Guide to VAX RPG II

Order Number: AA–JA05A–TE

August 1986

This manual describes language elements, programming constructs, and features of the VAX RPG II language.

Revision/Update Information: This revised manual supersedes *Programming in VAX RPG II* (Order No. AA–R431B–TE)

Operating System and Version: VAX/VMS Version 4.4 or higher
MicroVMS Version 4.4 or higher

Software Version: VAX RPG II Version 2.1

digital equipment corporation
maynard, massachusetts

First Printing, February 1984
Revised, November 1985
Revised, August 1986

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem-10	PDP	VT
DECSYSTEM-20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital

ZK-3253

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a trademark of the American Mathematical Society.

Contents

USING VAX RPG II FEATURES ON VMS

LANGUAGE REFERENCE

FIGURES

TABLES

Preface

This manual describes the features, uses, constructs, and syntax of the VAX RPG II programming language on VAX/VMS systems.

Structure of This Document

This manual contains 16 chapters and 3 appendixes.

Chapter 1 explains the VAX RPG II logic cycle.

Chapter 2 explains how to use the VAX RPG II editor to create and edit RPG II programs.

Chapter 3 explains how to compile, link, and run programs.

Chapter 4 explains the format of a listing file.

Chapter 5 explains how to use the VAX/VMS Debugger to debug VAX RPG II programs.

Chapter 6 explains how to perform screen activities with VAX RPG II programs.

Chapter 7 explains how to use VAX RPG II indicators.

Chapter 8 explains how to manage files.

Chapter 9 explains those elements that affect printer output files.

Chapter 10 explains how to create and access tables.

Chapter 11 explains how to create and access arrays.

Chapter 12 explains how to use the VAX RPG II CALL interface to access VAX/VMS Run-Time Library procedures, system services, and subprograms.

Chapter 13 explains how to improve the efficiency of programs.

Chapter 14 explains VAX RPG II elements and data types.

Chapter 15 lists specifications, allowable entries, and their functions.

Chapter 16 explains how to use VAX RPG II operation codes.

Appendix A lists the VAX RPG II character sets.

Appendix B explains the differences between VAX RPG II and PDP-11 RPG II.

Appendix C shows the VAX Performance Coverage Analyzer applied to a VAX RPG II program.

Intended Audience

This manual is intended for use by programmers familiar with computer programming fundamentals and the RPG II language. It is designed to be used both as a reference manual and as a user's guide.

Conventions Used in This Document

Conventions	Meaning
▮	The VAX RPG II editor cursor is represented by a box.
[]	Brackets enclose an optional portion of a format.
{ }	Braces enclose a mandatory portion of a format.
. . .	A vertical ellipsis indicates that not all of the program lines in an example are shown.
$ SHOW TIME 05-AUG-1986 11:55:22	Command examples show all output lines or prompting characters that the system prints or displays in black letters. All user-entered commands are shown in red letters.

Associated Documents

If you need additional information on VAX/VMS you should refer to the following manuals:

- *VAX/VMS Debugger Reference Manual*
- *VAX/VMS Linker Reference Manual*
- *VAX/VMS Librarian Reference Manual*

- *VAX/VMS Run-Time Library Routines Reference Manual*
- *VAX/VMS System Services Reference Manual*
- *VAX/VMS Utility Routines Reference Manual*
- *VAX/VMS System Manager's Reference Manual*
- *VAX/VMS Record Management Services Reference Manual*

Developing VAX RPG II Programs on VMS

This part of the manual provides information on the development of VAX RPG II programs:

- Creating the source
- Using the VAX RPG II editor
- Compiling, linking, and running the program
- Interpreting a VAX RPG II compiler listing
- Using the VAX/VMS Debugger

Chapter 1

Understanding the VAX RPG II Logic Cycle

VAX RPG II is a language processor that provides a convenient and efficient means of preparing a wide variety of reports as well as performing other commercial data processing tasks. VAX RPG II is an extended implementation of the RPG II language that was developed by IBM; it includes DIGITAL extensions for integration with the VAX/VMS architecture. VAX RPG II runs on the VAX/VMS or MicroVMS operating system and consists of a compiler, special editor, run-time support, conversion utility for existing screen definitions, and sample programs showing use of VAX RPG II in screen handling applications.

VAX RPG II is a nonprocedural language; that is, every program compiled by the VAX RPG II compiler executes according to a fixed plan. Unlike a procedural language such as COBOL, the logic of this plan is not supplied by the programmer, but is built into the compiler. This built-in logic is called the VAX RPG II **logic cycle**. The execution of a VAX RPG II program consists of a number of iterations of the logic cycle.

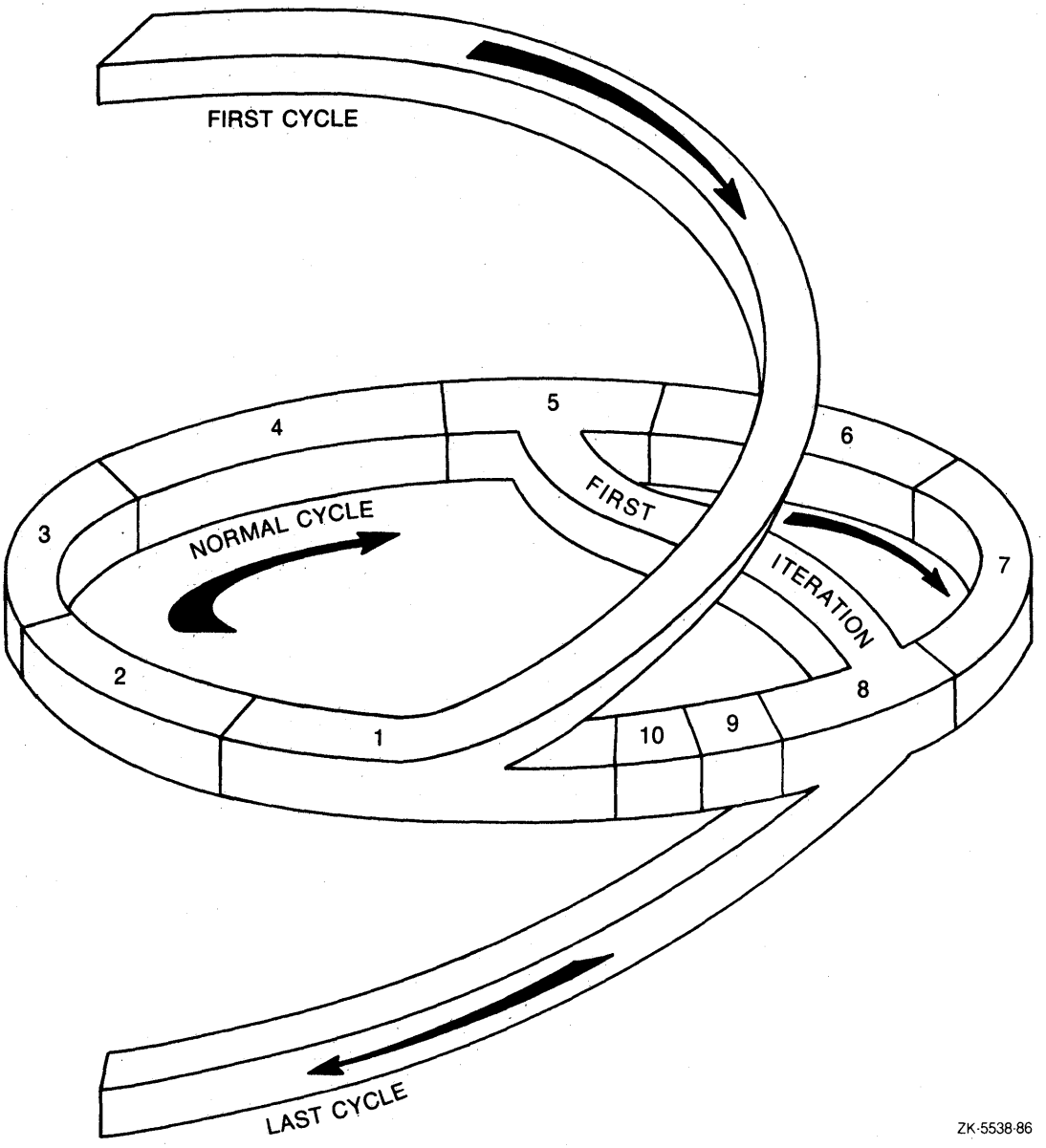
You provide directions to the program using VAX RPG II **specifications**. Specifications are the record-oriented instructions by which you define input and output formats, arithmetic processes, and special operations to be performed depending on the data input and desired output. There are seven different specification types. Each specification is structured in an 80-column format; the columns are grouped into fields according to the purpose of the specification. You choose a specification according to the program function you wish to do and supply input in the fields according to the specific application. See Chapter 15 for an explanation of each specification. The VAX RPG II specifications you include determine what happens within the various phases of the logic cycle, but do not change the basic sequence of program execution.

For example, you can write an Input specification to program VAX RPG II to recognize and process a particular input file record type; you *cannot* program VAX RPG II to read three input records in a row, print a report heading, load a table, write four different output records, and then perform total calculations. This series of steps, while perfectly acceptable in a COBOL program, does not fit into the structure of the VAX RPG II logic cycle.

The VAX RPG II fixed logic cycle was designed specifically to accommodate the sequence of operations needed to generate most common business reports and file maintenance functions. However, the fixed nature of the VAX RPG II logic cycle does not prevent you from controlling the set of functions performed for each input record and, to some extent, the sequence and timing of these functions. This control is provided by the use of **indicators**. For example, by setting various indicators on or off when certain conditions occur, you can affect the sequence of program execution within the phases of the normal logic cycle. See Chapter 7 for more information on indicators. Therefore, to write effective VAX RPG II specifications and to take advantage of what flexibility and control VAX RPG II does provide, you must thoroughly understand the structure and timing characteristics of the overall VAX RPG II logic cycle, and recognize both VAX RPG II's special capabilities and its limitations.

There are 10 fundamental operations that are performed during the operation (and reiterations) of the VAX RPG II logic cycle, as shown in Figure 1–1 and identified in the accompanying keyed list.

Figure 1–1: Logical Flow of the VAX RPG II Logic Cycle



Key to Figure 1–1:

In the **FIRST CYCLE**, several initialization tasks are accomplished. Files are opened, the local data area is loaded, the system date is obtained from the system, and counters, tables, and arrays are initialized. The program then transits into the **NORMAL CYCLE**.

❶ When the **NORMAL CYCLE** is entered, program instruction lines with special conditions for Output specifications are processed.

❷ All control-level and record-identifying indicators are set off.

❸ A record is read, and the appropriate record-identifying indicator is set on.

❹ If the control field of the record just read is different from the control field of the previous record, a *control break* occurs. When a control break does occur, the appropriate control-level indicator is set on, plus all lower control-level indicators.

❺ If VAX RPG II detects that this is the first iteration after the **FIRST CYCLE**, it branches to step **8**.

❻ Total calculations (conditioned by control-level indicators in the Calculation specifications) are processed if the appropriate control-level indicators are on.

❼ If totals are requested in the program, the program does total output operations.

❽ If the last-record indicator is on, the **LAST CYCLE** is entered and the program ends.

❾ Detail calculations and program output are processed from data read in this cycle.

❿ All remaining detail calculations are processed on the data from the current record read at the beginning of the **NORMAL CYCLE**. Then, the cycle continues around, repeating while there are records to process.

Upon detecting the last-record indicator, the program enters the **LAST CYCLE**, does one-time cleanup work including total calculations and total output operations, and the program ends.

1.1 The VAX RPG II General Logical Sequence

Every VAX RPG II program follows the same sequence of execution steps which form the general or **normal logic cycle**. Some of the programs you write will require you to use one or more of the additional operations of VAX RPG II: matching fields (described in Section 8.11, chaining (described in Section 16.7.1), overflow processing (described in Section 9.3.2), and look-ahead processing (described in Section 15.6.8). Each of these additional operations executes according to the fixed logic cycle within the general logical sequence of the program. These functions are discussed later in this chapter.

The VAX RPG II logic cycle is executed once for each input record. The logic cycle consists of the following three steps, performed in order for each record:

1. Inputting a record
2. Performing calculations
3. Outputting one or more records

Each logic cycle begins when a new record is input and ends just before the next record is input. The VAX RPG II specifications you develop determine the range and type of specific functions performed during each phase. During the calculation and output steps within each cycle, there are two distinct timing phases:

- Total time—operations are performed on summary data accumulated from a group of related records.
- Detail time—operations are performed on individual records.

Sections 1.4.1 and 1.4.2 describe total-time and detail-time characteristics and operations.

The first and last iterations of the VAX RPG II logic cycle are somewhat different from all other iterations. Sections 1.2 and 1.3 describe these differences and explain how you can take advantage of them.

1.2 The First Cycle

When program execution begins and before the first input record is read, several one-time-only operations are performed, which constitutes the **first cycle**. You can exert control over this process by providing detail-time output records conditioned by the first-page (1P) indicator, and by using Output specifications with either no conditioning indicators or with all negative conditioning indicators. (See Chapter 7 for more details on conditioning indicators.) During the first cycle, VAX RPG II performs the following initialization operations:

- Obtains the current (system) date (UDATE, UDAY, UMONTH, and UYEAR (see Chapter 9).
- Loads the local data area.
- Opens all files (see Chapters 6 and 8).
- Loads preexecution-time tables and arrays (see Chapters 10 and 11).
- Initializes page number counters.
- Prints heading and detail lines conditioned by the 1P indicator, by all negative indicators other than the 1P indicator, and by no indicators.

Although all iterations of the logic cycle (other than the first) include a total-time phase, VAX RPG II bypasses all total-time calculations and total-time steps during the first cycle unless the last-record (LR) indicator is on. This behavior, like the logic cycle, is built into VAX RPG II.

After initialization tasks are performed, VAX RPG II reads the first record in the primary file, if used, and then reads the first record in each secondary file, if used, and determines the type of each record read. (The distinction between primary, secondary, and demand files is complex and is discussed in Chapters 6 and 15.)

1.3 The Last Cycle

The **last cycle** is performed after all the records you specified for processing have been read from all primary and secondary files. When the last record from the last file has been read, VAX RPG II sets on the LR indicator and all the control-level indicators (L1 through L9). Then, after this last record has been processed, VAX RPG II performs the following operations:

- Performs total-time calculations.
- Writes total-time output.
- Outputs any tables or arrays that have output files associated with them.
- Closes all files. (Some files are left to be closed automatically by the VAX Record Management System (VAX RMS) when the program ends execution.)
- Outputs the local data area.
- Ends program execution.

1.4 A Normal Cycle

A **normal cycle** (sometimes referred to as the general cycle) in a VAX RPG II program can be defined as any cycle but the first or the last. During a normal cycle, VAX RPG II performs all operations necessary to process a single input record. Because of the nature of most VAX RPG II applications, a normal program cycle includes two special phases—total-time and detail-time. Total time occurs before detail time because detail time is for the *previous* record. When the first record is read, there is no previous record; hence there is no detail time. Note that the cycle is a closed loop. In a normal cycle, VAX RPG II performs the following sequence of steps:

1. Outputs heading lines, if specified
2. Outputs detail-time information pertaining to the previous record
3. Reads an input record
4. Performs total-time calculations for the previous record, if required
5. Performs total-time output

6. Checks the LR indicator; if it is on, VAX RPG II terminates the program (see Section 1.3)
7. Processes the record read in step 3; performs all detail-time calculations

Steps 4 and 5 constitute total time; steps 1, 2, and 7 constitute detail time.

This list of steps in a normal cycle is an overview only. Figure 1–2 is a detailed flowchart of a normal VAX RPG II logic cycle.

1.4.1 Total Time

During total time, VAX RPG II checks which control-level indicators (L1 through L9) you have defined and the control field you have associated with each. (See Chapter 7 for details on using control-level indicators.) For example, if your application involves the generation of a monthly sales report, you may have associated indicator L9 with the grand total of monthly sales, indicator L8 with total sales by region, indicator L7 with total sales by district office, and indicator L6 with total sales by salesperson. This is shown in the following example:

```
                                                    Field length
     Control level                                  | Decimal positions
     |                                              | |Half adjust (H)
     | Indicators          Operation                | ||
     | |                   |                        | ||Resulting
     | |         Factor    |    Factor    Result|   | |||indicators
     | |         1         |    2         field |   | |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   | |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *         *   . *          *     *--*** * * *
     CL9           MONTH     MULT  12         GRAND
     CL8           REGION    ADD   REGION     TOTREG
     CL7           DIST      ADD   DIST       TOTDIS
```

ZK-4386-85

If during a particular cycle VAX RPG II determines that the salesperson identification number in the record just read is different from the salesperson number in the previous record, a **control break** occurs at the salesperson total level. At this control break, your program outputs the accumulated total sales for the salesperson whose number was found in the previous record.

You might print each person's name and sales total on a separate line. Or, you might choose to print a page heading (with a date), and then print a salesperson's total sales, thus providing a separate one-page report for each salesperson. The Output and Calculation specifications you develop determine the contents, order, and appearance of your report.

During another cycle, it might be determined that the region identifier of the record just read is different from the region number in the previous record. Given the control-level indicators described in the preceding example (L9, L8, L7, L6), this means that a three-level control break has occurred. In this situation, you must first output the accumulated total for an individual salesperson (L6), then the accumulated total for a district office (L7) and, finally, the accumulated total for the region (L8).

Similarly, after the last input record in the file is read, a four-level control break occurs automatically. At that point, your program must first output the accumulated total for the last individual salesperson in the last district office (L6); then, the accumulated total for the last district office (L7); then, the accumulated total for the last region (L8); and, finally, the accumulated grand total of all sales for the month (L9).

After all control breaks have occurred, total time ends and detail time begins. Detail-time operations are for the record just read.

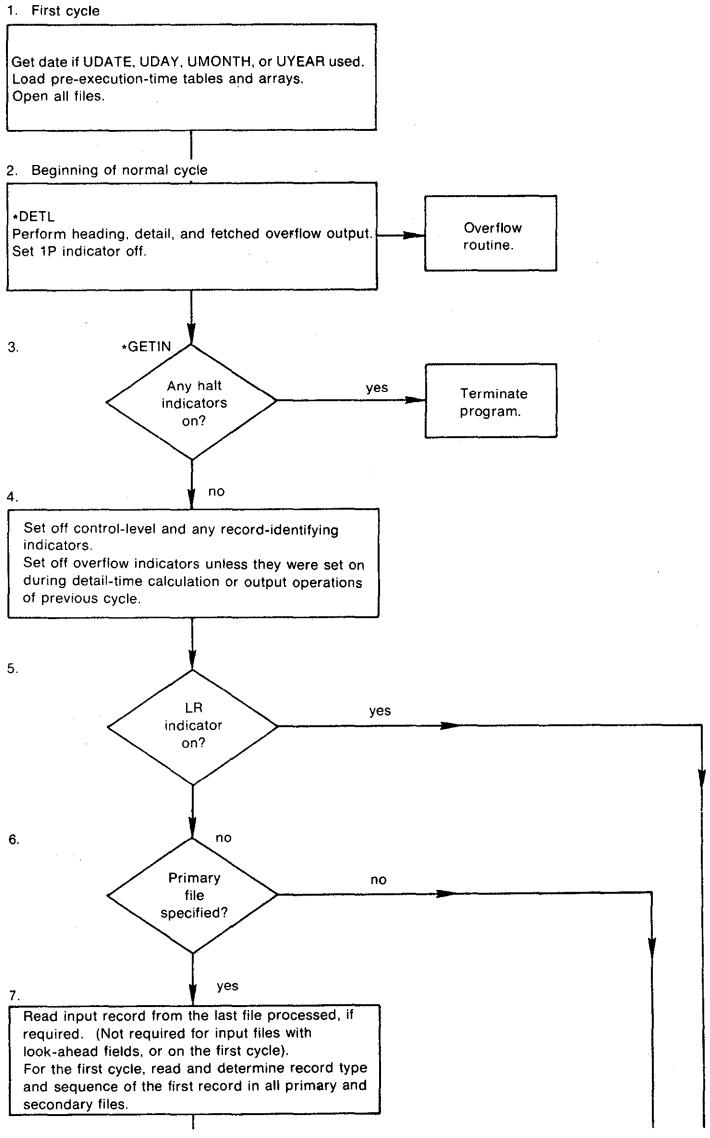
1.4.2 Detail Time

During detail time, your program performs operations specific to each individual record. In the example described in Section 1.4.1, each time a record is read, the detail-time operations might consist of the following steps:

1. Printing an output line on your report. For example, each record in your file might contain a weekly sales figure for a particular salesperson. The report would list the week's beginning and ending dates and the sales figure.
2. Adding the sales figure to all active accumulators. Then, when the next control break occurs, each accumulator will contain the correct amount.
3. Performing any other operations you defined in your specifications. These might include moving data fields and handling errors.

