe the formats of records in particular files and are similar to data definitions in COBOL. (A
typical record structure is shown in Figure 7-12.) Domains
are named groups of data containing records of a single
type. When using DATATRIVE with RMS-11, domains
and files may be regarded as synonymous; DATATRIVE
will always refer to the domain name. Collections are sub-
sets of domains consisting of records which share com-
mon characteristics (e.g., all employees with incomes
between $15,000 and $20,000); collections may consist of
no records, a single record, or many records, up to and in-
cluding the entire domain. DATATRIVE operates on col-
clections of records from RMS-11 files rather than on the
files themselves.

QL: SHOW YACHT
RECORD YACHT
USING
01 BOAT
03 TYPE.
06 MANUFACTURER PIC X(10)
QUERY-NAME IS BUILDER.
06 MODEL PIC X(10)
03 SPECIFICATIONS
QUERY-NAME SPECIFICATIONS.
06 RIG PIC X(6).
QUERY-NAME IS LOA.

Figure 7-12
Typical DATATRIVE Record Structure

Data definitions are maintained by DATATRIVE in the Da-
ta Dictionary. PDP-11 DATATRIVE/VAX also provides
commands to list the contents of the Data Dictionary, to
delete entries, and to control access to individual entries in
the Data Dictionary.

Command Language
In order to perform its query, file maintenance, report writ-
ing and other activities, PDP-11 DATATRIVE/VAX uses a
special command language. The commands are simple,
follow an English-type syntax, and are designed to be
learned easily by individuals with little or no programming
experience. Typical command statements might be
FIND SCREWS WITH PART-NUM = 223,
utilizing the FIND command, or
SELECT 7,
which utilizes the SELECT command to select the seventh
record in the collection.

PDP-11 DATATRIVE/VAX also provides more sophisti-
cated commands such as the IF THEN ELSE sequence,
and the BEGIN...END block, which allows groups of com-
mands to be combined into a single compound command
statement.

DATATRIVE Inquiry and Update Facilities
PDP-11 DATATRIVE/VAX's Inquiry and Update facilities
include the following features:

- Ability to reorder a collection via the SORT command.
- Ability to select a single record in a collection via the SE-
  LECT command.
- Ability to print the records of a collection via the PRINT
  command.

DATATRIVE Report Writer Facilities
PDP-11 DATATRIVE/VAX allows the user great discre-
tion in specifying formats and contents of printed reports.
In particular, the user can specify:

- report/page headers and trailers
- multiple control breaks
- automatic statistical functions (MAX, MIN, AVERAGE,
  TOTAL, COUNT)
- page and line limitations
- control headings and footings
- page totals, control totals, or report totals
- detail or summary reports

DATATRIVE inquiry and report writer statements may be
intermixed during a given session, the only restriction be-
ing that an inquiry statement must not occur within a re-
port writer sequence.

Data Protection
The PDP-11 DATATRIVE/VAX user can regulate access
to domains, records, and procedures through access re-
quirements recorded with the definitions in the Data Di-
cictionary. There are five types of access, called the user’s
“privileges.” These privileges are defined as follows:

- READ: User can only retrieve this resource (i.e., “read”
  the domain, record, or procedure).
- EXTEND: User can only add records to this resource.
- MODIFY: User can retrieve or change records in this re-
source.
- WRITE: User can do all of the above and can also erase
  records.
- CONTROL: User can issue data protection commands.

DATATRIVE controls access based upon the user’s pass-
word and UIC (user identification code). Hence one user
might have the ability to modify a particular domain, while
another might be permitted only to read the records in that
domain.

VAX-11 PROGRAM DEVELOPMENT TOOLS
The VAX-11/780 program development tools include text
editors, a linker, a librarian, a common run-time pro-
cedure library, and a debugger. These tools are available
to the programmer through the VAX/VMS command lan-
guage, as are the language compilers themselves.

The text editors can be used to create memos, documen-
tation, and data files, as well as source program modules
for any language processor. The linker, librarian, debug-
ger, and run-time procedure library described below are
used only in conjunction with the language processors that
produce native code. The language processors that pro-
duce compatibility mode code offer their own task
building, library, and debugging facilities, and they each
include their own object time system libraries.
Editors
The programmer can use either or both of two text editors: SOS and SLP. SOS is an interactive text editor that enables the programmer to create and modify text files using commands entered from either a hard-copy or video terminal. The user can insert, delete, and replace lines, find and substitute strings, or modify the text a character at a time. Lines can be identified by line number, relative position, or by contents. An adjacent group of lines can be copied or transferred from one place to another. Editing can be done in any order in the file. Editor parameters can be set to user-specified values and the current values can be shown. User-specific parameters can be set automatically at editor startup.

SLP is a programmed text editor that enables a user to modify an existing file by supplying a command file containing a list of the modifications to be made. The command file provides a reliable way to duplicate the changes made to a file at a later time or on another system. SLP provides a formal record of changes made to files, both in the source file and in an audit trail listing — a feature useful in tracking the stages of large programming projects.

Linker
The VAX/VMS linker accepts one or more native object modules produced by an assembler or compiler, resolves the symbols and procedure references between them, and produces an executable program image. The VAX/VMS linker also enables a programmer to create sharable images that can be linked subsequently with other modules to produce an executable image. Furthermore, the linker not only accepts object modules to produce executable or sharable images, it can also accept object module libraries, sharable images, and sharable image libraries.

The linker's ability to produce and use sharable images reduces program development time in much the same way as its ability to produce and use previously compiled object modules and object module libraries: it saves link time itself, since internal references in sharable images have been previously resolved, and it enables a programmer to take advantage of sharable code at both link time and run time. (For more information on sharable images, refer to the Operating System section.)

The linker has several other distinguishing features:
• It has no knowledge of how physical memory is allocated. Its concern is allocation of virtual addresses to an image.
• It keeps both an image file and the amount of virtual address space an image requires as small as possible.
• It allows the programmer to separate private code and data from potentially sharable code and data.
• It allows the programmer to separate code from read-only data and read/write data so that memory protection can be used to prevent and isolate programming errors.
• It allows both weak definitions and weak references for global symbols. Weak definitions and weak references are especially useful in creating libraries and language processors that use libraries.

Librarian
The librarian enables a programmer to create, update, modify, list and maintain library files. A library file can be a collection of object modules or sharable images. A programmer can request the linker to use one or more library files from which the linker can obtain modules to resolve references during linking.

Common Run-time Procedure Library
The run-time procedure library is a collection of general purpose and language-specific libraries available to any native program, regardless of the source language in which the program was written. The run-time library is a sharable image that allows:
• the choice of incorporating procedures from the library into an executable image, or mapping the global sections into a process virtual address space at run time
• a single copy of the library to be shared by all processes
• installation of a new library without the need to relink existing programs

The run-time library includes the following:
• a mathematical library
• a general utility library
• a condition handling facilities library
• a language-independent support library
• a FORTRAN IV-PLUS language specific support library

Symbolic Debugger
The debugger can be linked with a native program image to control program execution during development. The debugger can be used interactively or it can be controlled from a command procedure file. The debugging language is similar to the VAX/VMS command language. Expressions and data references are similar to those of the source language used to create the image being debugged. Debugging commands include the ability to start and interrupt program execution, to step through instruction sequences, to call routines, to set break or trace points, to set default modes, to define symbols, and to deposit, examine, or evaluate virtual memory locations.

RSX-11M PROGRAM DEVELOPMENT
Software bundled within the operating system allows VAX/VMS users to write, assemble or compile, and link RSX-11M task images. The task images can be written to execute on an RSX-11M or RSX-11S system or, if properly coded, they can be written to execute in the VAX-11/780 compatibility mode environment. (Refer to the Processor and Operating System sections for descriptions of compatibility mode.)

Programming Languages
The user can develop RSX-11M programs in several languages. The PDP-11 MACRO assembler, included in the operating system, makes RSX-11M executive directives and the PDP-11 instruction set directly available. In addition, the the PDP-11 BASIC-PLUS-2/VAX and PDP-11 COBOL/VAX compilers can be used to create task images for execution on RSX-11M or RSX-11S systems.

The FORTRAN-IV/VAX Cross-Development Compiler is an optional product which allows users to develop
FORTRAN IV programs for the RSX-11M and RSX-11S systems. The FORTRAN IV language allows the programmer to use RSX-11M directives. FORTRAN IV object modules can be linked with other FORTRAN IV or PDP-11 MACRO object modules into executable task images. FORTRAN IV includes an object time system (a library of commonly used FORTRAN routines such as math, error handling, and process I/O routines) as well as the RSX-11M executive directives.

RSX-11M Program Development Tools
VAX programmers can use the RSX-11M MCR command interface on the VAX-11/780 system. MCR enables the programmer to run the standard RSX-11M utilities such as the EDI and SLP editors, the PIP and FLX file transfer utilities, as well as the task builder, librarian, and patch utilities.

The RSX-11M task builder creates loadable task images from object modules created by the PDP-11 MACRO assembler and/or PDP-11 FORTRAN IV compiler, the PDP-11 COBOL compiler, or the PDP-11 BASIC-PLUS-2 compiler. It links relocatable object modules and resolves any references to global symbols, common areas, and shared libraries. The RSX-11M librarian provides the user with the ability to create and maintain disk-resident libraries of object modules and user-defined macros.

The RSX-11M on-line debugger (ODT) can be linked with PDP-11 MACRO or FORTRAN IV task images to aid in debugging programs. The RSX-11M ZAP task patch utility is used to examine and modify 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. The RSX-11M PAT object module patch utility allows the patching or updating of code in a relocatable binary object module.
ENVIRONMENT DIVISION.
[INPUT-OUTPUT SECTION.]
FILE-CONTROL.
SELECT file-name
  ASSIGN TO device-name-1 [, device-name-2] ...
  ORGANIZATION IS INDEXED
  [ ; ACCESS MODE IS {SEQUENTIAL RANDOM DYNAMIC} ]
  ; RECORD KEY IS data-name-1
  [ ; ALTERNATE RECORD KEY IS data-name-2 [WITH DUPLICATES] ] ...

DATA DIVISION.
[FILE SECTION.]
FD file-name
  [ ; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS CHARACTERS} ]
  [ ; RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
  ; LABEL {RECORD IS } {STANDARD}
  {RECORDS ARE} {OMITTED}
  [ ; DATA {RECORD IS } {RECORDS ARE} data-name-3 [ , data-name-4] ... ]

PROCEDURE DIVISION.
OPEN { INPUT file-name-1 [, file-name-2] ... } ...
  OUTPUT file-name-3 [, file-name-4] ... }
  I-O file-name-5 [, file-name-6] ...
VAX/VMS data management includes a file system that provides volume structuring and protection, and record management services that provide device-independent access to the VAX-11/780 peripherals.

The VAX/VMS on-disk structure provides a multiple-level hierarchy of named directories and subdirectories. Files can extend across multiple volumes and be as large as the volume set on which they reside. Volumes are mounted to identify them to the system. VAX/VMS also supports multi-volume ANS format magnetic tape files with transparent volume switching.

The VAX/VMS record management input/output system (RMS) provides device independent access to disks, tapes, unit record equipment, terminals, and mailboxes. RMS allows users and application programs to create, access, and maintain data files with efficiency and economy. Under RMS, records are regarded by the user program as logical data units that are structured and accessed in accordance with application requirements.

RMS provides sequential record access to sequential file organizations, and sequential, random, or combined record access to relative file organizations. Compatibility mode application programs can also process multi-key indexed files sequentially, randomly, or in combination using index keys. Multi-key indexed file processing includes incremental reorganization.
The following paragraphs discuss some of the features and functions of the file system, including the file structures, file naming facilities, and the file management utility programs. The remainder of this section describes the record management services programming environment.

**FILE MANAGEMENT**

VAX/VMS provides two file structures: one for disk volumes and one for magnetic tape volumes. From the user's point of view, the only differences between the two file structures are those imposed by the capabilities of the media. Volumes are mounted for identification, and files can extend across multiple volumes. The practical limit to file size is that they can be only as large as the volume set on which they reside.

Volume and file protection is based on User Identification Codes (UICs) assigned to accessors and the file or volume. The UICs establish the accessor's relationship to the data structure as Owner, the owner's Group, the System, or the World (all others). Depending on the relationship, the accessor may or may not have read, write, execute, or delete access to any given file.

Disk volumes are multi-user volumes. They can contain a multi-level directory hierarchy that is defined dynamically by the users of the volume. The on-disk file structure appears to a program to be a virtually contiguous set of blocks. The blocks of the file, however, may be scattered anywhere on a volume. Mapping information is maintained to identify all the blocks constituting a file. Figure 8-1 illustrates the file structure.

**Figure 8-1**

*Disk File Structure*
Disk files can be extended easily. The blocks of the file are allocated in physically contiguous sets, called **extents**. Users are not required to preallocate space, although they can do so. Users can specify placement on an allocation request, and they can control automatic allocation. For example, when a file is automatically extended, it can be extended by any given number of contiguous blocks. If desired, a file can be created as a contiguous file, in which case it is both virtually and physically contiguous.

The disk structure includes duplicates of its critical volume information. The system detects bad disk blocks dynamically and prevents re-use once the files to which they are allocated are deleted.

Magnetic tape volumes are single-user volumes. Magnetic tape files consist of physically contiguous blocks. Record blocking is under program control. Files have ANSI format labels. VAX/VMS also supports unlabeled (non-file structured) magnetic tapes.

**File Directories and Directory Structures**

A directory is a file containing a list of files on a given volume. A directory entry contains the name, type, version, and unique file ID for a particular file. A directory can list files having the same owner UIC or files having different owner UICs. The entries are listed alphabetically.

A disk volume contains at least one directory, called the master file directory. The system manager is responsible for creating a volume's master file directory. The master file directory can (and normally does) contain a list of directory files which form a second level of directories. The second level of directory files can list data files and/or other directory files, called **subdirectories**. Users can create subdirectories within the directories they own. The subdirectories can also list other directory files and/or data files. Figure 8-2 illustrates a multi-level directory structure.

Since directories of files on volumes are files themselves, they are assigned owner UICs and can be protected from certain kinds of access depending on the relationship established by an accessor's UIC. In the special case of directory files, the file protection fields control an accessor's ability to:

- look up files
- enter new files in the directory, including new versions of existing files
- remove files from the directory

**File Specifications**

A **file specification** identifies which file is to be used in a file processing operation. Programs use file specifications to identify the file they want to create, access, delete, or extend, and users supply the command interpreter with a file specification to identify the file they want to edit, compile, copy, or delete, etc. A complete file specification is a well-defined character string composed of the following fields:

- **Node Name** — The node of the network in which the volume containing the file is stored. The node name is followed by two colons (:) to delimit it from the remainder of the file specification.

- **Device Name** — The device on which the volume containing the file is mounted. The device name is followed by a single colon (:) to delimit it from the remainder of the file specification.

- **Directory Name** — The directory in which the file is listed. A directory name begins with an opening bracket (< or [) and ends with a closing bracket (> or ]). If the file is listed in a subdirectory, the directories to be searched are listed in the desired search order, with the names separated by periods, e.g.: [name1.name2.name3]
• **File Name** — The user-assigned name of the file.

• **File Type** — The type identification for the file. The type is preceded by a period (.) to delimit it from the remainder of the file specification.

• **File Version** — the generation number of the file. The file version is preceded by a semicolon (;) or period (.) to delimit it from the remainder of the file specification.

For example, a complete file specification might be:

```
NODE47::DBA1:[JONES]HANOI.FOR;2
```

In this case, NODE47 is the name of the network node, DBA1 is the name of the device (DB for disk pack device, A for disk controller, 1 for drive unit number), [JONES] is the directory name, HANOI is the file name, FOR is the file type (meaning that the file is a FORTRAN source file), and 2 is the version number.

Neither programs nor command language users need to provide a complete file specification to identify files. The system applies defaults to most fields of a file specification when they are not present. For example, if the node name is not present, the node is assumed to be the node on which the program is executing. If the version number is not present, the version is always assumed to be the latest version. Device name and directory name defaults for users and the programs they execute are supplied by the system manager in the user authorization file, and users can change the standard defaults at any time during their session on the system.

Some commands (such as COPY, PRINT, and DELETE) accept a wild card in one or more fields of a file specification. A wild card is an asterisk appearing in a file specification field that means "all."

File specifications also apply to non-file structured devices such as line printers, card readers, and terminals. In these cases, however, the user or program needs to supply only the node name and device name, as appropriate.

**Logical File Naming**

To provide both system and device independence, users and programs are not limited to identifying files by their file specifications. They can use logical names in place of a complete file specification, or in place of a portion of a file specification. For example, a user can assign a logical name to the left-most three fields of a file specification:

```
$ ASSIGN NODE47::DBA4:[JONES] to VOL
```

And then use the logical name VOL in a subsequent command:

```
$ TYPE VOL:HANOI.FOR
```

Defaults also apply when translating logical names, so that the user could have made the assignment:

```
$ ASSIGN NODE47::[JONES] to VOL
```

In this case, the user's default device name would be used to derive the complete file specification.

Logical name assignments can be made on a process, group, or system wide basis. Logical names can also be recursive, that is, a logical name can be assigned to another logical name, or to a logical name and a portion of a file specification.

For example, suppose a company's weekly payroll production run includes an application program that uses the current week's payroll changes data file. That data file may be located in the directory named [PAYROLL] one week, or in the payroll backup subdirectory, [PAYROLL.BACKUP], another week. The volume on which the file is stored may be mounted on disk pack drive unit number 1 one week, or on unit 2 another week.

The application programmer can write the program without knowing which directory the data file is listed in, or which drive the volume is mounted on. A series of logical name assignments provides the complete file specification. The assignments are the responsibility of the people who know what directory the file is listed in, and what drive the volume is mounted on.

In the example shown in Figure 8-3, the application program contains an OPEN statement for the payroll data file using the logical name WEEKLY_PAYROLL_CHANGES (note that underscore is a legal character). The application systems designer has created a command procedure file

```
Application Programmer:
OPEN ("WEEKLY_PAYROLL_CHANGES")

Application System Programmer:
Command Procedure: PAY_RUN.COM
accepts one parameter (P1): Week Number

$ ASSIGN WEEKLY_PAYROLL:'P1'.WPY "WEEKLY_PAYROLL_CHANGES"
$ RUN APPLICATION

Production Clerk:
$ @PAY_RUN WEEK09

Payroll Group Operations Manager:
$ ASSIGN/GROUP PAY_PACK:[PAYROLL.BACKUP] "WEEKLY_PAYROLL"

Local Operator:
$ ASSIGN/SYSTEM DBA2: PAY Pack
```

**Figure 8-3**

Logical Name Assignment

8-3
called PAY_RUN that controls the production run. The command procedure file includes a logical name assignment that obtains the file name as a parameter supplied by the operator or production clerk who starts the production run. The logical name used by the application program is given a value that consists of another logical name (WEEKLY_PAYROLL) and the file name and type specifications.

To complete the series of logical name assignments, the payroll group operations manager makes a group-wide logical name assignment: the payroll data files this week are stored in the PAYROLL.BACKUP subdirectory. The logical name assignment provides the directory name, using another logical name (PAY_PACK) known to the operator who mounts the payroll data files volume. The operator makes the system-wide logical name assignment when mounting the pack before the production run. Given the assignments shown in the example, the logical name used to open the file is translated to:

DBA2:[PAYROLL.BACKUP]WEEK09.WPY

(The local system node name and the latest version number are used as defaults to complete the file specification.) Should the directory name change, or the pack be mounted on another device that day, the only changes made are the logical name assignments. There is no need to modify either the application program or the command procedure controlling the production run.

**File Management**

The VAX/VMS system includes many services that aid in data management and maintenance. Some of these are described in the following paragraphs.

**Sorting Files**

The SORT program allows the user to rearrange, delete, and reformat records in a file. The user can arrange the records in the ascending or descending sequence of one or more fields within the records for subsequent sequential processing. SORT can also create several different index files for accessing a file according to these indices 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, 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.

- **ADDROUT Sort** produces an ordered address file that enables the user to access the records in that data file in the order of the address file.

- **Index Sort** produces an ordered key file that can be used to sequentially or randomly access records in the data file. SORT accepts two kinds of command formats: a simple keyboard-oriented command string, and a conventional specification file format. The latter format allows input record selection techniques based on user-defined record characteristics and field specifications. In addition, the input records can vary in format and the output records can be restructured.

**Comparing Files**

A file differences command contrasts two files by automatically aligning matching text, and optionally ignoring comments, empty records, trailing blanks, or multiple blanks. The output can be a file-by-file list of differences, an interleaved list of differences, a list with change bars, or a batch editor command input file.

**Backing Up Files and Volumes**

The Disk Save and Compress (DSC) utility enables a user to backup entire disk volumes to magnetic tape or to other disks. When backing up disk volumes to other disk volumes, or restoring disk volumes from magnetic tape, DSC combines unused blocks on disks into contiguous areas.

**Verifying File Structures**

The file verification utility checks the consistency and accuracy of the file structure on a Files-11 disk volume. It can also display the number of available blocks in a volume, locate files that could not otherwise be accessed, and list the names of files on the volume.

**Bad Block Locator**

The bad block locator utility determines the number and location of bad blocks on Files-11 disk volumes and stores this information in the bad block file on the volume so that the blocks can not be located. Running this utility before initializing a Files-11 volume is useful in ensuring a disk's integrity.

**RMS Utilities**

The record management services procedures are complemented by a number of utilities designed especially for RMS file creation and maintenance. They allow the user to:

- Create an RMS file and define the attributes of the file.
- List the attributes of a single file or a group of files, or list the contents of a backup magnetic tape.
- Convert a file with any file organization or record format to a file with any other file organization or record format.
- Backup a single file or group of files in a compact format (optionally by creation or revision date).
- Restore files previously backed up (optionally by creation or revision date).

**RECORD MANAGEMENT SERVICES**

The record management services (RMS) are a set of system procedures that provide efficient and flexible facilities for data storage, retrieval, and modification. When writing programs, the user can select processing methods from among the RMS file organizations and accessing techniques. The following sections discuss the RMS:

- file organizations
- file attributes
- program operations
- run-time environment

The manner in which RMS builds a file is called its organization. RMS provides three file organizations:

- sequential
- relative
- indexed
The sequential and relative file organizations can be processed using native and compatibility mode programming languages. The indexed file organization can be processed using compatibility mode programming languages only.

The sequential and relative file organizations can be processed using native and compatibility mode programming languages. The indexed file organization can be processed using compatibility mode programming languages only.

The organization of a file establishes the techniques one can use to retrieve and store data in the file. These techniques are known as record access modes. The record access modes that RMS supports are:

- sequential
- random
- Record's File Address (RFA)

An application program or a RMS utility can be used when creating a RMS file to specify the organization and characteristics of the file. Among the attributes specified are:

- storage medium
- file name and protection specifications
- record format and size
- file allocation information

After RMS creates a file according to the specified attributes, application programs can store, retrieve and modify the data. These program operations take place on the logical records in a file or the blocks comprising the file.

RMS FILE ORGANIZATIONS

A file is a collection of related information. For example, a file might contain a company’s personnel information (employee names, addresses, job titles). Within this file, the information is divided into records. All the information on a single employee might constitute a single record. Each record in the personnel file would be subdivided into discrete pieces of information known as fields. The user defines the number, locations within the record, and logical interpretations of these fields.

The user can completely control the grouping of fields into records and records into files. The relationship among fields and records is embedded in the logic of the programs. RMS does not know the logical relationships that exist within the information in the files.

RMS ensures that every record written into a file can subsequently be retrieved and passed to a requesting program as a single logical unit of data. The structure, or organization, of a file establishes the manner in which RMS stores and retrieves records. The way a program requests the storage or retrieval of records is known as the record access mode. The organization of a file determines which record access modes can be used.

Sequential File Organization

In sequential file organization, records appear in consecutive sequence. The order in which records appear is the order in which the records were originally written to the file by an application program or RMS utility. Sequential organization is the only file organization permitted for magnetic tape and unit record devices. Figure 8-4 illustrates sequential file organization.

Relative File Organization

When relative organization is selected, RMS structures a file as a series of fixed-size record cells. Cell size is based on the maximum length permitted for a record in the file. These cells are numbered from 1 (the first) to n (the last). A cell's number represents its location relative to the beginning of the file. Each cell in a relative file can contain a single record. There is no requirement, however, that every cell contain a record. Empty cells can be interspersed among cells containing records. Figure 8-5 illustrates a relative file organization.

Since cell numbers in a relative file are unique, they can be used to identify both a cell and the record (if any) occupying that cell. Thus, record number 1 occupies the first cell in the file, record number 17 occupies the seventeenth cell, and so forth. When a cell number is used to identify a record, it is also known as a relative record number.

![Sequential File Organization](image1)

![Relative File Organization](image2)
Indexed File Organization

The location of records in indexed file organization is transparent to the program. RMS completely controls the placement of records in an indexed file. The presence of keys in the records of the file governs this placement. A key is a field present in every record of an indexed file. The location and length of this field are identical in all records. When creating an indexed file, the user decides which field or fields in the file’s records are to be a key. Selecting such fields indicates to RMS that the contents (i.e., key value) of those fields in any particular record written to the file can be used by a program to identify that record for subsequent retrieval.

At least one key must be defined for an indexed file: the primary key. Optionally, additional keys or alternate keys can be defined. An alternate key value can also be used as a means of identifying a record for retrieval.

As programs write records into an indexed file, RMS builds a tree-structured table known as an index. An index consists of a series of entries containing a key value copied from a record that a program wrote into the file. Stored with each key value is a pointer to the location in the file of the record from which the value was copied. RMS builds and maintains a separate index for each key defined for the file. Each index is stored in the file. Thus, every indexed file contains at least one index, the primary key index. Figure 8-6 illustrates an indexed file organization with a primary key. When alternate keys are defined, RMS builds and stores an additional index for each alternate key.

RMS RECORD ACCESS MODES

The methods of retrieving and storing records in a file are called record access modes. A different record access mode can be used to process records within the file each time it is opened. A program can also change record access mode during the processing of a file. RMS permits only certain combinations of file organization and record access mode. Table 8-1 lists these combinations.

Table 8-1
Record Access Modes and File Organizations

<table>
<thead>
<tr>
<th>File Organization</th>
<th>Record Access Mode</th>
<th>Record</th>
<th>Key Value</th>
<th>RFA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Relative</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Indexed</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1Disk files only.
2Fixed length record format disk files only.

Sequential Record Access Mode

Sequential record access mode can be used to access all RMS files and all record-oriented devices, including mailboxes. Sequential record access means that records are retrieved or written in the sequence established by the organization of the file.

Sequential Access to Sequential Files — When using sequential record access mode in a sequentially organized file, physical arrangement establishes the order in which records are retrieved. To read a particular record in a file, say the fifteenth record, a program must open the file and access the first fourteen records before accessing the desired record. Thus each record in a sequential file can be retrieved only by first accessing all records that physically precede it. Similarly, once a program has retrieved the fifteenth record, it can read all the remaining records (from the sixteenth on) in physical sequence. It cannot, however, read any preceding record without closing and reopening the file and beginning again with the first record.

Sequential Record Access to Relative Files — During the sequential access of records in the relative file organization, the contents of the record cells in the file establish the order in which a program processes records. RMS recognizes whether successively numbered record cells are empty or contain records.

When a program issues read requests in sequential record access mode for a relative file, RMS ignores empty record cells and searches successive cells for the first one con-
taining a record. When a program adds new records in sequential record access mode to a relative file, RMS places a record in the cell whose relative number is one higher than the relative number of the previous request, as long as that cell does not already contain a record. RMS allows a program to write new records only into empty cells in the file.

*Sequential Record Access to Indexed Files* — A program can use the sequential record access mode to retrieve records from an indexed file in the order represented by any index. The entries in an index are arranged in ascending order by key values. If more than one key is defined for the file, each separate index associated with a key represents a different logical ordering of the records in the file.

When reading records in sequential record access mode from an indexed file, a program initially specifies a key (primary key, first alternate key, second alternate key, etc.) to RMS. Thereafter, RMS uses the index associated with that specified key to retrieve records in the sequence represented by the entries in the index. Each successive record RMS returns in response to a read request contains a value in the specified key field that is equal to or greater than that of the previous record returned.

When writing records to an indexed file, RMS uses the definition of the primary key field to place the record in the file.

**Random Record Access Mode**

In random record access mode, the program establishes the order in which records are processed. Each program request for access to a record operates independently of the previous record accessed. Each request in random record access mode identifies the particular record of interest. Successive requests in random mode can identify and access records anywhere in the file.

*Random Record Access to Sequential Files* — Native programs can access sequential files on disk using relative record number to randomly locate a record, provided that the records are in fixed-length record format.

*Random Record Access to Relative Files* — Programs can read or write records in a relative file by specifying relative record number. RMS interprets each number as the corresponding cell in the file. A program can read records at random by successively requesting, for example, record number 47, record number 11, record number 31, and so forth. If no record exists in a specified cell, RMS notifies the requesting program. Similarly, a program can store records in a relative file by identifying the cell in the file that a record is to occupy. If a program attempts to write a new record in a cell already containing a record, RMS notifies the program.

*Random Record Access to Indexed Files* — For indexed files, a key value rather than a relative record number identifies the record. Each program read request in random record access mode specifies a key value and the index (primary index, first alternate index, second alternate index, etc.) that RMS must search. When RMS finds the key value in the specified index, it reads the record that the index entry points to and passes the record to the user program.

Program requests to write records randomly in an indexed file do not require the separate specification of a key value. All key values (primary and, if any, alternate key values) are in the record itself. When an indexed file is opened, RMS retrieves all definitions stored in the file. RMS knows the location and length of each key field in a record. Before writing a record into the file, RMS examines the values contained in the key fields and creates new entries in the indices. In this way RMS ensures that the record can be retrieved by any of its key values.

**Record’s File Address (RFA) Record Access Mode**

Record’s File Address (RFA) record access mode can be used to retrieve records in any file organization as long as the file resides on a disk volume. Like random record access mode, RFA record access allows a specific record to be identified for retrieval, using the record’s unique address. The actual format of this address depends on the organization of the file.

After every successful read or write operation, RMS returns the RFA of the subject record to the program. The program can then save this RFA to use again to retrieve the same record. It is not required that this RFA be used only during the current execution of the program. RFAs can be saved and used at any subsequent time.

**Dynamic Access**

Dynamic access is not strictly an access mode. It is the ability to switch from one record access mode to another while processing a file. For example, a program can access a record randomly, then switch to sequential record access mode for processing subsequent records. There is no limitation on the number of times such switching can occur. The only limitation is that the file organization must support the record access mode selected.
FILE AND RECORD ATTRIBUTES

When creating an RMS file, a program or user defines its logical and physical characteristics, or attributes. These characteristics are defined by source language statements in an application program or by an RMS utility. The program or user assigns the file a name, the owner’s User Identification Code, and a protection code, and selects the file organization. The program or user also defines or selects other attributes, including:

- device
- file size
- file location
- record format and size
- keys (for indexed files only)

Selection of device is related to the organization of the file. Sequential files can be created on Files-11 disk volumes or ANS magnetic tape volumes. Sequential files can also be read from mailboxes, terminals, and card readers, and written to mailboxes, terminals, and line printers. Relative and indexed files can be created on Files-11 disk volumes.

The logical limit on file size is $2^{24}$ blocks, with a more realistic limit being the set of blocks on which a file can reside. When creating an RMS file on a disk volume, the user can specify an initial allocation size. If no file size is given, RMS allocates the minimum amount of storage needed to contain the defined attributes of the file. The initial size can be extended dynamically. The user can let RMS locate the file, or the user can allocate the file to specific locations on the disk to optimize disk access time. The file’s starting location can be specified optionally using a volume-relative block number, or a physical cylinder address.

When creating a file on a magnetic tape volume, a user or program does not specify an initial allocation size. The blocks are simply written one after another down the tape, beginning after the last file, if any, written on the tape. Once a tape file has been created, another file can replace it or be appended to it, but all subsequent files on the tape, if any, are lost.

Record Formats

The user provides the format and maximum size specifications for the records the file will contain. The specified format establishes how each record appears in the file. The size specification allows RMS to verify that records written into the file do not exceed the length specified when the file was created.

Fixed length record format refers to records of a file that are all equal in size. Each record occupies an identical amount of space in the file. All file organizations support fixed length record format.

Variable-length record format records can be either equal or unequal in length. All file organizations support variable-length record format. RMS prefixes a count field to each variable-length record it writes. The count field describes the length (in bytes) of the record. RMS removes this count field before it passes a record to the program. RMS produces two types of count fields, depending on the storage medium on which the file resides:

- Variable-length records in files on Files-11 disk volumes have a 2-byte binary count field preceding the data field portion of each record. The specified size excludes the count field.
- Variable-length records on ANS magnetic tapes have 4-character decimal count fields preceding the data portion of each record. The specified size includes the count field. In the context of ANS tapes, this record format is known as D format.

Variable-with-fixed-control (VFC) records consist of two distinct parts, the fixed control area and a variable-length data record. Although stored together, the two parts are returned to the program separately when the record is read. The size of the fixed control area is identical for all records of the file. The contents of the fixed control area are completely under the control of the program and can be used for any purpose. For example, fixed control areas can be used to store the identifier (relative record number or RFA) of related records. Indexed file organizations do not support VFC record format.

Key Definitions For Indexed Files

To define a key for an indexed file, the user specifies the position and length of particular data fields within the records. At least one key, the primary key, must be defined for an indexed file. Additionally, up to 254 alternate keys can be defined. In general, most files have two or three keys. Because indices require storage space and RMS updates indices as records are added or modified, no more than 6 to 8 keys should be defined where storage space or access time is important.

Each primary and alternate key represents from 1 to 255 bytes in each record of the file. RMS permits 6 key field data types.

- string
- signed 15-bit integer
- unsigned 16-bit binary
- signed 31-bit integer
- unsigned 32-bit binary
- packed decimal

The string key field can be composed of simple or segmented keys. A simple key is a single, contiguous string of characters in the record; in other words, a single field. A segmented key, however, can consist of from 2 to 8 fields within records. These fields need not be contiguous. When processing records that contain segmented keys, RMS treats the separate fields (segments) as a logically contiguous character string. The integer, binary, and packed decimal data types can only be simple keys.

When defining keys at file creation time, two characteristics for each key can be specified:

- Duplicate key values are or are not allowed.
- Key value can or cannot change.

When duplicate key values are allowed, more than one record can have the same value in a given key. For example, the creator of a personnel file could define the department name field as an alternate key. As programs wrote records into the file, the alternate index for the department name key field would contain multiple entries for each key value (e.g., PAYROLL, SALES, ADMINISTRATION), since departments are composed of more than one employee.
When such duplication occurs, RMS stores the records so that they can be retrieved in first-in/first-out (FIFO) order. If key values can change, records can be read and then written back into the file with a modified key value. For example, this specification would allow a program to access a record in the personnel file and change the contents of a department name field to reflect the transfer of an employee from one department to another. This characteristic can be specified only for alternate keys. If key values can change, the user must also specify that the duplicate key values are allowed. If the primary key value can change, the user may not change the record length.

Figures 8-7 and 8-8 show excerpts from a COBOL program which operates upon an indexed customer information file via the dynamic access method. The program searches through the file and generates various reports based upon the customer's financial status and additional input typed in by the user at the terminal. In Figure 8-7, the program describes the organization of the file and specifies the access method to be used. In Figure 8-8, the program searches for the first non-zero customer number. Using the "approximate key" match facility (greater than), the program searches for the first non-zero customer number, the program changes access method and the file is read sequentially.

**INPUT-OUTPUT SECTION.**

**FILE-CONTROL.**

**SELECT CUSTOMER-FILE**

ASSIGN TO "CUSTOM.DAT"

ORGANIZATION IS INDEXED

ACCESS MODE IS DYNAMIC

RECORD KEY IS CUST-CUSTOMER-NUMBER

ALTERNATE RECORD IS KEY IS CUST-CUSTOMER-NAME

FILE STATUS IS CUSTOMER-FILE-STATUS.

---

**Figure 8-7**

**ISAM File Description**

OPEN INPUT CUSTOMER-FILE.

MOVE "000000" TO CUST-CUST-NUMBER.

START CUSTOMER-FILE

   KEY IS &CUST-CUST-NUMBER.

OPEN OUTPUT STATEMENT-REPORT.

*-----------------------------

**MAINLINE SECTION.**

**SBEGIN.**

READ CUSTOMER-FILE NEXT AT END

   GO TO END-PROCESS.

   ADD 1 TO RECORD-COUNT.

*-----------------------------

**Figure 8-8**

**Dynamic Access Processing**

**PROGRAM OPERATIONS ON RMS FILES**

After RMS has created a file according to the user's description of file characteristics, a program can access the file and store and retrieve data.

When a program accesses the file as a logical structure (i.e., a sequential, relative, or indexed file), it uses record I/O operations such as add, update, and delete record. The organization of the file determines the types of record operations permitted.

If the record accessing capabilities of RMS are not used, programs can access the file as an array of virtual blocks. To process a file at this level, programs use a type of access known as block I/O.

**File Processing**

At the file level, that is, independent of record processing, a program can:

- create a file
- open an existing file
- modify file attributes
- extend a file
- close the file
- delete a file

Once a program has opened a file for the first time, it has access to the unique internal ID for the file. If the program intends to open the file subsequently, it can use that internal ID to open the file and avoid any directory search.

**Record I/O Processing**

The organization of a file, defined when the file is created, determines the types of operations that the program can perform on records. Depending on file organization, RMS permits a program to perform the following record operations:

- Read a record. RMS returns an existing record within the file to the program.
- Write a record. RMS adds a new record that the program constructs to the file. The new record cannot replace an already existing record.
- Find a record. RMS locates an existing record in the file. It does not return the record to the program, but establishes a new current position in the file.
- Delete a record. RMS removes an existing record from the file. The delete record operation is not valid for the sequential file organization.
- Update a record. The program modifies the contents of a record in the file. RMS writes the modified record into the file, replacing the old record. The update record operation is not valid for sequential file organizations, except for sequentially organized disk files.

**Sequential File Record I/O** — In a sequential file organization, a program can read existing records from the file using sequential, RFA, or, if the file contains fixed-length records, random record access mode. New records can be added only to the end of the file and only through the use of sequential or random record access mode.

The find operation is supported in both sequential and RFA record access modes. In sequential record access mode the program can use a find operation to skip records. In RFA record access mode, the program can use the find operation to establish a random starting point in the file for sequential read operations.

The sequential file organization does not support the delete operation, since the structure of the file requires that
records be adjacent in and across virtual blocks. A program can, however, update existing records in sequential disk files as long as the modification of a record does not alter its size.

*Relative File Record I/O* — Relative file organization permits programs greater flexibility in performing record operations than does sequential organization. A program can read existing records from the file using sequential, random, or RFA record access mode.

New records can be sequentially or randomly written as long as the intended record cell does not already contain a record. Similarly, any record access mode can be used to perform a find operation. After a record has been found or read, RMS permits the delete operation. Once a record has been deleted, the record cell is available for a new record. A program can also update records in the file. If the format of the records is variable, update operations can modify record length up to the maximum size specified when the file was created.

*Indexed File Record I/O* — Indexed file organization provides the greatest flexibility in performing record operations. A program can read existing records from the file in sequential, RFA, or random record access mode. When reading records in random record access mode, the program can choose one of four types of matches that RMS performs using the program-provided key value. The four types of matches are:

- exact key match
- approximate key match
- generic key match
- approximate and generic key match

An exact key match requires that the contents of the key in the record retrieved precisely match the key value specified in the program read operation.

The approximate match facility allows the program to select either of the following relationships between the key of the record retrieved and the key value specified by the program:

- equal to or greater than
- greater than

The advantage of this kind of match is that if the requested key value does not exist in any record of the file, RMS returns the record that contains the next higher key value. This allows the program to retrieve records without knowing an exact key value.

Generic key match means that the program need specify only an initial portion of the key value. RMS returns to the program the first occurrence of a record whose key contains a value beginning with those characters. This allows the program to retrieve a class of records, for example, all employee records in the personnel file with a name field beginning with M.

The final type of key match combines both generic and approximate facilities. The program specifies only an initial portion of the key value, as with generic match. Additionally, a program specifies that the key data field of the record retrieved must be either:

- equal to or greater than the program-supplied value

RMS also allows any number of new records to be written into an indexed file. It rejects a write operation only if the value contained in a key of the record violates a user-defined key characteristic (e.g., duplicate key values not permitted).

The find operation, similar to the read operation, can be performed in sequential, RFA, or random record access mode. When finding records in random record access mode, the program can specify any one of the four types of key matches provided for read operations.

In addition to read, write, and find operations, the program can delete any record in an indexed file and update any record. The only restriction RMS applies during an update operation is that the contents of the modified record must not violate any user-defined key characteristic (e.g., key values cannot change and duplicate key values are not permitted).

**Block I/O Processing**

Block I/O allows a program to bypass the record processing capabilities of RMS entirely. Rather than performing record operations through the use of supported record access modes, a program can process a file as a structure consisting solely of blocks.

Using block I/O, a program reads or writes blocks by identifying a starting virtual block number in the file and a transfer length. Regardless of the organization of the file, RMS accesses the identified block or blocks on behalf of the program.

Since RMS files, particularly relative and indexed files, contain internal information meaningful only to RMS itself, DIGITAL does not recommend that a file be modified by using block I/O. The presence of the block I/O facility, however, does permit user-created record formats on a Files-11 disk volume or ANS magnetic tape volume.
RMS RUN-TIME ENVIRONMENT

The environment within which a program processes RMS files at run time has two levels, the file processing level and the record processing level.

At the file processing level, RMS and the operating system provide an environment permitting concurrently executing programs to share access to the same file. RMS ascertains the amount of sharing permissible from information provided by the programs themselves. Additionally, at the file processing level, RMS provides facilities allowing programs to exercise as little or as much control over buffer space requirements for file processing as desired.

At the record processing level, RMS allows programs to access records in a file through one or more record access streams. Each record access stream represents an independent and simultaneously active series of record operations directed toward the file. Within each stream, programs can perform record operations synchronously or asynchronously. That is, RMS allows programs to choose between receiving control only after a record operation request has been satisfied (synchronous operation) or receiving control before the request has been satisfied (asynchronous operation).

For both synchronous and asynchronous record operations, RMS provides two record transfer modes, move mode and locate mode. Move mode causes RMS to copy a record to/from an I/O buffer from/to a program-provided location. Locate mode allows programs to process retrieved records directly in an I/O buffer.

Run-time File Processing

RMS allows executing programs to share files rather than requiring them to process files serially. The manner in which a file can be shared depends on the organization of the file. Program-provided information further establishes the degree of sharing of a particular file.

File Organization and Sharing — With the exception of magnetic tape files, which cannot be shared, an RMS file can be shared by any number of programs that are reading, but not writing, the file. Sequential disk files can be shared by multiple readers and multiple writers, but they are responsible for any record locking required to handle multiple readers and writers properly.

Program Sharing Information — A program specifies what kind of sharing actually occurs at run time. The user controls the sharing of a file through information the program provides RMS when it opens the file. First, a program must declare what operations (e.g., read, write, delete, update) it intends to perform on the file. Second, a program must specify whether other programs can read the file or both read and write the file concurrently with this program.

These two types of information allow RMS to determine if multiple user programs can access a file at the same time. Whenever a program's sharing information is compatible with the corresponding information another program provides, both programs can access the file concurrently.

Buffer Handling — To a program, record processing under RMS appears as the direct movement of records between a file and the program itself. Transparently to the program, however, RMS reads or writes the blocks of a file into or from internal memory areas known as I/O buffers. Records within these buffers are then made available to the program. Users can control the number and size of buffers. For sequential record access, users can choose an optional I/O read-ahead and write-behind buffer management. For magnetic tape file access, they can control the number of buffers for multiple buffering. For sequential disk files, users can specify the number of blocks that are to be transferred whenever RMS performs an I/O operation.

Run-time Record Processing

After opening a file, a program can access records in the file through the RMS record processing environment. This environment provides three facilities:

- record access streams
- synchronous or asynchronous record operations
- record transfer modes

Record Access Streams — In the record processing environment, a program accesses records in a file through a record access stream. A record access stream is a serial sequence of record operation requests. For example, a program can issue a read request for a particular record, receive the record from RMS, modify the contents of the record, and then issue an update request that causes RMS to write the record back into the file. The sequence of read and update record operation requests can then be performed for a different record, or other record operations can be performed, again in a serial fashion. Thus, within a record access stream, there is at most one record being processed at any time.

For relative and indexed files, RMS permits a program to establish multiple record access streams for record operations to the same file. The presence of such multiple record access streams allows programs to process in parallel more than one record of a file. Each stream represents an independent and concurrently active sequence of record operations.

As an example of multiple record access streams, a program could open an indexed file and establish two record access streams to the file. The program could use one record access stream to access records in the file in random access mode through the primary index. At the same time, the program could use the second record access stream to access records sequentially in the order specified by an alternate index.