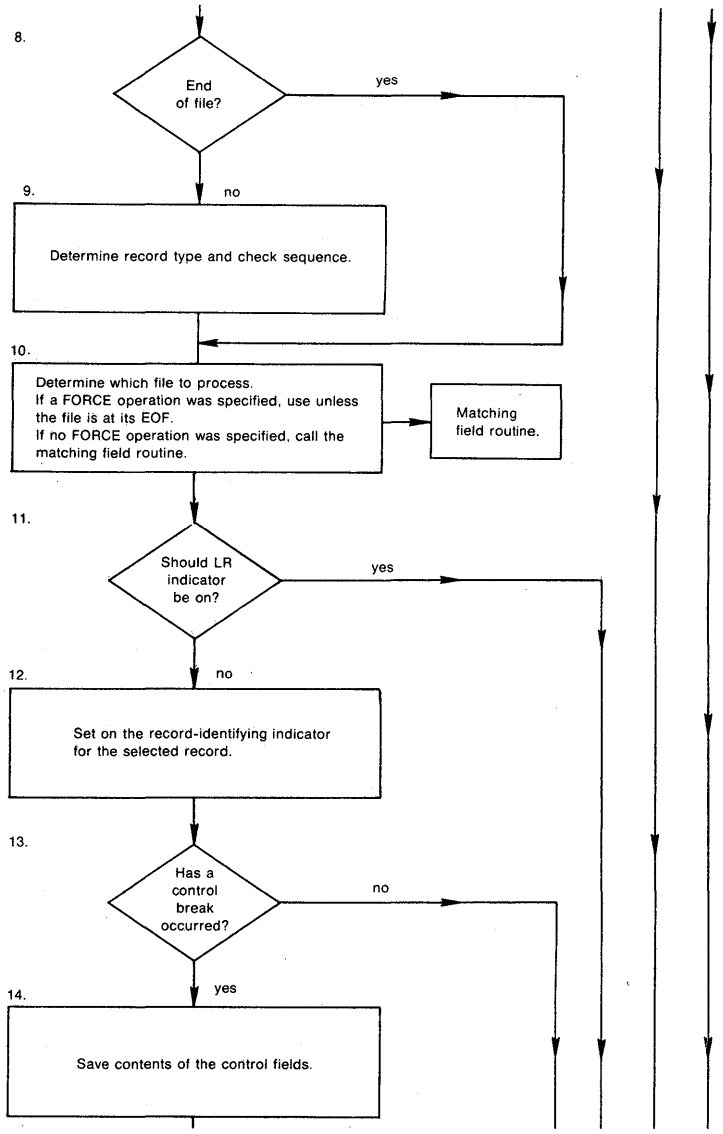
Figure 1–2 is a detailed flowchart of a complete, normal VAX RPG II program cycle. Each processing and decision box is numbered; the numbers are keyed to the annotations that immediately follow the figure.

Figure 1–2: Detailed VAX RPG II Logic Cycle Flowchart



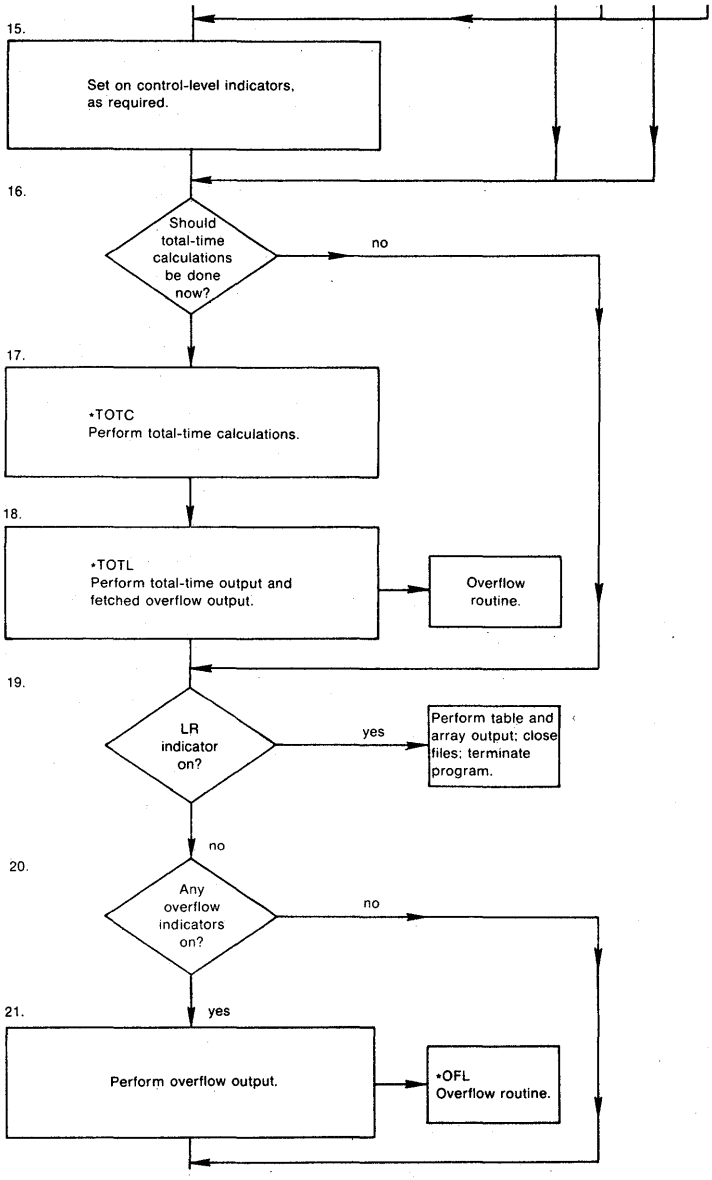
(Continued on next page)

Figure 1–2 (Cont.): Detailed VAX RPG II Logic Cycle Flowchart



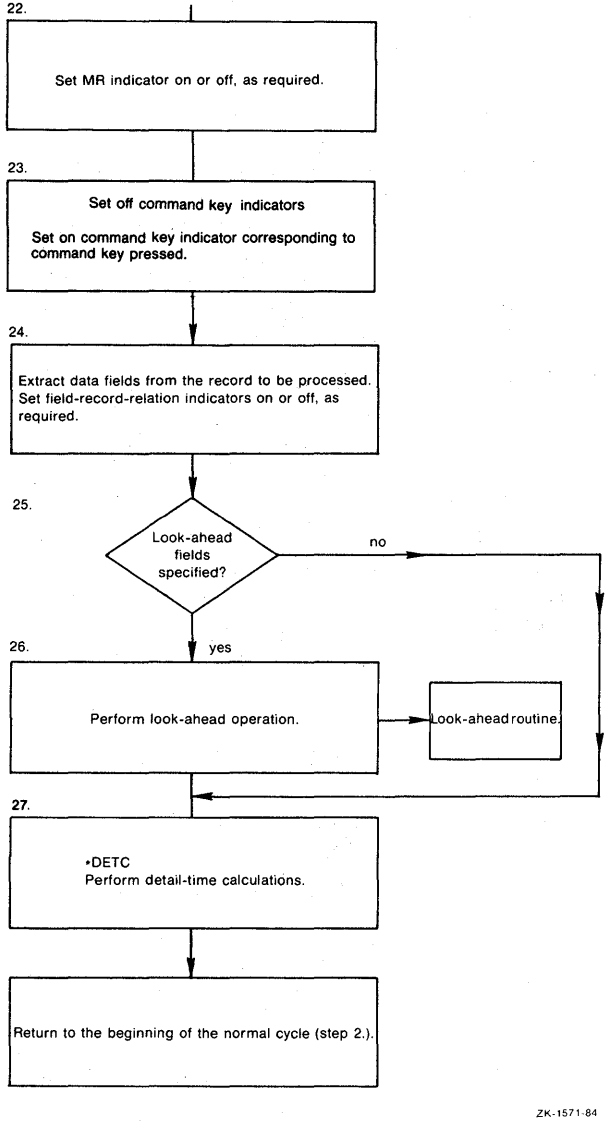
(Continued on next page)

Figure 1–2 (Cont.): Detailed VAX RPG II Logic Cycle Flowchart



(Continued on next page)

Figure 1–2 (Cont.): Detailed VAX RPG II Logic Cycle Flowchart



Key to Figure 1–2:

❶ This step is executed only during the first cycle. It initializes your program for execution. Initialization consists of retrieving the date (if you specified UDATE, UDAY, UMONTH, or UYEAR), opening all files, and loading all preexecution-time tables and arrays.

❷ VAX RPG II writes heading and detail lines (identified by H or D in column 15 (Type) of the Output specification). Heading and detail lines are always executed at the same time. If conditioning indicators are specified, the conditions for the indicator must be satisfied. If the fetch overflow logic is specified and the overflow indicator is on, RPG II writes the appropriate overflow lines. If the 1P indicator is on (during the first cycle only), VAX RPG II prints all lines conditioned by it, then sets the 1P indicator off. VAX RPG II executes this step at the beginning of the program so that heading lines can be printed before processing begins.

❸ VAX RPG II checks whether any halt indicators (H1 through H9) are on; if any are, the program terminates. If you do not want your program to terminate here, you must set all halt indicators off previous to this step. You can set halt indicators on, however, at any time during the program.

❹ VAX RPG II sets control-level indicators (L1 through L9) and all indicators used as record-identifying indicators off. VAX RPG II also sets overflow indicators (OA through OG, and OV) off, unless they were set on during detail time (detail-time calculations or output operations) in the preceding cycle. All other types of indicators that are on remain on.

❺ Here, VAX RPG II determines whether the LR indicator is on. If it is, RPG II branches to step 15 and sets on control-level indicators L1 through L9, if used.

❻ VAX RPG II determines whether a primary file was specified by the program. If not, VAX RPG II proceeds directly to step 16.

❼ If required, VAX RPG II reads an input record from the last primary or secondary file processed. If this was an input file with look-ahead fields, the record is already available; therefore, no read operation may be necessary at this time. On the first cycle, a record is read from each primary and secondary file.

❽ VAX RPG II tests the file just read for end-of-file. If end-of-file has been encountered, the program bypasses step 9.

❾ If VAX RPG II reads a record from a file, the record type is determined and the record sequence is checked. If the record type cannot be determined, or the record is out of sequence, the program terminates.

⑩ In this step, VAX RPG II determines which file to process. If a FORCE operation was executed during the previous cycle, the forced file is selected for processing. (All records processed with a FORCE operation are processed with the matching-records (MR) indicator set off.) However, if the forced file is at end-of-file (EOF), the normal multifile logic selects the next record for processing. If no forced file was specified, VAX RPG II determines whether matching fields were specified. If so, the matching-fields routine is given control (see Figure 1–3). Otherwise, all records in a primary file are processed first, then the records from each secondary file in order of their specification.

⑪ Here, VAX RPG II determines whether the LR indicator should be set on. The LR indicator is set on when the program has reached the end of all the files that you have specified for processing until the end-of-file, and when all the records from secondary files that match the last primary record have been processed. If the LR indicator should be set on, VAX RPG II branches to step 15 and sets on indicators L1 through L9.

⑫ VAX RPG II sets on the record-identifying indicator for the record selected for processing.

⑬ VAX RPG II determines whether the record selected for processing has caused a control break to occur. A control break occurs when the value in the control field of the record being processed differs from the previous value of the control field. See Section 1.4.1 for more information.

⑭ If a control break has occurred, VAX RPG II saves the contents of all appropriate control fields.

⑮ If a control break has occurred, VAX RPG II sets the appropriate control-level indicator (L1 through L9) on; at the same time, VAX RPG II sets all lower-level control-level indicators on. The L1 through L9 indicators can be used for conditioning only if they have been defined as conditioning indicators.

⑯ VAX RPG II determines whether total-time calculation and output operations should be performed. If control-level indicators are not specified in columns 59 and 60 (control-level) of the Input specification, VAX RPG II bypasses total-time calculation and output operations during the first cycle only; after the first cycle, VAX RPG II performs total-time calculation and output operations for every cycle.

If control-level indicators are specified, VAX RPG II bypasses total-time calculation and output operations until after the first record with control fields is processed. When the LR indicator is on, VAX RPG II always performs total-time calculation and output operations.

⓱ In this step, VAX RPG II performs all total-time calculations conditioned by a control-level indicator or containing L0 in columns 7 and 8 of the Calculation specification. Total-time calculations can include CHAIN operations, in which a record is immediately retrieved from an input file (see Figure 1–4), or READ operations, in which the next record is retrieved from a demand file.

⓲ Here, VAX RPG II writes all total-time output lines that satisfy the conditions specified by the indicators. If an overflow indicator (OA through OG, or OV) is on, and fetch overflow is specified, VAX RPG II writes the overflow lines as well.

⓳ VAX RPG II determines whether the LR indicator is on. If it is on, VAX RPG II performs table and array output, closes all files, and terminates the program.

⓴ VAX RPG II checks to determine whether any overflow indicators (OA through OG, or OV) are on.

㉑ If any overflow indicators are on, the overflow routine is given control (see Figure 1–5). VAX RPG II outputs all lines conditioned by those overflow indicators that are on. However, VAX RPG II outputs these lines only if they were not output by fetch overflow logic (step 2 or step 18).

㉒ VAX RPG II determines whether the MR indicator should be set on. If this is a multifile program and the record being processed is a matching record, VAX RPG II sets the MR indicator on; it remains on for the duration of the cycle during which the matching record is processed. If these conditions are not present, VAX RPG II sets the MR indicator off.

㉓ If the program contains a WORKSTN file, VAX RPG II sets off the KA through KZ and K0 through K9 indicators. If form input was terminated by a command key, VAX RPG II sets on the corresponding command key indicator. Note that if an error occurred on the read, the command key indicators are not changed.

㉔ VAX RPG II extracts data fields from the record to be processed and sets the field indicators on or off, as appropriate, for those fields.

㉕ VAX RPG II then determines whether look-ahead fields are specified in the last file processed and whether it is an input file.

㉖ If the last file processed was an input file with look-ahead fields, VAX RPG II passes control to the VAX RPG II look-ahead routine (see Figure 1–6). In this routine, VAX RPG II retrieves the look-ahead record and extracts the look-ahead fields. If look-ahead fields are not specified, VAX RPG II continues with detail-time calculations (step 27).

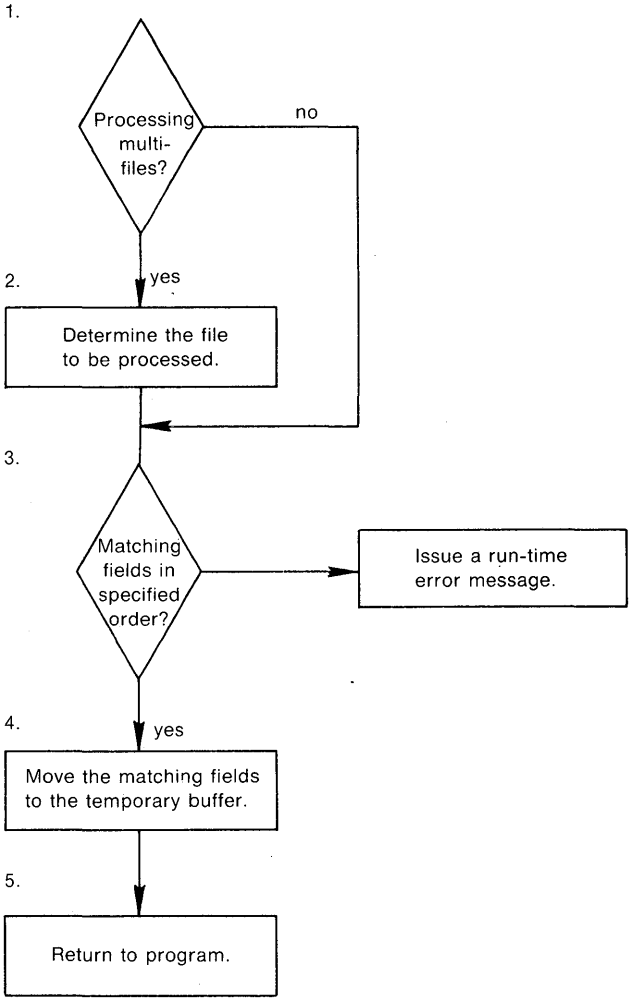
㉗ This is the detail-time calculations step. Here, VAX RPG II performs all conditioned detail-time calculations and subroutines. The calculations may include CHAIN and READ operations (see Figure 1–4). Detail-time calculations complete the VAX RPG II logic cycle. Then, the cycle branches to step 2 to begin again.

1.5 VAX RPG II Detail Program Logic Cycle

This section consists of annotated flowchart diagrams that show, in detail, various routines within the VAX RPG II logic cycle. The following figures are provided:

- Figure 1–3 shows the VAX RPG II matching-fields routine.
- Figure 1–4 shows VAX RPG II file processing for chained and demand files.
- Figure 1–5 shows VAX RPG II overflow processing.
- Figure 1–6 shows VAX RPG II look-ahead processing.

Figure 1–3: Logic Cycle for the Matching-Fields Routine



Key to Figure 1–3:

❶ VAX RPG II determines whether the program uses more than one primary and secondary file. If multifile processing is in effect, processing continues with step 2. Otherwise, the program branches to step 3.

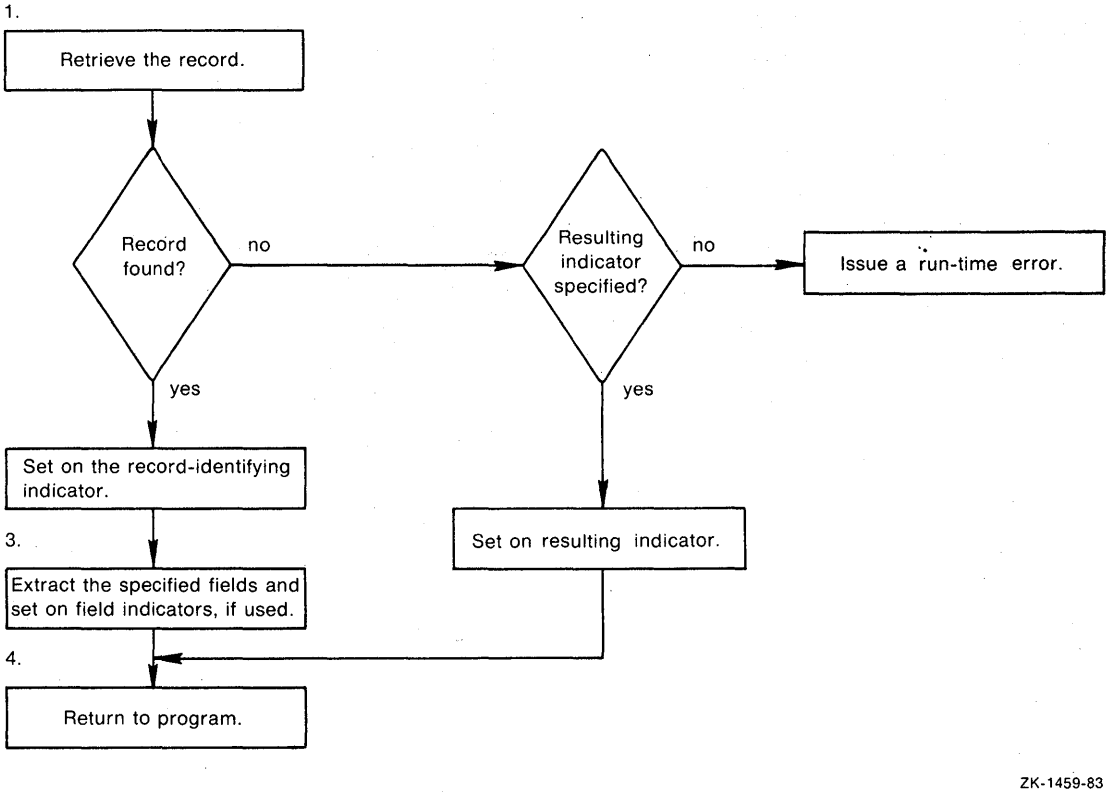
❷ VAX RPG II compares the matching fields to determine which file is to be processed. VAX RPG II extracts the matching fields and checks their sequence.

❸ If the matching fields are not in sequence, a run-time error occurs and the program terminates.

❹ VAX RPG II moves the matching fields into a temporary buffer. The next record is selected, based on the value of the matching fields.

❺ VAX RPG II returns to the program.

Figure 1–4: Logic Cycle for Chained and Demand Files



Key to Figure 1–4:

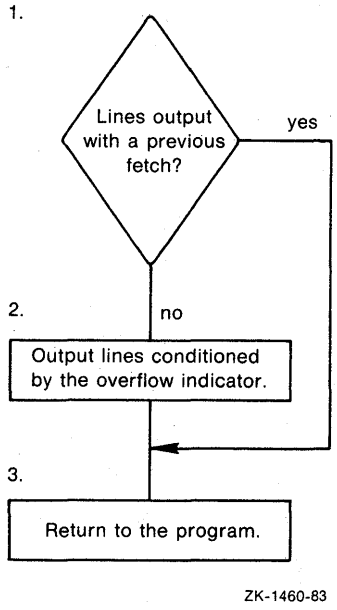
❶ VAX RPG II retrieves the next record in the file specified by the CHAIN or READ operation code. If the record is not found on a CHAIN operation or an end-of-file occurs during a READ operation and a resulting indicator is not specified, a run-time error occurs. If the record is not found on a CHAIN operation or an end-of-file occurs during a READ operation and a resulting indicator has been specified, the indicator is set on and control returns to the program.

❷ VAX RPG II sets on the record-identifying indicator associated with the chained or demand file for the record type read.

❸ VAX RPG II extracts the fields from the record just retrieved. Also, VAX RPG II sets on any field indicators associated with the record.

❹ VAX RPG II returns to the program.

Figure 1–5: Logic Cycle for Overflow Processing



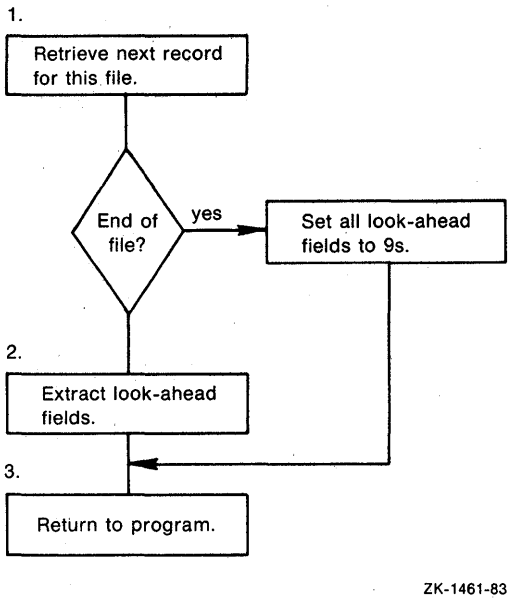
Key to Figure 1–5:

❶ VAX RPG II uses the fetch overflow routine to determine whether the overflow lines were written previously. If the overflow lines were written previously, the program branches to the specified return point; otherwise, it continues with step 2.