Synchronous and Asynchronous Record Operations — Within each record access stream, a program can perform any record operation either synchronously or asynchronously. When a record operation is performed synchronously, RMS returns control to a program only after the record operation request has been satisfied (e.g., a record has been read and passed on to the program).

If the programming language allows asynchronous processing, RMS can return control to a program before the record operation request has been satisfied. A program can use the time required for the physical transfer to perform other computations. A program cannot, however, issue a second record operation through the same stream until the first record operation has completed. To ascertain when a record operation has actually been performed, a program can specify completion routines or issue a wait
request and regain control when the record operation is complete.

**Record Transfer Modes** — A program can use either of two record transfer modes to gain access to each record in memory:

- move mode
- locate mode

Move mode means that the individual records are copied between the I/O buffer and a program. For read operations, RMS reads a block into an I/O buffer, finds the desired record within the buffer, and moves the record to a program-specified location in its work space. For write operations, the program builds or modifies a record in its own work space and RMS moves the record to an I/O buffer. RMS supports move mode record operations for all file organizations.

Locate mode enables programs to read records directly in an I/O buffer. Locate mode reduces the amount of data movement, thereby saving processing time. RMS provides the program with the address and size of the record in the I/O buffer. RMS supports locate mode record transfers on all file organizations for read operations only.

**RMS Record Locking**

VAX-11 RMS provides a record locking capability for files that use the relative and indexed organization. This provides control over operation when the file is being accessed simultaneously by more than one program and/or more than one stream in a program. Record locking makes certain that when a program is adding, deleting, or modifying a record on a given stream, another program or stream is not allowed access to the same record or record cell. There are two varieties of record locking and unlocking:

- **Automatic Record Locking** — The lock occurs on every execution of a $FIND or $GET macro instruction, and the lock is released when the next record is accessed, the current record is updated or deleted, the record stream is disconnected, or the file is closed. The $FREE macro instruction explicitly unlocks all records previously locked for a particular record stream. The $RELEASE macro instruction explicitly unlocks a specified record in a record stream.

- **Manual Record Locking** — In manual record locking, varying degrees of locking may be specified by setting bits in the record processing options field (ROP) of the RAB. The ULK bit specifies manual (as opposed to automatic) locking and unlocking. This bit specifies that locking will occur on the execution of a $GET, $FIND, or $PUT macro instruction and that unlocking may take place explicitly only via a $FREE or $RELEASE macro instruction. The NLK bit specifies that the record accessed with either a $GET or $FIND macro instruction is not to be locked, while the RLK specifies that a record may be accessible for read purposes but may not otherwise be accessed.
9
The
Network
Services
DECnet is a family of network products developed by Digital Equipment Corporation that adds networking capability to DIGITAL's computer families and operating systems. Using DECnet, various kinds of computer system networks can be constructed to facilitate remote communications, resource sharing, and distributed computation. DECnet is highly modular and flexible. It can be viewed as a set of tools or services from which a user selects those appropriate to build a network to satisfy the requirements of a particular application.

DIGITAL Network Architecture (DNA) provides the common network structure upon which all DECnet products are built. The architecture is designed to handle a broad range of application requirements because all the functions of the network from the user interface to physical link control are completely modular. DNA allows nodes to operate as switches, front-ends, terminal concentrators or hosts.

DECnet/VAX:
- provides an interprocess communication facility that is highly transparent and easy to use
- provides a high-level language programming interface
- allows programs to access files at other systems
- allows users and programs to transfer files between systems
- allows users to transmit command files to be executed in other systems
- allows an operator to down-line load RSX-11S system images into other systems
INTRODUCTION

With DECnet, a variety of computer networks can be implemented. They typically fall into one of three classes:

- **Communications Networks:** These networks exist to move data from one, often distant, physical location to another. The data may be file-oriented (as is often the case for remote job entry systems) or record-oriented (as occurs with the concentration of interactive terminal data). Interfaces to common carriers, using both switched and leased-line facilities, are normally a part of such networks. Such networks are often characterized by the concentration of all user applications programs and database systems on one or two large host systems in the network. Figure 9-1 illustrates such a network.

- **Resource-Sharing Networks:** These networks exist to permit the sharing of expensive computer resources among several computer systems. Shared resources not only include peripherals such as mass storage devices, they can also include logical entities, such as a centralized database which is made available to other systems in the network. Such networks are often characterized by the concentration of high-performance peripherals, extensive data bases, and large programs on one or two host systems in the network, while the satellite systems have less expensive peripherals and smaller programs. Figure 9-2 illustrates a resource-sharing network.

- **Distributed Computing Networks:** These networks coordinate the activities of several independent computing systems and exchange data between them. Networks of this nature may have specific geometries (star, ring, hierarchy), but often have no regular arrangement of links and nodes. Such networks are usually configured so that the resources of a system are close to the users of those resources. Distributed computing networks are usually characterized by multiple computers with applications programs and data bases distributed throughout the network. Figure 9-3 and 9-4 illustrate two such networks.

When DECnet is used to connect heterogeneous systems, each node of the network has both common DECnet attributes and system-specific attributes. The attributes provided by DECnet/VAX include:

- **Interprocess (Task-to-Task) Communication:** Programs executing on one system can converse with programs executing on other systems.

- **Inter-System File Transfer:** A program or a command language user can transfer an entire data file from one system to another.

- **Inter-System Resource Sharing:** Programs executing on one system can access files physically located at other systems in the network. Access to devices in other systems is provided only through the file system of the target node and is subject to that system’s DAP implementation or file system restrictions.
• Down-Line System Loading: Initial load images for RSX-11S systems in the network can be stored on the host VAX-11/780 system, and loaded on request into PDP-11 systems configured for the RSX-11S operating system.

• Down-Line Command File Loading: Programs or command language users can send command files to a remote node to be executed there. However, no status information or error messages are returned.

• High-Level Language Interface: This facility allows programs written in any VAX-11 native programming language to access some of the network facilities.

When a VAX/VMS system operates as a node in a DECnet configuration, the types of functions it can perform will be somewhat dependent on other DECnet products in the configuration. This is because two nodes communicating with each other will always be restricted to those functions which are common to both of them. Table 9-1 lists the various DECnet systems which may be configured with DECnet/VAX and the functions each may perform.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Task-to-Task</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Intersystem File Transfer</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Command/Batch File Submission</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Command/Batch File Execution</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Remote File Access</td>
<td>YES</td>
<td>YES ²</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Down-Line System Loading</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Down-Line Task Loading</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

¹ Cannot submit files to DECnet/E systems. Can tell DECnet/E to execute batch files already at the DECnet node.
² Offers local users network access to remote file systems. Does not allow users on remote systems to access local files.

DIGITAL NETWORK ARCHITECTURE

The DIGITAL Network Architecture (DNA) is a set of protocols (rules) governing the format, control, and sequencing of message exchange for all DECnet implementations. DNA controls all data that travels throughout a DECnet network and provides a modular design for DECnet.

Its functional components are defined within four distinct layers: User Layer, Logical Link Control Layer, Physical Link Control Layer, and Hardware Layer. Each layer performs a well-defined set of network functions (via network protocols) and presents a level of abstraction and capability to the layer above it.

User Layer: At the user layer, a message can be generated to be transmitted to another process. The message can be generated at a terminal (such as a request to transfer a file), or in a process (using RMS or System Services). It is this layer that is "visible" to the user. All other layers are transparent, thus allowing network use without the necessity of learning the many protocols required to transmit data between processes.

Logical Link Control Layer: Processes access the network at the logical link layer. The Network Services protocol (NSP) accepts the data from the user layer, and formats the data for multiplexed transmission by the communications hardware. NSP also accepts incoming data from remote processes, separates the data into logical link streams, and delivers the data to the user process.

Physical Link Control Layer: DIGITAL Data Communications Message Protocol (DDCMP) controls physical links. Like other line protocols, DDCMP maintains control over a physical communications link to provide an error-free, sequential data path over a generally error-prone medium. DDCMP accomplishes this with a Cyclic Redundancy Check (CRC) for error detection, transmission for error corrections, and numbered data messages to ensure sequential transfer of data.

Hardware Layer: The hardware layer transmits and receives data over the physical link. Device drivers or interrupt service routines exchange data with the interface and handle synchronization and modem control for the hardware.
PROGRAMMING AND OPERATING INTERFACE
The following paragraphs describe the ways in which programmers, operators, and terminal users can interact with DECnet. Essentially, there are three ways in which users can employ DECnet features:

- Interprocess or “task-to-task” communication. The DECnet/VAX programming interface allows both transparent and non-transparent interprocess communication.
- File and record operations. DECnet/VAX allows file and device access to programs and terminal users. A terminal user can create, transfer, and delete files in other nodes. Programs also have this capability, with the added ability to read and write records. These capabilities provide the tools for creation of distributed data bases.
- Down-line loading. DECnet/VAX can down-line load RSX-11S system images, provided the target system includes a ROM bootstrap. No peripheral device for loading, such as floppy disk or paper tape, is required for such nodes.

INTERPROCESS COMMUNICATION
Interprocess communication is the one feature common to all DECnet implementations. Interprocess communication allows programs (tasks, processes, etc.) to create one or more logical links, which are full-duplex virtual data paths. Programs have the capability to create these connections, transmit and receive data over them, and destroy them. Data transmissions can be done on a normal or priority (i.e., interrupt) basis.

In DECnet/VAX, the interface itself can be transparent (each program looks like a sequential device to the other), or non-transparent (each program knows that it is using DECnet and has the opportunity to acquire information about the network). Each access method has its advantages. Transparent access is easier to learn, and it allows great flexibility in that the location of files, devices, and the program itself need not be determined until run time. Under non-transparent access, the programmer can take advantage of known properties of the protocols, providing the ability to transmit and receive interrupt messages, connect, disconnect, and initiate disconnect notification.

Transparent Interprocess Communication
In transparent access, the program opens the network interchange as if it were preparing device access, and then performs a series of reads and writes, just as it would to a pair of serial devices, one for input (reception) and the other for output (transmission). By its very nature, transparent access has no calls specifically associated with DECnet. The calls used for interprocess communication are the same as the calls used for accessing a sequential file in a high-level language: OPEN, CLOSE, READ, WRITE, etc. The programmer can choose to include the target node name in the OPEN statement, or can defer assignment using logical names.

Non-transparent Interprocess Communication
In non-transparent access, a program can obtain information about the network status to control the nature of its communication with other processes or tasks. Non-transparent access is available only through calls to operating system service procedures. A program can issue the following requests:

- CONNECT — Establish a logical link (the analog of OPEN).
- CONNECT REJECT — Reject a connect initiate.
- RECEIVE — Receive a message (the analog of GET).
- SEND — Transmit a message (the analog of PUT).
- SEND INTERRUPT MESSAGE — Transmit a high-priority message.
- DISCONNECT — Terminate a conversation (the analog of CLOSE).

The process can send optional data along with the connect request, for example, the size or number of messages that it wants to send. The receiving process or task can accept or reject the connect initiate. A process can accept multiple connect requests.

A process can send or receive unsolicited messages to or from another process or task. Unsolicited message traffic is essentially no different: from solicited message traffic except that it uses a system's software interrupt mechanism to transmit a message. A logical link therefore has two subchannels over which messages can be transmitted: one for normal messages and another for high-priority messages. In DECnet/VAX, an interrupt message is written to a mailbox that a process supplies for that purpose. The process can request that an asynchronous system trap routine be executed when a message is queued to that mailbox.

A program can issue a synchronous disconnect, which guarantees the receiver that it got every message that was sent, or it can issue a disconnect abort, which terminates the logical link immediately.

In DECnet/VAX, a program using non-transparent access normally opens a control path directly to the Networks Ancillary Control Process (NETACP), and designates one or more mailboxes for receiving information from the NETACP about the logical or physical links over which the process is communicating. The NETACP can notify a process when:

- A partner requests a synchronous disconnect.
- A partner requests a disconnect abort.
- A partner exits.
- A physical link goes down.
- An NSP protocol error is detected.

If the process has the diagnostic privilege, it can also stop and start DDCMP protocol over a physical link.

DATA TRANSFER OPERATIONS
Using the VAX/VMS command language, operators and terminal users can:

- copy files from the VAX-11 system to another system
- delete files in other systems
- transfer command files for executing on other systems

Programs have access to all the above file operations, and they can also read and write records sequentially and randomly.
Command Language

DECNет/VAX-11 supports the following VAX/VMS commands:

$ APPEND
$ ASSIGN
$ COPY
$ DEASSIGN
$ DEFINE
$ DELETE
$ SUBMIT
$ TYPE

The following examples illustrate the $ COPY and $ SUBMIT commands:

$ COPY BOSTON::DBA1:TEST.DAT DENVER::DMA2:
transfers a file named TEST.DAT from the disk (DBA1:) to
the node named BOSTON to the disk (DMA2:) at the node
named DENVER.

Using the VAX/VMS command SUBMIT, a terminal user
can send a command file to another system in a network,
and have it executed at that node. For example, the com-
mand:

$ SUBMIT/REMOTE:WASHDC.INITIAL.COM

transfers the command file named INITIAL.COM from the
host system to the node named WASHDC, where the com-
mand file is executed. Command files must be written in
the command language of the target system. No status in-
formation or messages are returned to the sender.

Programmed Data Transfer

For more complicated operations on remote files, DECNет
supports the following RMS macros for both sequential or
relative file access:

- $CREATE — Create and open a file.
- $ERASE — Delete a closed file.
- $OPEN — Open a file.
- $CLOSE — Close an open file.
- $CONNECT — Establish a record processing stream.
- $DISCONNECT — Terminate a record processing
  stream.
- $GET — Read a record.
- $PUT — Write a record.
- $UPDATE — Modify an existing record.
- $DELETE — Delete a record from a relatively organized
  file.
- $READ — Read a block.
- $WRITE — Write a block.
- $FIND — Position to a specified record.
- $REWIND — Position to the beginning of a file.
- $SPACE — Position to the next block.

Figure 9-6, showing a MACRO program transferring a se-
quential file from one device to another, illustrates RMS
file transfer under DECNет.

.TITLE DEMO1 - RMS FILE TRANSFER EXAMPLE

This program transfers a sequential file with variable length
records from one device to another. The devices are specified
by the logical names SRC and DST. For example, to display file
INVENTORY.DAT at node ALBANY on the line printer at node
BOSTON, execute the following procedure:

$ DEFINE SRC ALBANY::DBB3(XYXCO.STOCK)INVENTORY.DAT
$ DEFINE DST BOSTON::LPA0:
$DEMO1

.SBTTL CONTROL BLOCK AND BUFFER STORAGE
.PSECT IMPURE NOEXE.LONG

Define the source file FAB and RAB control blocks.

SRC_FAB:

$FAB FAC=<GET>- FOP=SQQ- FNA=SRC_NAM- FNS=SRC_NAM-SIZ

SRC_RAB:

$RAB FAB=SRC_FAB- RAC=SEQ- UHF=BUFFER- USZ=BUFFER_SIZ

Define the destination file FAB and RAB control blocks.

DST_FAB:

$FAB FAC=<PUT>- FOP=SQQ- FNA=DST_NAM- FNS=DST_NAM-SIZ- ORG=SEQ- RFM=VAR- RAT=CR

DST_RAB:

$RAB FAB=DST_FAB- RAC=SEQ

Define logical names for the source and destination files.

SRC_NAM:

.ASCII /SRC/
.SCR_NAM_SIZ=.-SRC_NAM

DST_NAM:

.ASCII /DST/
.DST_NAM_SIZ=.-DST_NAM

Allocate buffer space to be size of largest record.

BUFFER:

.BLKB 132
.BUFFER_SIZ=.-BUFFER

.SBTTL MAINLINE
.PSECT CODE NONRT,NYTB

..ENTRY DEMO01, |M<>

Put FAB and RAB addresses in registers for efficiency.

MOVAB WSRSC_FAB,R6
MOVAB WSRSC_RAB,R7
MOVAB WSDST_FAB,R8
MOVAB WSDST_RAB,R9

Open the SRC and DST files.

$OPEN FAB=R6
BLBC R0,30S
$CONNECT RAB=R7
BLBC R0,30S
$CREATE FAB=R8

9-4
The following example shows how a FORTRAN IV-PLUS program could retrieve records from a file at a node DALLAS and print them on the line printer at node BOSTON. If node names are omitted from the file specifications, the same routine prints a local file on a local line printer.

```
CHARACTER*133 REC
OPEN (UNIT=1,NAME='DALLAS::DBA1.FILE.DAT',TYPE='OLD')
OPEN (UNIT=2,NAME='BOSTON::LPA1:')
10 READ (1,100,END=300)REC
100 FORMAT (A)
     WRITE (2,100)REC
     GO TO 10
300 CLOSE (UNIT=1)
     CLOSE (UNIT=2)
END
```

**FORTRAN IV-PLUS INTERTASK COMMUNICATION**

There are three major steps in FORTRAN IV-PLUS inter-task communication:

- Create a logical link between tasks.
- Send and receive messages (each message can be 1 to 512 bytes in length).
- Destroy the link at the end of the message dialog.

**Creating a Logical Link Between Processes**

The establishment of a logical link between processes requires "cooperation" between the two processes. That is, one process (the source process) must request that a logical link be created, and the other process (the target process) must agree to accept the request.

The source task issues a logical link connect request by including a task specifier in the source task's OPEN statement. This is illustrated in the following example:

```
OPEN (UNIT=7, NAME='DENVER::"TASK=ACC",ERR=200')
```

Here the NAME argument requests a logical link connection to target task ACC on node DENVER. The local node passes the logical link connect request to the remote node (using DECnet/VAX services). The remote node creates a process for the target task, and places the source task identifier in the logical name table under the name SYS$NET.

The target task identifies the source task requesting the logical link connect request by specifying SYS$NET as the NAME argument in the OPEN statement.

An example of the target task OPEN statement would be:

```
OPEN (UNIT=2,NAME=SYS$NET,ERR=700)
```

**Sending and Receiving Messages**

After the logical link has been created, the tasks must "cooperate" with each other. That is, for each message sent by a task (WRITE statement), the receiving task must issue a corresponding receive (READ statement).

In addition, the tasks must ensure that enough buffer space is allocated for messages, that the end of dialog can be determined, and which of the tasks will disconnect the logical link (CLOSE statement).

**Disconnecting the Logical Link**

Either task can disconnect the logical link by calling CLOSE. CLOSE aborts all pending sends and receives, disconnects the link immediately, and frees the channel number associated with the logical link.
MACRO NON-TRANSPARENT INTERTASK COMMUNICATION

The VAX-11 MACRO language permits the user to perform non-transparent intertask communication as well as transparent communication, allowing greater flexibility and control in network operations. Non-transparent intertask communication includes the following capabilities:

- Associate a mailbox with the I/O channel (over which the logical link will be created). The mailbox can then receive unsolicited messages sent by a remote task, or notifications affecting the status of the logical link. For example, the remote task accepted or rejected a connect request, or the cooperating task disconnected or destroyed the link.

- A task can declare itself as a network task to accept multiple logical link connect requests.

- A source task can send a logical link connect request to the target task. The source task can optionally send 1 to 16 bytes of data to the target task at the same time it issues the connect request.

- The target task can accept or request the connect request. It can send 1 to 16 bytes of optional data back to the source task at the same time it accepts or rejects the connect request.

- A non-transparent task can also accept or reject connect requests received from tasks written using transparent intertask communication system service calls.

- Either task can send or receive a 1- to 16-byte interrupt message after the logical link is created.

- Either task can abort the link immediately, or issue a synchronous disconnect. The task disconnecting or aborting the logical link can also send 1 to 16 bytes of optional data to the remote task at the same time it disconnects or aborts a logical link.

Creating a logical link in non-transparent intertask communication requires that both the source and target tasks call the $ASSIGN system service in order to:

- Assign to device NETO:

- Request a channel number for the logical link.

- Associate a previously created mailbox with the channel.

In addition to $ASSIGN, the following VAX/VMS system services can be used in non-transparent intertask communication. These are:

- $QIO(IO$ _ACCESS) — Request a Logical Link Connection

- $QIO(IO$ _ACCESS) — Accept a Logical Link Connection Request

- $QIO(IO$ _ACCESS$IO$ _M_ABORT) — Reject A Logical Link Connection Request

- $QIO(IO$ _WRITEVBLK) — Send a Message to a Remote Task

- $QIO(IO$ _READVBLK) — Receive a Message from a Remote Task

- $INPUT — Read a Message

- $QIO(IO$ _WRITEVBLK$IO$ _M_INTERRUPT) — Send an Interrupt Message to a Remote Task

- $QIO(IO$ _DEACCESS$IO$ _M_SYNCH) — Synchronously Disconnect the Logical Link

- $QIO(IO$ _DEACCESS$IO$ _M_ABORT) — Abort a Logical Link

- $QIO(IO$ _ACPCONTROL) — Declare a Network Name

- $DASSGN — Disconnect the Logical Link

DOWN-LINE LOADING AND NETWORK STATUS

This interface for the operator at the host VAX-11/780 system is a function of the NCP utility that accepts parameters such as the location and name of the RSX-11S system image and the name of the node to be sent the load image. For example:

```
$ NCP
#1> LOAD NODE DALLAS
#2> LOAD NODE BOSTON FROM SYSMON.EXE
```

In the first request, NCP starts down-line loading the default operating system image to node DALLAS, while in the second request, NCP starts loading a specific system image.

Booting the RSX-11S operating system requires the presence of a read-only memory bootstrap program and, for cold start, an operator to power-on the system and switch the DMC11 to "remote load detect." For warm start, when an operating system is already running, no operator intervention is required, as the system automatically jumps to the bootstrap.

For cold start, the operator of the RSX-11S system starts the ROM bootstrap. It sends a special BOOT-ME message to the host, which automatically sends the proper system over the link.

For warm start, the VAX-11/780 host operator may decide to load a different system in the satellite machine. In such cases, the utility will send over the appropriate BOOT messages, and the satellite will automatically stop any current processing and go into BOOT mode. No operator is required at the RSX-11S satellite in this case.

Finally, a watchdog timer is available that will automatically put the RSX-11S system into BOOT mode whenever the operating system fails to reset a bit within a specified period. In this case, no operator intervention is required at either end. In all cases, the satellite system can start automatically after a boot operation.

NCP also enables the operator at the VAX/VMS system to:

- SHOW PATHS — Identify the physical links connected to the system.

- SHOW COUNTS — Display the current network throughput and error statistics.

- ZERO COUNTS — Initialize the statistics counters.

- SHOW STATUS — Display the status of a node in the network or a particular physical link on any node. Node status includes: ON (allow logical links), SHUT (active, but allow no new logical links), MAINTENANCE (no logical links, but physical links may be active; for example, down-line load in progress), and OFF. Line status includes ON, MAINTENANCE, and OFF.

- SET STATE — Set the status for each physical link and for the local system.
• LOOP — Perform loop-back tests with a particular node or with a particular physical link in the local system. The operator can, for example, test the local NSP functions, the local modem, the remote modem, or the remote NICE functions.
10
The Support Services
DIGITAL offers special services to help before, during and after system installation. DIGITAL's sales force is the primary contact for all products and services.

First, with your sales representative, you study your application and requirements. During this time, you work closely with the salesperson to determine your computing needs. The sales representative may call in software and hardware specialists to help answer specific questions. These specialists are trained to design systems using DIGITAL's standard products, and if necessary, to suggest alternate solutions for special situations.

Once you have determined the exact nature of your requirements, your sales representative helps you select a system configuration. You review the site requirements outlined in the *Site Preparation Guide* (assuring adequate floor space, electrical capacity, air conditioning and humidity control, etc.). You select a Field Service maintenance plan that is appropriate to your needs and budget. Your salesperson writes up your order and helps arrange financing.

While waiting for delivery of your system, you can train your personnel by taking advantage of DIGITAL's extensive educational programs. By purchasing a system, you obtain training credits which can be applied to the cost of some of DIGITAL's courses.

On delivery, DIGITAL's Hardware Field Service and Software Support organizations are on hand to ensure smooth installation. Specialists install the hardware and the software and run extensive tests to ensure that the system is correctly installed and performing properly.

Following installation, DIGITAL's support organizations are available to help with any special needs that may arise both during and after the warranty period. The hardware Field Service organization includes representatives in locations throughout the world who can provide preventive and remedial maintenance to keep your system hardware performing optimally. Software specialists, trained and proficient in the use of your software system, can be called in to help with any questions that arise in its operation or use. In addition, software consultants and the Computer Special Systems group may be contracted to help design and build systems to meet individual requirements.
INSTALLATION

Upon delivery of your system, your Field Service account representative will schedule installation of the hardware components. During installation, Field Service engineers supervise the uncrating and placement of equipment, cable connections, and powering-up of components. They test the hardware by running a system exerciser — a complete diagnostic package. Once hardware reliability is confirmed, the Field Service engineers coordinate with software support personnel to install and test the operating system. They use the User Environmental Test Package to exercise the system software components. This package runs customer-type routines (compilations and assemblies) and serves as both a final test of the system and as a demonstration of system operation.

Finally, Field Service completes DIGITAL forms certifying successful installation. You acknowledge system acceptance by signing the Field Service Labor Accounting and Reporting System form.

HARDWARE SERVICES

On-site warranty begins 30 days after the system was shipped or upon customer acceptance, whichever occurs first. Hardware components in a packaged system are covered under warranty for 90 days. OEMs must buy back a minimum installation and 30-day on-site warranty; an optional installation and 90-day on-site warranty is available. During the warranty period, DIGITAL will perform remedial maintenance on any defective components.

Once the warranty period ceases, your maintenance plan becomes effective.

Most customers select a DECService Agreement. The DECService Agreement is for customers who require high availability, whose system is critical for daily operation. This plan includes all materials and labor, preventive maintenance, installation of engineering changes, and the use of licensed diagnostics. It guarantees 4-hour response on service calls and continuous service until repairs are complete.

If your availability requirements are not stringent, you may choose a Basic Agreement. The Basic Agreement includes all materials and labor, installation of engineering changes, and the use of licensed diagnostics. However, preventive maintenance occurs less frequently than under the DECService Agreement.

With either of these service agreements, Remote Diagnosis is available. Remote diagnosis service capabilities are normally installed and used by Field Service during the warranty period. Remote Diagnosis allows a DIGITAL Field Service engineer, with your permission, to access your system from a remote site for the purpose of exercising diagnostics to determine hardware problems. Its main benefit is increased system availability, due to shortened diagnosis and repair time. Under both the DECService Agreement and the Basic Agreement, you may obtain coverage from 8 hours a day 5 days a week, up to 24 hours a day 7 days a week. Your sales representative is trained to help you select the best coverage.

Two non-contract maintenance options are also available: Per-Call and Self-Maintenance.

Per-Call service is provided for customers who do not need immediate response or continuous effort. Generally
per-call arrangements are useful for installations that are non-critical, have redundant backup equipment, or are self-maintained. Per-call service is also offered as a supplementary service (on a “best-efforts” basis) for service agreement customers, if remedial service is required outside the hours of contract coverage.

Off-Site Services are available for customers who prefer to maintain their own equipment. DIGITAL offers many kinds of assistance, including off-site repair of major equipment and modules; recommended spares service, spare parts, tool kits, and test equipment; maintenance documentation and engineering updates; emergency parts service; and hardware maintenance training. A license is required to use DIGITAL diagnostics when no service agreement is in effect.

If you have a service agreement, you need only contact Field Service by dialing the phone number provided during installation whenever you require Field Service assistance to solve a hardware-related problem. If you do not have a service agreement, you can still obtain help. Call your local Field Service branch office for details.

SOFTWARE SERVICES

Software Services is committed to maintaining a high level of support for the VAX-11/780 software. Software specialists have been specially trained in VAX-11/780 software in order to provide the expert knowledge and experience necessary to analyze your needs, to identify those DIGITAL services that will help you meet those needs, and to deliver those services through your local software specialist. In addition, your local software specialist has access to backup support at the regional and corporate levels, when necessary. This means that DIGITAL’s total software resources and expertise are available to support your VAX-11/780 software product.

Software Warranty

The VAX-11/780 software package is covered by DIGITAL’s Software Warranty. This warranty consists of installation of the software, 90 days of remedial support from your local DIGITAL office, and one year of software maintenance service.

During the 90 days of remedial service, if you encounter a problem that DIGITAL determines to be a defect in the current unaltered release of the software, DIGITAL will provide the following remedial services, on site where necessary:

- If the software is inoperable, apply a temporary correction or make a reasonable attempt to develop an emergency by-pass.
- Help you prepare and submit a Software Performance Report.

During the one year of software maintenance services, the following service components are in effect:

- Software Dispatch — This monthly publication is automatically distributed and contains information on reported software problems and their solutions, general software information, system enhancements, and new software products.
- Software Updates — For a media charge only, you can receive the most recent version of the existing software containing the most recent fixes and new features. You are informed through the Software Dispatch before a new release is delivered so that you can prepare your operation.
- Software Performance Report (SPR) Service — SPRs are a formal problem-reporting service which you may use to document any detected software problems. You are assured a response to every SPR submitted to DIGITAL. SPRs are used by DIGITAL as a basis for software remedy and enhancement. SPRs of a general nature are printed in the Software Dispatch.

Post-Warranty

Once the warranty period expires, you may elect to continue maintenance services by purchasing a yearly subscription to the Binary Program Update Service. This service is a continuation of the three maintenance services you receive under the software warranty, namely:

- Software Dispatch
- Software Update
- Software Performance Reports

Appropriate order forms will be supplied automatically when your warranty is about to expire.

Consulting Services

If you need expert software assistance, DIGITAL provides software consultants. Software consultants are professionals trained in DIGITAL products. They are experienced in designing and writing custom software as well as tailoring DIGITAL software to meet specific needs. Their expertise can be applied to any phase of any application. They are specially trained to help resolve compatibility and conversion problems with your application. They can provide operational and administrative services, such as developing an operations manual or providing on-site training for your staff. They can perform system analysis after your system has been in operation for some months.

Consulting services are available in a variety of plans:
• Resident Consulting — for those customers who need a full-time resident specialist. Residents are particularly useful in new, complex installations, or in critical, long-term projects. Residents are usually on site for 6-month or 12-month periods. Arrangements can be made to adjust the length of service to suit your needs (with a minimum of 6 months required).

• Weekly Consulting — for customers with definite, but short-term consulting needs. Weekly consulting includes 40 hours of on-site support. Consecutive weeks of service may be ordered as needed.

• Per-Call Consulting — for customers with irregular or infrequent consulting needs. Per-call (hourly) services may be ordered as needed and generally extend from a few hours to a few days.

To request consulting services, or for more information, contact your local DIGITAL office.

EDUCATIONAL SERVICES

To train your personnel before, during, and after installation, DIGITAL provides comprehensive educational programs. Trained instructors give standard courses in system management, operations, hardware, and software at DIGITAL’s worldwide training centers. In addition, special on-site training and custom courses can be arranged.

Courses fall into four general categories:

• General Interest Computer Science Courses — These are not geared to a specific DIGITAL system or product, but provide a technical foundation for personnel who have little previous computer experience.

• Software Systems Courses — These courses are designed to train users, programmers, and operators in the efficient and knowledgeable use of DIGITAL’s operating systems, languages, and utilities. Courses are available for both beginning and advanced users. For the most part, these courses assume that the student has general computer and programming knowledge.

• Hardware Courses — These are designed for customers who intend to service their own equipment or who simply want a general understanding of the components in their system. Courses are available in general hardware familiarization, hardware troubleshooting, and hardware maintenance.

• Audio/Visual Courses — A/V courses are designed for students who wish to learn computer fundamentals at a self-paced rate. They are portable, self-contained, and modular in format. Among the A/V courses offered are Introduction to Minicomputers, Introduction to the PDP-11, and Introduction to Data Communications Concepts. Standard courses are given regularly at DIGITAL’s training centers. However, other options are available for customers with special needs:

• On-site courses — If you have a group of individuals to train, Educational Services can conduct courses at any convenient location, such as your company offices. On-site instruction eliminates travel expenses, and allows DIGITAL instructors to emphasize points of particular value to your application and operation.

• Custom courses — If you have a unique application, Educational Services can create a custom course tailored to your needs and schedule. These courses can be modifications of existing courses or completely new programs.

VAX-11/780 Specific Courses

DIGITAL’s Educational Services offers a comprehensive series of courses to provide VAX-specific training to all levels of users. These range from introductory material describing the VAX-11/780 hardware and operating systems concepts to advanced courses designed for high-level systems programmers. The diagram below illustrates a typical curriculum for three kinds of VAX-11/780 users: the system manager, a high-level language programmer, and a MACRO programmer.

Figure 10-1
VAX-11/70 Courses
Following are brief descriptions of the courses shown in Figure 10-1.

**Introduction to VAX-11/780 Concepts:** Basic hardware and operating system components.

**Introduction to VAX-11/780 Instruction Set:** Introduction to addressing modes and use and coding of VAX-11/780 instructions.

**VAX/VMS User:** Topics useful in application program development. Editors, language translators, linkers, and utilities; VAX/VMS Command Language.

**VAX/VMS System Manager:** Topics useful in management of VAX/VMS system; system generation, updating, and tuning, authorization of users, error analysis and batch and I/O queue management.

**VAX-11 MACRO Assembly Language:** Layout conventions, assembler directives, libraries, linker, debugging techniques; advanced programming techniques.

**VAX/VMS FORTRAN/MACRO Programming:** Features and use of the system services provided by VAX/VMS. Discussion of run-time library, condition and exit handlers, RMS structures and directives, and interprocess communication. Scientific and real-time orientation.

**VAX/VMS System Programming:** Detailed study of VAX/VMS operating system, including paging, swapping, scheduling, I/O operations, job controller, and management of batch and spooling jobs. Techniques for optimizing application programs, improving system throughput, and adding custom software.

**VAX/VMS Device Drivers:** Techniques for writing and incorporating a device driver into the VMS operating system. Course is particularly geared towards non-standard peripheral devices.

Additional information concerning course content and availability of training credits may be obtained from your sales representative.

**DECUS**

DECUS, Digital Equipment Computer Users Society, is the largest and most active user group in the computer industry. It is supported by DIGITAL, but actively controlled by individuals who have purchased, leased, have on-order, or use a DIGITAL computer. Membership is free and voluntary.

The goals of DECUS are to:

- Advance the art of computation through mutual education and exchange of ideas and information.
- Establish standards and provide channels to facilitate the exchange of computer programs.
- Provide feedback to DIGITAL on hardware and software needs.
- Advance the effective utilization of DIGITAL computers, peripherals, and software by promoting the interchange of information concerning their use.

To further these goals, DECUS holds regular symposia, maintains a program library, publishes newsletters and symposia proceedings, and has a number of special interest and local subgroups:

- Program Library — Programs submitted by users are available to all DECUS members. Last year, the library distributed over 80,000 programs.
- Symposia — These are regularly scheduled meetings held in the United States, Canada, Europe, and Australia. DIGITAL personnel and DECUS members attend these meetings and present papers which are available to all members.
- Subgroups — Special interest groups focus on operating systems, languages, processors, and applications; in addition, numerous local users groups exist worldwide.

Your sales representative can provide additional details and the necessary application forms.

**COMPUTER SPECIAL SYSTEMS**

DIGITAL's Computer Special Systems (CSS) group provides solutions for special needs. You may contract this group, which includes analysts, engineers, programmers, and manufacturing specialists, to obtain hardware and software tailored to your application.

CSS builds hardware, software, and system components that supplement DIGITAL's standard product mix. Products and systems are analyzed, designed, and implemented according to your goals and requirements, and may range from simple processor interfaces to complete turn-key hardware and software systems. CSS services are available on a fixed-price contract basis.

- Special hardware — CSS interfaces DIGITAL hardware with that of other manufacturers; modifies standard hardware, and designs and builds new equipment. All special hardware can be manufactured in any quantity; spares and support are guaranteed.
- Special Software — CSS designs and produces diagnostic, systems, and applications software. They modify and expand standard DIGITAL software systems, or build new software according to your needs. When required, CSS will supply software on a turn-key basis.
- Special Systems — CSS will build complete hardware and software solutions for your special application. CSS project managers oversee the analysis, design, and implementation of the system. They work with you to ensure proper installation and start-up.
- Support — CSS solutions receive DIGITAL's standard support. Hardware is supported by DIGITAL's extensive Field Service organization, and training is available on all aspects of CSS systems.

For information on how you may be able to take advantage of the services provided by CSS, contact your sales representative.

**CUSTOMER FINANCING**

To simplify the financial problems in acquiring a new computer, DIGITAL provides leasing and financial counseling. The Customer Finance Department is organized to help you acquire a DIGITAL system using a lease, conditional sale, or similar financing agreement, rather than through outright cash purchase.

For private organizations or commercial businesses, DIGITAL has developed a program, known as DIGITAL Leas-
ing, with the U.S. Leasing Corporation of San Francisco. DIGITAL Leasing is a division of U.S. Leasing committed solely to the financing of DIGITAL computers. Representatives are located in or near many of the DIGITAL District Sales Offices.

Federal, state, and local government agencies have special contractual needs, and in some cases can benefit from special tax privileges. For example, a state or municipal agency qualifies for special interest rates on Conditional Sales Agreements significantly below those charged to commercial customers; the interest income is free from federal, and in some cases state, income taxes.

The following three types of financing are available: Full Payout Lease, Conditional Sales Agreement, and Federal Government Lease to Ownership Agreement.

- Full Payout Lease — This financing is used primarily by commercial customers. It involves a non-cancelable three to seven year term, usually with a 10% purchase option at the end. No down payment is required, and title remains with the lessor. Lease payment schedules are flexible, and can be tailored to your needs.

- Conditional Sales Agreement — This type of financing is used primarily by non-profit institutions, state and local governments. It also involves a non-cancelable three to seven year term. Title passes to the customer, but DIGITAL retains a security interest. The customer owns the equipment free and clear at the end of the term. Federal funding provisions are available for state and local governments.

- Federal Government Lease to Ownership Agreement — This financing is available only to approved federal government agencies. It involves a three to five year term cancelable at the end of each fiscal year and at the government's convenience. Ownership passes to the customer at the end of the term.

DIGITAL's customer financing group can provide financial counseling to help decide which arrangement is best for you. For more information, contact your sales representative.

OTHER SERVICES

The following services are provided for customers who wish to perform their own maintenance, or who want to supplement their system configuration by ordering additional components.

Software Components

Software Components — software, hardware and software manuals, engineering drawings and print sets, diagnostic listings, and source listings — are available for DIGITAL software users and can be ordered from the DIGITAL Software Distribution Center in Maynard, Massachusetts. You may obtain a catalog of available materials from your salesperson.

Customer Spares

DIGITAL's Customer Spares group supports those customers who choose to perform their own computer maintenance. Customer Spares' inventory includes standard loose-piece component and module spares, tool kits, and required test equipment. In addition, Customer Spares provides comprehensive spares kits for the majority of installed processors, tape and disk drives, terminals, and other peripherals — currently 260 devices. All spares are produced directly by volume manufacturing and are inspected to the same rigid criteria as the original product.

Recommended spares lists are available for individual devices or entire systems. Maintenance documentation is also available (by subscription) and includes all standard documents (except engineering drawings) used in the maintenance of DIGITAL's hardware. Spares specialists are located at many of the DIGITAL district offices and can assist you in selecting and maintaining proper spares inventories. Contact your local DIGITAL sales office for more information and for literature that further describes Customer Spares products and services.

DIGITAL Supplies

DIGITAL's Supplies Group maintains a complete line of operating computer supplies and accessories specifically designed for use with DIGITAL computer systems. You may order such supplies as magnetic disk media (disk packs, floppy disks, etc.), nylon and mylar ribbons, and acrylic filters for use on video terminals. In addition, you may order custom-configured cabinetry to store and protect supplies. Supplies specialists are located at many of the district offices and can help you plan your supply needs. Contact your local sales office for more information.

Logic Products

The Logic Products Group is your supplier for add-on equipment: power supplies, cable assemblies, line interfaces, terminals, ribbons, and paper. All logic products are described in the Logic Handbook and are listed in the Direct Sales Catalog. Ask your sales representative to help you obtain copies. You may order products directly from the catalog, or you may place your order through your local sales office.
glossary (gl
basic technical,
ject or field, wit
glossa gloss²)
—glos/sa·rist,
abort  An exception that occurs in the middle of an instruction and potentially leaves the registers and memory in an indeterminate state, such that the instruction can not necessarily be restarted.

absolute indexed mode  An indexed addressing mode in which the base operand specifier is addressed in absolute mode.

absolute mode  In absolute mode addressing, the PC is used as the register in autoincrement deferred mode. The PC contains the address of the location containing the actual operand.

absolute time  Time values expressing a specific date (month, day, and year) and time of day. Absolute time values are always expressed in the system as positive numbers.

access mode  1. Any of the four processor access modes in which software executes. Processor access modes are, in order from most to least privileged and protected: kernel (mode 0), executive (mode 1), supervisor (mode 2), and user (mode 3). When the processor is in kernel mode, the executing software has complete control of, and responsibility for, the system. When the processor is in any other mode, the processor is inhibited from executing privileged instructions. The Processor Status Longword contains the current access mode field. The operating system uses access modes to define protection levels for software executing in the context of a process. For example, the executive runs in kernel and executive mode and is most protected. The command interpreter is less protected and runs in supervisor mode. The debugger runs in user mode and is not more protected than normal user programs. 2. See record access mode.

access type  1. The way in which the processor accesses instruction operands. Access types are: read, write, modify, address, and branch. 2. The way in which a procedure accesses its arguments.

access violation  An attempt to reference an address that is not mapped into virtual memory or an attempt to reference an address that is not accessible by the current access mode.

account name  A string that identifies a particular account used to accumulate data on a job's resource use. This name is the user's accounting charge number, not the user's UIC.

address  A number used by the operating system and user software to identify a storage location. See also virtual address and physical address.

address access type  The specified operand of an instruction is not directly accessed by the instruction. The address of the specified operand is the actual instruction operand. The context of the address calculation is given by the data type of the operand.

addressing mode  The way in which an operand is specified; for example, the way in which the effective address of an instruction operand is calculated using the general registers. The basic general register addressing modes are called: register, register deferred, autoincrement, autodecrement, displacement, and displacement deferred. In addition, there are six indexed addressing modes using two general registers, and literal mode addressing. The PC addressing modes are called: immediate (for register deferred mode using the PC), absolute (for autoincrement deferred mode using the PC), and branch.

address space  The set of all possible addresses available to a process. Virtual address space refers to the set of all possible virtual addresses. Physical address space refers to the set of all possible physical addresses sent out on the SBI.

allocate a device  To reserve a particular device unit for exclusive use. A user process can allocate a device only when that device is not allocated by any other process.

alphanumeric character  An upper or lower case letter (A-Z, a-z), a dollar sign ($), an underscore (_), or a decimal digit (0-9).

American Standard Code for Information Interchange (ASCII)  A set of 8-bit binary numbers representing the alphabet, punctuation, numerals, and other special symbols used in text representation and communications protocol.

Ancillary Control Process (ACP)  A process that acts as an interface between user software and an I/O driver. An ACP provides functions supplemental to those performed in the driver, such as file and directory management. Three examples of ACPs are: the File-11 ACP (F11ACP), the magnetic tape ACP (MTACP), and the networks ACP (NETACP).

Argument Pointer  General register 12 (R12). By convention, AP contains the address of the base of the argument list for procedures initiated using the CALL instructions.

assign a channel  To establish the necessary software linkage between a user process and a device unit before a user process can transfer any data to or from that device. A user process requests the system to assign a channel and the system returns a channel number.

asynchronous record operation  A mode of record processing in which a user program can continue to execute after issuing a record retrieval or storage request without having to wait for the request to be fulfilled.

Asynchronous System Trap  A software-simulated interrupt to a user-defined service routine. ASTs enable a user process to be notified asynchronously with respect to its execution of the occurrence of a specific event. If a user process has defined an AST routine for an event, the system interrupts the process and executes the AST routine when that event occurs. When the AST routine exits, the system resumes the process at the point where it was interrupted.