❷ VAX RPG II evaluates all overflow lines and writes those lines that satisfy the conditions of the indicators.

❸ VAX RPG II returns to the program.

Figure 1–6: Logic Cycle for Look-Ahead Processing



Key to Figure 1–6:

❶ VAX RPG II reads the next record for the file being processed. If the end-of-file has been reached, all look-ahead fields are filled with 9s and control is returned to the program.

❷ VAX RPG II extracts the look-ahead fields from the record.

❸ VAX RPG II returns to the program.

Chapter 2

Using the VAX RPG II Editor

This chapter explains how to use the VAX RPG II editor to create, edit, and read (or view) VAX RPG II programs.

The VAX RPG II editor is available on the VT100, VT200, and VK100 (GIGI) terminals.

The VAX RPG II editor allows overstriking; that is, you can change a program line by placing the cursor in the column where you want to make a change and typing a new character, without affecting any characters to the right of the cursor.

The cursor is represented as a box (▮) in the examples throughout this chapter.

All examples in this chapter assume a terminal page size of 24 lines, unless otherwise noted.

2.1 Qualifiers

Invoke the VAX RPG II editor by typing the RPG/EDIT command. To create a file, provide a file specification, as shown in the following example:

```
$ RPG/EDIT FIRSTTRY
```

You do not have to supply the RPG file type, because it is the default.

To edit or read a file, include the name of the file you want to edit or read when you invoke the VAX RPG II editor. See Section 2.8.2 for an example.

The following error message may be displayed when you invoke the VAX RPG II editor:

```
%TPU-E-NONANSICRT, SYS$INPUT must be an ANSI CRT
```

If this error occurs, leave the VAX RPG II editor and make sure that the VAX/VMS terminal characteristics are set properly for your terminal by typing the SET TERMINAL/INQUIRE command.

When you invoke the VAX RPG II editor, if the number of columns (SET TERMINAL/WIDTH=m) is less than 80 or the number of lines (SET TERMINAL/PAGE=n) is less than 6, the VAX RPG II editor will display the following message, then will exit:

```
At least 6 lines and 80 columns on the screen are required
```

See the *VAX/VMS DCL Dictionary* for information on the SET TERMINAL command.

Note that the SET TERMINAL command must be entered before invoking the VAX RPG II editor.

If you are using a VK100 (GIGI) terminal and the terminal screen does not appear to update correctly, leave the VAX RPG II editor and type the SHOW TERMINAL command to make sure that the device is a VK100. If it is not, type the SET TERMINAL/INQUIRE command to make sure that the VAX/VMS terminal characteristics are set properly for your terminal.

If the file you specify when invoking the VAX RPG II editor is a new file, the VAX RPG II editor displays the following message:

```
File not found
```

If the file you specify when invoking the VAX RPG II editor is an existing file, the VAX RPG II editor displays the following message:

```
n lines read from file device:[directory]filename.type;version
```

Finally, the VAX RPG II editor displays the following message:

```
Press the PF2 key to get help information
```

If the terminal page size is less than 17 lines, the initial HELP message is not displayed. If HELP is requested using the HELP key or a SET HELP command in a start-up command file, and the terminal page size is less than 17 lines, the following message is displayed and the usual HELP action will not be performed:

```
At least 17 lines on the screen are required by the editor to provide HELP
```

Table 2–1 lists the qualifiers that you can use with the RPG/EDIT command. If you precede a qualifier with /NO, that qualifier is not in effect.

Table 2–1: RPG/EDIT Command Qualifiers

Qualifier	Negative Form	Default
/COMMAND	/NOCOMMAND	/COMMAND=RPGINI
/CREATE	/NOCREATE	/CREATE
/JOURNAL	/NOJOURNAL	/JOURNAL
/OUTPUT	/NOOUTPUT	/OUTPUT
/READ_ONLY	/NOREAD_ONLY	/NOREAD_ONLY
/RECOVER	/NORECOVER	/NORECOVER
/START_POSITION	/NOSTART_POSITION	/START_POSITION=(1,1)

Sections 2.1.1 through 2.1.7 describe the qualifiers and their use.

2.1.1 /COMMAND Qualifier

The /COMMAND qualifier causes the VAX RPG II editor to execute a specified file in the start-up command file. Its format is as follows:

```
/COMMAND[=file-spec]
```

The VAX RPG II editor reads commands from any file specified by the qualifier. Each command in the specified file is treated as if the /COMMAND qualifier were used.

The /COMMAND qualifier is present by default, with a default value of RPGINI. If the /NOCOMMAND qualifier is used, then no command file is executed. See Section 2.7.2 for information on start-up command files.

2.1.2 /CREATE Qualifier

The /CREATE qualifier creates a file for the editing session. If the specified file already exists, that file is opened. Its format is as follows:

```
/CREATE
```

The /CREATE qualifier is present by default. If the /NOCREATE qualifier is used, the file is not created. However, if the file already exists, it is opened.

2.1.3 /JOURNAL Qualifier

The /JOURNAL qualifier creates a journal file for the current editing session. Its format is as follows:

```
/JOURNAL[=file-spec]
```

If you should leave an editing session abnormally, you can use the journal file to reexecute all the commands you issued during the session. To do this, type the RPG/EDIT/RECOVER command.

The /JOURNAL qualifier is present by default. If you do not provide a file specification with /JOURNAL, the VAX RPG II editor creates a journal file with the same name as your input file and the default file type JOU.

2.1.4 /OUTPUT Qualifier

The /OUTPUT qualifier defines the name of the output file. Its format is as follows:

```
/OUTPUT[=file-spec]
```

The /OUTPUT qualifier is the default. If you do not provide a file specification with /OUTPUT, the VAX RPG II editor creates an output file with the same node, device, directory, name, and type as the input file, whose version number is one higher than the highest existing version of the input file.

If you use the /NOOUTPUT qualifier, the VAX RPG II editor does not create an output file. In this case, you must either use the QUIT command or specify a file specification with the EXIT command to leave the VAX RPG II editor.

2.1.5 /READ_ONLY Qualifier

The /READ_ONLY qualifier tells the VAX RPG II editor not to create a journal file or an output file for the file you are currently editing. Its format is as follows:

```
/READ_ONLY
```

You can use the /READ_ONLY qualifier when you want to view a file without changing its contents. In this case, you must either use the QUIT command or specify a file specification with the EXIT command to leave the editor.

The /NOREAD_ONLY qualifier is the default and automatically creates a journal file and output file for the file you are currently editing (unless you leave the VAX RPG II editor using the QUIT command).

Using the /READ_ONLY qualifier has the same effect as using both the /NOOUTPUT and the /NOJOURNAL qualifiers with the RPG/EDIT command.

2.1.6 /RECOVER Qualifier

The /RECOVER qualifier reads the commands from a journal file and reexecutes all the edits you made during an editing session. Its format is as follows:

```
/RECOVER
```

When the recovery is done, the VAX RPG II editor responds with:

```
Recovery complete
```

At this time, you can continue editing your file.

If the name of the recovery journal file is different from the default (the same file name as the input file with the JOU file type), use /JOURNAL=file-spec and /RECOVER to specify another name, as shown in the following example:

```
$ RPG/EDIT/JOURNAL=FILE1.JOU/RECOVER FILE2.RPG
```

In this example, the journal file name is FILE1.JOU, and the name of both the input and output files is FILE2.RPG. If you do not use /JOURNAL, the journal file name is FILE2.JOU.

The /NORECOVER qualifier is the default.

2.1.7 /START_POSITION Qualifier

The /START_POSITION qualifier determines where the VAX RPG II editor starts in the editing buffer. Its format is as follows:

```
/START_POSITION[=(line,column)]
```

The /START_POSITION qualifier is the default. The setting is line 1, column 1.

The /NOSTART_POSITION qualifier is equivalent to START_POSITION=(1,1).

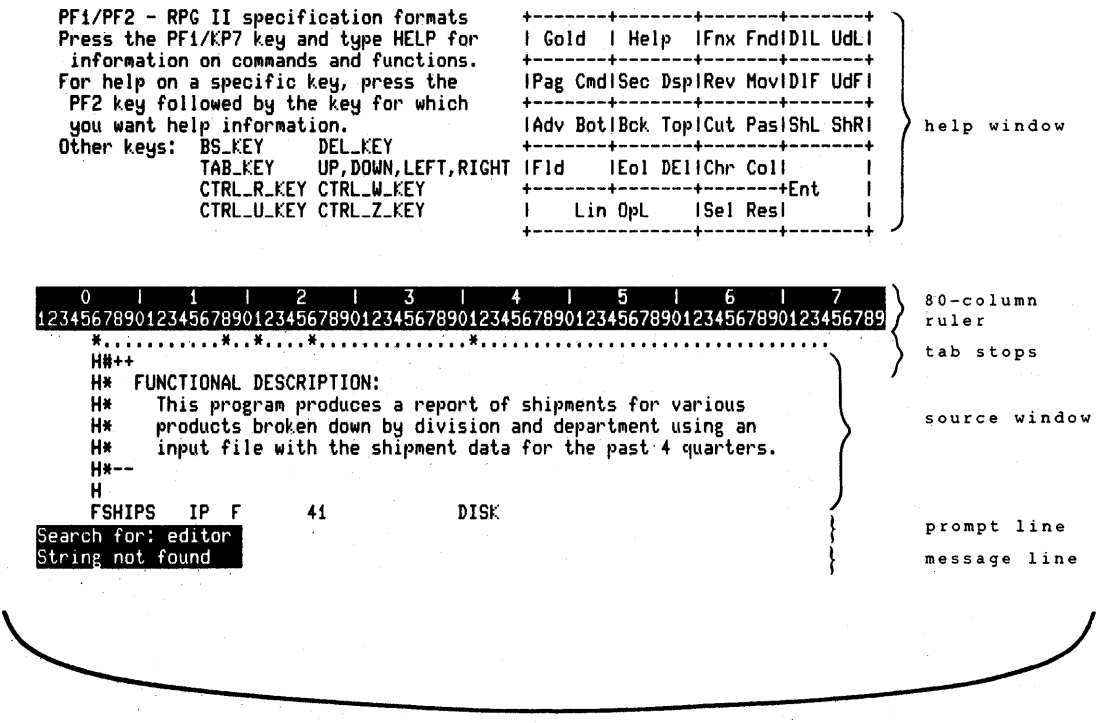
2.2 The Editor Screen

The VAX RPG II editor screen consists of the following components:

- The HELP window
- An 80-column ruler
- Tab stops
- The editing window
- The prompt line
- The message line

Note that when you use a terminal without scrolling regions (for example, VK100 (GIGI)), the VAX RPG II editor must redisplay the information on the screen rather than scrolling new information onto the screen.

The following screen shows an example of each of the VAX RPG II editor screen components. Note that all screens shown are based on a default setting of 24 lines for the page size with a top ruler. If you want to change the page size, see the SET TERM/PAGE commands in the *VAX/VMS DCL Dictionary*.



ZK-4666-85

When you use the HELP function (default = PF2), the HELP window appears on the screen. It includes a diagram of the keypad and other key functions.

When the keypad diagram is displayed and you enter the HELP function again, the following message appears:

Press the key for which you want further HELP information

You can press the HELP key and any other key listed in the keypad diagram to display HELP information on that key in the HELP window. See Section 2.5.2 for an example.

The HELP_SPECS function (default = PF1/PF2[1]) causes the HELP window to display the specification format for the current line. See Section 2.5.3 for an example. See the DISPLAY function, Section 2.5.11, for information on how to remove HELP from the screen.

If you press CTRL/C while HELP information from the VAX RPG II editor HELP command is being written to the terminal, you must type a terminator to resume editing.

If you do not request HELP information, the VAX RPG II editor displays the program in the entire screen except for the ruler and tab stops and the prompt and message lines, as shown in the following example:

```
    0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
    *.....*..*....*.............*.................................
    H*++
    H*  FUNCTIONAL DESCRIPTION:
    H*    This program produces a report of shipments for various
    H*    products broken down by division and department using an
    H*    input file with the shipment data for the past 4 quarters.
    H*--
    H
    FSHIPS   IP  F      41           DISK
    FSUMREP  O   F      98          LPRINTER
    E                    QTY        4  2 0
    LSUMREP   55FL 500L
    ISHIPS   AA  01
    I                                        1   5 DIV   L2
    I                                        6   7 DEPT  L1
    I                                        8  16 PROD
    I                                       17  24 QTY
    C*
    C   01               XFOOTQTY       PROQTY  30

Press the PF2 key to get help information
```

ZK-4330-85

[1] PF1/PF2 indicates that these two keys should be pressed in succession.

An 80-column ruler in reverse video is displayed above or below the source window. See Section 2.6.8.4 for information on setting the ruler. Below the ruler, a tab stop marks the beginning of each field for the specification of the current line.

An asterisk (*) indicates a tab stop where you can enter a field value. A dot (.) indicates that the column must be left blank. A dash (-) after an asterisk indicates that the field must contain numeric data. Numeric data must be right-justified. Blanks after an asterisk indicate that the field must contain alphanumeric data. Alphanumeric data must be left-justified.

The VAX RPG II editor marks the line after the last line in the editing buffer with the end-of-buffer [EOB] symbol. The [EOB] symbol will not appear in the output file.

The last two lines of the screen consist of the prompt line and the message line. The prompt line displays prompts in reverse video for input when you use functions such as FIND and COMMAND. The message line displays informational and error messages. The following example shows what the VAX RPG II editor screen looks like with the specification format for the current line, the prompt for the FIND function, and an informational message.

```
                    Currency symbol
                    |  Inverted print (DIJ)
                    |  |    Alternate collating sequence (SE)
                    |  |    |              1P forms position (1)
                    |  |    |              |
          H         |  |    |              |
    0   |   1   |   2   |   3   |   4   |   5   |   6   |   7
1234567890123456789012345678901234567890123456789012345678901234567890123456
          *...........*..*....*.............*.................................
          H*++
          H*  FUNCTIONAL DESCRIPTION:
          H*    This program produces a report of shipments for various
          H*    products broken down by division and department using an
          H*    input file with the shipment data for the past 4 quarters.
          H*--
          H
          FSHIPS   IP  F      41           DISK
Search for:
New mail on node MYNODE from SYSTEM
```

ZK-4331-85

2.3 The Cursor

The VAX RPG II editor cursor is represented as a box (▮) or an underscore (_), depending on the cursor setting for your terminal. The cursor is displayed in the current column on the current line. Note, however, that if the current column is column 81, the cursor is displayed in column 80 on the current line. If you try to move the cursor to the right of column

81 or to the left of column 1, the current column remains unchanged and one of the following messages is displayed on the message line:

```
Attempt to move past column 81

Attempt to move before column 1
```

2.4 The Buffers

The VAX RPG II editor uses the following four buffers:

- Editing

  The editing buffer contains the file of source code that is displayed on the VAX RPG II editor screen.

- Deleted-field

  The deleted-field buffer contains the field deleted when you use the DELETE_FIELD function (default = MINUS). See Section 2.5.14 for information on the DELETE_FIELD function. You can access the contents of the deleted-field buffer by using the UNDELETE_FIELD function. See Section 2.5.15 for information on the UNDELETE_FIELD function.

- Deleted-line

  The deleted-line buffer contains the line deleted by the DELETE_LINE function (default = PF4). See Section 2.5.6 for more information on the DELETE_LINE function. You can access the contents of the deleted-line buffer by pressing the UNDELETE_LINE function (default = PF1/PF4). See Section 2.5.7 for more information on the UNDELETE_LINE function.

- Paste

  The paste buffer contains the range of lines delimited by the SELECT (default = PERIOD) and CUT (default = KP6) functions (see Section 2.5.32). You can access the contents of the paste buffer by using the PASTE function (default = PF1/KP6). See Section 2.5.21 for more information on the PASTE function.

2.5 Keys and Functions

To make sure the VAX RPG II editor is using the correct VAX/VMS terminal characteristics for your terminal, type the DCL command SET TERM/INQUIRE before invoking the VAX RPG II editor. The following diagram represents the VAX RPG II editor keypad:

VT100/VT200/VK100(GIGI) Keypad

PF1	PF2	PF3	PF4
7	8	9	—
4	5	6	,
1	2	3	Enter
0		.	

ZK-1605-84

VAX RPG II Editor Keypad

Gold	HELP	Fnx Fnd	DIL UdL
Pag Cmd	Sec Dsp	Rev Mov	DIF UdF
Adv Bot	Bck Top	Cut Pas	ShL ShR
Fld	Eol Del	Chr Col	Ent
Lin Opl		Sel Res	

ZK-5550-86

This chapter refers to those keys with numbers and symbols as KPn, where KP means keypad and n is the number of the key shown on the VT100, VT200, and VK100 (GIGI) keypad. For example, KP6 refers to the keypad key numbered 6. Table 2–2 lists the name and default function of each key.

Note that many keys have alternate functions. An alternate function is enabled when you press the GOLD key (default = PF1) followed by the key you want to use. This sequence is referred to in this chapter as PF1/[key_name].

Table 2–2: VAX RPG II Editor Define Key Defaults

COMMAND	KEY	DEFAULT
DEFINE KEY	PF1	GOLD
DEFINE KEY	↑	UP
DEFINE KEY	↓	DOWN
DEFINE KEY	←	LEFT
DEFINE KEY	→	RIGHT
DEFINE KEY	PF2	HELP_KEYPAD
DEFINE KEY/GOLD	PF2	HELP_SPECIFICATIONS
DEFINE KEY	PF3	FIND_NEXT
DEFINE KEY/GOLD	PF3	FIND
DEFINE KEY	PF4	DELETE_LINE
DEFINE KEY/GOLD	PF4	UNDELETE_LINE
DEFINE KEY	KP7	PAGE
DEFINE KEY/GOLD	KP7	COMMAND
DEFINE KEY	KP8	SECTION
DEFINE KEY/GOLD	KP8	DISPLAY
DEFINE KEY	KP9	REVIEW_ERROR
DEFINE KEY/GOLD	KP9	MOVE_TO_RULER
DEFINE KEY	-	DELETE_FIELD
DEFINE KEY/GOLD	-	UNDELETE_FIELD
DEFINE KEY	KP4	ADVANCE
DEFINE KEY/GOLD	KP4	BOTTOM
DEFINE KEY	KP5	BACKUP
DEFINE KEY/GOLD	KP5	TOP
DEFINE KEY	KP6	CUT
DEFINE KEY/GOLD	KP6	PASTE
DEFINE KEY	,	SHIFT_LEFT
DEFINE KEY/GOLD	,	SHIFT_RIGHT
DEFINE KEY	KP1	FIELD
DEFINE KEY	KP2	END_OF_LINE

Table 2–2 (Cont.): VAX RPG II Editor Define Key Defaults

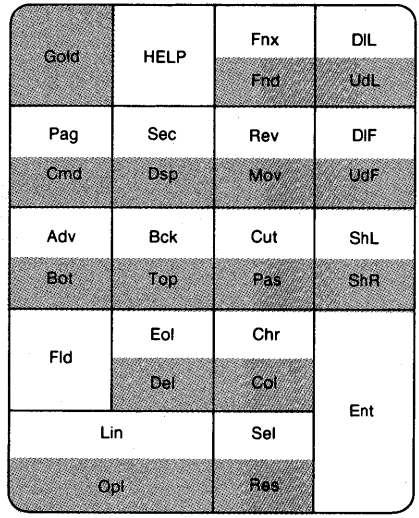
COMMAND	KEY	DEFAULT
DEFINE KEY/GOLD	KP2	DELETE_TO_END_OF_LINE
DEFINE KEY	KP3	CHARACTER
DEFINE KEY/GOLD	KP3	COLUMN
DEFINE KEY	ENTER	ENTER
DEFINE KEY	KP0	LINE
DEFINE KEY/GOLD	KP0	OPEN_LINE
DEFINE KEY	.	SELECT
DEFINE KEY/GOLD	.	RESET
DEFINE KEY	CTRL/H	FIELD_BACKWARD
DEFINE KEY	CTRL/I	FIELD_FORWARD
DEFINE KEY	RETURN	NEW_LINE
DEFINE KEY	CTRL/R	REFRESH_SCREEN
DEFINE KEY	CTRL/U	DELETE_TO_BEGINNING_OF_LINE
DEFINE KEY	CTRL/W	REFRESH_SCREEN
DEFINE KEY	CTRL/Z	EXIT
DEFINE KEY	<X	DELETE_CHARACTER
DEFINE KEY	F10	EXIT
DEFINE KEY	F12	FIELD_BACKWARD
DEFINE KEY	F15	HELP_KEYPAD
DEFINE KEY	F16	ENTER
DEFINE KEY	FIND	FIND
DEFINE KEY	INSERT HERE	PASTE
DEFINE KEY	REMOVE	CUT
DEFINE KEY	SELECT	SELECT
DEFINE KEY	PREV SCREEN	SECTION_BACKWARD
DEFINE KEY	NEXT SCREEN	SECTION_FORWARD

See DEFINE KEY (Section 2.6.2) for a complete list of definable keys. Sections 2.5.10 through 2.5.44 describe these functions and explain how to use them.