Asynchronous System Trap level (ASTLVL)  A value kept in an internal processor register that is the highest access mode for which an AST is pending. The AST does not occur until the current access mode drops in priority (raises in numeric value) to a value greater than or equal to.
ASTLVL. Thus, an AST for an access mode will not be serviced while the processor is executing in a higher priority access mode.

authorization file See user authorization file.

autodecrement indexed mode An indexed addressing mode in which the base operand specifier uses autodecrement mode addressing.

autodecrement mode In autodecrement mode addressing, the contents of the selected register are decremented, and the result is used as the address of the actual operand for the instruction. The contents of the register are decremented according to the data type context of the register: 1 for byte, 2 for word, 4 for longword and floating, 8 for quadword and double floating.

autoincrement deferred indexed mode An indexed addressing mode in which the base operand specifier uses autoincrement deferred mode addressing.

autoincrement deferred mode In autoincrement deferred mode addressing, the specified register contains the address of a longword which contains the address of the actual operand. The contents of the register are incremented by 4 (the number of bytes in a longword). If the PC is used as the register, this mode is called absolute mode.

autoincrement indexed mode An indexed addressing mode in which the base operand specifier uses autoincrement mode addressing.

autoincrement mode In autoincrement mode addressing, the contents of the specified register are used as the address of the operand, then the contents of the register are incremented by the size of the operand.

balance set The set of all process working sets currently resident in physical memory. The processes whose working sets are in the balance set have memory requirements that balance with available memory. The balance set is maintained by the system swapper process.

base operand address The address of the base of a table or array referenced by index mode addressing.

base operand specifier The register used to calculate the base operand address of a table or array referenced by index mode addressing.

base priority The process priority that the system assigns a process when it is created. The scheduler never schedules a process below its base priority. The base priority can be modified only by the system manager or the process itself.

base register A general register used to contain the address of the first entry in a list, table, array, or other data structure.

binding See linking.

bit string See variable-length bit field.

block 1. The smallest addressable unit of data that the specified device can transfer in an I/O operation (512 contiguous bytes for most disk devices). 2. An arbitrary number of contiguous bytes used to store logically related status, control, or other processing information.

block I/O A data accessing technique in which the program manipulates the blocks (physical records) that make up a file, instead of its logical records.

bootstrap block A block in the index file on a system disk that contains a program that can load the operating system into memory and start its execution.

branch access type An instruction attribute which indicates that the processor does not reference an operand, but that the operand is a branch displacement. The size of the branch displacement is given by the data type of the operand.

branch mode In branch addressing mode, the instruction operand specifier is a signed byte or word displacement. The displacement is added to the contents of the updated PC (which is the address of the first byte beyond the displacement), and the result is the branch address.

bucket A storage structure of 1 to 32 blocks that is used to store and transfer blocks of data in files with a relative file organization.

bucket locking A facility that prevents access to any record in a bucket by more than one user until that user releases the bucket.

buffered I/O See system buffered I/O.

bug check The operating system's internal diagnostic check. The system logs the failure and crashes the system.

byte A byte is eight contiguous bits starting on an addressable byte boundary. Bits are numbered from the right, 0 through 7, with bit 0 the low-order bit. When interpreted arithmetically, a byte is a two's complement integer with significance increasing from bits 0 through 6. Bit 7 is the sign bit. The value of the signed integer is in the range -128 to 127 decimal. When interpreted as an unsigned integer, significance increases from bits 0 through 7 and the value of the unsigned integer is in the range 0 to 255 decimal. A byte can be used to store one ASCII character.

cache memory A small, high-speed memory placed between slower main memory and the processor. A cache increases effective memory transfer rates and processor speed. It contains copies of data recently used by the processor, and fetches several bytes of data from memory in anticipation that the processor will access the next sequential series of bytes.

call frame See stack frame.

call instructions The processor instructions CALLG (Call Procedure with General Argument List) and CALLS (Call Procedure with Stack Argument List).

call stack The stack, and conventional stack structure, used during a procedure call. Each access mode of each process context has one call stack, and interrupt service context has one call stack.

channel A logical path connecting a user process to a physical device unit. A user process requests the operating system to assign a channel to a device so the process can transfer data to or from that device.

character A symbol represented by an ASCII code. See also alphanumeric character.

character string A contiguous set of bytes. A character string is identified by two attributes: an address and a length. Its address is the address of the byte containing the first character of the string. Subsequent characters are
stored in bytes of increasing addresses. The length is the number of characters in the string.

**character string descriptor** A quadword data structure used for passing character data (strings). The first word of the quadword contains the length of the character string. The second word can contain type information. The remaining longword contains the address of the string.

**cluster** 1. A set of contiguous blocks that is the basic unit of space allocation on a Files-11 disk volume. 2. A set of pages brought into memory in one paging operation. 3. An event flag cluster.

**command** An instruction, generally an English word, typed by the user at a terminal or included in a command file which requests the software monitoring a terminal or reading a command file to perform some well-defined activity. For example, typing the COPY command requests the system to copy the contents of one file into another file.

**command file** A file containing command strings. See also command procedure.

**command interpreter** Procedure-based system code that executes in supervisor mode in the context of a process to receive, syntax check, and parse commands typed by the user at a terminal or submitted in a command file.

**command parameter** The positional operand of a command delimited by spaces, such as a file specification, option, or constant.

**command procedure** A file containing commands and data that the command interpreter can accept in lieu of the user typing the commands individually on a terminal.

**command string** A line (or set of continued lines), normally terminated by typing the carriage return key, containing a command and, optionally, information modifying the command. A complete command string consists of a command, its qualifiers, if any, and its parameters (file specifications, for example), if any, and their qualifiers, if any.

**common** A FORTRAN term for a program section that contains only data.

**common event flag cluster** A set of 32 event flags that enables cooperating processes to post event notification to each other. Common event flag clusters are created as they are needed. A process can associate with up to two common event flag clusters.

**compatibility mode** A mode of execution that enables the central processor to execute non-privileged PDP-11 instructions. The operating system supports compatibility mode execution by providing an RSX-11M programming environment for an RSX-11M task image. The operating system compatibility mode procedures reside in the control region of the process executing a compatibility mode image. The procedures intercept calls to the RSX-11M executive and convert them to the appropriate operating system functions.

**condition** An exception condition detected and declared by software. For example, see failure exception mode.

**condition codes** Four bits in the Processor Status Word that indicate the results of previously executed instructions.

**condition handler** A procedure that a process wants the system to execute when an exception condition occurs. When an exception condition occurs, the operating system searches for a condition handler and, if found, initiates the handler immediately. The condition handler may perform some action to change the situation that caused the exception condition and continue execution for the process that incurred the exception condition. Condition handlers execute in the context of the process at the access mode of the code that incurred the exception condition.

**condition value** A 32-bit quantity that uniquely identifies an exception condition.

**context** The environment of an activity. See also process context, hardware context, and software context.

**context indexing** The ability to index through a data structure automatically because the size of the data type is known and used to determine the offset factor.

**context switching** Interrupting the activity in progress and switching to another activity. Context switching occurs as one process after another is scheduled for execution. The operating system saves the interrupted process' hardware context in its hardware process control block (PCB) using the Save Process Context instruction, loads another process' hardware PCB into the hardware context using the Load Process Context instruction, and scheduling that process for execution.

**continuation character** A hyphen at the end of a command line signifying that the command string continues on to the next command line.

**console** The manual control unit integrated into the central processor. The console includes an LSI-11 microprocessor and a serial line interface connected to a hard copy terminal. It enables the operator to start and stop the system, monitor system operation, and run diagnostics.

**console terminal** The hard copy terminal connected to the central processor console.

**control region** The highest-addressed half of process space (the P1 region). Control region virtual addresses refer to the process-related information used by the system to control the process, such as: the kernel, executive, and supervisor stacks, the permanent I/O channels, exception vectors, and dynamically used system procedures (such as the command interpreter and RSX-11M programming environment compatibility mode procedures). The user stack is also normally found in the control region, although it can be relocated elsewhere.

**Control Region Base Register (P1BR)** The processor register, or its equivalent in a hardware process control block, that contains the base virtual address of a process control region page table.

**Control Region Length Register (P1LR)** The processor register, or its equivalent in a hardware process control block, that contains the number of non-existent page table entries for virtual pages in a process control region.

**copy-on-reference** A method used in memory management for sharing data until a process accesses it, in which case it is copied before the access. Copy-on-reference allows sharing of the initial values of a global section whose pages have read/write access but contain pre-initialized data available to many processes.
counted string A data structure consisting of a byte-sized length followed by the string. Although a counted string is not used as a procedure argument, it is a convenient representation in memory.

current access mode The processor access mode of the currently executing software. The Current Mode field of the Processor Status Longword indicates the access mode of the currently executing software.

cylinder The tracks at the same radius on all recording surfaces of a disk.

data base 1. All the occurrences of data described by a data base management system. 2. A collection of related data structures.

data structure Any table, list, array, queue, or tree whose format and access conventions are well-defined for reference by one or more images.

data type In general, the way in which bits are grouped and interpreted. In reference to the processor instructions, the data type of an operand identifies the size of the operand and the significance of the bits in the operand. Operands and data types include: byte, word, longword, and quadword integer, floating and double floating, character string, packed decimal string, and variable-length bit field.

defferred echo Refers to the fact that terminal echoing does not occur until a process is ready to accept input entered by type ahead.

delta time A time value expressing an offset from the current date and time. Delta times are always expressed in the system as negative numbers whose absolute value is used as an offset from the current time.

demand zero page A page, typically of an image stack or buffer area, that is initialized to contain all zeros when dynamically created in memory as a result of a page fault. This feature eliminates the waste of disk space that would otherwise be required to store blocks (pages) that contain only zeros.

descriptor A data structure used in calling sequences for passing argument types, addresses and other optional information. See character string descriptor.

detached process A process that has no owner. The parent process of a tree of subprocesses. Detached processes are created by the job controller when a user logs on the system or a when a batch job is initiated. The job controller does not own the user processes it creates; these processes are therefore detached.

device The general name for any physical terminus or link connected to the processor that is capable of receiving, storing, or transmitting data. Card readers, line printers, and terminals are examples of record-oriented devices. Magnetic tape devices and disk devices are examples of mass storage devices. Terminal line interfaces and interprocessor links are examples of communications devices.

device interrupt An interrupt received on interrupt priority level 16 through 23. Device interrupts can be requested only by devices, controllers, and memories.

device name The field in a file specification that identifies the device unit on which a file is stored. Device names also include the mnemonics that identify an I/O peripheral device in a data transfer request. A device name consists of a mnemonic followed by a controller identification letter (if applicable), followed by a unit number (if applicable). A colon (:) separates it from following fields.

device queue See spool queue.

device register A location in device controller logic used to request device functions (such as I/O transfers) and/or report status.

device unit One drive, and its controlling logic, of a mass storage device system. A mass storage system can have several drives connected to it.

diagnostic A program that tests logic and reports any faults it detects.

direct I/O An I/O operation in which the system locks the pages containing the associated buffer in memory for the duration of the I/O operation. The I/O transfer takes place directly from the process buffer. Contrast with system buffered I/O.

direct mapping cache A cache organization in which only one address comparison is needed to locate any data in the cache because any block of main memory data can be placed in only one possible position in the cache. Contrast with fully associative cache.

directory A file used to locate files on a volume that contains a list of file names (including type and version number) and their unique internal identifications.

directory name The field in a file specification that identifies the directory file in which a file is listed. The directory name begins with a left bracket ([] or <) and ends with a right bracket ([] or >).

displacement deferred indexed mode An indexed addressing mode in which the base operand specifier uses displacement deferred mode addressing.

displacement deferred mode In displacement deferred mode addressing, the specifier extension is a byte, word, or longword displacement. The displacement is sign extended to 32 bits and added to a base address obtained from the specified register. The result is the address of a longword which contains the address of the actual operand. If the PC is used as the register, the updated contents of the PC are used as the base address. The base address is the address of the first byte beyond the specifier extension.

displacement indexed mode An indexed addressing mode in which the base operand specifier uses displacement mode addressing.

displacement mode In displacement mode addressing, the specifier extension is a byte, word, or longword displacement. The displacement is sign extended to 32 bits and added to a base address obtained from the specified register. The result is the address of the actual operand. If the PC is used as the register, the updated contents of the PC are used as the base address. The base address is the address of the first byte beyond the specifier extension.

double floating datum Eight contiguous bytes (64 bits), starting on an addressable byte boundary, which are interpreted as containing a floating point number. The bits are labeled from right to left, 0 to 63. A four-word floating point number is identified by the address of the byte con-
taining bit 0. Bit 15 contains the sign of the number. Bits 14 through 7 contain the excess 128 binary exponent. Bits 63 through 16 and 6 through 0 contain a normalized 56-bit fraction with the redundant most significant fraction bit not represented. Within the fraction, bits of decreasing significance go from 6 through 0, 31 through 16, 47 through 32, then 63 through 48. Exponent values of 1 through 255 in the 8-bit exponent field represent true binary exponents of -128 to 127. An exponent value of 0 together with a sign bit of 0 represent a floating value of 0. An exponent value of 0 with a sign bit of 1 is a reserved representation; floating point instructions processing this value return a reserved operand fault. The value of a floating datum is in the approximate range (+ or -) 0.29 x 10^-38 to 1.7 x 10^38. The precision is approximately one part in 2^58 or sixteen decimal digits.

drive  The electro-mechanical unit of a mass storage device system on which a recording medium (disk cartridge, disk pack, or magnetic tape reel) is mounted.

driver  The set of code that handles physical I/O to a device.

dynamic access  A technique in which a program switches from one record access mode to another while processing a file.

echo  A terminal handling characteristic in which the characters typed by the terminal user on the keyboard are also displayed on the screen or printer.

effective address  The address obtained after indirect or indexing modifications are calculated.

entry mask  A word whose bits represent the registers to be saved or restored on a subroutine or procedure call using the call and return instructions.

entry point  A location that can be specified as the object of a call. It contains an entry mask and exception enables known as the entry point mask.

equivalence name  The string associated with a logical name in a logical name table. An equivalence name can be for example, a device name, another logical name, or a logical name concatenated with a portion of a file specification.

error logger  A system process that empties the error log buffers and writes the error messages into the error file. Errors logged by the system include memory system errors, device errors and timeouts, and interrupts with invalid vector addresses.

escape sequence  An escape is a transition from the normal mode of operation to a mode outside the normal mode. An escape character is the code that indicates the transition from normal to escape mode. An escape sequence refers to the set of character combinations starting with an escape character that the terminal transmits without interpretation to the software set up to handle escape sequences.

event  A change in process status or an indication of the occurrence of some activity that concerns an individual process or cooperating processes. An incident reported to the scheduler that affects a process' ability to execute. Events can be synchronous with the process' execution (a wait request), or they can be asynchronous (I/O completion). Some other events include: swapping; wake request; page fault.

event flag  A bit in an event flag cluster that can be set or cleared to indicate the occurrence of the event associated with that flag. Event flags are used to synchronize activities in a process or among many processes.

event flag cluster  A set of 32 event flags which are used for event posting. Four clusters are defined for each process: two process-local clusters, and two common event flag clusters. Of the process-local flags, eight are reserved for system use.

exception  An event detected by the hardware (other than an interrupt or jump, branch, case, or call instruction) that changes the normal flow of instruction execution. An exception is always caused by the execution of an instruction or set of instructions (whereas an interrupt is caused by an activity in the system independent of the current instruction). There are three types of hardware exceptions: traps, faults, and aborts. Examples are: attempts to execute a privileged or reserved instruction, trace traps, compatibility mode faults, breakpoint instruction execution, and arithmetic traps such as overflow, underflow, and divide by zero.

exception condition  A hardware- or software-detected event other than an interrupt or jump, branch, case, or call instruction that changes the normal flow of instruction execution.

exception dispatcher  An operating system procedure that searches for a condition handler when an exception condition occurs. If no exception handler is found for the exception or condition, the image that incurred the exception is terminated.

exception enables  See trap enables.

exception vector  See vector.

executable image  An image that is capable of being run in a process. When run, an executable image is read from a file for execution in a process.

executive  The generic name for the collection of procedures included in the operating system software that provides the basic control and monitoring functions of the operating system.

executive mode  The second most privileged processor access mode (mode 1). The record management services (RMS) and many of the operating system's programmed service procedures execute in executive mode.

exit  An image exit is a rundown activity that occurs when image execution terminates either normally or abnormally. Image rundown activities include deassigning I/O channels and disassociation of common event flag clusters. Any user- or system-specified exit handlers are called.

exit handler  A procedure executed when an image exits. An exit handler enables a procedure that is not on the call stack to gain control and clean up procedure-own data bases before the actual image exit occurs.

extended attribute block (XAB)  An RMS user data structure that contains additional file attributes beyond those expressed in the file access block (FAB), such as boundary types (aligned on cylinder, logical block number, virtual block number) and file protection information.
extension  The amount of space to allocate at the end of a file each time a sequential write exceeds the allocated length of the file.

extent  The contiguous area on a disk containing a file or a portion of a file. Consists of one or more clusters.

failure exception mode  A mode of execution selected by a process indicating that it wants an exception condition declared if an error occurs as the result of a system service call. The normal mode is for the system service to return an error status code for which the process must test.

fault  A hardware exception condition that occurs in the middle of an instruction and that leaves the registers and memory in a consistent state, such that elimination of the fault and restarting the instruction will give correct results.

field  1. See variable-length bit field. 2. A set of contiguous bytes in a logical record.

file access block (FAB)  An RMS user data structure that represents a request for data operations related to the file as a whole, such as OPEN, CLOSE, or CREATE.

file header  A block in the index file describing a file on a Files-11 disk structure. The file header identifies the locations of the file's extents. There is a file header for every file on the disk.

file name  The field preceding a file type in a file specification that contains a 1- to 9-character logical name for a file.

filename extension  See file type.

file organization  The particular file structure used to record the data comprising a file on a mass storage medium. RMS file organizations are: sequential, relative, direct, and indexed.

Files-11  The name of the on-disk structure used by the RSX-11, IAS and VAX operating systems. Volumes created under this structure are transportable between these operating systems.

file specification  A unique name for a file on a mass storage medium. It identifies the node, the device, the directory name, the file name, the file type, and the version number under which a file is stored.

file structure  The way in which the blocks forming a file are distributed on a disk or magnetic tape to provide a physical accessing technique suitable for the way in which the data in the file is processed.

file system  A method of recording, cataloging, and accessing files on a volume.

file type  The field in a file specification that is preceded by a period or dot (.) and consists of a zero- to three-character type identification. By convention, the type identifies a generic class of files that have the same use or characteristics, such as ASCII text files, binary object files, etc.

fixed control area  An area associated with a variable length record available for controlling or assisting record access operations. Typical uses include line numbers and printer format control information.

fixed length record format  A file format in which all records have the same length.

floating (point) datum  Four contiguous bytes (32 bits) starting on an addressable byte boundary. The bits are labeled from right to left from 0 to 31. A two-word floating point number is identified by the address of the byte containing bit 0. Bit 15 contains the sign of the number. Bits 14 through 7 contain the excess 128 binary exponent. Bits 31 through 16 and 6 through 0 contain a normalized 24-bit fraction with the redundant most significant fraction bit not represented. Within the fraction, bits of decreasing significance go from bit 6 through 0, then 31 through 16. Exponent values of 1 through 255 in the 8-bit exponent field represent true binary exponents of -128 to 127. An exponent value of 0 together with a sign bit of 0 represent a floating point value of 0. An exponent value of 0 with a sign bit of 1 is a reserved representation; floating point instructions processing this value return a reserved operand fault. The value of a floating datum is in the approximate range (+ or -) 0.29 x 10^-38 to 1.7 x 10^38. The precision is approximately one part in 2^23 or seven decimal digits.

foreign volume  Any volume other than a Files-11 formatted volume which may or may not be file structured.

fork process  A dynamically created system process such as a process that executes device driver code or the timer process. Fork processes have minimal context. Fork processes are scheduled by the hardware rather than by the software. The timer process is dispatched directly by software interrupt. I/O driver processes are dispatched by a fork dispatcher. Fork processes execute at software interrupt levels and are dispatched for execution immediately. Fork processes remain resident until they terminate.

frame pointer  General register 13 (R13). By convention, FP contains the base address of the most recent call frame on the stack.

fully associative cache  A cache organization in which any block of data from main memory can be placed anywhere in the cache. Address comparision must take place against each block in the cache to find any particular block. Constrast with direct mapping cache.

general register  Any of the sixteen 32-bit registers used as the primary operands of the native mode instructions. The general registers include 12 general purpose registers which can be used as accumulators, as counters, and as pointers to locations in main memory, and the Frame Pointer (FP), Argument Pointer (AP), Stack Pointer (SP), and Program Counter (PC) registers.

generic device name  A device name that identifies the type of device but not a particular unit; a device name in which the specific controller and/or unit number is omitted.

giga  Metric term used to represent the number 1 followed by nine zeros.

global page table  The page table containing the master page table entries for global sections.

global section  A data structure (e.g., FORTRAN global common) or sharable image section potentially available to all processes in the system. Access is protected by privilege and/or group number of the UIC.

global symbol  A symbol defined in a module that is potentially available for reference by another module. The linker resolves (matches references with definitions) global symbols. Contrast with local symbol.
global symbol table (GST)  In a library, an index of strongly defined global symbols used to access the modules defining the global symbols. The linker will also put global symbol tables into an image. For example, the linker appends a global symbol table to executable images that are intended to run under the symbolic debugger, and it appends a global symbol table to all sharable images.

**group** 1. A set of users who have special access privileges to each other's directories and files within those directories (unless protected otherwise), as in the context "system, owner, group, world," where group refers to all members of a particular owner's group. 2. A set of jobs (processes and their subprocesses) who have access privileges to a group's common event flags and logical name tables, and may have mutual process controlling privileges, such as scheduling, hibernation, etc.

**group number** The first number in a User Identification Code (UIC).

**hardware context** The values contained in the following registers while a process is executing: the Program Counter (PC); the Processor Status Longword (PSL); the 14 general registers (R0 through R13); the four processor registers (P0BR, P0LR, P1BR and P1LR) that describe the process virtual address space; the Stack Pointer (SP) for the current access mode in which the processor is executing; plus the contents to be loaded in the Stack Pointer for every access mode other than the current access mode. While a process is executing, its hardware context is continually being updated by the processor. While a process is not executing its hardware context is stored in its hardware PCB.

**hardware process control block (PCB)** A data structure known to the processor that contains the hardware context when a process is not executing. A process' hardware PCB resides in its process header.

**hibernation** A state in which a process is inactive, but known to the system with all of its current status. A hibernating process becomes active again when a wake request is issued. It can schedule a wake request before hibernating, or another process can issue its wake request. A hibernating process also becomes active for the time sufficient to service any AST it may receive while it is hibernating. Contrast with suspension.

**home block** A block in the index file that contains the volume identification, such as volume label and protection.

**image** An image consists of procedures and data that have been bound together by the linker. There are three types of images: executable, sharable, and system.

**image activator** A set of system procedures that prepare an image for execution. The image activator establishes the memory management data structures required both to map the image's virtual pages to physical pages and to perform paging.

**image exit** See exit.

**image I/O segment** That portion of the control region that contains the RMS internal file access blocks (IFAB) and I/O buffers for the image currently being executed by a process.

**image name** The file name of the file in which an image is stored.

**image privileges** The privileges assigned to an image when it is linked. See process privileges.

**image section (isect)** A group of program sections (psects) with the same attributes (such as read-only access, read/write access, absolute, relocatable, etc.) that is the unit of virtual memory allocation for an image.

**immediate mode** In immediate mode addressing, the PC is used as the register in autoincrement mode addressing.

**indexed addressing mode** In indexed mode addressing, two registers are used to determine the actual instruction operand: an index register and a base operand specifier. The contents of the index register are used as an index (offset) into a table or array. The base operand specifier supplies the base address of the array (the base operand address or BOA). The address of the actual operand is calculated by multiplying the contents of the index register by the size (in bytes) of the actual operand and adding the result to the base operand address. The addressing modes resulting from index mode addressing are formed by adding the suffix "indexed" to the addressing mode of the base operand specifier: register deferred indexed, autoncrement indexed, autoncrement deferred indexed (or absolute indexed), autodecrement indexed, displacement indexed, and displacement deferred indexed.

**indexed file organization** A file organization in which a file contains records and a primary key index (and optionally one or more alternate key indices) used to process the records sequentially by index or randomly by index.

**index file** The file on a Files-11 volume that contains the access information for all files on the volume and enables the operating system to identify and access the volume.

**index file bit map** A table in the index file of a Files-11 volume that indicates which file headers are in use.

**index register** A register used to contain an address offset.

**indirect command file** See command procedure.

**input stream** The source of commands and data. One of: the user's terminal, the batch stream, or an indirect command file.

**instruction buffer** An 8-byte buffer in the processor used to contain bytes of the instruction currently being decoded and to pre-fetch instructions in the instruction stream. The control logic continuously fetches data from memory to keep the 8-byte buffer full.

**interleaving** Assigning consecutive physical memory addresses alternately between two memory controllers.

**interprocess communication facility** A common event flag, mailbox, or global section used to pass information between two or more processes.

**interrecord gap** A blank space deliberately placed between data records on the recording surface of a magnetic tape.

**interrupt** An event other than an exception or branch, jump, case, or call instruction that changes the normal flow of instruction execution. Interrupts are generally external to the process executing when the interrupt occurs. See also device interrupt, software interrupt, and urgent interrupt.
interrupt priority level (IPL)  The interrupt level at which the processor executes when an interrupt is generated. There are 31 possible interrupt priority levels. IPL 1 is lowest, 31 highest. The levels arbitrate contention for processor service. For example, a device cannot interrupt the processor if the processor is currently executing at an interrupt priority level greater than the interrupt priority level of the device's interrupt service routine.

interrupt service routine  The routine executed when a device interrupt occurs.

interrupt stack  The system-wide stack used when executing in interrupt service context. At any time, the processor is either in a process context executing in user, supervisor, executive or kernel mode, or in system-wide interrupt service context operating with kernel privileges, as indicated by the interrupt stack and current mode bits in the PSL. The interrupt stack is not context switched.

interrupt stack pointer  The stack pointer for the interrupt stack. Unlike the stack pointers for process context stacks, which are stored in the hardware PCB, the interrupt stack pointer is stored in an internal register.

interrupt vector  See vector.

I/O driver  See driver.

I/O function  An I/O operation that is interpreted by the operating system and typically results in one or more physical I/O operations.

I/O function code  A 6-bit value specified in a Queue I/O Request system service that describes the particular I/O operation to be performed (e.g., read, write, rewind).

I/O function modifier  A 10-bit value specified in a Queue I/O Request system service that modifies an I/O function code (e.g., read terminal input no echo).

I/O lockdown  The state of a page such that it cannot be paged or swapped out of memory until any I/O in progress to that page is completed.

I/O rundown  An operating system function in which the system cleans up any I/O in progress when an image exits.

I/O space  The region of physical address space that contains the configuration registers, and device control/status and data registers.

I/O status block  A data structure associated with the Queue I/O Request system service. This service optionally returns a status code, number of bytes transferred, and device- and function-dependent information in an I/O status block. It is not returned from the service call, but filled in when the I/O request completes.

job  1. A job is the accounting unit equivalent to a process and the collection of all the subprocesses, if any, that it and its subprocesses create. Jobs are classified as batch and interactive. For example, the job controller creates an interactive job to handle a user's requests when the user logs onto the system and it creates a batch job when the system manager passes a command input file to it. 2. A print job.

job controller  The system process that establishes a job's process context, starts a process running the LOGIN image for the job, maintains the accounting record for the job, manages symbionts, and terminates a process and its subprocesses.

job queue  A list of files that a process has supplied for processing by a specific device, for example, a line printer.

kernel mode  The most privileged processor access mode (mode 0). The operating system's most privileged services, such as I/O drivers and the pager, run in kernel mode.

lexical function  A command language construct that the command interpreter evaluates and substitutes before it performs expression analysis on a command string. Lexical functions return information about the current process, such as UIC or default directory; and about character strings, such as length or substring locations.

librarian  A program that allows the user to create, update, modify, list, and maintain object library, image library, and assembler macro library files.

library file  A direct access file containing one or more modules of the same module type.

limit  The size or number of given items requiring system resources (such as mailboxes, locked pages, I/O requests, open files, etc.) that a job is allowed to have at any one time during execution, as specified by the system manager in the user authorization file. See also quota.

line number  A number used to identify a line of text in a file processed by a text editor.

linker  A program that reads one or more object files created by language processors and produces an executable image file, a sharable image file, or a system image file.

linking  The resolution of external references between object modules used to create an image, the acquisition of referenced library routines, service entry points, and data for the image, and the assignment of virtual addresses to components of an image.

literal mode  In literal mode addressing, the instruction operand is a constant whose value is expressed in a 6-bit field of the instruction. If the operand data type is byte, word, longword, or quadword, the operand is zero extended and can express values in the range 0 through 63 (decimal). If the operand data type is floating or double floating, the 6-bit field is composed of two 3-bit fields, one for the exponent and the other for the fraction. The operand is extended to floating or double floating format.

locality  See program locality.

local symbol  A symbol meaningful only to the module that defines it. Symbols not identified to a language processor as global symbols are considered to be local symbols. A language processor resolves (matches references with definitions) local symbols. They are not known to the linker and can not be made available to another object module. They can, however, be passed through the linker to the symbolic debugger. Contrast with global symbol.

locate mode  A record access technique in which a program reads records in an RMS block buffer working storage area to reduce overhead. See also move mode.

locking a page in memory  Making a page in an image ineligible for either paging or swapping. A page stays locked in memory until it is specifically unlocked.

locking a page in the working set  Making a page in an image ineligible for paging out of the working set for the
image. The page can be swapped when the process is swapped. A page stays locked in a working set until it is specifically unlocked.

logical block A block on a mass storage device identified using a volume-relative address rather than its physical (device-oriented) address or its virtual (file-relative) address. The blocks that constitute the volume are labeled sequentially starting with logical block 0.

logical I/O function A set of I/O operations (e.g., read and write logical block) that allow restricted direct access to device level I/O operations using logical block addresses.

logical name A user-specified name for any portion or all of a file specification. For example, the logical name INPUT can be assigned to a terminal device from which a program reads data entered by a user. Logical name assignments are maintained in logical name tables for each process, each group, and the system. A logical name can be created and assigned a value permanently or dynamically.

logical name table A table that contains a set of logical names and their equivalence names for a particular process, a particular group, or the system.

logical I/O functions A set of I/O functions that allow restricted direct access to device level I/O operations.

logical record A group of related fields treated as a unit.

longword Four contiguous bytes (32 bits) starting on an addressable byte boundary. Bits are numbered from right to left with 0 through 31. The address of the longword is the address of the byte containing bit 0. When interpreted arithmetically, a longword is a 2's complement integer with significance increasing from bit 0 to bit 30. When interpreted as a signed integer, bit 31 is the sign bit. The value of the signed integer is in the range -2,147,483,648 to 2,147,483,647. When interpreted as an unsigned integer, significance increases from bit 0 to bit 31. The value of the unsigned integer is in the range 0 through 4,294,967,295.

macro A statement that requests a language processor to generate a predefined set of instructions.

mailbox A software data structure that is treated as a record-oriented device for general interprocess communication. Communication using a mailbox is similar to other forms of device-independent I/O. Senders perform a write to a mailbox, the receiver performs a read from that mailbox. Some system-wide mailboxes are defined: the error logger and OPCOM read from system-wide mailboxes.

main memory See physical memory.

mapping window A subset of the retrieval information for a file that is used to translate virtual block numbers to logical block numbers.

mass storage device A device capable of reading and writing data on mass storage media such as a disk pack or a magnetic tape reel.

member number The second number in a user identification code that uniquely identifies that code.

memory management The system functions that include the hardware's page mapping and protection and the operating system's image activator and pager.

Memory Mapping Enable (MME) A bit in a processor register that governs address translation.

modify access type The specified operand of an instruction or procedure is read, and is potentially modified and written, during that instruction's or procedure's execution.

module 1. A portion of a program or program library, as in a source module, object module, or image module. 2. A board, usually made of plastic covered with an electrical conductor, on which logic devices (such as transistors, resistors, and memory chips) are mounted, and circuits connecting these devices are etched, as in a logic module.

Monitor Console Routine (MCR) The command interpreter in an RSX-11 system.

mount a volume 1. To logically associate a volume with the physical unit on which it is loaded (an activity accomplished by system software at the request of an operator). 2. To load or place a magnetic tape or disk pack on a drive and place the drive on-line (an activity accomplished by a system operator).

move mode A record I/O access technique in which a program accesses records in its own working storage area. See also locate mode.

mutex A semaphore that is used to control exclusive access to a region of code that can share a data structure or other resource. The mutex (mutual exclusion) semaphore ensures that only one process at a time has access to the region of code.

name block (NAM) An RMS user data structure that contains supplementary information used in parsing file specifications.

native image An image whose instructions are executed in native mode.

native mode The processor's primary execution mode in which the programmed instructions are interpreted as byte-aligned, variable-length instructions that operate on byte, word, longword, and quadword integer, floating and double floating, character string, packed decimal, and variable-length bit field data. The instruction execution mode other than compatibility mode.

network A collection of interconnected individual computer systems.

nibble The low-order or high-order four bits of a byte.

node An individual computer system in a network.

null process A small system process that is the lowest priority process in the system and takes one entire priority class. One function of the null process is to accumulate idle processor time.

numeric string A contiguous sequence of bytes representing up to 31 decimal digits (one per byte) and possibly a sign. The numeric string is specified by its lowest addressed location, its length, and its sign representation.

object module The binary output of a language processor such as the assembler or a compiler, which is used as input to the linker.

object time system (OTS) See Run Time Procedure Library.
offset A fixed displacement from the beginning of a data structure. System offsets for items within a data structure normally have an associated symbolic name used instead of the numeric displacement. Where symbols are defined, programmers always reference the symbolic names for items in a data structure instead of using the numeric displacement.

on-disk structure-1 (ODS-1) Refer to Files-11 structure Level 1.

on-disk structure-2 (ODS-2) Refer to Files-11 structure Level 2.

opcode The pattern of bits within an instruction that specify the operation to be performed.

operand specifier The pattern of bits in an instruction that indicate the addressing mode, a register and/or displacement, which, taken together, identify an instruction operand.

operand specifier type The access type and data type of an instruction's operand(s). For example, the test instructions are of read access type, since they only read the value of the operand. The operand can be of byte, word, or longword data type, depending on whether the opcode is for the TSTB (test byte), TSTW (test word), or TSTL (test longword) instruction.

Operator Communication Manager (OPCOM) A system process that is always active. OPCOM receives input from a process that wants to inform an operator of a particular status or condition, passes a message to the operator, and tracks the message.

operator's console Any terminal identified as a terminal attended by a system operator.

owner In the context "system, owner, group, world," an owner is the particular member (of a group) to which a file, global section, mailbox, or event flag cluster belongs.

owner process The process (with the exception of the job controller) or subprocess that created a subprocess.

packed decimal A method of representing a decimal number by storing a pair of decimal digits in one byte, taking advantage of the fact that only four bits are required to represent the numbers zero through nine.

packed decimal string A contiguous sequence of up to 16 bytes interpreted as a string of nibbles. Each nibble represents a digit except the low-order nibble of the highest addressed byte, which represents the sign. The packed decimal string is specified by its lowest addressed location and the number of digits.

page A set of 512 contiguous byte locations used as the unit of memory mapping and protection. 2. The data between the beginning of file and a page marker, between two markers, or between a marker and the end of a file.

page fault An exception generated by a reference to a page which is not mapped into a working set.

page fault cluster size The number of pages read in on a page fault.

page frame number (PFN) The address of the first byte of a page in physical memory. The high-order 21 bits of the physical address of the base of a page.

page marker A character or characters (generally a form feed) that separates pages in a file that is processed by a text editor.

pager A set of kernel mode procedures that executes as the result of a page fault. The pager makes the page for which the fault occurred available in physical memory so that the image can continue execution. The pager and the image activator provide the operating system’s memory management functions.

page table entry (PTE) The data structure that identifies the location and status of a page of virtual address space. When a virtual page is in memory, the PTE contains the page frame number needed to map the virtual page to a physical page. When it is not in memory, the page table entry contains the information needed to locate the page on secondary storage (disk).

paging The action of bringing pages of an executing process into physical memory when referenced. When a process executes, all of its pages are said to reside in virtual memory. Only the actively used pages, however, need to reside in physical memory. The remaining pages can reside on disk until they are needed in physical memory. In this system, a process is paged only when it references more pages than it is allowed to have in its working set. When the process refers to a page not in its working set, a page fault occurs. This causes the operating system’s pager to read in the referenced page if it is on disk (and, optionally, other related pages depending on a cluster factor), replacing the least recently faulted pages as needed. A process pages only against itself.

parameter See command parameter.

per-process address space See process address space.

physical address The address used by hardware to identify a location in physical memory or on directly addressable secondary storage devices such as a disk. A physical memory address consists of a page frame number and the number of a byte within the page. A physical disk block address consists of a cylinder or track and sector number.

physical address space The set of all possible 30-bit physical addresses that can be used to refer to locations in memory (memory space) or device registers (I/O space).

physical block A block on a mass storage device referred to by its physical (device-oriented) address rather than a logical (volume-relative) or virtual (file-relative) address.

physical I/O functions A set of I/O functions that allow access to all device level I/O operations except maintenance mode.

physical memory The memory modules connected to the SBI that are used to store: 1) instructions that the processor can directly fetch and execute, and 2) any other data that a processor is instructed to manipulate. Also called main memory.

position dependent code Code that can execute properly only in the locations in virtual address space that are assigned to it by the linker.

position independent code Code that can execute properly without modification wherever it is located in
virtual address space, even if its location is changed after it has been linked. Generally, this code uses addressing modes that form an effective address relative to the PC.

**primary vector** A location that contains the starting address of a condition handler to be executed when an exception condition occurs. If a primary vector is declared, that condition handler is the first handler to be executed.

**private section** An image section of a process that is not sharable among processes. See also global section.

**privilege** See process privilege, user privilege, and image privilege.

**privileged instructions** In general, any instructions intended for use by the operating system or privileged system programs. In particular, instructions that the processor will not execute unless the current access mode is kernel mode (e.g., HALT, SVPCTX, LDPCTX, MTPR, and MFPR).

**procedure** 1. A routine entered via a call instruction. 2. See command procedure.

**process** The basic entity scheduled by the system software that provides the context in which an image executes. A process consists of an address space and both hardware and software context.

**process address space** See process space.

**process context** The hardware and software contexts of a process.

**process control block (PCB)** A data structure used to contain process context. The hardware PCB contains the hardware context. The software PCB contains the software context, which includes a pointer to the hardware PCB.

**process header** A data structure that contains the hardware PCB, accounting and quota information, process section table, working set list, and the page tables defining the virtual layout of the process.

**process header slots** That portion of the system address space in which the system stores the process headers for the processes in the balance set. The number of process header slots in the system determines the number of processes that can be in the balance set at any one time.

**process identification (PID)** The operating system's unique 32-bit binary value assigned to a process.

**process I/O segment** That portion of a process control region that contains the process permanent RMS internal file access block for each open file, and the I/O buffers, including the command interpreter's command buffer and command descriptors.

**process name** A 1- to 15-character ASCII string that can be used to identify processes executing under the same group number.

**processor register** A part of the processor used by the operating system software to control the execution states of the computer system. They include the system base and length registers, the program and control region base and length registers, the system control block base register, the software interrupt request register, and many more.

**Processor Status Longword (PSL)** A system programmed processor register consisting of a word of privileged processor status and the PSW. The privileged processor status information includes: the current IPL (interrupt priority level), the previous access mode, the current access mode, the interrupt stack bit, the trace trap pending bit, and the compatibility mode bit.

**Processor Status Word (PSW)** The low-order word of the Processor Status Longword. Processor status information includes: the condition codes (carry, overflow, zero, negative), the arithmetic trap enable bits (integer overflow, decimal overflow, floating underflow), and the trace enable bit.

**process page tables** The page tables used to describe process virtual memory.

**process priority** The priority assigned to a process for scheduling purposes. The operating system recognizes 32 levels of process priority, where 0 is low and 31 high. Levels 16 through 31 are used for time-critical processes. The system does not modify the priority of a time-critical process (although the system manager or process itself may). Levels 0 through 15 are used for normal processes. The system may temporarily increase the priority of a normal process based on the activity of the process.

**process privileges** The privileges granted to a process by the system, which are a combination of user privileges and image privileges. They include, for example, the privilege to: affect other processes associated with the same group as the user's group, affect any process in the system regardless of UIC, set process swap mode, create permanent event flag clusters, create another process, create a mailbox, perform direct I/O to a file-structured device.

**process section** See private section.

**process space** The lowest-addressed half of virtual address space, where per-process instructions and data reside. Process space is divided into a program region and a control region.

**Program Counter (PC)** General register 15 (R15). At the beginning of an instruction's execution, the PC normally contains the address of a location in memory from which the processor will fetch the next instruction it will execute.

**program locality** A characteristic of a program that indicates how close or far apart the references to locations in virtual memory are over time. A program with a high degree of locality does not refer to many widely scattered virtual addresses in a short period of time.

**programmer number** See member number.

**program region** The lowest-addressed half of process address space (P0 space). The program region contains the image currently being executed by the process and other user code called by the image.

**Program region Base Register (POBR)** The processor register, or its equivalent in a hardware process control block, that contains the base virtual address of the page table entry for virtual page number 0 in a process program region.

**Program region Length Register (POLR)** The processor register, or its equivalent in a hardware process control block, that contains the number of entries in the page table for a process program region.

**program section (psect)** A portion of a program with a given protection and set of storage management attrib-
utes. Program sections that have the same attributes are gathered together by the linker to form an image section.

**project number**  See group number or account number.

**pure code**  See reentrant code.

**quadword**  Eight contiguous bytes (64 bits) starting on an addressable byte boundary. Bits are numbered from right to left, 0 to 63. A quadword is identified by the address of the byte containing the low-order bit (bit 0). When interpreted arithmetically, a quadword is a 2's complement integer with significance increasing from bit 0 to bit 62. Bit 63 is used as the sign bit. The value of the integer is in the range $-2^{63}$ to $2^{63}-1$.

**qualifier**  A portion of a command string that modifies a command verb or command parameter by selecting one of several options. A qualifier, if present, follows the command verb or parameter to which it applies and is in the format: "/*/qualifier:option". For example, in the command string "PRINT filename */COPIES:3", the COPIES qualifier indicates that the user wants three copies of a given file printed.

**queue**  1. n. A circular, doubly-linked list. See system queues. v. To make an entry in a list or table, perhaps using the INSOLE instruction. 2. See job queue.

**queue priority**  The priority assigned to a job placed in a spooler queue or a batch queue.

**quota**  The total amount of a system resource, such as CPU time, that a job is allowed to use in an accounting period, as specified by the system manager in the user authorization file. See also limit.

**random access by record's file address**  The retrieval of a record by its unique address, which is provided to the program by RMS. This method of access can be used to randomly access a sequentially organized file containing variable length records.

**random access by relative record number**  The retrieval or storage of a record by specifying its position relative to the beginning of the file.

**read access type**  An instruction or procedure operand attribute indicating that the specified operand is only read during instruction or procedure execution.

**record access block (RAB)**  An RMS user data structure that represents a request for a record access stream. A RAB relates to operations on the records within a file, such as UPDATE, DELETE, or GET.

**record access mode**  The method used in RMS for retrieving and storing record in a file. One of three methods: sequential, random, and record's file address.

**record cell**  A fixed-length area in a relatively organized file that is used to contain one record.

**Record Management Services**  A set of operating system system procedures that are called by programs to process files and records within files. RMS allows programs to issue READ and WRITE requests at the record level (record I/O) as well as read and write blocks (block I/O). RMS is an integral part of the system software. RMS procedures run in executive mode.

**record-oriented device**  A device such as a terminal, a line printer, or a card reader, on which the largest unit of data a program can access in one I/O operation is the device's physical record.

**record's file address**  The unique address of a record in a file that allows records to be accessed randomly regardless of file organization.

**record slot**  A fixed length area in a relatively organized file that is used to contain one record.

**reentrant code**  Code that is never modified during execution. It is possible to let many users share the same copy of a procedure or program written as reentrant code.

**register**  A storage location in hardware logic other than main memory. See also general register, processor register, and device register.

**register deferred indexed mode**  An indexed addressing mode in which the base operand specifier uses register deferred mode addressing.

**register deferred mode**  In register deferred mode addressing, the contents of the specified register are used as the address of the actual instruction operand.

**register mode**  In register mode addressing, the contents of the specified 'register are used as the actual instruction operand.

**relative file organization**  A file organization in which the file contains fixed length record slots. Each slot is assigned a consecutive number that represents its position relative to the beginning of a file. Records within each slot can be as big as or smaller than the slot. Relative file organization permits sequential record access, random record access by record number, and random record access by record's file address.

**resource**  A physical part of the computer system such as a device or memory, or an interlocked data structure such as a mutex. Quotas and limits control the use of physical resources.

**resource wait mode**  An execution state in which a process indicates that it will wait until a system resource becomes available when it issues a service request requiring a resource. If a process wants notification when a resource is not available, it can disable resource wait mode during program execution.

**return status code**  See status code.

**Run Time Procedure Library**  The collection of procedures available to native mode images at run time. These library procedures (such as trigonometric functions, etc.) are common to all native mode images, regardless of the language processor used to compile or assemble the program.

**scatter/gather**  The ability to transfer in one I/O operation data from contiguous pages in memory to contiguous blocks on disk, or data from contiguous blocks on disk to contiguous pages in memory.

**secondary storage**  Random access mass storage.

**secondary vector**  A location that identifies the starting address of a condition handler to be executed when a condition occurs and the primary vector contains zero or the handler to which the primary vector points chooses not to handle the condition.

**section**  A portion of process virtual memory that has common memory management attributes (protection, ac-
cess, cluster factor, etc.). It is created from an image section, a disk file, or as the result of a Create Virtual Address Space system service. See global section, private section, image section, and program section.