2.5.1 GOLD Function

The GOLD function (default = PF1) enables you to select the alternate function of a key. In the following diagram of the keypad, the alternate key names appear in the shaded areas:

VAX RPG II Editor Keypad



ZK-5550-86

2.5.2 HELP_KEYPAD Function

The HELP_KEYPAD function (default = PF2) displays the keypad diagram in the HELP window, as shown in the following example.

```
PF1/PF2 - RPG II specification formats      +-------+-------+-------+-------+
Press the PF1/KP7 key and type HELP for    | Gold  | Help  |Fnx Fnd|DIL UdL|
 information on commands and functions.    +-------+-------+-------+-------+
For help on a specific key, press the      |Pag Cmd|Sec Dsp|Rev Mov|DIF UdF|
 PF2 key followed by the key for which     +-------+-------+-------+-------+
 you want help information.                |Adv Bot|Bck Top|Cut Pas|ShL ShR|
Other keys:  BS_KEY      DEL_KEY           +-------+-------+-------+-------+
             TAB_KEY     UP,DOWN,LEFT,RIGHT|Fld    |Eol DEl|Chr Col|       |
             CTRL_R_KEY  CTRL_W_KEY        +-------+-------+-------+Ent    |
             CTRL_U_KEY  CTRL_Z_KEY        |    Lin OpL    |Sel Res|       |
                                           +---------------+-------+-------+

   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *.....*..*....*.............*.............................
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41            DISK
```

ZK-4333-85

If HELP is requested while the terminal page size is less than 17 lines, the following message will be displayed and the usual HELP action will not be performed:

```
At least 17 lines on the screen are required by the editor to provide HELP
```

HELP cannot be displayed unless there are enough lines on the screen to position the HELP window and still keep the ruler, prompt line, message line, and one line of the editing window visible.

If the keypad diagram is already displayed, you can get HELP information on any function (except GOLD) by using HELP_KEYPAD (default = PF2) and the key for which you want HELP information. HELP information will appear in the HELP window. The following example displays HELP for the CUT (default = KP6) and PASTE (default = PF1/KP6) functions:

```
KP6

The CUT function moves the selected range of lines to the paste buffer.  The
selected range of lines consists of the line identified by the SELECT
function up to the current line.  The line following the selected range of
lines becomes the current line.  The current column remains unchanged.

The PASTE function inserts the contents of the paste buffer directly in
front of the current line.  The current line is moved down to accommodate
the lines from the paste buffer.  The current column and line remain
unchanged.
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     *...........*..*....*.............*.............................
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41           DISK
```

ZK-4332-85

2.5.3 HELP_SPECIFICATIONS Function

The HELP_SPECIFICATIONS function (default = PF1/PF2) displays the specification format for the current line. In the following example, if the current line is line 100, the VAX RPG II editor displays the Control specification format when you use HELP_SPECIFICATIONS.

```
                  Currency symbol
                  |  Inverted print (DIJ)
                  |  |    Alternate collating sequence (SE)
                  |  |    |              1P forms position (1)
                  |  |    |              |
     H            |  |    |              |
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     *...........*..*....*............*.................................
100H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41           DISK
```

ZK-4334-85

To restore the editing buffer to the entire screen, as shown in the following example, use the DISPLAY function (default = PF1/KP8).

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     *.....*..*....*.............*.............................
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41            DISK
     FSUMREP  O   F      98           LPRINTER
     E                    QTY         4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                        1   5 DIV   L2
     I                                        6   7 DEPT  L1
     I                                        8  16 PROD
     I                                       17  24 QTY
     C*
     C   01               XFOOTQTY       PROQTY  30
     C   01      PROQTY   ADD  DEPQTY    DEPQTY  30
```

ZK-4335-85

The VAX RPG II editor automatically updates the tab stops and the specification format, if displayed, for the specification type of the current line after a terminator (such as TAB) is typed.

2.5.4 FIND_NEXT Function

The FIND_NEXT function (default = PF3) moves the cursor to the first character position of the next or last occurrence of the search string, depending on the current direction (ADVANCE or BACKUP). Use the FIND function to enter the search string. If the current direction is ADVANCE, the VAX RPG II editor will try to locate the next occurrence of the search string by searching forward from the current column and line to the end of the editing buffer. If the current direction is BACKUP, the VAX RPG II editor will try to locate the next occurrence of the search string by searching backward from the current column and line to the beginning of the editing buffer. If the VAX RPG II editor cannot locate the search string, the current column and line remain unchanged and an error message is displayed on the message line. See Section 2.8.2 for an example of the FIND_NEXT function.

2.5.5 FIND Function

The FIND function (default = PF1/PF3) locates the search string you specify. The VAX RPG II editor moves the cursor forward or backward to the beginning of the nearest occurrence of the search string, depending on the current direction (ADVANCE or BACKUP). If the current direction is ADVANCE, the VAX RPG II editor will try to locate the search string by searching forward from the current column and line toward the end of the editing buffer. If the current direction is BACKUP, the VAX RPG II editor will try to locate the search string by searching backward from the current column and line toward the beginning of the editing buffer.

When you use the FIND function, the VAX RPG II editor displays the following prompt on the prompt line:

```
Search for:
```

You can enter up to 63 characters for the search string. If no search string is entered, the VAX RPG II editor will search for the last search string specified. Note that you cannot use control characters (RETURN, FORM FEED, TAB, and so on) in the search string.

If the VAX RPG II editor cannot locate the search string, the current column and line remain unchanged and the following error message is displayed on the message line:

```
String not found
```

Terminate the search string by pressing either the RETURN key or the ENTER key.

See Section 2.8.2 for an example of the FIND function.

2.5.6 DELETE_LINE Function

The DELETE_LINE function places the current line in the deleted-line buffer, at the same time removing it from the screen. The line following the deleted line becomes the current line. The current column remains unchanged. If there is no line following the deleted line, the cursor is left in column 1 at the [EOB] mark.

2.5.7 UNDELETE_LINE Function

The UNDELETE_LINE function (default = PF1/PF4) inserts the contents of the deleted-line buffer before the current line. The new line becomes the current line, and the current column remains unchanged.

If the deleted-line buffer is empty, no action is taken but an error message is displayed on the message line.

2.5.8 PAGE Function

The PAGE function (default = KP7) causes the editing buffer to move forward or backward to the next or preceding page, depending on the current direction (ADVANCE or BACKUP). A page is the start or finish of a section with the same specification type (column 6).

In the following example, (1) the current cursor position is in column 34 on line 120, (2) the current direction is ADVANCE, (3) the current setting for the SET STARTCOLUMN command is 7, and (4) when you use the PAGE function, the VAX RPG II editor will move the cursor to column 7 on line 170.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *          *     *--*** * * *
  10H*++
  20H*  FUNCTIONAL DESCRIPTION:
  30H*     This program produces a report of shipments for various
  40H*     products broken down by division and department using an
  50H*     input file with the shipment data for the past 4 quarters.
  60H*--
  70H
  80FSHIPS   IP  F      41            DISK
  90FSUMREP  O   F      98           LPRINTER
 100E                    QTY         4  2 0
 110LSUMREP   55FL 500L
 120ISHIPS   AA  01
 130I                                          1   5 DIV   L2
 140I                                          6   7 DEPT  L1
 150I                                          8  16 PROD
 160I                                         17  24 QTY
 170C
 180C   01               XFOOTQTY       PROQTY  30
 190C   01      PROQTY   ADD  DEPQTY    DEPQTY  30
```

ZK-4336-85

2.5.9 COMMAND Function

The COMMAND function (default = PF1/KP7) allows you to execute a VAX RPG II editor command. The VAX RPG II editor displays the following prompt:

```
Command:
```

The following commands can be entered:

- COMPILE
- DEFINE KEY

- EXIT
- HELP
- INCLUDE
- QUIT
- RESEQUENCE
- SET
- SHOW
- SUBSTITUTE

Sections 2.6.1 through 2.6.10 describe these VAX RPG II editor commands and explain how to use them.

2.5.10 SECTION Function

The SECTION function (default = KP8) causes the editing buffer to move forward or backward the number of lines specified by the current setting of the SET SECTION command. The direction of the movement depends on the current direction (ADVANCE or BACKUP). The current column remains unchanged. See Section 2.6.8.6 for information on changing the SECTION value. See Sections 2.5.16 and 2.5.18 for information on setting the current direction.

2.5.11 DISPLAY Function

The DISPLAY function (default = PF1/KP8) removes any HELP information from the screen.

2.5.12 REVIEW_ERROR Function

If you use the VAX RPG II editor COMPILE command to compile your program, and your program contains errors, the VAX RPG II editor moves the cursor to the column and line where the first error occurs and displays the error text on the message line. The REVIEW_ERROR function (default = KP9) moves the cursor to the column and line where the next error occurs and displays the error message for that error on the message line. You can edit the line to correct the error and use the REVIEW_ERROR function again to move the cursor to the next error.

If you use REVIEW_ERROR and there are no more errors, the VAX RPG II editor displays the following message on the message line:

```
No more errors found
```

If you added or deleted a line in the program while correcting errors, when REVIEW_ERROR is used again the VAX RPG II editor will display the following message:

```
Reissue the editor COMPILE command
```

2.5.13 MOVE_TO_RULER Function

The MOVE_TO_RULER function (default = PF1/KP9) places the cursor as close as possible to the top of the ruler (if the editing window is above it) or toward the bottom of the ruler (if the editing window is below it). The current column remains unchanged. Movement is restricted to the boundaries of the SET SCROLL offsets. If the ruler is positioned above the editing window and the last line of the buffer appears, movement is stopped. If the ruler is positioned below the editing window and the first line of the buffer appears, movement is stopped. The MOVE_TO_RULER function will have no effect if no ruler is visible.

2.5.14 DELETE_FIELD Function

The DELETE_FIELD function (default = MINUS) places all the characters between the cursor and the next field (forward or backward, depending on the current direction) into the deleted-field buffer and replaces the characters with spaces.

2.5.15 UNDELETE_FIELD Function

The UNDELETE_FIELD function (default = PF1/MINUS) replaces the current field with the contents of the deleted-field buffer. If the contents of the deleted-field buffer are longer than the current field, the VAX RPG II editor copies to the current field until it is filled. If the contents of the deleted-field buffer are shorter than the current field, the VAX RPG II editor fills the current field to the right with spaces. Also, the cursor moves to the next field, depending on the current direction (ADVANCE or BACKUP).

2.5.16 ADVANCE Function

The ADVANCE function (default = KP4) sets the current direction to forward, that is, to the right and down, toward the end of the editing buffer. ADVANCE sets the direction for the following functions:

- CHARACTER
- DELETE_FIELD
- UNDELETE_FIELD
- FIELD
- END_OF_LINE
- FIND
- FIND_NEXT
- LINE
- PAGE
- SECTION

2.5.17 BOTTOM Function

The BOTTOM function (default = PF1/KP4) moves the cursor to the last line in the editing buffer. The current column remains unchanged.

2.5.18 BACKUP Function

The BACKUP function (default = KP5) sets the current direction to backward, that is, to the left and up, toward the beginning of the editing buffer. BACKUP sets the direction for the same functions as ADVANCE.

2.5.19 TOP Function

The TOP function (default = PF1/KP5) moves the cursor to the first line in the editing buffer. The current column remains unchanged.

2.5.20 CUT Function

The CUT function (default = KP6) moves the selected range of lines to the paste buffer. The selected range of lines consists of the line identified by the SELECT function (default = PERIOD) through the current line. The line following the selected range of lines becomes the current line. The current column remains unchanged. If there is no line following the selected range, the cursor is left in column 1 at the [EOB] mark. See Section 2.8.2 for an example using CUT.

2.5.21 PASTE Function

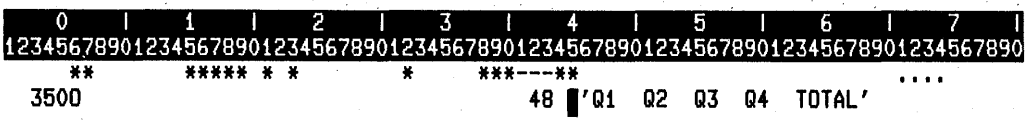
The PASTE function (default = PF1/KP6) inserts the contents of the paste buffer directly in front of the current line. The current line is moved down to accommodate the lines from the paste buffer. The current column and line remain unchanged. See Section 2.8.2 for an example using PASTE.

2.5.22 SHIFT_LEFT Function

The SHIFT_LEFT function (default = COMMA) deletes the character in the current column. All characters to the right of the current column are moved one column to the left, and the cursor position remains the same.

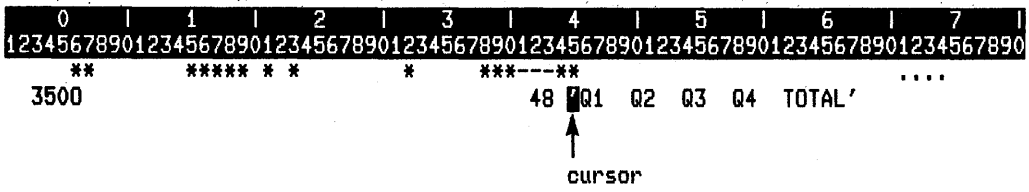
In the following example if the cursor is in column 45 on line 350 and SHIFT_LEFT is used, the VAX RPG II editor deletes the blank in column 45, moves all the characters to the right of the cursor one column to the left, and inserts a blank in column 80.

Before using SHIFT_LEFT:



ZK-4337-85

After using SHIFT_LEFT:



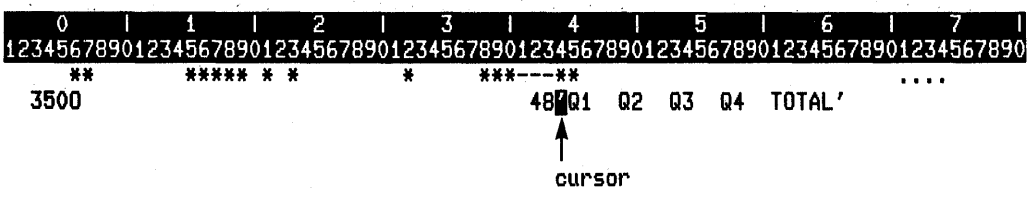
ZK-4338-85

2.5.23 SHIFT_RIGHT Function

The SHIFT_RIGHT function (default = PF1/COMMA) moves all characters in the current column through the end of the line one column to the right. A space is placed in the current column, and the cursor location remains unchanged.

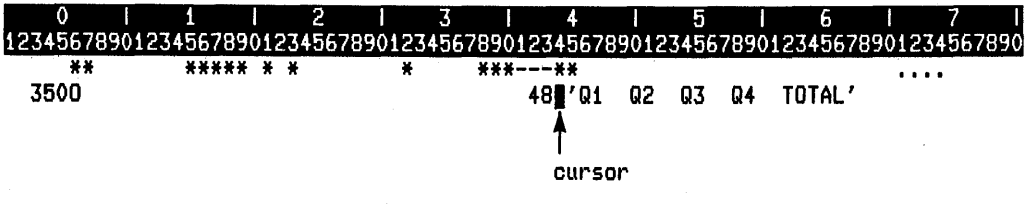
In the following example if the cursor is in column 44 on line 350 and SHIFT_RIGHT is used, the VAX RPG II editor moves all characters one column to the right of the cursor and inserts a blank in column 44. The blank in column 80 is lost.

Before using SHIFT_RIGHT:



ZK-4339-85

After using SHIFT_RIGHT:

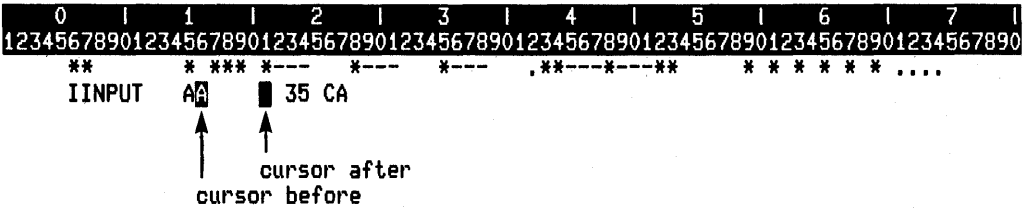


ZK-4340-85

2.5.24 FIELD Function

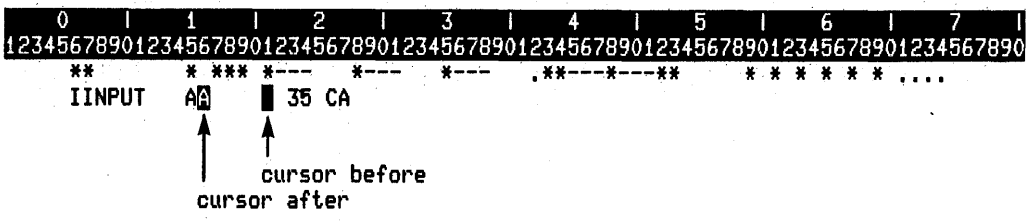
The FIELD function (default = KP1) moves the cursor to the nearest character in the next nonblank field. If the current direction is ADVANCE, using FIELD moves the cursor to the beginning of the next nonblank field following the current column. If the current direction is BACKUP, using FIELD moves the cursor to the end of the next nonblank field preceding the current column.

In the following example if the cursor is in column 16 and the current direction is ADVANCE, using FIELD moves the cursor to column 21.



ZK-4341-85

In the following example if the cursor is in column 21 and the current direction is BACKUP, using FIELD moves the cursor to column 16.



ZK-4342-85

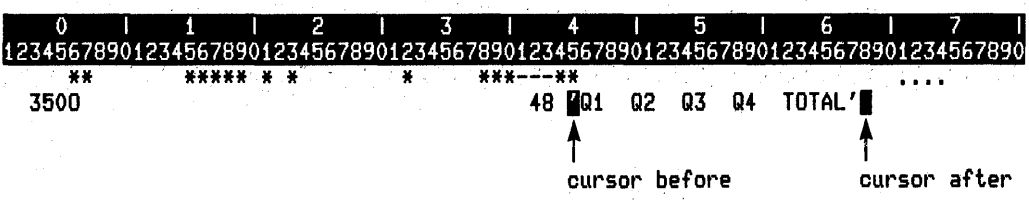
Note that you cannot use FIELD to move from one program line to another.

2.5.25 END_OF_LINE Function

The END_OF_LINE function (default = KP2) moves the cursor one column to the right of the end of the current line (the last nonblank character) if ADVANCE is the current direction. If the current direction is BACKUP, using END_OF_LINE moves the cursor one column to the right of the end of the preceding line.

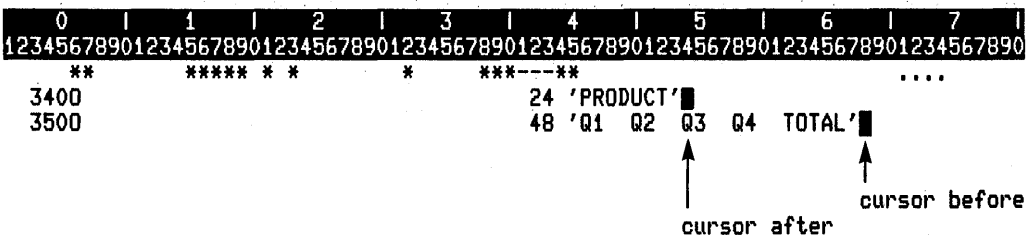
If the cursor is already at the end of the current line and the current direction is ADVANCE, using END_OF_LINE moves the current column one column to the right of the next line.

In the following example if the cursor is in column 45 and the current direction is ADVANCE and if you use END_OF_LINE, the VAX RPG II editor moves the cursor to column 68.



ZK-4343-85

In the following example if the cursor is in column 68 of line 350 and the current direction is BACKUP and if you use END_OF_LINE, the VAX RPG II editor moves the cursor to column 54 in line 340.



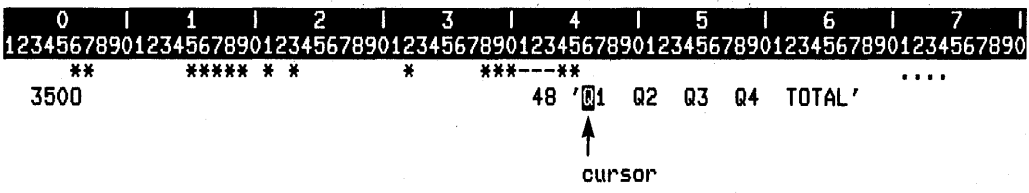
ZK-4344-85

2.5.26 DELETE_TO_END_OF_LINE Function

The DELETE_TO_END_OF_LINE function (default = PF1/KP2) deletes the characters from the current column to the end of the line. The cursor position remains unchanged.

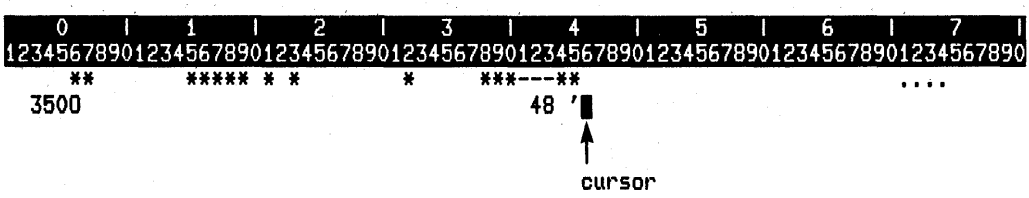
In the following example if the cursor is in column 46 and you use DELETE_TO_END_OF_LINE, the VAX RPG II editor deletes the characters in columns 46 through 67.

Before using DELETE_TO_END_OF_LINE:



ZK-4345-85

After using DELETE_TO_END_OF_LINE:

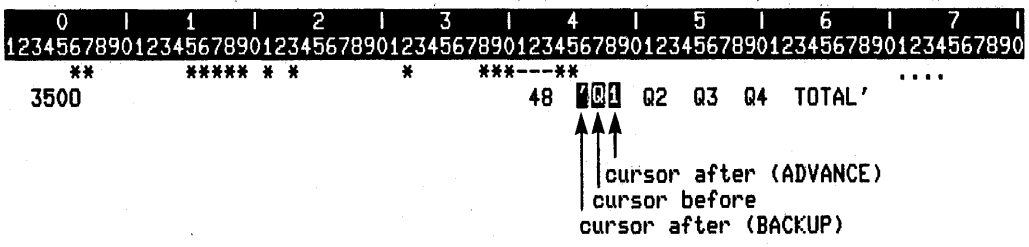


ZK-4346-85

2.5.27 CHARACTER Function

The CHARACTER function (default = KP3) moves the cursor position to the right or left, depending on the current direction (ADVANCE or BACKUP). If you attempt to move the cursor to the right of column 81 or to the left of column 1, no action is taken and an error message is displayed on the message line.

In the following example, if the cursor is in column 47 and the current direction is ADVANCE and if you use CHARACTER, the VAX RPG II editor moves the cursor to column 48. If the cursor is in column 47 and the current direction is BACKUP and if you use CHARACTER, the VAX RPG II editor moves the cursor to column 46.



ZK-4347-85

2.5.28 COLUMN Function

The COLUMN function (default = PF1/KP3) highlights the number of the current column by causing the column number in the 80-column ruler to blink. The column function takes no action if no ruler is visible.

On a terminal without Advanced Video Option (AVO), the COLUMN function performs no action.

On the VK100 (GIGI) terminal, the blinking for the COLUMN function is sometimes wider than the width of one character.

2.5.29 ENTER Function

The ENTER function (default = ENTER) terminates the FIND function (see Section 2.5.5) and VAX RPG II editor commands (see Section 2.5.9).

The ENTER function also clears any information on the message line.

2.5.30 LINE Function

The LINE function (default = KP0) causes the cursor to move one line up or down, depending on the current direction (ADVANCE or BACKUP). The cursor is moved to the current setting for the SET STARTCOLUMN command.

2.5.31 OPEN_LINE Function

The OPEN_LINE function (default = PF1/KP0) creates a new line above the current line. The new line becomes the current line, and the cursor is moved to the current setting for the SET STARTCOLUMN command. If the current setting for the SET STARTCOLUMN command is greater than 6, the new line will have the same specification format as the previous line. See Section 2.8.2 for an example of the OPEN_LINE function.

2.5.32 SELECT Function

The SELECT function (default = PERIOD) marks the current line as the beginning of the range of lines you are selecting (select range). The SELECT function highlights column 1 of the current line in reverse video. You can use SELECT to select a range of lines to be deleted or moved. You can then use CUT to move the selected lines from the editing buffer to the paste buffer (see Section 2.5.20), and you can use PASTE to reinsert them into the editing buffer at another location (see Section 2.5.21). The cursor position in the line does not matter—the entire line will be moved into the paste buffer when CUT is used.

If you select a line as the beginning of a select range and then delete that line, the select range will no longer be in effect and a message will be displayed on the message line.

You cannot select the line where [EOB] appears. If you select a range of lines that includes [EOB], [EOB] will not be placed in the paste buffer.

See Section 2.8.2 for an example of the SELECT function.

2.5.33 RESET Function

You can clear the current setting for the SELECT function by using the RESET function (default = PF1/PERIOD).

2.5.34 UP Function

The UP function (default = UP) causes the cursor to move up one line. The current column remains unchanged. If the current line is the first line in the editing buffer, the cursor will not be moved and an error message will be displayed.

2.5.35 DOWN Function

The DOWN function (default = DOWN) causes the cursor to move down one line. The current column remains unchanged. If the current line is the last line in the editing buffer, the cursor will not be moved and an error message will be displayed.

2.5.36 RIGHT Function

The RIGHT function (default = RIGHT) moves the cursor to the right one column. If the current column is 80, the cursor is not moved and column 81 becomes the current column. If the current column is 81, the cursor will not be moved and an error message will be displayed.

2.5.37 LEFT Function

The LEFT function (default = LEFT) moves the cursor to the left one column. If the current column is column 1, the cursor will not be moved and an error message will be displayed.

2.5.38 FIELD_BACKWARD Function

The FIELD_BACKWARD function (default = BS_KEY) moves the cursor to the tab stop preceding the current column. Or, if the cursor is before the first tab stop, it moves the cursor to column 1. If the current column is 1, the cursor will not be moved and an error message will be displayed.

2.5.39 DELETE_CHARACTER Function

The DELETE_CHARACTER function (default = DEL_KEY) replaces the character to the left of the cursor with a space and moves the cursor one column to the left. If you try to delete a character to the left of column 1, the cursor will not be moved and an error message will be displayed.

2.5.40 NEW_LINE Function

The NEW_LINE function (default = RET_KEY) creates a new line following the current line. The lines following the current line are moved down to accommodate the new line. If the current line is the last line in the current buffer, a new last line is created. The cursor is moved to the current setting for the SET STARTCOLUMN command. If the current setting for the SET STARTCOLUMN command is greater than 6, the new line will have the same specification format as the previous line.

2.5.41 FIELD_FORWARD Function

The FIELD_FORWARD function (default = TAB_KEY) moves the cursor to the next tab stop after the current column. If the cursor has already passed the last tab stop, FIELD_FORWARD moves the cursor to column 81. If the current column is column 81, the cursor will not be moved and an error message will be displayed.

2.5.42 REFRESH_SCREEN Function

The REFRESH_SCREEN function (default = CTRL_R_KEY and CTRL_W_KEY) rewrites the screen display. The cursor location remains unchanged.

2.5.43 DELETE_TO_BEGINNING_OF_LINE Function

The DELETE_TO_BEGINNING_OF_LINE function (default = CTRL_U_KEY) replaces the characters from the current column to column 1 with spaces. The cursor location remains unchanged.

2.5.44 EXIT Function

The EXIT function (default = CTRL_Z_KEY) writes the editing buffer to an output file as described in Section 2.1.4. If a journal file was created, it is not saved.

If you have issued the VAX RPG II editor COMPILE command, and then leave the VAX RPG II editor using EXIT, the following message will be displayed:

```
Subprocess terminated
```

If you invoked the VAX RPG II editor with the /NOOUTPUT qualifier or the /READ_ONLY qualifier, the following message will be displayed:

```
Use EXIT with an output file specification or QUIT
```

The EXIT function performs the same function as the EXIT/NOSAVE command.

2.6 Editor Commands

This section describes the VAX RPG II editor commands and explains how to use them. You must issue the COMMAND function before executing a VAX RPG II editor command. Section 2.5.9 discusses the COMMAND function.

The following conditions exist when executing VAX RPG II editor commands:

- If you type a command with a missing required parameter, you will receive a prompt to supply the missing parameter.
- Qualifiers can appear anywhere on the line; they do not have to immediately follow the command and can appear in any order.
- Qualifiers can be negated.
- Command line input can be in uppercase, lowercase, or both.
- Abbreviations are allowed. You must type enough information to resolve any ambiguity.
- You can enter full line comments, end-of-line comments, and blank lines in a command line.
- You can continue a command line by entering a hyphen (-) at the end of the line. You will get a prompt for more input.
- Terminate a command by pressing either the RETURN key or the ENTER key.

2.6.1 COMPILE Command

The COMPILE command compiles the source code in the editing buffer and displays the following messages:

```
Subprocess activated
Beginning compilation
```

The message "Subprocess activated" appears only when the COMPILE command is issued for the first time during an editing session.

The format of the COMPILE command is as follows:

```
COMPILE [/LIST]
```

The following message is displayed to indicate how many errors were found:

```
Compilation complete - n errors found
```

If n is 0, no errors were found and you can leave the VAX RPG II editor, then link and run your program.

If the compilation encounters errors, the error text associated with the first error is displayed on the message line and the cursor is moved to the column and line where the first error occurs. If there is more than one error, use the REVIEW_ERROR function to move the cursor to the column and line causing the next error. See Section 2.5.12 for more information on the REVIEW_ERROR function.

You can use only the /LIST qualifier with the COMPILE command to create a listing file for the compiled source code. The default is /NOLIST. The /OBJECT qualifier is always in effect. However, if the compilation encounters fatal errors, an object module will not be produced.

You can specify a symbol definition at the DCL command level to change the defaults for a compilation. When you issue the VAX RPG II editor COMPILE command, the compiler will use these settings. In the following example, the symbol RPG is defined to compile a program and generate a listing file with machine-generated code. The compiler will also generate code in the program to check for blanks in numerics.

```
$ RPG :== RPG/LIST/MAC/CHECK:BLANKS_IN_NUMERICS
```

You must use COMPILE/LIST in the VAX RPG II editor to get a program listing.

To use the debugger after you enter the COMPILE command, you must first define the following command before invoking the VAX RPG II editor:

```
$ RPG := RPG/DEBUG
```

See Chapter 5 for information on how to set the appropriate source file for debugging.

The COMPILE command requires each line in the editing buffer to be 140 characters or less.

If you define RPG to invoke something other than the VAX RPG II compiler, or if the VAX RPG II compiler encounters an unexpected error, the following message is displayed on the message line:

```
Unexpected error during compilation - leave editor and try DCL RPG command
```

2.6.2 DEFINE KEY Command

The DEFINE KEY command allows you to bind specific keys to specific VAX RPG II editor functions. These functions are listed with their default key definitions in Table 2-2 at the beginning of Section 2.5.

You can bind the following keys in the VAX RPG II editor:

- Control keys
- Cursor keys
- Editing keys (LK201 except Rainbow)
- Function keys (LK201 except Rainbow)
- Keypad keys
- Gold versions of all these keys

Exceptions:

The following list contains seven control key restrictions. These are special functions of the VAX/VMS operating system.

- CTRL_C_KEY
- CTRL_O_KEY
- CTRL_T_KEY
- CTRL_Q_KEY
- CTRL_X_KEY
- CTRL_S_KEY
- CTRL_Y_KEY

Note that key redefinition does not cause automatic update to the VAX RPG II editor keypad diagram and key-specific HELP text.

The format of the DEFINE KEY command is as follows:

```
DEFINE KEY[/GOLD] key_name function
```

In this command, /GOLD indicates that you must press the GOLD key followed by key_name to execute the chosen function. For example:

```
DEFINE KEY/GOLD KP5 CUT
```

When you enter this command and then press the GOLD key, followed by pressing the KP5 key, the CUT function is executed.

If key_name is not a valid definable key, or if function is not a valid VAX RPG II editor function that you can bind to a key, an error message is displayed.

To redefine the GOLD key, enter the following line at the command prompt:

```
DEFINE KEY key_name GOLD
```

To remove the GOLD key completely, enter the following line at the command prompt:

```
DEFINE KEY/GOLD PF1 GOLD
```

Note that if you use a key name other than PF1 with this command, it will be treated as if PF1 had been entered.

Note also that you must redefine the GOLD key (default = PF1) before you can define the PF1 key to a function other than GOLD.

See Table 2–2 for a list of default key definitions. This table provides a list of definitions that you can bind to keys. Note that in some cases, more than one key is bound to the same procedure. Note also that TAB_KEY and CTRL_I_KEY (the default settings for FIELD_FORWARD), and the RETURN_KEY and CTRL_M_KEY (default settings for RETURN), can only be bound to the same function, while the F10 key and CTRL_Z_KEY (the default settings for EXIT) may be bound to separate functions.

The SECTION_FORWARD and the SECTION_BACKWARD functions are not bound by default to any key on the VT100 and VK100 (GIGI) terminal keyboards. However, you can bind any of the valid definable keys to those functions.

Table 2–3 contains additional keys that you can bind to the functions listed in Table 2–2.

Table 2–3: VAX RPG II Keynames for Valid Definable Keys

VAX RPG II Keyname	LK201	VT100 Family VK100 (GIGI)
PF1	PF1	PF1
PF2	PF2	PF2
PF3	PF3	PF3
PF4	PF4	PF4

Table 2–3 (Cont.): VAX RPG II Keynames for Valid Definable Keys

VAX RPG II Keyname	LK201	VT100 Family VK100 (GIGI)
KP0,KP1 . . . KP9	0,1 . . . 9	0,1 . . . 9
PERIOD	.	.
COMMA	,	,
MINUS	-	-
ENTER	ENTER	ENTER
UP	↑	↑
DOWN	↓	↓
LEFT	←	←
RIGHT	→	→
E1	FIND /E1	
E2	INSERT HERE /E2	
E3	REMOVE /E3	
E4	SELECT /E4	
E5	PREV SCREEN /E5	
E6	NEXT SCREEN /E6	
HELP	HELP /F15	
DO	DO /F16	
F7 . . . F20	F7 . . . F20	
TAB_KEY	Tab	Tab
RET_KEY	RETURN	RETURN
DEL_KEY	<X	DELETE

Table 2–3 (Cont.): VAX RPG II Keynames for Valid Definable Keys

VAX RPG II Keyname	LK201	VT100 Family VK100 (GIGI)
LF_KEY		Line-feed
BS_KEY		Back-space
CTRL_A_KEY	CTRL/A	CTRL/A
.	.	.
.	.	.
.	.	.
CTRL_Z_KEY	CTRL/Z	CTRL/Z

Note the list of exceptions at the beginning of this section.

You can modify the key bindings shown in Table 2-2 at VAX RPG II editor start-up by creating a start-up command file with the desired DEFINE KEY commands. For more information on using DEFINE KEY, see Section 2.7.2.

2.6.3 EXIT Command

The EXIT command writes the editing buffer to the output file, leaves the VAX RPG II editor, and returns to the DCL command prompt ($), as shown in the following example.

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        * *** *---    *---    *---   .**---*---**     * * * * * * ....
   10H*++
   20H*  FUNCTIONAL DESCRIPTION:
   30H*    This program produces a report of shipments for various
   40H*    products broken down by division and department using an
   50H*    input file with the shipment data for the past 4 quarters.
   60H*--
   70H
   80FSHIPS   IP  F      41            DISK
   90FSUMREP  O   F      98          LPRINTER
  100E                   QTY         4  2 0
  110LSUMREP   55FL 500L
  120ISHIPS   AA  01
  130I                                          1   5 DIV   L2
  140I                                          6   7 DEPT  L1
  150I                                          8  16 PROD
  160I                                         17  24 QTY
  170C*
  180C   01               XFOOTQTY       PROQTY  30
  190C   01      PROQTY    ADD  DEPQTY    DEPQTY  30
53 records written to file MYDISK:[MYDIRECTORY]MYFILE.RPG;2

$
```

ZK-4348-85

The format of the EXIT command is as follows:

```
EXIT [/SAVE] [file-spec]
```

The output file is one of the following:

- The file name you supplied with the EXIT command
- The file name you supplied with the /OUTPUT qualifier to the RPG/EDIT command
- The same file name as the input file you specified when you invoked the VAX RPG II editor, if both the /READ_ONLY and /NOOUTPUT qualifiers were not used with the RPG/EDIT command

The VAX RPG II editor will write the editing buffer to the output file even if no changes have been made.

You can use the /SAVE qualifier with the EXIT command to save the journal file, if one was created. The file name of the journal file is the name of the output file, if specified, with the JOU file type. If a journal file name was not specified, the VAX RPG II editor uses the same file name as the input file. See Section 2.1.3 for information on journal files.

If an error occurs during the execution of an EXIT/SAVE command and you resume editing, the journal facility will still be in effect.

If you have issued the VAX RPG II editor COMPILE command and then leave the VAX RPG II editor by typing the EXIT command, the following message will be displayed on the message line:

```
Subprocess terminated
```

2.6.4 HELP Command

The HELP command displays information on VAX RPG II editor functions and commands in the HELP window of the VAX RPG II editor screen. The following example shows what the screen looks like after you issue the COMMAND function, type the HELP command, and press either the RETURN key or the ENTER key.

```
This help information describes the VAX RPG II editor.  Information  is
included on editing functions performed by pressing keypad keys and control
keys.  Editor commands issued with the COMMAND function are described.
Brief notes describing the journaling facility are included.  You can read
this help information while using the editor by typing the HELP command or
using the help functions.


Additional information available:

Commands   Cursor     Functions  Help       Journal    Keypad
Specifications
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *...........*..*....*.............*.............................
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*     This program produces a report of shipments for various
     H*     products broken down by division and department using an
     H*     input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41           DISK
```

ZK-4349-85

The format of the HELP command is as follows:

```
HELP [/FULL] [/PAGE] [/PROMPT] list-of-topics
```

The /PAGE qualifier is similar to the DCL HELP command /PAGE qualifier. If the HELP text does not fit in a logical page (in this case, the HELP window), the text is displayed one page at a time and you must press the RETURN key to advance to the next page. The default is /NOPAGE.

The /PROMPT qualifier is similar to the DCL HELP command /PROMPT qualifier. After HELP for the given list of topics is displayed, you are prompted for additional topics, which are then linked to the current list of topics. Press the RETURN key repeatedly to back up through the levels of HELP text. Press CTRL/Z to terminate the HELP command. The default is /NOPROMPT.

The /FULL qualifier uses the entire screen, except for the prompt and message lines, to display HELP text. When the requested HELP text has been displayed, the previous screen layout is restored. You are prompted to press the RETURN key before the screen is repainted. If the previous screen contained HELP text, it is not restored. Instead, the last 11 lines of text from the current HELP command is left in the HELP window. The default is /NOFULL.

Note that /FULL, /PAGE, and /PROMPT are positional qualifiers. If they occur after a topic or subtopic, they are interpreted as subtopics on which HELP is desired.

There is no fixed number on the list of topics. Whatever can fit on the command line is valid. If you use the /PROMPT qualifier, you can extend the depth indefinitely.

By default, the VAX RPG II editor searches its own HELP library (SYS$HELP:RPGEDIHLP) for the given list of topics.

You can access other libraries in the following ways:

- If the first topic has the form @filespec, that library is searched instead.
- If you define logical names of the form HLP$LIBRARY, HLP$LIBRARY_1 . . . HLP$LIBRARY_999, the LIBRARIAN searches them in the following order: root library, main library, process libraries, group libraries, and system libraries.

The following example shows what the screen looks like after you issue the COMMAND function, type HELP COMMANDS, and press either the RETURN key or the ENTER key:

```
COMMANDS

  Editor commands are executed by pressing the COMMAND function (PF1/KP7 - see
  information for FUNCTIONS).  Any command, parameter or qualifier can be
  abbreviated so that the information typed is unambiguous.  The prompt
  "Command: " is displayed in reverse video on the prompt line.  Any
  characters that can normally be typed in the editor may be typed at the
  prompt.

  Qualifiers can be negated and can also appear in any order on a command line
  after the name of the command.

  Blank command lines are ignored.  Also any text on a command line after an
  exclamation point ("!") is ignored.


  Additional information available:

  COMPILE     DEFINE      EXIT        HELP        INCLUDE     QUIT
  RESEQUENCE  SET         SHOW        SUBSTITUTE
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *.....*..*....*.............*.................................
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*     This program produces a report of shipments for various
     H*     products broken down by division and department using an
     H*     input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41           DISK
```

ZK-4350-85

After you press either the RETURN key or the ENTER key to execute a HELP command and HELP information is displayed, the VAX RPG II editor returns the cursor to its current column and line so you can resume editing.

If you terminate the HELP display by pressing CTRL/C, you must type a terminator to resume editing.

2.6.5 INCLUDE Command

The INCLUDE command copies a text file into the source buffer using the VAX RPG II editor. The format of the INCLUDE command is as follows:

```
INCLUDE file-spec
```

The file is copied into the editing buffer, immediately before the current line. The cursor position remains unchanged. Note that the lines read in are not syntax checked.

If the INCLUDE command is successful, the number of records read in is displayed on the message line.

2.6.6 QUIT Command

The QUIT command allows you to leave the VAX RPG II editor and return to the DCL command level, without writing the editing buffer to the output file, as shown in the following example.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **       * *** *---   *---   *---   .**---*---**     * * * * * * ....
   10H*++
   20H*  FUNCTIONAL DESCRIPTION:
   30H*    This program produces a report of shipments for various
   40H*    products broken down by division and department using an
   50H*    input file with the shipment data for the past 4 quarters.
   60H*--
   70H
   80FSHIPS   IP  F      41            DISK
   90FSUMREP  O   F      98           LPRINTER
  100E                   QTY          4  2 0
  110LSUMREP   55FL 500L
  120ISHIPS   AA  01
  130I                                          1   5 DIV   L2
  140I                                          6   7 DEPT  L1
  150I                                          8  16 PROD
  160I                                         17  24 QTY
  170C*
  180C   01               XFOOTQTY        PROQTY  30
  190C   01      PROQTY    ADD  DEPQTY     DEPQTY  30
Command: QUIT

$
```

ZK-4351-85

The format of the QUIT command is as follows:

```
QUIT [/SAVE]
```

Use the QUIT command if you have made no changes to the editing buffer or if you have decided not to save the changes you made. If you have made changes, or if you have pressed keys to move the cursor past the last nonblank character in any line, the VAX RPG II editor displays the following message:

```
Buffer modifications will not be saved, continue quitting (Y or N)?
```

You can respond by typing Y, YE, or YES. Any other response will continue the editing session. If you resume editing, a journal file for your edits will not be created. To resume journaling, you must leave the VAX RPG II editor, and invoke the VAX RPG II editor again.

You can use the /SAVE qualifier with the QUIT command to save the journal file, if one was created.

If you issue the VAX RPG II editor COMPILE command and then leave the VAX RPG II editor by typing the QUIT command, the following message will be displayed on the message line:

```
Subprocess terminated
```

2.6.7 RESEQUENCE Command

The RESEQUENCE command either generates a new line number for each program line in the editing buffer or resequences existing line numbers. The format of the RESEQUENCE command is as follows:

```
RESEQUENCE [/REMOVE] [initial-value [increment]]
```

The RESEQUENCE command renumbers program lines up to the first line containing a double slash (//) and a blank or a double asterisk (**) and a blank in columns 1 through 3. Lines are numbered beginning at the initial value (default = 10) and are incremented by the increment value (default = 10).

The maximum line number is 99,999. If during resequencing, a line number plus the increment exceeds 99,999, that line and all remaining lines are numbered 99,999. In this case, reissue the RESEQUENCE command with smaller values for the initial value and increment by the increment value.