**sequential access mode** The retrieval or storage of records in which a program successively reads or writes records one after the other in the order in which they appear, starting and ending at any arbitrary point in the file.

**sequential file organization** A file organization in which records appear in the order in which they were originally written. The records can be fixed length or variable length. Although one does not speak of record slots with sequentially organized files, for purposes of comparison with relatively organized files one can say that the record itself is the same as its record slot, and its record number is the same as its relative slot number. Sequential file organization permits sequential record access and random access by record’s file address. Sequential file organization with fixed length records also permits random access by relative record number.

**sharable image** An image that has all of its internal references resolved, but which must be linked with an object module(s) to produce an executable image. A sharable image cannot be executed. A sharable image file can be used to contain a library of routines. A sharable image can be used to create a global section by the system manager.

**shell process** A predefined process that the job initiator copies to create the minimum context necessary to establish a process.

**signal** 1. An electrical impulse conveying information. 2. The software mechanism used to indicate that an exception condition was detected.

**slave terminal** A terminal from which it is not possible to issue commands to the command interpreter. A terminal assigned to application software.

**small process** A system process that has no control region in its virtual address space and has an abbreviated context. Examples are the working set swapper and the null process. A small process is scheduled in the same manner as user processes, but must remain resident during its execution.

**software context** The context maintained by the operating system that describes a process. See software process control block (PCB).

**software interrupt** An interrupt generated on interrupt priority level 1 through 15, which can be requested only by software.

**software process control block (PCB)** The data structure used to contain a process’ software context. The operating system defines a software PCB for every process when the process is created. The software PCB includes the following kinds of information about the process: current state; storage address if it is swapped out of memory; unique identification of the process, and address of the process header (which contains the hardware PCB). The software PCB resides in system region virtual address space. It is not swapped with a process.

**software priority** See process priority and queue priority.

**spooling** Output spooling: The method by which output to a low-speed peripheral device (such as a line printer) is placed into queues maintained on a high-speed device (such as disk) to await transmission to the low-speed device. Input spooling: The method by which input from a low-speed peripheral (such as the card reader) is placed into queues maintained on a high-speed device (such as disk) to await transmission to a job processing that input.

**spool queue** The list of files supplied by processes that are to be processed by a spooler. For example, a line printer queue is a list of files to be printed on the line printer.

**stack** An area of memory set aside for temporary storage, or for procedure and interrupt service linkages. A stack uses the last-in, first-out concept. As items are added to ("pushed on") the stack, the stack pointer decrements. As items are retrieved from ("popped off") the stack, the stack pointer increments.

**stack frame** A standard data structure built on the stack during a procedure call, starting from the location addressed by the FP to lower addresses, and popped off during a return from procedure. Also called call frame.

**Stack Pointer** General register 14 (R14). SP contains the address of the top (lowest address) of the processor-defined stack. Reference to SP will access one of the five possible stack pointers, kernel, executive, supervisor, user, or interrupt, depending on the value in the current mode and interrupt stack bits in the Processor Status Longword (PSL).

**state queue** A list of processes in a particular processing state. The scheduler uses state queues to keep track of processes’ eligibility to execute. They include: processes waiting for a common event flag, suspended processes, and executable processes.

**status code** A longword value that indicates the success or failure of a specific function. For example, system services always return a status code in R0 upon completion.

**store through** See write through.

**strong definition** Definition of a global symbol that is explicitly available for reference by modules linked with the module in which the definition occurs. The linker always lists a global symbol with a strong definition in the symbol portion of the map. The librarian always includes a global symbol with a strong definition in the global symbol table of a library.

**strong reference** A reference to a global symbol in an object module that requests the linker to report an error if it does not find a definition for the symbol during linking. If a library contains the definition, the linker incorporates the library module defining the global symbol into the image containing the strong reference.

**subprocess** A subsidiary process created by another process. The process that creates a subprocess is its owner. A subprocess receives resource quotas and limits from its owner. When an owner process is removed from the system, all its subprocesses (and their subprocesses) are also removed.

**supervisor mode** The third most privileged processor access mode (mode 2). The operating system's command interpreter runs in supervisor mode.
suspension  A state in which a process is inactive, but known to the system. A suspended process becomes active again only when another process requests the operating system to resume it. Contrast with hibernation.

swap mode  A process execution state that determines the eligibility of a process to be swapped out of the balance set. If process swap mode is disabled, the process working set is locked in the balance set.

swapping  The method for sharing memory resources among several processes by writing an entire working set to secondary storage (swap out) and reading another working set into memory (swap in). For example, a process' working set can be written to secondary storage while the process is waiting for I/O completion on a slow device. It is brought back into the balance set when I/O completes. Contrast with paging.

switch  See (command) qualifier.

symbiont  A full process that transfers record-oriented data to or from a mass storage device. For example, an input symbiont transfers data from a card reader to a disk. An output symbiont transfers data from a disk to a line printer.

symbiont manager  The function (in the system process called the job controller) that maintains spool queues, and dynamically creates symbiont processes to perform the necessary I/O operations.

symbol  See local symbol, global symbol, and universal global symbol.

Synchronous Backplane Interconnect (SBI)  The part of the hardware that interconnects the processor, memory controllers, MASSBUS adapters, the UNIBUS adapter.

synchronous record operation  A mode of record processing in which a user program issues a record read or write request and then waits until that request is fulfilled before continuing to execute.

system  In the context "system, owner, group, world," the system refers to the group numbers that are used by operating system and its controlling users, the system operators, and system manager.

system address space  See system space and system region.

System Base Register (SBR)  A processor register containing the physical address of the base of the system page table.

system buffered I/O  An I/O operation, such as terminal or mailbox I/O, in which an intermediate buffer from the system buffer pool is used instead of a process-specified buffer. Contrast with direct I/O.

System Control Block (SCB)  The data structure in system space that contains all the interrupt and exception vectors known to the system.

System Control Block Base register (SCBB)  A processor register containing the base address of the system control block.

system device  The random access mass storage device unit on which the volume containing the operating system software resides.

system dynamic memory  Memory reserved for the operating system to allocate as needed for temporary storage. For example, when an image issues an I/O request, system dynamic memory is used to contain the I/O request packet. Each process has a limit on the amount of system dynamic memory that can be allocated for its use at one time.

System Identification Register  A processor register which contains the processor type and serial number.

system image  The image that is read into memory from secondary storage when the system is started up.

System Length Register (SLR)  A processor register containing the length of the system page table in longwords, that is, the number of page table entries in the system region page table.

System Page Table (SPT)  The data structure that maps the system region virtual addresses, including the addresses used to refer to the process page tables. The system page table (SPT) contains one page table entry (PTE) for each page of system region virtual memory. The physical base address of the SPT is contained in a register called the SBR.

system process  A process that provides system-level functions. Any process that is part of the operating system. See also small process, fork process.

system programmer  A person who designs and/or writes operating systems, or who designs and writes procedures or programs that provide general purpose services for an application system.

system queue  A queue used and maintained by operating system procedures. See also state queues.

system region  The third quarter of virtual address space. The lowest-addressed half of system space. Virtual addresses in the system region are sharable between processes. Some of the data structures mapped by system region virtual addresses are: system entry vectors, the system control block (SCB), the system page table (SPT), and process page tables.

system services  Procedures provided by the operating system that can be called by user processes.

system space  The highest-addressed half of virtual address space. See also system region.

system virtual address  A virtual address identifying a location mapped by an address in system space.

system virtual space  See system space.

task  An RSX-11/IAS term for a process and image bound together.

terminal  The general name for those peripheral devices that have keyboards and video screens or printers. Under program control, a terminal enables people to type commands and data on the keyboard and receive messages on the video screen or printer. Examples of terminals are the LA36 DECwriter hard-copy terminal and VT52 video display terminal.

time-critical process  A process assigned to a software priority level between 16 and 31, inclusive. The scheduling priority assigned to a time-critical process is never modified by the scheduler, although it can be modified by the system manager or process itself.

timer  A system fork process that maintains the time of day and the date. It also scans for device timeouts and
performs time-dependent scheduling upon request.

**track** A collection of blocks at a single radius on one recording surface of a disk.

**transfer address** The address of the location containing a program entry point (the first instruction to execute).

**translation buffer** An internal processor cache containing translations for recently used virtual addresses.

**trap** An exception condition that occurs at the end of the instruction that caused the exception. The PC saved on the stack is the address of the next instruction that would normally have been executed. All software can enable and disable some of the trap conditions with a single instruction.

**trap enables** Three bits in the Processor Status Word that control the processor's action on certain arithmetic exceptions.

**two's complement** A binary representation for integers in which a negative number is one greater than the bit complement of the positive number.

**two-way associative cache** A cache organization which has two groups of directly mapped blocks. Each group contains several blocks for each index position in the cache. A block of data from main memory can go into any group at its proper index position. A two-way associative cache is a compromise between the extremes of fully associative and direct mapping cache organizations that takes advantage of the features of both.

**type ahead** A terminal handling technique in which the user can enter commands and data while the software is processing a previously entered command. The commands typed ahead are not echoed on the terminal until the command processor is ready to process them. They are held in a type ahead buffer.

**unit record device** A device such as a card reader or line printer.

**universal global symbol** A global symbol in a sharable image that can be used by modules linked with that sharable image. Universal global symbols are typically a subset of all the global symbols in a sharable image. When creating a sharable image, the linker ensures that universal global symbols remain available for reference after symbols have been resolved.

**unwind the call stack** To remove call frames from the stack by tracing back through nested procedure calls using the current contents of the FP register and the FP register contents stored on the stack for each call frame.

**urgent interrupt** An interrupt received on interrupt priority levels 24 through 31. These can be generated only by the processor for the interval clock, serious errors, and power fail.

**user authorization file** A file containing an entry for every user that the system manager authorizes to gain access to the system. Each entry identifies the user name, password, default account, User Identification Code (UIC), quotas, limits, and privileges assigned to individuals who use the system.

**user environment test package (UETP)** A collection of routines that verify that the hardware and software systems are complete, properly installed, and ready to be used.

**User File Directory (UFD)** See directory.

**User Identification Code (UIC)** The pair of numbers assigned to users and to files, global sections, common event flag clusters, and mailboxes that specifies the type of access (read and/or write access, and in the case of files, execute and/or delete access) available to the owners, group, world, and system. It consists of a group number and a member number separated by a comma.

**user mode** The least privileged processor access mode (mode 3). User processes and the Run Time Library procedures run in user mode.

**user name** The name that a person types on a terminal to log on to the system.

**user number** See member number.

**user privileges** The privileges granted a user by the system manager. See process privileges.

**utility** A program that provides a set of related general purpose functions, such as a program development utility (an editor, a linker, etc.), a file management utility (file copy or file format translation program), or operations management utility (disk backup/restore, diagnostic program, etc.).

**value return registers** The general registers R0 and R1 used by convention to return function values. These registers are not preserved by any called procedures. They are available as temporary registers to any called procedure. All other registers (R2, R3,...; R11, AP, FP, SP, PC) are preserved across procedure calls.

**variable-length bit field** A set of zero to 32 contiguous bits located arbitrarily with respect to byte boundaries. A variable bit field is specified by four attributes: 1) the address A of a byte, 2) the bit position P of the starting location of the bit field with respect to bit 0 of the byte at address A, 3) the size, in bits, of the bit field, and 4) whether the field is signed or unsigned.

**variable-length record format** A file format in which records are not necessarily the same length.

**variable with fixed-length control (VFC) record format** A file format in which records of variable length contain an additional fixed-length control area. The control area may be used to contain file line numbers and/or print format controls.

**vector** 1. A interrupt or exception vector is a storage location known to the system that contains the starting address of a procedure to be executed when a given interrupt or exception occurs. The system defines separate vectors for each interrupting device controller and for classes of exceptions. Each system vector is a longword. 2. For the purposes of exception handling, users can declare up to two software exception vectors (primary and secondary) for each of the four access modes. Each vector contains the address of a condition handler. 3. A one-dimensional array.

**version number** 1. The field following the file type in a file specification. It begins with a period (.) and is followed by a number which generally identifies it as the latest file created of all files having the identical file specification but for version number. 2. The number used to identify the revision level of program.
virtual address  A 32-bit integer identifying a byte “location” in virtual address space. The memory management hardware translates a virtual address to a physical address. The term virtual address may also refer to the address used to identify a virtual block on a mass storage device.

virtual address space  The set of all possible virtual addresses that an image executing in the context of a process can use to identify the location of an instruction or data. The virtual address space seen by the programmer is a linear array of 4,294,967,296 ($2^{32}$) byte addresses.

virtual block  A block on a mass storage device referred to by its file-relative address rather than its logical (volume-oriented) or physical (device-oriented) address. The first block in a file is always virtual block 1.

virtual I/O functions  A set of I/O functions that must be interpreted by an ancillary control process.

virtual memory  The set of storage locations in physical memory and on disk that are referred to by virtual addresses. From the programmer’s viewpoint, the secondary storage locations appear to be locations in physical memory. The size of virtual memory in any system depends on the amount of physical memory available and the amount of disk storage used for non-resident virtual memory.

virtual page number  The virtual address of a page of virtual memory.

volume  A mass storage medium such as a disk pack or reel of magnetic tape.

volume set  The file-structured collection of data residing on one or more mass storage media.

wait  To become inactive. A process enters a process wait state when the process suspends itself, hibernates, or declares that it needs to wait for an event, resource, mutex, etc.

wake  To activate a hibernating process. A hibernating process can be awakened by another process or by the timer process, if the hibernating process or any other process scheduled a wake-up call.

weak definition  Definition of a global symbol that is not explicitly available for reference by modules linked with the module in which the definition occurs. The librarian does not include a global symbol with a weak definition in the global symbol table of a library. Weak definitions are often used when creating libraries to identify those global symbols that are needed only if the module containing them is otherwise linked with a program.

weak reference  A reference to a global symbol that requests the linker not to report an error or to search the default library’s global symbol table to resolve the reference if the definition is not in the modules explicitly supplied to the linker. Weak references are often used when creating object modules to identify those global symbols that may not be needed at run time.

wild card  A symbol, such as an asterisk, that is used in place of a file name, file type, directory name, or version number in a file specification to indicate “all” for the given field.

window  See mapping window.

word  Two contiguous bytes (16 bits) starting on an addressable byte boundary. Bits are numbered from the right, 0 through 15. A word is identified by the address of the byte containing bit 0. When interpreted arithmetically, a word is a 2’s complement integer with significance increasing from bit 0 to bit 14. If interpreted as a signed integer, bit 15 is the sign bit. The value of the integer is in the range -32768 to 32767. When interpreted as an unsigned integer, significance increases from bit 0 through bit 15 and the value of the unsigned integer is in the range 0 through 65535.

working set  The set of pages in process space to which an executing process can refer without incurring a page fault. The working set must be resident in memory for the process to execute. The remaining pages of that process, if any, are either in memory and not in the process working set or they are on secondary storage.

working set swapper  A system process that brings process working sets into the balance set and removes them from the balance set.

world  In the context “system, owner, group, world,” world refers to all users, including the system operators, the system manager, and users both in an owner’s group and in any other group.

write access type  The specified operand of an instruction or procedure is written only during that instruction’s or procedure’s execution.

write allocate  A cache management technique in which cache is allocated on a write miss as well as on the usual read miss.

write back  A cache management technique in which data from a write operation to cache is copied into main memory only when the data in cache must be overwritten. This results in temporary inconsistencies between cache and main memory. Contrast with write through.