The RESEQUENCE/REMOVE command will remove all line numbers in the editing buffer.

The following command renumbers the line numbers in the editing buffer beginning with 100 and increments each number by 20:

```
RESEQUENCE 100 20
```

See Section 2.8.2 for another example of the RESEQUENCE command.

2.6.8 SET Command

The VAX RPG II editor has eight optional functions which are controlled by the SET command:

- COMMAND
- DEFAULT
- HELP
- RULER
- SCROLL
- SECTION
- STARTCOLUMN
- SYNTAXCHECK

The SET command controls VAX RPG II editor options. Once set, these options are in effect until you leave the VAX RPG II editor or reissue the SET command.

You can include SET commands in a start-up command file. See Section 2.7.2 for information on start-up command files.

The format of the SET command is as follows:

```
SET option
```

2.6.8.1 COMMAND Option

The COMMAND option allows you to process additional start-up command files at the beginning of the VAX RPG II editing session. The format of the COMMAND option is as follows:

```
SET COMMAND file-spec
```

For information on the SET COMMAND option, see Section 2.7.2.

2.6.8.2 DEFAULT Option

The DEFAULT option allows you to determine the default value of qualifiers used in other editor commands. The format of the DEFAULT option is as follows:

```
SET DEFAULT option
```

For example, the following command means that any later HELP command uses the PAGE and PROMPT options by default. You can turn defaults off by using the negated form of an option. (For example, SET DEFAULT NOPROMPT.)

```
SET DEFAULT PAGE,PROMPT
```

2.6.8.3 HELP Option

The HELP option allows you to choose a variety of settings. The format of the HELP option is as follows:

```
SET HELP { KEYPAD | NONE | SPECIFICATIONS }
```

The HELP KEYPAD option has the same effect as using the HELP_KEYPAD function (default = PF2). See Section 2.5.2 for information on HELP_KEYPAD.

The HELP NONE option allows you to start up as if you used the DISPLAY function. See Section 2.5.11 for information on the DISPLAY function.

The HELP SPECIFICATIONS option has the same effect as using the HELP_SPECIFICATIONS function (default = PF1/PF2). See Section 2.5.3 for information on HELP_SPECIFICATIONS.

2.6.8.4 RULER Option

The RULER option moves the three-line 80-column ruler with tab stops as a unit, to either the top or bottom of the current window. The SET RULER NONE option removes the ruler from the screen.

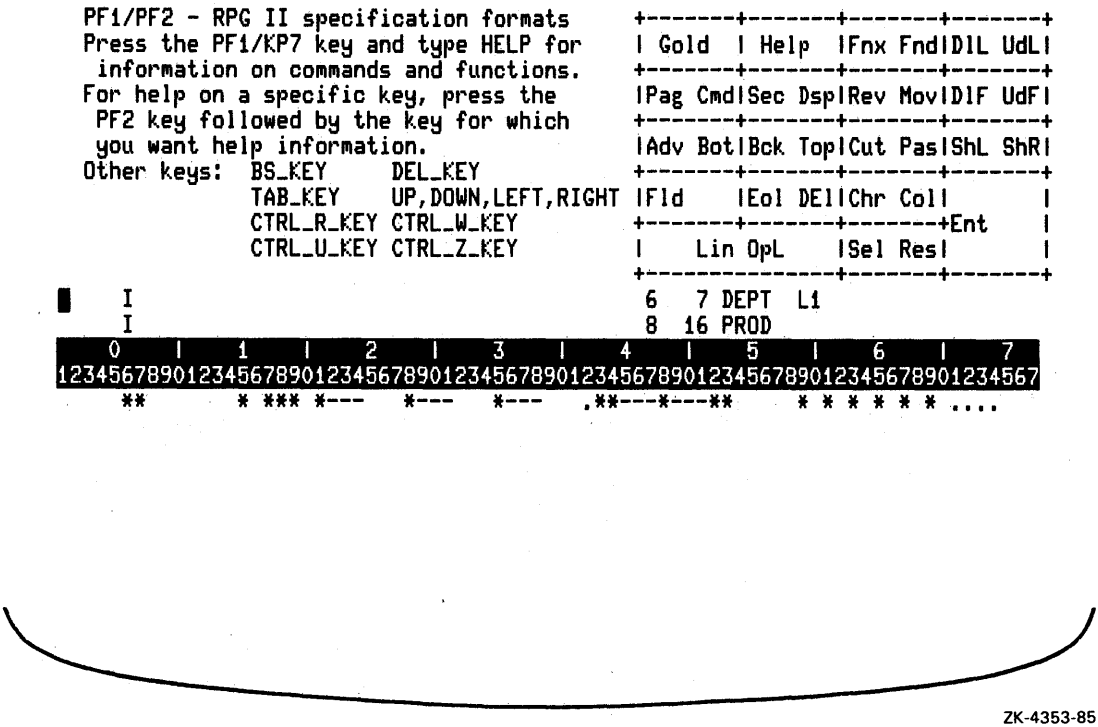
The format of the RULER option is as follows:

```
SET RULER { TOP | BOTTOM | NONE }
```

The following example shows an editor screen as it appears after a SET RULER BOTTOM command, with an 18-line terminal page size. The next example shows the same screen followed by a HELP request.

```
     FSHIPS   IP  F      41            DISK
     FSUMREP  O   F      98          LPRINTER
     E                    QTY        4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                        1   5 DIV   L2
     I                                        6   7 DEPT  L1
     I                                        8  16 PROD
     I                                       17  24 QTY
     C*
     C   01                 XFOOTQTY       PROQTY  30
     C   01       PROQTY    ADD  DEPQTY    DEPQTY  30
     C*
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
     **        * *** *---   *---    *---    .**---*---**      * * * * * * ....
```

ZK-4352-85



ZK-4353-85

2.6.8.5 SCROLL Option

The SCROLL option specifies the region within the editing window where the cursor will stay.

The format of the SCROLL option is as follows:

```
SET SCROLL [top-offset [bottom-offset]]
```

Top-offset is the number of lines from the top of the editing window to the top of the scrolling region. Bottom-offset is the number of lines from the bottom of the scrolling region to the bottom of the editing window. If bottom-offset is omitted, the current offset from the bottom is not changed. If the top and bottom offsets are omitted, both offsets are set to the initial editor defaults of zero.

If you enter SET SCROLL 0 1, then the cursor will move from the line next to the ruler on the top, to within one line above the bottom of the editing window. If you want to keep the cursor from hitting the top line, enter a number greater than zero for top-offset. The higher the number, the greater the number of source code lines that will remain on the screen between the cursor and the top or bottom of the editing window.

2.6.8.6 SECTION Option

The SECTION option specifies the number of lines the VAX RPG II editor will move the cursor (forward or backward) when the SECTION function (default = KP8) is used and there is no HELP information displayed. You can specify any value between 1 and 5 less than the terminal page length. By default, the bottom line of the editing window moves to the top (just under the tab stops), regardless of the size of the window.

The format of the SECTION option is as follows:

```
SET SECTION lines
```

If you specify a SECTION value other than the default (when the HELP window is displayed), the SECTION value is proportional to the visible number of lines in the editing window.

2.6.8.7 STARTCOLUMN Option

The STARTCOLUMN option specifies the current column for the following functions:

- NEW_LINE
- OPEN_LINE
- LINE
- PAGE

The format of the STARTCOLUMN option is as follows:

```
SET STARTCOLUMN column
```

The default value is column 7. When the setting for the STARTCOLUMN option is greater than 6, the RETURN key and OPEN_LINE (default = PF1/KP0) function supply a specification type in column 6 that is the same as is present on the current line.

2.6.8.8 SYNTAXCHECK Option

The SYNTAXCHECK option specifies that syntax checking and automatic right justification of numeric fields will occur.

The format of the SYNTAXCHECK option is as follows:

```
SET SYNTAXCHECK { ON | OFF | PROMPT }
```

By default, the VAX RPG II editor starts up with the SYNTAXCHECK option set on. This setting can be changed in a start-up command file, or interactively.

If you modify a line when syntax checking is on, and then attempt to move off the line, one of the following events will occur:

- If there are no errors on the line and all numeric entries are properly justified, the requested action takes place.
- If there are no errors on the line, but one or more numeric entries are not properly justified, the numeric entries are justified, the justified fields are highlighted, and the requested action takes place. The highlighting is removed from the fields when the next line is syntax checked.
- If a syntax error is detected, the requested action does not take place. The cursor is positioned at the column of the error, and the error message is displayed on the message line. You can either correct the error or ignore the error by immediately moving off the line. Another syntax check will take place on that line only if you modify it again.

If you enter table and array data while SYNTAXCHECK is set on, there is a risk that the data will be right-justified as if it were part of the source program. This will yield unexpected results. Therefore, it is recommended that SYNTAXCHECK be set off while entering table and array data, or that you use the PROMPT argument with SYNTAXCHECK. When the PROMPT argument is in effect, the VAX RPG II editor will highlight any proposed numeric right justification before it occurs and will prompt you to confirm whether you want right justification.

Note that SYNTAXCHECK detects all errors, regardless of the severity. Therefore, depending on the qualifiers you use when compiling, it is possible that informational messages that you see in the editing session will not be issued by the compiler. You can ensure that the errors will be reported by compiling at DCL with the /WARNINGS=ALL qualifier. This will cause both warning and informational messages to be sent. See Section 3.1.2.8 for additional information on this qualifier.

2.6.9 SHOW Command

The SHOW command displays the current settings for the following options:

- DEFAULT
- SCROLL
- SECTION
- STARTCOLUMN
- SYNTAXCHECK
- VERSION

The format of the SHOW command is as follows:

```
SHOW option
```

The current settings appear on the message line, as shown in the following examples:

```
COMMAND: SHOW DEFAULT PAGE,PROMPT
Current defaults are NOPAGE,NOPROMPT

Command: SHOW SCROLL
Scroll offset from top is 0, from bottom is 0

Command: SHOW SECTION
Section length is: 18 or when HELP is displayed: 7

Command: SHOW STARTCOLUMN
STARTCOLUMN value is: 7

Command: SHOW SYNTAXCHECK
SYNTAXCHECK is ON
```

The VERSION argument displays the current version of the VAX RPG II editor and a VAX RPG II copyright statement, as shown in the following example:

```
Command: SHOW VERSION
VAX RPG II V2.1 editor  COPYRIGHT (C) DIGITAL EQUIPMENT CORPORATION 1986
```

2.6.10 SUBSTITUTE Command

The SUBSTITUTE command allows you to substitute text using the VAX RPG II editor. The format of the SUBSTITUTE command is as follows:

```
SUBSTITUTE search-argument replace-argument [/SELECT] [/QUERY]
```

The SUBSTITUTE command replaces all occurrences of the search-argument with the replace-argument in the specified range. If the /SELECT qualifier is specified, the command applies to all lines in the select range. Otherwise, it applies to all lines in the buffer.

Only exact matches of the search-argument with text in the editing buffer are performed.

Only equal length substitutions are performed. If one argument is shorter than the other, it is padded on the right with spaces before searching and replacing begins.

If you specify the /QUERY qualifier, then at each occurrence of the string to be substituted the following occurs:

- The string to be substituted is highlighted.
- A "Substitute this occurrence (YES, NO, ALL, or QUIT)?" prompt is displayed on the prompt line.
- You may answer YES, NO, ALL, or QUIT.
- If you answer YES, the text is replaced and the VAX RPG II editor finds the next occurrence.
- If you answer NO, the text is not replaced and the VAX RPG II editor finds the next occurrence.
- If you answer ALL, the current text is replaced as well as any further occurrences of the text, without additional prompting.

- If you answer QUIT, the text is not replaced and the SUBSTITUTE command terminates.
- If you make any other response, the sequence is repeated from the point where the prompt message is displayed.

If the SYNTAXCHECK option is on, the current line is syntax checked after each change is made. If a syntax error is found, the substitution is terminated.

The command does not display the lines on which substitutions are made (except in /QUERY mode).

Upon completion of this command, the message "Substitutions: n" is displayed in the message area, where 'n' indicates the number of substitutions performed.

Upon completion of this command when the /SELECT qualifier was specified, the select range is removed.

The SUBSTITUTE command ignores the current editing direction. It always proceeds from the beginning of the range to the end. The current editing direction is not changed; it is ignored for the duration of the command.

The cursor is returned to where it was before the command was issued.

Rules for Specifying Search-Argument and Replace-Argument

- The search-argument must contain at least one nonblank character. If it does not, the message "The search string must contain at least one nonblank character" is displayed in the message area.
- If lowercase characters are desired in the substitution, the argument must be enclosed within double quotation marks (for example, "string"). Otherwise, lowercase characters are converted to uppercase.
- If the argument contains a terminator, such as a blank space or a slash (/), the argument must be enclosed within double quotation marks (for example, " " and "/").
- If the argument contains a double quotation mark ("), two double quotation marks ("") must be entered.
- Single quotation marks (') are not treated like double quotation marks ("").
- Control characters cannot be entered in arguments.

2.7 Customizing the VAX RPG II Editor

This section describes several VAX RPG II editor commands that are available to you. These commands enable you to customize your editing environment.

2.7.1 Using Editor Commands

For the purpose of this example, you want the ruler to lie on the bottom of the screen and the HELP keypad to show in the HELP window. Because you are entering a program with a compile-time table or array, you want to be prompted before any numeric fields are right justified. Because you have chosen a small scrolling region, you want the SECTION function to give you 10 lines. Finally, you want to use CTRL/P to review errors.

You would use the COMMAND function (default = PF1/KP7) to enter each of the following commands:

- SET RULER BOTTOM
- SET SCROLL 2 2
- SET SECTION 10
- SET HELP KEYPAD
- SET SYNTAXCHECK PROMPT
- DEFINE KEY CTRL_P_KEY REVIEW_ERROR

See Section 2.6.8.4 for an example of a screen with the ruler on the bottom and the HELP keypad displayed.

2.7.2 Start-up Command Files

Start-up command files allow you to specify a set of commands to be executed automatically each time you begin an editing session. A start-up command file can contain any of the VAX RPG II editor commands. It can also contain comment and blank lines to improve readability. Each command is executed as if the COMMAND function were used.

The VAX RPG II editor uses the /COMMAND qualifier to find a start-up file. This qualifier is present by default, with a default value of RPGINI.

The uses of the /COMMAND qualifier and their effects are as follows:

- If /COMMAND=filespec is used, the specified file is executed.
- If the /NOCOMMAND qualifier is used, no command file is executed.

All start-up files are opened with a default file type of RPG. The value for the /COMMAND qualifier can be a full or partial file specification, or a logical name that translates to a file specification.

Control can be passed from one start-up file to another by using the COMMAND option of the SET command. When the VAX RPG II editor is executing commands from a start-up command file and encounters a SET COMMAND command, it tries to find the associated file by translating logical names, if necessary. If a file is found, the contents of that file are then executed in the same way as the original start-up file. The rest of the commands in the start-up file are not executed. If the file is not found, the rest of the commands in the start-up file are executed.

Following are several ways of using these options to customize your editing environment.

If you do not want to execute any start-up file, your command line should be as follows:

```
$ RPG/EDIT/NOCOMMAND filespec
```

To execute your own start-up commands, create a file of VAX RPG II editor commands and define the logical name RPGINI to reference it. For example, create the file MYSTART-UP.RPG to contain the following:

```
SET DEFAULT PAGE,PROMPT
SET HELP KEYPAD
SET RULER NONE
DEFINE KEY CTRL_N_KEY REVIEW_ERROR
```

Add to your LOGIN.COM file the following:

```
$ DEFINE RPGINI MYDISK:[MYDIRECTORY]MYSTARTUP.RPG
```

Then, whenever you invoke the VAX RPG II editor, your commands will be executed.

One way to establish a customized environment for many users at the same time is described here. A system-wide start-up command file can be established by defining the logical name RPGINI in the system logical name table. In the following example, the file SYSRPGINI.RPG is in the directory addressed by SYS$PUBLIC.

```
! System-wide start-up commands
SET HELP SPECIFICATIONS
SET RULER BOTTOM
SET COMMAND RPGINI.RPG
```

If RPGINI was defined by the following command then, by default, all users on the system would have that set of commands executed automatically. The last command shown would mean that after executing the system-wide commands, the VAX RPG II editor would also execute any commands found in the RPGINI.RPG file in the default directory.

```
$ DEFINE/SYSTEM RPGINI SYS$PUBLIC:SYSRPGINI.RPG
```

2.7.3 Modifying Screen Length

You can determine the number of lines on the terminal screen that are used by the VAX RPG II editor. This is a useful option for a variety of reasons. If you have a VT100 terminal that does not have Advanced Video Option (AVO), there are only 14 lines in 132-column mode. It is also useful if you have a terminal with more than 24 lines. Also, if you have a terminal that runs at a slow baud rate, you can control the number of lines displayed on the VAX RPG II editor screen. This improves performance over a slow communication line by decreasing the number of lines on the screen that must be updated during an editing session.

Use the DCL command SET TERMINAL/PAGE=n to set the length of the page on your terminal screen. You can also set the width of the page with SET TERMINAL/WIDTH=n. If you set the width to 132 columns, you will get the full text of the VAX RPG II editor error messages.

Note that there must be at least six lines on the screen to allow for the two-line ruler, tab stop line, prompt line, message line, and one line in the source editing window.

2.8 Creating and Editing Programs

This section contains a sample VAX RPG II program and some of the output it might produce. Section 2.8.1 shows you how to create a program using the VAX RPG II editor, and Section 2.8.2 shows you how to edit a program using the VAX RPG II editor. Both sections use the following sample program.

Note that this example assumes a 24-line screen and no start-up file.

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
H*++
H*  FUNCTIONAL DESCRIPTION:
H*     This program produces a report of shipments for various
H*     products broken down by division and department using an
H*     input file with the shipment data for the past 4 quarters.
H*--
H
FSHIPS   IP  F      41            DISK
FSUMREP  O   F      98           LPRINTER
E                    QTY         4  2 0
LSUMREP   55FL 500L
ISHIPS   AA  01
I                                          1   5 DIV   L2
I                                          6   7 DEPT  L1
I                                          8  16 PROD
I                                         17  24 QTY
C*
C   01                 XFOOTQTY       PROQTY  30
C   01       PROQTY    ADD  DEPQTY    DEPQTY  30
C*
CL1          DEPQTY    ADD  DIVQTY    DIVQTY  30
CL1                    Z-ADD0         DEPQTY
CL2          DIVQTY    ADD  FINQTY    FINQTY  40
C*
```

ZK-4354-85

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
OSUMREP  H  001   1P
O                                          48 'PRODUCT SHIPMENT REPORT'
O        H 02     1P
O                          UDATE Y   12
O                                          48 'PRODUCT SHIPMENT REPORT'
O        H  1     1P
O                                          42 'SHIPMENTS'
O        H  2     1P
O                                          15 'DIVISION  DEPT'
O                                          24 'PRODUCT'
O                                          48 'Q1  Q2  Q3  Q4  TOTAL'
O        D  1     01
O                 L2       DIV        8
O                 L1       DEPT      14
O                          PROD      25
O                          QTY   Z   41
O                          PROQTYZ   48
O        T  1     L1
O        T  0     L2
O                          DIV       69
O        T  0     L2
O                          DIV       69
O        T 02     L2
O                          DIVQTYZB  48
O                                          63 '<== Total for'
O                          DIV       69
O        T  0     LR
O                          FINQTY1   48
O                                          65 '<== GRAND TOTAL'
```

ZK-4355-85

A sample of the output from this program might appear as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    9/09/83                PRODUCT SHIPMENT REPORT
                                    SHIPMENTS
 DIVISION   DEPT  PRODUCT    Q1  Q2  Q3  Q4  TOTAL
   East      12   CPU-19     12  13  14  15     54
                  CPU-20     11  11  11  10     43

             13   TERM-12    12  34  34  35    115
                  TERM-13    23  24  25  26     98
                  TERM-20    11  12  13  14     50

                                                360  <== Total for East
```

```
North     23  DISK-45     18  17  15  14     64
              DISK-48     12  14  20  35     81
              DISK-60     10  10  10  11     41

          24  TAPE-12      8   7   6   3     24
              TAPE-13      1   2   4  11     18
              TAPE-32     10  10  10  11     41
              TAPE-33      4   4   4   5     17

                                            286  <== Total for North

South     25  MEMORY-11   19  20  21  21     81
              MEMORY-16   19  18  17  16     70
              MEMORY-17   12  13  13  12     50

                                            201  <== Total for South

West      39  SOFT-12     11  13  13  12     49
              SOFT-14      6   7   8   8     29
              SOFT-23     13  14  20  19     66

          40  SOFT-24     15  14  14  13     56
              SOFT-25      3   3   4   7     17

                                            217  <== Total for West

                                          1,064  <== GRAND TOTAL
```

2.8.1 Creating a New Program

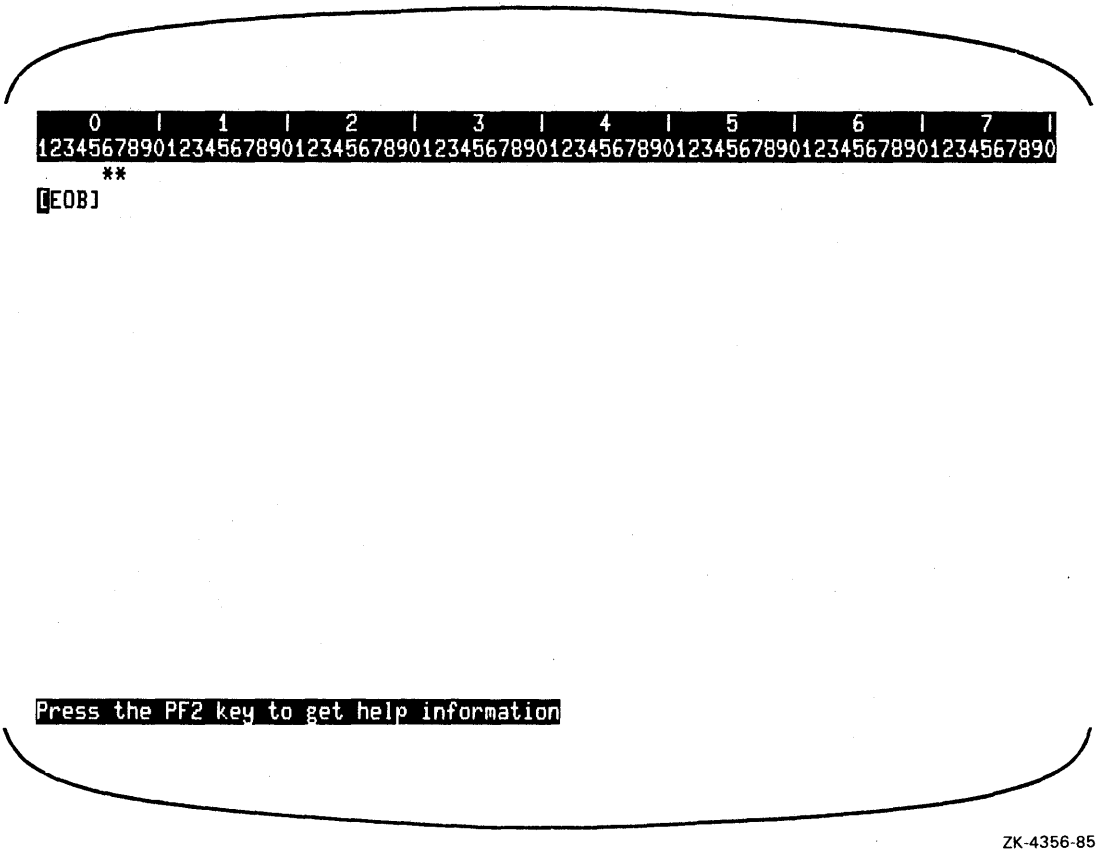
Invoke the VAX RPG II editor by typing the following command:

```
$ RPG/EDIT MYFILE
```

The VAX RPG II editor displays the following message:

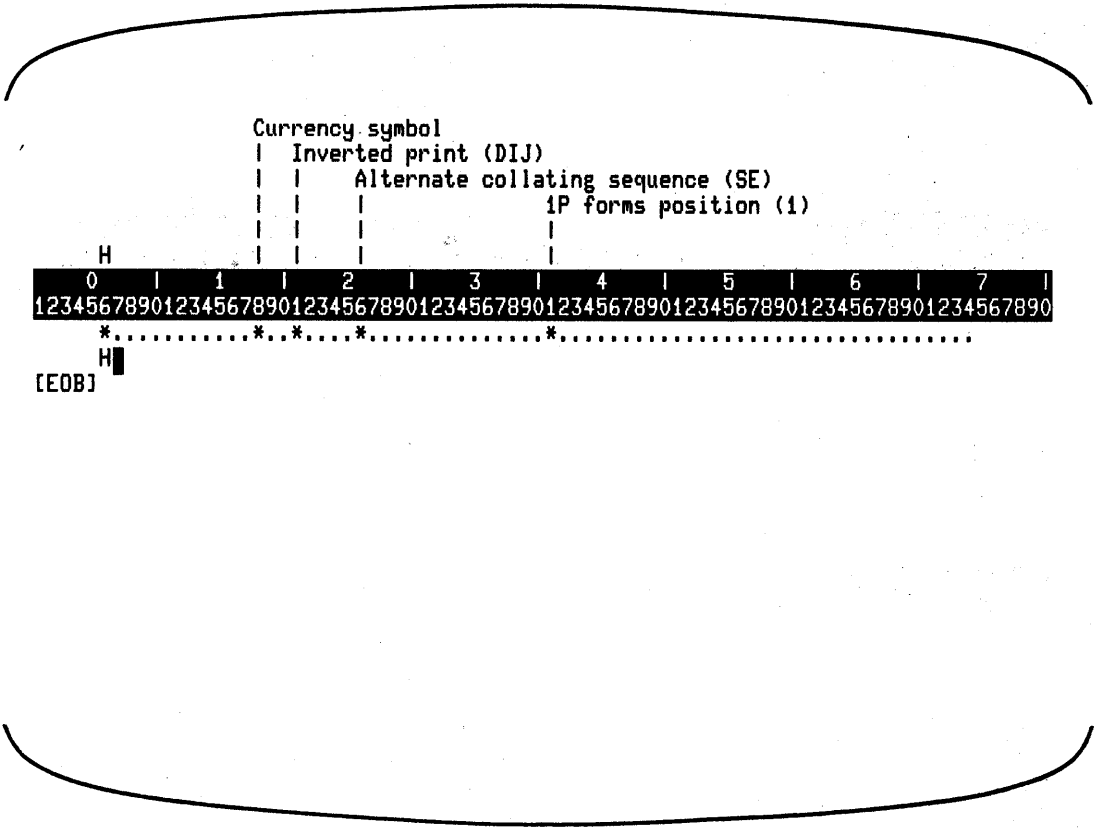
```
File not found
```

The following screen is displayed:



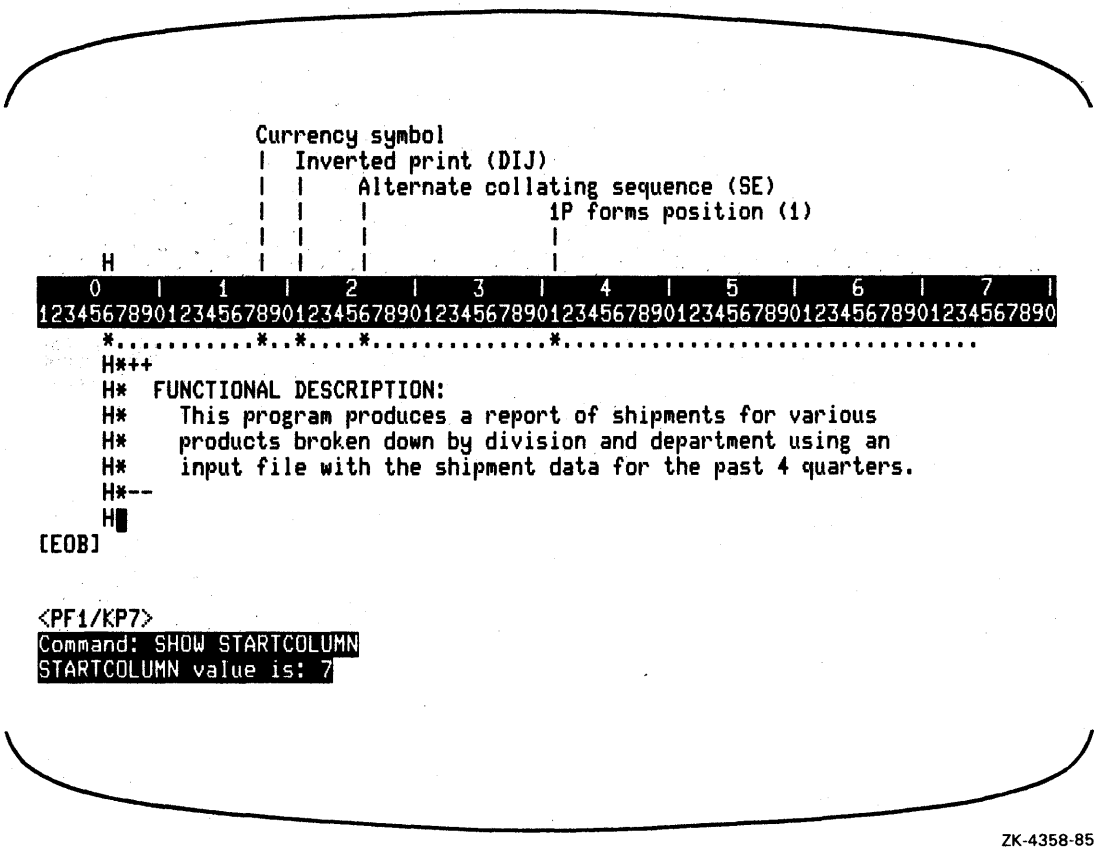
ZK-4356-85

Although VAX RPG II does not require a Control specification (see Chapter 15 for information on all VAX RPG II specifications), it is useful to place an asterisk in column 7 of a Control specification to include a comment describing what your program does. Press either the TAB key or the RIGHT key repeatedly to move the cursor to column 6. Type H in column 6. Use the HELP_SPECS function (default = PF1/PF2) to display the specification format for the Control specification. Because the current line is a Control specification, the VAX RPG II editor automatically displays the tab stops for the Control specification. HELP information will be displayed in the HELP window of the VAX RPG II editor screen, as shown in the following example:



ZK-4357-85

Enter an asterisk (*) in column 7. Type the description of the program. Press the RETURN key at the end of each line. After the RETURN key is pressed, the VAX RPG II editor moves the current line on the screen one line up, if necessary; automatically enters H in column 6; and moves the cursor to column 7. (Column 7 is the default setting for the SET STARTCOLUMN command.) To display the current default for the SET STARTCOLUMN command, issue the COMMAND function (default = PF1/KP7), type SHOW STARTCOLUMN, and press the RETURN key, as shown in the following example:



ZK-4358-85

The specification type for the current line will be duplicated until you enter a new specification type in column 6. H is automatically entered in column 6 and the cursor is moved to column 7. The next specification needed is the File Description specification. To replace H, move the cursor to column 6 by using the FIELD_BACKWARD function (default = BS_KEY), and enter F (File description).

Enter the name of the file, beginning in column 7, and then use the FIELD_FORWARD function, hereafter referred to by its default setting, TAB_KEY. Because HELP information is displayed in the HELP window of the VAX RPG II editor screen, after TAB_KEY or any terminator is pressed the VAX RPG II editor displays the specification format and tab stops for the File Description specification, as shown in the following example:

```
                          Mode (LR)
                          |Key length
               Type (IOUDC) || Record address type (APIB)          Addtn(AU)
               |Des(PSRCTDF)|| |Organization (IT,1-9)              |Expand
               ||EOF (E)    || ||Overflow indicator  Continue (K) ||Shr(SR)
               |||Seq (AD)  || ||| Key location      |Opt   Entry ||| Rewnd
      File     ||||Fmt (FV) || ||| |   Extension (EL)||     |     ||| |
      name     |||||Blk Rec || ||| |   |Device Symb  Tape   Core  ||| |File
       |       ||||||len len || ||| |   |code   dev   label  index ||| |cond
     F|        ||||||   |   || ||| |   ||      |     |      |     ||| ||
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       ******---*---**-*** *---**      *     *......*-----***.** ..
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   
[EOB]
```

ZK-4359-85

Note that you can press TAB_KEY to move the cursor to the next tab stop. Enter the rest of the File Description specifications as shown in the following example:

```
                          Mode (LR)
                          |Key length
                Type (IOUDC) || Record address type (APIB)          Addtn(AU)
                |Des(PSRCTDF)|| |Organization (IT,1-9)              |Expand
                ||EOF (E)    || ||Overflow indicator  Continue (K) ||Shr(SR)
                |||Seq (AD)  || ||| Key location      |Opt   Entry |||Rewnd
      File      ||||Fmt (FV) || ||| |   Extension (EL)||     |     |||  |
      name      |||||Blk Rec || ||| |   |Device Symb  Tape   Core  |||  |File
       |        ||||||len len || ||| |   |code   dev   label  index |||  |cond
     F|         ||||||   |   || ||| |   ||     |     |      |     |||  ||
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       *******---*---**-*** *---**      *     *......*-----***.** ..
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     FSHIPS   IP  F      41            DISK
     FSUMREP  O   F      98          LPRINTER
     F
[EOB]
```

ZK-4360-85

Replace F in column 6 with E (Extension) and press TAB_KEY. The VAX RPG II editor displays the specification format and tab stops for the Extension specification. Then, enter the rest of the entries for the Extension specification, as shown in the following example:

```
                                          ------F = Format (PB)
                                         | -----D = Decimal positions
                                         || ----S = Sequence (AD)
                                         |||
                                         |||Alternating table or array
          From     To       Table EntEnt Len|||name  Len
          file     file     or    perin  of F|||     of F
          name     name     array RecTbl Ent|D||     Ent|D
          |        |        name  |  |   |  ||S|     |  ||S
     E    |        |        |     |  |   |  ||||     |  |||+-- Comments ---+
    0    |    1   |    2   |    3   |    4   |    5   |    6   |    7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     *....*        *        *      *--*---*--****     *--****
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     FSHIPS   IP  F       41            DISK
     FSUMREP  O   F       98           LPRINTER
     E                     QTY         4  2 0
     E
[EOB]
```

ZK-4361-85

Enter L (Line Counter) in column 6. Then, enter the rest of the entries for the Line Counter specification, as shown in the following example:

```
                Form length (1-112)
      File      |  FL (if Form length used)
      name      |  | Overflow line number (1-112)
       |        |  |  |  OL (if Overflow line used)
     L|         |  |  |  |
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         *--* *--* ..............................................
     H*     products broken down by division and department using an
     H*     input file with the shipment data for the past 4 quarters.
     H*--
     FSHIPS    IP  F       41             DISK
     FSUMREP   O   F       98           LPRINTER
     E                      QTY          4  2 0
     LSUMREP    55FL 500L
     L
[EOB]
```

ZK-4362-85

Enter the Input specifications, as shown in the following example:

```
            Sequence (AA-ZZ, 01-99)
            | Number (1-N)
            | |Optional/External (OU)            Decimal positions
            | ||Record identifying indicator     |      Control level
            | |||                                |      | Match field
            | ||| + Identifying codes + Format   |      | | Fld rec rel
     File   | ||| |                     | | (PB)   |Field | | |
     name   | ||| |    C      C      C  | |Field   |name  | | | Field
      |     | ||| |    Z      Z      Z  | ||location||      | | | indicatrs
     I|     | ||| Pos NDcPos NDcPos NDc |Fr  To  ||      | | | + - 0
   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890
     **     * *** *---   *---   *---   .**---*---**     * * * * * * ....
     E                  QTY         4  2 0
     LSUMREP   55FL 500L
     ISHIPS    AA  01
     I                                        1   5 DIV   L2
     I                                        6   7 DEPT  L1
     I                                        8  16 PROD
     I                                       17  24 QTY
     I
[EOB]
```

ZK-4363-85

Use the DISPLAY function (default = PF1/KP8) to display the program on the entire screen (except lines 1 through 3, 23, and 24), as shown in the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        * *** *---    *---    *---    .**---*---**      * * * * * * ....
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*     This program produces a report of shipments for various
     H*     products broken down by division and department using an
     H*     input file with the shipment data for the past 4 quarters.
     H*--
     FSHIPS   IP  F      41            DISK
     FSUMREP  O   F      98           LPRINTER
     E                    QTY         4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                        1   5 DIV   L2
     I                                        6   7 DEPT  L1
     I                                        8  16 PROD
     I                                       17  24 QTY
     I
[EOB]
```

ZK-4364-85

Enter the Calculation specifications without displaying the specification format in the HELP window, as shown in the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *         *          *    *         *     *--*** * * *
     H*--
     FSHIPS   IP  F       41            DISK
     FSUMREP  O   F       98           LPRINTER
     E                     QTY         4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                          1   5 DIV   L2
     I                                          6   7 DEPT  L1
     I                                          8  16 PROD
     I                                         17  24 QTY
     C*
     C   01               XFOOTQTY       PROQTY  30
     C   01       PROQTY  ADD  DEPQTY    DEPQTY  30
     C*
     CL1          DEPQTY  ADD  DIVQTY    DIVQTY  30
     CL1                  Z-ADD0         DEPQTY
     CL2          DIVQTY  ADD  FINQTY    FINQTY  40
     C*
     C
[EOB]
```

ZK-4365-85

Enter the Output specifications as shown in the following example. Note that the VAX RPG II editor screen can display only 19 source lines at a time when the terminal has 24 lines and when the ruler is displayed. After you enter more than 19 lines, the VAX RPG II editor moves the editing window up.

```
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        ***** * *        *      ***---**                          ....
     OSUMREP   H  001    1P
     O                                        48 'PRODUCT SHIPMENT REPORT'
     O         H 02       1P
     O                             UDATE Y    12
     O                                        48 'PRODUCT SHIPMENT REPORT'
     O         H  1       1P
     O                                        42 'SHIPMENTS'
     O         H  2       1P
     O                                        15 'DIVISION  DEPT'
     O                                        24 'PRODUCT'
     O                                        48 'Q1  Q2  Q3  Q4  TOTAL'
     O         D  1       01
     O                             DIV         8
     O                   L1        DEPT       14
     O                             PROD       25
     O                             QTY    Z   41
     O                             PROQTYZ    48
     O         T  1       L1
     O
```

ZK-4366-85

Enter the rest of the Output specifications. Use the EXIT function (default = CTRL_Z_KEY) to save the contents of the editing buffer and to leave the VAX RPG II editor. When EXIT is used, the VAX RPG II editor displays the following message as shown in the next example:

```
45 records written to file MYDISK:[MYDIRECTORY]MYFILE.RPG;1
```

```
    0   |    1   |    2   |    3   |    4   |    5   |    6   |    7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       ***** * *       *     ***---**                          ....
     O        H 02     1P
     O                          UDATE Y   12
     O                                    48 'PRODUCT SHIPMENT REPORT'
     O        H  1     1P
     O                                    42 'SHIPMENTS'
     O        H  2     1P
     O                                    15 'DIVISION  DEPT'
     O                                    24 'PRODUCT'
     O                                    48 'Q1  Q2  Q3  Q4  TOTAL'
     O        D  1     01
     O                          DIV        8
     O                 L1       DEPT      14
     O                          PROD      25
     O                          QTY   Z   41
     O                          PROQTYZ   48
     O        T  1     L1
     O        T  0     LR
     O                          FINQTY1   48
     O                                    65 '<== GRAND TOTAL'

45 records written to file MYDISK:[MYDIRECTORY]MYFILE.RPG;1

$
```

ZK-4367-85

2.8.2 Editing an Existing Program

When you invoke the VAX RPG II editor to edit the program created in Section 2.8.1, the VAX RPG II editor displays the following message:

```
45 records read from file MYDISK:[MYDIRECTORY]MYFILE.RPG;1
```

The following screen is displayed:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *...........*..*....*.............*.................................
#    H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41           DISK
     FSUMREP  O   F      98          LPRINTER
     E                    QTY        4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                        1   5 DIV   L2
     I                                        6   7 DEPT  L1
     I                                        8  16 PROD
     I                                       17  24 QTY
     C*
     C   01               XFOOTQTY       PROQTY  30
     C   01      PROQTY   ADD  DEPQTY    DEPQTY  30

Press the PF2 key to get help information
```

ZK-4368-85

In this session, the control-level indicator L2 must condition the DIV field in the detail record Output specification. Use the FIND function (default = PF1/PF3) to locate DIV. The VAX RPG II editor displays the command prompt "Search for: ". Enter the search string DIV and press the ENTER key, as shown in the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *...........*..*....*.............*..............................
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41            DISK
     FSUMREP  O   F      98           LPRINTER
     E                    QTY         4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                          1   5 DIV   L2
     I                                          6   7 DEPT  L1
     I                                          8  16 PROD
     I                                         17  24 QTY
     C*
     C   01               XFOOTQTY       PROQTY  30
     C   01      PROQTY   ADD  DEPQTY    DEPQTY  30
Search for: DIV
```

ZK-4369-85

The VAX RPG II editor responds by moving the cursor to the first character of the first occurrence of the search string DIV (see the comment line), as shown in the following example:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *...........*..*....*.............*................................
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41            DISK
     FSUMREP  O   F      98           LPRINTER
     E                    QTY         4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                          1   5 DIV   L2
     I                                          6   7 DEPT  L1
     I                                          8  16 PROD
     I                                         17  24 QTY
     C*
     C   01               XFOOTQTY       PROQTY  30
     C   01      PROQTY   ADD  DEPQTY    DEPQTY  30
```

ZK-4370-85

Because this is not the correct string, use the FIND_NEXT function (default = PF3). The VAX RPG II editor moves the cursor to the first character of the next occurrence of the string DIV, as shown in the following example:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **       * *** *---   *---   *---   .**---*---**     * * * * * * ....
     H*++
     H*  FUNCTIONAL DESCRIPTION:
     H*    This program produces a report of shipments for various
     H*    products broken down by division and department using an
     H*    input file with the shipment data for the past 4 quarters.
     H*--
     H
     FSHIPS   IP  F      41            DISK
     FSUMREP  O   F      98           LPRINTER
     E                    QTY         4  2 0
     LSUMREP   55FL 500L
     ISHIPS   AA  01
     I                                          1   5 DIV   L2
     I                                          6   7 DEPT  L1
     I                                          8  16 PROD
     I                                         17  24 QTY
     C*
     C   01               XFOOTQTY       PROQTY  30
     C   01      PROQTY   ADD  DEPQTY    DEPQTY  30
```

ZK-4371-85

Again, this occurrence of the string DIV is incorrect, so issue the FIND_NEXT function five more times to move the cursor to the correct occurrence. You could have specified DIV and a blank as the search string to avoid duplicating key strokes. L2 must be entered in columns 24 and 25. To do this, move the cursor to column 24 by pressing BS_KEY to column 23, then use the RIGHT function (default = RIGHT) once. Enter the string L2 in columns 24 and 25, as shown in the following example:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        ***** * *         *      ***---**                          ....
     C   01      PROQTY    ADD  DEPQTY    DEPQTY  30
     C*
     CL1         DEPQTY    ADD  DIVQTY    DIVQTY  30
     CL1                   Z-ADD0         DEPQTY
     CL2         DIVQTY    ADD  FINQTY    FINQTY  40
     C*
     OSUMREP  H  001   1P
     O                                    48 'PRODUCT SHIPMENT REPORT'
     O        H 02     1P
     O                        UDATE Y     12
     O                                    48 'PRODUCT SHIPMENT REPORT'
     O        H  1     1P
     O                                    42 'SHIPMENTS'
     O        H  2     1P
     O                                    15 'DIVISION  DEPT'
     O                                    24 'PRODUCT'
     O                                    48 'Q1  Q2  Q3  Q4  TOTAL'
     O        D  1     01
     O                 L2     DIV         8
```