write through  A cache management technique in which data from a write operation is copied in both cache and main memory. Cache and main memory data are always consistent. Contrast with write back.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACP</td>
<td>Ancillary Control Process</td>
</tr>
<tr>
<td>ANS</td>
<td>American National Standard</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>AST</td>
<td>Asynchronous System Trap</td>
</tr>
<tr>
<td>ASTLVL</td>
<td>Asynchronous System Trap level</td>
</tr>
<tr>
<td>CCB</td>
<td>Channel Control Block</td>
</tr>
<tr>
<td>CM</td>
<td>Compatibility Mode bit in the hardware PSL</td>
</tr>
<tr>
<td>CRB</td>
<td>Cyclic Request Block</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>DAP</td>
<td>Data Access Protocol</td>
</tr>
<tr>
<td>DDB</td>
<td>Device Data Block</td>
</tr>
<tr>
<td>DDCMP</td>
<td>DIGITAL Data Communications Message Protocol</td>
</tr>
<tr>
<td>DDT</td>
<td>Driver Data Table</td>
</tr>
<tr>
<td>DV</td>
<td>Decimal Overflow trap enable bit in the PSW</td>
</tr>
<tr>
<td>ECB</td>
<td>Exit Control Block</td>
</tr>
<tr>
<td>ECC</td>
<td>Error Correction Code</td>
</tr>
<tr>
<td>ESP</td>
<td>Executive Mode Stack Pointer</td>
</tr>
<tr>
<td>ESR</td>
<td>Exception Service Routine</td>
</tr>
<tr>
<td>F11ACP</td>
<td>Files-11 Ancillary Control Process</td>
</tr>
<tr>
<td>FAB</td>
<td>File Access Block</td>
</tr>
<tr>
<td>FCA</td>
<td>Fixed Control Area</td>
</tr>
<tr>
<td>FCB</td>
<td>File Control Block</td>
</tr>
<tr>
<td>FCS</td>
<td>File Control Services</td>
</tr>
<tr>
<td>FDT</td>
<td>Function Decision Table</td>
</tr>
<tr>
<td>FP</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>FPD</td>
<td>First Part (of an instruction) Done</td>
</tr>
<tr>
<td>FU</td>
<td>Floating Underflow trap enable bit in the PSW</td>
</tr>
<tr>
<td>GSD</td>
<td>Global Section Descriptor</td>
</tr>
<tr>
<td>GST</td>
<td>Global Symbol Table</td>
</tr>
<tr>
<td>IDB</td>
<td>Interrupt Dispatch Block</td>
</tr>
<tr>
<td>IPL</td>
<td>Interrupt Priority Level</td>
</tr>
<tr>
<td>IRP</td>
<td>I/O Request Packet</td>
</tr>
<tr>
<td>ISECT</td>
<td>Image Section</td>
</tr>
<tr>
<td>ISD</td>
<td>Image Section Descriptor</td>
</tr>
<tr>
<td>ISP</td>
<td>Interrupt Stack Pointer</td>
</tr>
<tr>
<td>IS</td>
<td>Interrupt Stack bit in PSL</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt Service Routine</td>
</tr>
<tr>
<td>IV</td>
<td>Integer Overflow trap enable bit in the PSW</td>
</tr>
<tr>
<td>KSP</td>
<td>Kernel Mode Stack Pointer</td>
</tr>
<tr>
<td>MBA</td>
<td>MASSBUS Adapter</td>
</tr>
<tr>
<td>MBZ</td>
<td>Must Be Zero</td>
</tr>
<tr>
<td>MCR</td>
<td>Monitor Console Routine</td>
</tr>
<tr>
<td>MFD</td>
<td>Master File Directory</td>
</tr>
<tr>
<td>MFPR</td>
<td>Move From Process Register instruction</td>
</tr>
<tr>
<td>MME</td>
<td>Memory Mapping Enable</td>
</tr>
<tr>
<td>MTPR</td>
<td>Move To Process Register instruction</td>
</tr>
<tr>
<td>MUTEX</td>
<td>Mutual Exclusion semaphore</td>
</tr>
<tr>
<td>NSP</td>
<td>Network Services Protocol</td>
</tr>
<tr>
<td>OPCOM</td>
<td>Operator Communication Manager</td>
</tr>
<tr>
<td>P0BR</td>
<td>Program region base register</td>
</tr>
<tr>
<td>P0LR</td>
<td>Program region length register</td>
</tr>
<tr>
<td>P0PT</td>
<td>Program region page table</td>
</tr>
<tr>
<td>P1BR</td>
<td>Control region base register</td>
</tr>
<tr>
<td>P1LR</td>
<td>Control region limit register</td>
</tr>
<tr>
<td>P1PT</td>
<td>Control region page table</td>
</tr>
<tr>
<td>PC</td>
<td>Program Counter</td>
</tr>
<tr>
<td>PCB</td>
<td>Process Control Block</td>
</tr>
<tr>
<td>PCBB</td>
<td>Process Control Block Base register</td>
</tr>
<tr>
<td>PFN</td>
<td>Page Frame Number</td>
</tr>
<tr>
<td>PID</td>
<td>Process Identification Number</td>
</tr>
<tr>
<td>PME</td>
<td>Performance Monitor Enable bit in PCB</td>
</tr>
<tr>
<td>PSECT</td>
<td>Program Section</td>
</tr>
<tr>
<td>PSL</td>
<td>Processor Status Longword</td>
</tr>
<tr>
<td>PSW</td>
<td>Processor Status Word</td>
</tr>
<tr>
<td>PTE</td>
<td>Page Table Entry</td>
</tr>
<tr>
<td>QIO</td>
<td>Queue Input/Output Request system service</td>
</tr>
<tr>
<td>RAB</td>
<td>Record Access Block</td>
</tr>
<tr>
<td>RFA</td>
<td>Record's File Address</td>
</tr>
<tr>
<td>RMS</td>
<td>Record Management Services</td>
</tr>
<tr>
<td>RWED</td>
<td>Read, Write, Execute, Delete</td>
</tr>
<tr>
<td>SBI</td>
<td>Synchronous Backplane Interconnect</td>
</tr>
<tr>
<td>SBR</td>
<td>System Base Register</td>
</tr>
<tr>
<td>SCB</td>
<td>System Control Block</td>
</tr>
<tr>
<td>SCBB</td>
<td>System Control Block Base register</td>
</tr>
<tr>
<td>SLR</td>
<td>System Length Register</td>
</tr>
<tr>
<td>SP</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>SPT</td>
<td>System Page Table</td>
</tr>
<tr>
<td>SSP</td>
<td>Supervisor Mode Stack Pointer</td>
</tr>
<tr>
<td>SVA</td>
<td>System virtual address</td>
</tr>
<tr>
<td>TP</td>
<td>Trace trap Pending bit in PSL</td>
</tr>
<tr>
<td>UBA</td>
<td>UNIBUS Adapter</td>
</tr>
<tr>
<td>UCB</td>
<td>Unit Control Block</td>
</tr>
<tr>
<td>UETP</td>
<td>User Environment Test Package</td>
</tr>
<tr>
<td>UFD</td>
<td>User File Directory</td>
</tr>
<tr>
<td>UIC</td>
<td>User Identification Code</td>
</tr>
<tr>
<td>USP</td>
<td>User Mode Stack Pointer</td>
</tr>
<tr>
<td>VCB</td>
<td>Volume Control Block</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Page Number</td>
</tr>
<tr>
<td>WCB</td>
<td>Window Control Block</td>
</tr>
<tr>
<td>WCS</td>
<td>Writable Control Store</td>
</tr>
<tr>
<td>WDCS</td>
<td>Writable Diagnostic Control Store</td>
</tr>
</tbody>
</table>
Absolute mode addressing .................................. 4-7
Access modes ........................................... 2-2, 4-17, 4-18
Accounting statistics ..................................... 3-11
ACP (ancillary control process) ........................... 6-17, 6-18
Address
  manipulation instructions ............................. 4-10, 4-13
  mapping registers .................................... 4-17
  physical ........................................... 4-17 to 4-19
Address
  virtual ............................................. 4-16, 4-17
Addressing modes ....................................... 4-2, 4-5 to 4-7
Address sort ............................................ 7-15, 8-4
Address space
  physical ........................................... 4-18, 4-19
  virtual ............................................ 4-2, 4-4, 4-5, 4-18, 4-19, 4-6, 4-5
Address translation buffer .............................. 4-26
Ancillary control process (ACP) ........................ 6-17, 6-18
Application programming ............................... 3-1 to 3-9
Arbitration SBI ......................................... 4-25
Argument Pointer (AP) .................................. 4-5, 4-7, 4-8
Arguments passing ..................................... 4-4, 4-7, 4-8
Arithmetic traps ........................................ 4-8
Assembler (see also VAX-11, MACRO; Instruction set) .................................................. 7-1, 7-2
Asynchronous system trap processing services .... 6-6, 6-9
Autodecrement addressing mode ........................ 4-5, 4-6
Autoincrement addressing mode ......................... 4-5, 4-6
Autoincrement Deferred addressing mode ............. 4-5, 4-6
Automatic recovery power failure ...................... 3-13
Automatic restart ...................................... 2-4
Backup disks .......................................... 2-4
Backup files .......................................... 8-4
Bad block locator ...................................... 8-4
Bad blocks ............................................ 2-3, 5-2
Balance set ........................................... 6-16
Bandwidth memory ...................................... 2-3
Base priority BASIC-PLUS-2/VAX ................. 3-5, 7-10, 7-11
Batch processing ...................................... 3-5, 12-3, 13-13
Battery backup ........................................ 2-4, 4-25
Bit field ................................................ 4-4
BLISS-32 ............................................. 3-5, 17-13 to 7-15
Block I/O ............................................. 8-10
Blocks bad ............................................ 2-3, 5-2
Bootstrap .............................................. 2-4
Branch instructions ................................. 4-10, 13-14, 14-1
Buffers ............................................... 4-26
BUS
  MASSBUS ........................................... 2-1, 2-1, 2-1
  synchronous system ................................ 4-25
  UNIBUS ........................................... 2-1, 2-1, 2-1
Byte .................................................. 4-2, 4-3
Cache memory ........................................ 4-25, 4-26
CALL facility ......................................... 7-9, 7-11
Call frame ............................................ 4-8
Card reader .......................................... 5-3
Case instruction ...................................... 4-10, 4-13, 4-14
Character data ........................................ 4-4
Character string instructions ......................... 4-9, 4-11, 4-12
Clocks ................................................ 2-1
COBOL-74 ........................................... 3-5, 7-7 to 7-10
Command language ................................... 2-2, 2-5, 3-1 to 3-5
Command procedures ................................ 2-2, 2-5, 3-3-4
Command terminal ................................... 5-3
Commercial system example ........................... 3-6, 3-7
Communication between processes ..................... 2-2, 2-5, 6-6, 9
  interprocessor .................................... 5-5, 5-6
  network ........................................... 9-1
Compatibility mode .................................. 2-2, 4-2, 4-14, 4-16
  16, 18, 19, 20, 2-7
Computer Special Systems (CSS) group ............ 10-4
Condition codes ..................................... 4-4
Condition handlers ................................... 4-8, 6-9
Console ............................................. 2-1, 4-24, 4-25
Context .............................................. 4-5, 6-2
Context switching .................................... 4-16
Control region ....................................... 4-5, 6-4, 6-5
CR11 card reader .................................... 5-3
CSS (Computer Special Systems) group ............. 10-4
Customer Spares ..................................... 10-5
Data
  integrity ......................................... 2-3, 2-4
  management services ............................. 8-1 to 8-12
  protection transfer ................................ 6-4
  transfer DECnet .................................. 9-3 to 9-5
  throughput types .................................. 2-3
  COBOL-74 ......................................... 7-7
  VAX-11/180 ...................................... 4-2 to 4-4
DATATRIEVE/VAX .................................. 7-16, 7-17
DDCMP (DIGITAL Data Communications Message Protocol) ........................................ 5-5, 9-2
Debugger ............................................. 7-18
Debugger
  BASIC-PLUS-2 ..................................... 7-11
  BLISS-32 ......................................... 7-15
  COBOL-74 ......................................... 7-9
  FORTRAN IV-PLUS ................................ 7-6
  RPG II ............................................ 7-13
Decimal data .......................................... 4-4
DECnet ............................................... 9-1 to 9-7
DECUS .............................................. 10-4
Default device name .................................. 3-10
Definitions ........................................... G-1 to G-16
Detached process ..................................... 6-3
Device drivers ................................. 4-23, 4-24, 4-1, 4-6
  independence ..................................... 3-6
  name ............................................. 6-8
Diagnosics on-line ................................... 3-13
  peripherals ........................................ 2-4
  remote ............................................ 2-4, 13
DIGITAL Data Communications Message Protocol (DDCMP) ........................................... 5-5, 9-2
Digital Equipment Computer Users Society ......... 10-4
DIGITAL Network Architecture (DNA) ................. 9-2
Direct memory access (DMA) devices ................. 5-1, 5-4-5-5
Directory ............................................ 3-10, 8-2
Disks
  backup ............................................ 2-4
  supported ........................................ 5-1, 5-2
Disk Save and Compress (DSC) utility ............... 8-4
Displacement addressing mode ......................... 4-5, 4-6
Displacement Deferred addressing mode .............. 4-6
Distributed computing network ..................... 9-1
DMA (direct memory access) devices ................. 5-1, 5-4-5-5
DMC11 communications link ........................... 5-5, 5-6
DNA (DIGITAL Network Architecture) ................ 9-2
Down-line loading .................................... 9-6, 9-7
DR11-B direct memory access digital interface .... 5-5
Drivers device ...................................... 4-23, 4-24, 4-5, 4-6
DSC (Disk Save and Compress) utility ............... 8-4
Dynamic access ..................................... 8-7
DZ11 terminal line interface ........................ 5-4
Edit instruction ..................................... 4-11
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>2-3</td>
</tr>
<tr>
<td>Performance analysis statistics</td>
<td>3-11</td>
</tr>
<tr>
<td>Peripheral controller interfaces</td>
<td>4-26</td>
</tr>
<tr>
<td>Peripheral devices</td>
<td>2-1,5-1 to 5-6</td>
</tr>
<tr>
<td>Per-process space</td>
<td>4-5,6-5</td>
</tr>
<tr>
<td>Physical address</td>
<td>4-17 to 4-19</td>
</tr>
<tr>
<td>Physical memory</td>
<td>2-1</td>
</tr>
<tr>
<td>Position independent code</td>
<td>4-7</td>
</tr>
<tr>
<td>Power failure</td>
<td>3-13</td>
</tr>
<tr>
<td>Priority</td>
<td>2-2,2,5-4,16,4-23,6-15,6-16</td>
</tr>
<tr>
<td>Private pages</td>
<td>2-3</td>
</tr>
<tr>
<td>Privilege</td>
<td>3-10,3-11,6-4</td>
</tr>
<tr>
<td>Privileged instructions</td>
<td>4-10,4-18</td>
</tr>
<tr>
<td>Privileged operating system control services</td>
<td>6-7,6-8,6-16,8-17</td>
</tr>
<tr>
<td>Procedure control instructions</td>
<td>4-10,4-14,4-15</td>
</tr>
<tr>
<td>Process communication</td>
<td>2-2,3-10,6-11,9-3</td>
</tr>
<tr>
<td>control blocks</td>
<td>4-24</td>
</tr>
<tr>
<td>control services</td>
<td>6-16,8-17</td>
</tr>
<tr>
<td>context</td>
<td>4-1</td>
</tr>
<tr>
<td>description</td>
<td>3-5,3-6,4-16,2</td>
</tr>
<tr>
<td>mapping</td>
<td>6-12</td>
</tr>
<tr>
<td>page table</td>
<td>4-20,4-21</td>
</tr>
<tr>
<td>scheduling</td>
<td>6-15 to 6-17</td>
</tr>
<tr>
<td>virtual address space</td>
<td>4-2,4-4,4-5</td>
</tr>
<tr>
<td>virtual memory</td>
<td>6-12</td>
</tr>
<tr>
<td>Processing performance</td>
<td>2-3</td>
</tr>
<tr>
<td>Processor</td>
<td>4-17,4-18</td>
</tr>
<tr>
<td>access modes</td>
<td>2-1,4-1</td>
</tr>
<tr>
<td>Process-oriented paging (see also Paging)</td>
<td>2-1,2-2</td>
</tr>
<tr>
<td>Processor Status Longword (PSL)</td>
<td>4-17</td>
</tr>
<tr>
<td>Processor Status Word</td>
<td>4-8</td>
</tr>
<tr>
<td>Process state control system services</td>
<td>6-7,6-10,6-11</td>
</tr>
<tr>
<td>Program Counter (PC)</td>
<td>4-2,4-5,4-7</td>
</tr>
<tr>
<td>Program application</td>
<td>3-1 to 3-9</td>
</tr>
<tr>
<td>development tools</td>
<td>2-2,3-6,7,17 to 7-19</td>
</tr>
<tr>
<td>region</td>
<td>4-5,6,4-6,5</td>
</tr>
<tr>
<td>Programmable real-time clock</td>
<td>2-1</td>
</tr>
<tr>
<td>Programmed interrupt request devices</td>
<td>5-1</td>
</tr>
<tr>
<td>Programmed text editor</td>
<td>7-18</td>
</tr>
<tr>
<td>Programming interfaces</td>
<td>2-5</td>
</tr>
<tr>
<td>languages</td>
<td>2-1,2-2,3-5,7-1 to 7-19</td>
</tr>
<tr>
<td>Protection</td>
<td>2-3,4-17,4-18</td>
</tr>
<tr>
<td>Protection code</td>
<td>2-10</td>
</tr>
<tr>
<td>PSL (Processor Status Longword)</td>
<td>4-17</td>
</tr>
<tr>
<td>Quadword</td>
<td>4-2,4-3</td>
</tr>
<tr>
<td>Queue control</td>
<td>3-12</td>
</tr>
<tr>
<td>Queue instructions</td>
<td>4-10,4-12,4-13</td>
</tr>
<tr>
<td>Queue I/O Request processing</td>
<td>6-18,6-19</td>
</tr>
<tr>
<td>Quota</td>
<td>3-11</td>
</tr>
<tr>
<td>Random record access mode</td>
<td>8-7</td>
</tr>
<tr>
<td>Read error</td>
<td>5-2</td>
</tr>
<tr>
<td>Real-time clock</td>
<td>2-1</td>
</tr>
<tr>
<td>Real-time flight simulation example</td>
<td>3-8,3-9</td>
</tr>
<tr>
<td>Real-time processes memory management</td>
<td>3-6</td>
</tr>
<tr>
<td>priority</td>
<td>2-2,2-5</td>
</tr>
<tr>
<td>resource allocation</td>
<td>6-1,6-15,6-16</td>
</tr>
<tr>
<td>Record</td>
<td>8-7</td>
</tr>
<tr>
<td>access modes</td>
<td>8-6,8-7</td>
</tr>
<tr>
<td>attributes</td>
<td>8-8,8-9</td>
</tr>
<tr>
<td>processing</td>
<td>8-11,8-12</td>
</tr>
<tr>
<td>RMS</td>
<td>8-9,8-10</td>
</tr>
<tr>
<td>Record I/O</td>
<td>8-9,8-10</td>
</tr>
<tr>
<td>Record Management Services</td>
<td>2-5,3-6,17,8-4 to 8-12</td>
</tr>
<tr>
<td>Record's File Address (RFA) access mode</td>
<td>8-7</td>
</tr>
<tr>
<td>Record sort</td>
<td>7-15,8-4</td>
</tr>
<tr>
<td>Regions</td>
<td>4-5</td>
</tr>
<tr>
<td>Register addressing mode</td>
<td>4-5</td>
</tr>
<tr>
<td>Register Deferred addressing mode</td>
<td>4-5,4-6</td>
</tr>
<tr>
<td>Register manipulation instructions</td>
<td>4-10,4-13</td>
</tr>
<tr>
<td>Registers</td>
<td>4-2,4-5</td>
</tr>
<tr>
<td>Relative Files</td>
<td>8-5,8-10</td>
</tr>
<tr>
<td>Reliability features</td>
<td>2-3,2-4</td>
</tr>
<tr>
<td>Remote diagnostics</td>
<td>2-4,3-13</td>
</tr>
<tr>
<td>Resource allocation</td>
<td>6-3,6-4</td>
</tr>
<tr>
<td>quota</td>
<td>3-11</td>
</tr>
<tr>
<td>Resource-sharing network</td>
<td>9-1</td>
</tr>
<tr>
<td>Restart</td>
<td>2-4</td>
</tr>
<tr>
<td>RFA (Record’s File Address) access mode</td>
<td>8-7,8-9</td>
</tr>
<tr>
<td>RK06 disk drive</td>
<td>5-1,5-2</td>
</tr>
<tr>
<td>RK07 disk drive</td>
<td>5-1,5-2</td>
</tr>
<tr>
<td>RM03 disk drive</td>
<td>5-1</td>
</tr>
<tr>
<td>RMS-11 files</td>
<td>7-16,7-17</td>
</tr>
<tr>
<td>record management services</td>
<td>6-19</td>
</tr>
<tr>
<td>RMS (Record Management Services)</td>
<td>2-5,3-6,17,8-4 to 8-12</td>
</tr>
<tr>
<td>RP06 disk drive</td>
<td>5-1</td>
</tr>
<tr>
<td>RPG II/VAX (see PDP-11,compatibility)</td>
<td>3-5,7-11 to 7-13</td>
</tr>
<tr>
<td>RSX-11M (see PDP-11,compatibility)</td>
<td>7-13</td>
</tr>
<tr>
<td>SBI (Synchronous Backplane Interconnect)</td>
<td>4-1,4-25</td>
</tr>
<tr>
<td>Scatter/gather transfers</td>
<td>4-26</td>
</tr>
<tr>
<td>Scheduling</td>
<td>2-2,2-5,6,2-15 to 6-17</td>
</tr>
<tr>
<td>Sequential files</td>
<td>8-5,8,9-10</td>
</tr>
<tr>
<td>Sequential record access mode</td>
<td>8-6,8-7</td>
</tr>
<tr>
<td>Serial line multiplexer</td>
<td>5-4</td>
</tr>
<tr>
<td>Sharable image</td>
<td>6-11</td>
</tr>
<tr>
<td>SLP programmed text editor</td>
<td>7-18</td>
</tr>
<tr>
<td>Software components</td>
<td>10-5</td>
</tr>
<tr>
<td>Software process control block</td>
<td>4-24</td>
</tr>
<tr>
<td>Software services</td>
<td>10-2,10-3</td>
</tr>
<tr>
<td>SORT</td>
<td>7-15,7-16,8-4</td>
</tr>
<tr>
<td>SOS interactive text editor</td>
<td>7-18</td>
</tr>
<tr>
<td>Special function instructions</td>
<td>4-10,4-15</td>
</tr>
<tr>
<td>Spooling</td>
<td>2-4,3-12</td>
</tr>
<tr>
<td>Stack</td>
<td>4-4</td>
</tr>
<tr>
<td>Stack frame</td>
<td>4-4,4-8</td>
</tr>
<tr>
<td>Stack Pointer (SP)</td>
<td>4-5,4-7,4-8</td>
</tr>
<tr>
<td>String handling</td>
<td>7-10</td>
</tr>
<tr>
<td>BASIC-PLUS-2</td>
<td>7-7</td>
</tr>
<tr>
<td>COBOL-74</td>
<td>7-4</td>
</tr>
<tr>
<td>FORTRAN IV-PLUS</td>
<td>7-4</td>
</tr>
<tr>
<td>Subprocess</td>
<td>6-2</td>
</tr>
<tr>
<td>Subprograms</td>
<td>4-10,4-14</td>
</tr>
<tr>
<td>BASIC-PLUS-2</td>
<td>7-11</td>
</tr>
<tr>
<td>COBOL-74</td>
<td>7-9</td>
</tr>
<tr>
<td>Subroutine control instructions</td>
<td>4-10,4-14</td>
</tr>
<tr>
<td>Supplies</td>
<td>10-5</td>
</tr>
<tr>
<td>Support services</td>
<td>10-1 to 10-5</td>
</tr>
<tr>
<td>Swapping</td>
<td>6-16</td>
</tr>
<tr>
<td>Symbolic debugger</td>
<td>7-18</td>
</tr>
<tr>
<td>Symbolic Traceback Facility</td>
<td>7-6,7-9</td>
</tr>
<tr>
<td>Synchronous Backplane Interconnect (SBI)</td>
<td>4-1,4-25</td>
</tr>
<tr>
<td>System</td>
<td>3-13</td>
</tr>
<tr>
<td>automatic recovery</td>
<td>6-15</td>
</tr>
<tr>
<td>power failure</td>
<td>3-9 to 3-12</td>
</tr>
<tr>
<td>event</td>
<td>3-12,3-13</td>
</tr>
<tr>
<td>manager</td>
<td>3-9,4-21</td>
</tr>
<tr>
<td>operator</td>
<td>4-20,4-21</td>
</tr>
<tr>
<td>page table</td>
<td>4-16 to 4-24</td>
</tr>
<tr>
<td>region</td>
<td>4-5,6-4,6-5</td>
</tr>
<tr>
<td>services</td>
<td>6-5 to 6-11</td>
</tr>
<tr>
<td>space</td>
<td>4-5</td>
</tr>
<tr>
<td>System Control Block</td>
<td>4-21,4-22</td>
</tr>
<tr>
<td>Tag sort</td>
<td>7-15,8-4</td>
</tr>
<tr>
<td>TE16 tape storage system</td>
<td>5-2</td>
</tr>
<tr>
<td>Terminals</td>
<td>5-3,5-4</td>
</tr>
<tr>
<td>Terms</td>
<td>G-1 to G-16</td>
</tr>
<tr>
<td>definitions</td>
<td>7-18</td>
</tr>
<tr>
<td>Text editors</td>
<td>7-18</td>
</tr>
<tr>
<td>Time-of-year clock</td>
<td>2-1</td>
</tr>
<tr>
<td>Traceback</td>
<td>7-6,7-9</td>
</tr>
<tr>
<td>Trace traps</td>
<td>4-8</td>
</tr>
<tr>
<td>Transparent interprocess communication</td>
<td>9-3</td>
</tr>
<tr>
<td>Traps</td>
<td>4-8</td>
</tr>
<tr>
<td>TU45 tape storage system</td>
<td>5-2</td>
</tr>
<tr>
<td>Type-ahead</td>
<td>5-3</td>
</tr>
<tr>
<td>UETP (User Environment Test Package)</td>
<td>3-13,3-14</td>
</tr>
<tr>
<td>UIC (user identification code)</td>
<td>3-10,6-4</td>
</tr>
<tr>
<td>UNIBUS</td>
<td>2-1,4-26,5-2</td>
</tr>
<tr>
<td>UNIBUS space</td>
<td>4-23</td>
</tr>
<tr>
<td>Topic</td>
<td>Pages</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Unit record devices</td>
<td>5-2,5-3</td>
</tr>
<tr>
<td>User authorization</td>
<td>3-10</td>
</tr>
<tr>
<td>User authorization file</td>
<td>6-4</td>
</tr>
<tr>
<td>User Environment Test Package (UETP)</td>
<td>3-13,3-14</td>
</tr>
<tr>
<td>User identification code (UIC)</td>
<td>3-10,6-4</td>
</tr>
<tr>
<td>Users society</td>
<td>10-4</td>
</tr>
<tr>
<td>Variable-length bit field instructions</td>
<td>4-9,4-12</td>
</tr>
<tr>
<td>VAX-11</td>
<td>3-5,7-7 to 7-10</td>
</tr>
<tr>
<td>FORTRAN IV-PLUS</td>
<td>3-5,7-2 to 7-6,9-5</td>
</tr>
<tr>
<td>MACRO</td>
<td>7-1,7-2,9-6</td>
</tr>
<tr>
<td>SORT</td>
<td>7-15,7-16,8-4</td>
</tr>
<tr>
<td>VAX-11/780</td>
<td>7-1 to 7-19</td>
</tr>
<tr>
<td>languages</td>
<td>2-1 to 2-5</td>
</tr>
<tr>
<td>overview</td>
<td>2-3</td>
</tr>
<tr>
<td>performance</td>
<td>7-17 to 7-19</td>
</tr>
<tr>
<td>program development tools</td>
<td>2-3,2-4</td>
</tr>
<tr>
<td>reliability features</td>
<td>10-1 to 10-5</td>
</tr>
<tr>
<td>support services</td>
<td>G-1 to G-17</td>
</tr>
<tr>
<td>terminology</td>
<td></td>
</tr>
<tr>
<td>VAX/VMS</td>
<td>2-2,2-5,3-1 to 3-5</td>
</tr>
<tr>
<td>command language</td>
<td></td>
</tr>
<tr>
<td>DEBUG program</td>
<td>7-6</td>
</tr>
<tr>
<td>operating system (see also Operating system)</td>
<td>2-1 to 2-3,6-1 to 6-20</td>
</tr>
<tr>
<td>Vectors</td>
<td>4-21,4-22</td>
</tr>
<tr>
<td>Video terminal</td>
<td>5-4</td>
</tr>
<tr>
<td>Virtual addressing</td>
<td>4-2,4-4,4-5,4-16 to 4-19,6-4,6-5</td>
</tr>
<tr>
<td>Virtual memory</td>
<td></td>
</tr>
<tr>
<td>operating system</td>
<td>2-1 to 2-3</td>
</tr>
<tr>
<td>process</td>
<td>6-12</td>
</tr>
<tr>
<td>programming considerations</td>
<td>6-14,6-15</td>
</tr>
<tr>
<td>VT100 video terminal</td>
<td>5-4</td>
</tr>
<tr>
<td>Word</td>
<td>4-2,4-3</td>
</tr>
<tr>
<td>Working set</td>
<td>6-12,6-13</td>
</tr>
</tbody>
</table>