ZK-4372-85

Number the program lines for reference by issuing the COMMAND function (default = PF1/KP7) and typing the RESEQUENCE command, as shown in the following example:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *          *      ***---**                          ....
  190C   01       PROQTY    ADD  DEPQTY    DEPQTY  30
  200C*
  210CL1           DEPQTY    ADD  DIVQTY    DIVQTY  30
  220CL1                     Z-ADDO         DEPQTY
  230CL2           DIVQTY    ADD  FINQTY    FINQTY  40
  240C*
  250OSUMREP  H  001   1P
  2600                                     48 'PRODUCT SHIPMENT REPORT'
  2700        H 02     1P
  2800                        UDATE Y      12
  2900                                     48 'PRODUCT SHIPMENT REPORT'
  3000        H  1     1P
  3100                                     42 'SHIPMENTS'
  3200        H  2     1P
  3300                                     15 'DIVISION  DEPT'
  3400                                     24 'PRODUCT'
  3500                                     48 'Q1  Q2  Q3  Q4  TOTAL'
  3600        D  1     01
  3700                 L2#       DIV        8
Command: RESEQUENCE
```

ZK-4373-85

Use the SECTION function (default = KP8) to move the cursor the number of lines set by the SET SECTION command, as shown in the following example:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
    **       ***** * *       *     ***---**                              ....
  2800                                UDATE Y   12
  2900                                          48 'PRODUCT SHIPMENT REPORT'
  3000        H  1     1P
  3100                                          42 'SHIPMENTS'
  3200        H  2     1P
  3300                                          15 'DIVISION  DEPT'
  3400                                          24 'PRODUCT'
  3500                                          48 'Q1  Q2  Q3  Q4  TOTAL'
  3600        D  1     01
  3700                 L2             DIV        8
  3800                 L1             DEPT      14
  3900                                PROD      25
  4000                                QTY   Z   41
  4100                                PROQTYZ   48
  4200        T  1     L1
  4300        T  0     LR
  4400                                FINQTY1   48
  4500                                          65 '<== GRAND TOTAL'
[EOB]

Attempt to move past end of buffer
```

ZK-4374-85

Enter two Output specifications between lines 420 and 430 by using the following functions:

1. UP (default = UP) to line 430
2. OPEN_LINE (default = PF1/KP0) to create a new line

Use the OPEN_LINE function to create a line preceding the current line. The VAX RPG II editor automatically places the specification type of the current line in column 6 and moves the cursor to column 7. Enter the new specifications, as shown in the following example:

```
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
    **       ***** * *       *     ***---**                              ....
2900                                      48 'PRODUCT SHIPMENT REPORT'
3000        H  1     1P
3100                                      42 'SHIPMENTS'
3200        H  2     1P
3300                                      15 'DIVISION  DEPT'
3400                                      24 'PRODUCT'
3500                                      48 'Q1  Q2  Q3  Q4  TOTAL'
3600        D  1     01
3700                 L2        DIV         8
3800                 L1        DEPT       14
3900                           PROD       25
4000                           QTY   Z    41
4100                           PROQTYZ    48
4200        T  1     L1
   0        T  0     L2
   0                           DIV        69
4300        T  0     LR
4400                           FINQTY1    48
4500                                      65 '<== GRAND TOTAL'
```

ZK-4375-85

Enter two more specifications (identical to the two specifications just entered) by using the following functions:

1. SELECT (default = PERIOD) to mark the beginning of the selected region
2. UP (default = UP) once
3. CUT (default = KP6) to place the selected region into the paste buffer
4. PASTE (default = PF1/KP6) twice

The following example shows the effects of the procedure just described:

```
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
   **       ***** * *        *      ***---**                              ....
2900                                          48 'PRODUCT SHIPMENT REPORT'
3000         H  1     1P
3100                                          42 'SHIPMENTS'
3200         H  2     1P
3300                                          15 'DIVISION  DEPT'
3400                                          24 'PRODUCT'
3500                                          48 'Q1  Q2  Q3  Q4  TOTAL'
3600         D  1     01
3700                  L2       DIV            8
3800                  L1       DEPT          14
3900                           PROD          25
4000                           QTY   Z       41
4100                           PROQTYZ       48
4200         T  1     L1
   0         T  0     L2
   0                           DIV           69
   0         T  0     L2
   0                           DIV           69
4300         T  0     LR
```

ZK-4376-85

Enter four more specifications. Then, remove the line numbers, as shown in the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        ***** * *        *     ***---**                            ....
     O         H  2     1P
     O                                          15 'DIVISION  DEPT'
     O                                          24 'PRODUCT'
     O                                          48 'Q1  Q2  Q3  Q4  TOTAL'
     O         D  1     01
     O                  L2       DIV            8
     O                  L1       DEPT          14
     O                           PROD          25
     O                           QTY   Z       41
     O                           PROQTYZ       48
     O         T  1     L1
     O         T  0     L2
     O                           DIV           69
     O         T  0     L2
     O                           DIV           69
     O         T 02     L2
     O                           DIVQTYZB      48
     O                                         63 '<== Total for'
     O                           DIV           69
Command: RESEQUENCE/REMOVE
```

ZK-4377-85

Use the COMMAND function and type the EXIT command to save the modified program, as shown in the following example:

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **       ***** * *       *     ***---**                                  ....
     0        H  2     1P
     0                                      15 'DIVISION  DEPT'
     0                                      24 'PRODUCT'
     0                                      48 'Q1  Q2  Q3  Q4  TOTAL'
     0        D  1     01
     0                 L2       DIV          8
     0                 L1       DEPT        14
     0                          PROD        25
     0                          QTY   Z     41
     0                          PROQTYZ     48
     0        T  1     L1
     0        T  0     L2
     0                          DIV         69
     0        T  0     L2
     0                          DIV         69
     0        T 02     L2
     0                          DIVQTYZB    48
     0                                      63 '<== Total for'
     0                          DIV         69
Command: EXIT
53 records written to file MYDISK:[MYDIRECTORY]MYFILE.RPG;2

$
```

ZK-4378-85

Chapter 3

Processing VAX RPG II Programs

You can create a source program using the VAX RPG II editor; then, you must compile, link, and run the program with commands to the VAX/VMS operating system. You can optionally create an object file before leaving the VAX RPG II editor using the VAX RPG II editor COMPILE command. If your VAX RPG II program does not execute correctly, you must modify it and repeat these steps until it does.

When you compile a VAX RPG II program, the VAX RPG II compiler creates an object module file. When you link your program, use the VAX/VMS Linker. The linker reads the object module file and uses libraries to replace external references with the address of the executable code that defines it. Then the linker places that code in an executable image file. When you execute your program, the system executes that image.

3.1 Compiling Programs

To compile a source program, use the RPG command as follows.

```
$ RPG[/qualifier(s)] file-spec-list[/qualifier(s)]
```

/qualifier(s)
Specifies special actions the compiler is to perform. See Sections 3.1.2.1 through 3.1.2.8 for information on qualifiers.

file-spec-list

Specifies the source files to be compiled. Normally, you would specify a single source file, but if you need to create a single object file from more than one source file, separate the file specifications with plus (+) signs. VAX RPG II appends the files in the order you specify. If you separate source file specifications with commas, VAX RPG II compiles the programs separately and creates a single object file for each source file.

When you execute the RPG command, VAX RPG II compiles the program and generates an object module with the specified file name and the default file type OBJ. The compiler can also generate other output files, depending on the qualifiers you supply.

When you compile a source file and specify only its file name, the compiler searches for a source file with the specified name that is stored on the default device in the default directory and has a file type of RPG.

If more than one file meets these conditions, the compiler chooses the one with the highest version number.

For example, assume that your default device is DBA0:, your default directory is [SMITH], and you give this command:

```
$ RPG FIRSTTRY
$
```

The appearance of the second DCL command prompt ($) indicates that the compilation is finished.

The compiler searches device DBA0: in directory [SMITH], seeking the highest version of FIRSTTRY.RPG. If you do not specify an output file, the compiler generates the file FIRSTTRY.OBJ and stores it on device DBA0: in directory [SMITH], with a version number that is one higher than any existing version number for FIRSTTRY.OBJ.

3.1.1 Default Compiler Options

When you compile a program, you can specify optional qualifiers such as /LIST or /NOWARNINGS. The qualifiers you get when you do not specify them are called defaults.

You can change these defaults for your own programs by using qualifiers with the RPG command. The RPG command accepts qualifiers to change the defaults for a single compilation, as shown in the following example:

```
$ RPG/LIST/NOOBJECT MYPROG
```

This RPG command instructs VAX RPG II to compile a single source file (MYPROG.RPG) and overrides the default compiler settings for:

- Listing—the VAX RPG II compiler will produce a compiler listing.
- Object file—the VAX RPG II compiler will not produce an object file.

You can specify other defaults by defining RPG as a symbol, as shown in the following example:

```
$ RPG :== "RPG/CHECK/LIST/CROSS"
```

If you then type RPG MYPROG, the /CHECK, /LIST, and /CROSS qualifiers are in effect.

3.1.2 Compiler Qualifiers

This section describes the RPG command; Sections 3.1.2.1 through 3.1.2.8 describe the RPG command qualifiers and list their default values.

You can change defaults by using qualifiers with the RPG command. Qualifiers have the form:

```
/qualifier[=value]
```

Many qualifiers have a corresponding form that negates the action specified by the qualifier. The negative form is as follows:

```
/NOqualifier
```

For example, /LIST tells the compiler to produce a listing file; /NOLIST suppresses the listing.

You can specify qualifiers so that they affect either all files in the command, or only certain files. If the qualifier immediately follows the RPG command, it applies to all files, as shown in the following example:

```
$ RPG/LIST ABC,XYZ,RST
```

This command specifies listing files for ABC.RPG, XYZ.RPG, and RST.RPG.

Qualifiers following a file specification (with some exceptions) affect only the associated file, as shown in the following example:

```
$ RPG/LIST ABC,XYZ/NOLIST,RST
```

The preceding RPG command specifies listing files for ABC.RPG and RST.RPG, but not for XYZ.RPG. Qualifiers to a single file specification in an appended list of file specifications are exceptions to this rule. (A list of file specifications separated by plus signs (+) is called an appended list or plus list.) See Example 5 in the following list.

1. `$ RPG/LIST AAA,BBB,CCC`

   VAX RPG II compiles source files AAA.RPG, BBB.RPG, and CCC.RPG as separate files and produces three object files (AAA.OBJ, BBB.OBJ, and CCC.OBJ) and three listing files (AAA.LIS, BBB.LIS, and CCC.LIS).

2. `$ RPG XXX+YYY+ZZZ`

   VAX RPG II appends source files XXX.RPG, YYY.RPG, and ZZZ.RPG and compiles them as a single program. This command produces one object file named XXX.OBJ, but does not produce a listing file.

3. `$ RPG/OBJECT=SQUARE CIRCLE`

   VAX RPG II compiles source file CIRCLE.RPG and produces object file SQUARE.OBJ. This command produces no listing file.

4. `$ RPG AAA+BBB,CCC/LIST`

   VAX RPG II produces two object files: AAA.OBJ (created from AAA.RPG and BBB.RPG) and CCC.OBJ (created from CCC.RPG). VAX RPG II also produces the listing file CCC.LIS.

5. `$ RPG ABC+DEF/NOOBJECT+XYZ`

   VAX RPG II appends and compiles the source files ABC.RPG, DEF.RPG, and XYZ.RPG. Because qualifiers in a list of appended files affect all files in the list, this command suppresses the creation of an object file.

Table 3–1 lists the qualifiers you can use with the RPG command.

Table 3–1: VAX RPG II Command Qualifiers

Qualifier	Negative Form	Default
/CHECK=[NO]BOUNDS [NO]RECURSION [NO]BLANKS_IN_NUMERICS ALL NONE	/NOCHECK	/NOCHECK
/CROSS_REFERENCE	/NOCROSS_REFERENCE	/NOCROSS_REFERENCE
/DEBUG=[NO]SYMBOLS [NO]TRACEBACK ALL NONE	/NODEBUG	/DEBUG= (NOSYMBOLS, TRACEBACK)
/LIST[=file-spec]	/NOLIST	/NOLIST (interactive) /LIST (batch)
/MACHINE_CODE	/NOMACHINE_CODE	/NOMACHINE_CODE
/OBJECT[=file-spec]	/NOOBJECT	/OBJECT
/SEQUENCE_CHECK	/NOSEQUENCE_CHECK	/NOSEQUENCE_CHECK
/WARNINGS=[NO]OTHER [NO]INFORMATION ALL NONE	/NOWARNINGS	/WARNINGS= (OTHER, NOINFORMATION)

Sections 3.1.2.1 through 3.1.2.8 describe VAX RPG II command qualifiers in detail.

3.1.2.1 /CHECK Qualifier

The /CHECK qualifier causes VAX RPG II to generate code to check for run-time errors in array indexes, recursive calls to subroutines, and blanks in overpunched numeric fields. The /CHECK qualifier format is as follows:

```
/CHECK[=(option[,...])]
```

option

Can be one of the following:

- [NO]BOUNDS
- [NO]RECURSION
- [NO]BLANKS_IN_NUMERICS
- ALL
- NONE

BOUNDS

Checks array indexes at run time to make sure they are within array boundaries specified by the program.

RECURSION

Verifies at run time that subroutines are not called recursively.

BLANKS_IN_NUMERICS

Converts blanks in overpunched numeric fields to zeros at run time. This option is the default if the program contains a WORKSTN file. Use the RPG/CHECK=BLANKS_IN_NUMERICS command to convert blanks in numeric data to zeros if you run your program and receive the following message:

```
A numeric field contains invalid data
```

ALL

Indicates that RECURSION, BOUNDS, and BLANKS_IN_NUMERICS checking will be performed.

NONE

Indicates that RECURSION, BOUNDS, and BLANKS_IN_NUMERICS checking will not be performed.

Specifying the /CHECK qualifier is equivalent to specifying /CHECK=ALL; /NOCHECK is equivalent to /CHECK=NONE. The /NOCHECK qualifier is the default.

Use /CHECK=(RECURSION,BOUNDS) for programs only during initial program debugging, because compiling with this qualifier results in additional code and, consequently, takes more time to process. Using /NOCHECK means that no error will be signaled at run time for an array reference outside the bounds of an array or for a subroutine that has been called recursively. Therefore, using /NOCHECK may result in your program getting a memory-management or access-violation error at run time.

3.1.2.2 /CROSS_REFERENCE Qualifier

The /CROSS_REFERENCE qualifier causes the compiler to include cross-reference information in the listing file for the compiled source file. Cross-reference information lists variable names, indicators, and the program lines on which they were referenced. Its format is as follows:

```
/CROSS_REFERENCE
```

When you use the /CROSS_REFERENCE qualifier, you must also use the /LIST qualifier, or /LIST must be in effect (default for batch mode) to produce a listing file. The /NOCROSS_REFERENCE qualifier is the default.

3.1.2.3 /DEBUG Qualifier

The /DEBUG qualifier causes the compiler to provide information for the VAX/VMS Debugger and the system run-time error traceback mechanism. Its format is as follows:

```
/DEBUG[=(option[,...])]
```

option

Can be one of the following:

- [NO]SYMBOLS
- [NO]TRACEBACK
- ALL
- NONE

SYMBOLS
Causes the compiler to provide the debugger with local symbol definitions for user-defined names (including dimension information for arrays). If you use SYMBOLS, you can refer to data entities by their names when you use the debugger.

TRACEBACK
Causes the compiler to provide an address correlation table so that the debugger and the run-time error traceback mechanism can translate absolute addresses into source program routine names and line numbers.

ALL
Causes the compiler to provide both local symbol definitions and an address correlation table.

NONE
Prevents the compiler from providing debugging information.

Neither the /TRACEBACK qualifier nor the /SYMBOLS qualifier affects a program's executable code.

Specifying the /DEBUG qualifier is equivalent to specifying /DEBUG=ALL; /NODEBUG is equivalent to /DEBUG=NONE. The /DEBUG=TRACEBACK qualifier is the default. For information on debugging, see Chapter 5.

3.1.2.4 /LIST Qualifier

The /LIST qualifier controls whether VAX RPG II produces a listing file for the compiled program. The listing file contains the source program and a compilation summary. If you also use the /MACHINE_CODE qualifier, the listing file will include the compiler-generated object code for the compiled program. If you also use the /CROSS_REFERENCE qualifier, the listing file will include cross-reference information. The format of the /LIST qualifier is as follows:

```
/LIST[=file-spec]
```

You can include a file specification for the listing file. Otherwise, the output file defaults to the name of the first source file and the file type LIS.

If the RPG command is executed in interactive mode, the default qualifier is /NOLIST. If the RPG command is executed in batch mode, the default qualifier is /LIST.

The listing file uses a listing page length which depends on the logical SYS$LP_LINES. Any value between 30 and 255 can be used for SYS$LP_LINES. The listing page length uses three-line top and bottom margins. If the logical SYS$LP_LINES is not defined, the default page length will be 66 lines (60 listing lines after the three-line top and bottom margins are subtracted).

It is possible to enter nonprintable characters when using an editor other than the VAX RPG II editor to create or edit a program. If a source line in the compiler listing contains one or more periods (.) where you have not entered a period on the program line, it is because the program line contains a nonprintable character (for example, a TAB character or a null character).

3.1.2.5 /MACHINE_CODE Qualifier

The /MACHINE_CODE qualifier specifies that the listing file include the compiler-generated object code. Its format is as follows:

```
/MACHINE_CODE
```

When you use the /MACHINE_CODE qualifier, you must also use the /LIST qualifier, or /LIST must be in effect (default for batch mode) to produce a listing file. The /NOMACHINE_CODE qualifier is the default.

3.1.2.6 /OBJECT Qualifier

The /OBJECT qualifier causes VAX RPG II to produce an object module and optionally specifies its file name. Its format is as follows:

```
/OBJECT[=file-spec]
```

The default qualifier is /OBJECT.

By default, the compiler generates object files as follows:

- If you specify one source file, VAX RPG II generates one object file.
- If you specify multiple source files separated by plus signs (+), VAX RPG II appends the files and generates one object file.
- If you specify multiple source files separated by commas (,), VAX RPG II compiles and generates a separate object file for each source file.

You can use both plus signs (+) and commas (,) in the same command line to produce different combinations of appended and separated object files. See Examples 1 through 5 in Section 3.1.2.

To produce an object file with an explicit file specification, you must use the /OBJECT qualifier in the form /OBJECT=file-spec. Otherwise, the object file has the same name as its corresponding source file and the default file type OBJ. By default, the object file produced from appended source files has the name of the first source file specified. All other file specification attributes (node, device, directory, and version number) assume the default values.

During the early stages of program development, you may find it useful to suppress the production of object files until your source program compiles without errors. Use the /NOOBJECT qualifier to do this.

3.1.2.7 /SEQUENCE_CHECK Qualifier

The /SEQUENCE_CHECK qualifier causes the compiler to check the line numbers in columns 1 through 5 of every program line to make sure they are in ascending line-number sequence. If the line numbers are not in sequence, the compiler issues a warning message. Its format is as follows:

```
/SEQUENCE_CHECK
```

The /NOSEQUENCE_CHECK qualifier is the default.

3.1.2.8 /WARNINGS Qualifier

The /WARNINGS qualifier allows you to specify whether VAX RPG II displays informational and warning messages. Its format is as follows:

```
/WARNINGS[=(option[,...])]
```

option
Can be one of the following:

- [NO]OTHER
- [NO]INFORMATION
- ALL
- NONE

OTHER
Causes VAX RPG II to display warning messages.

INFORMATION
Causes VAX RPG II to display informational messages.

ALL
Causes VAX RPG II to display both warning and informational messages.

NONE
Prevents VAX RPG II from displaying warning or informational messages.

Specifying the /WARNINGS qualifier is equivalent to specifying /WARNINGS=ALL; /NOWARNINGS is equivalent to /WARNINGS=NONE. The /WARNINGS= (NOINFORMATION,OTHER) qualifier is the default.

For information on how /WARNINGS affects the VAX RPG II editor SYNTAXCHECK option, see also Section 2.6.8.8.

3.2 Linking and Running Programs

The VAX/VMS Linker uses the object module produced by the VAX RPG II compiler as input and produces an executable image file as output. This file has the same name as your program and the default file type EXE.

When your program calls other programs—that is, when it is comprised of more than one program module—the linker takes multiple object files and creates a single executable image from them. For information on subprograms, see Section 12.6.

You use the LINK command to invoke the VAX/VMS Linker. The format of the LINK command is as follows:

```
LINK[/command-qualifier(s)] file-spec-list[/file-qualifier(s)]
```

command-qualifier(s)
Specifies output file options. Use the /DEBUG qualifier to provide information for the VAX/VMS Debugger. See Chapter 5 for information on debugging VAX RPG II programs. For information about other command qualifiers, see the *VAX/VMS Linker Reference Manual.*

file-spec-list
Specifies a file or the files to be linked.

file-qualifier(s)
Specifies input file options. For information on file qualifiers, see the *VAX/VMS Linker Reference Manual.*

When you type LINK, the system prompts with:

```
_File:
```

Respond by typing the file specifications. If multiple file specifications do not fit on a single line, type a hyphen (-) as the last character on the line and continue on the next line.

For example, to link the object file created from the program FIRSTTRY in Section 3.1, type:

```
$ LINK FIRSTTRY
$
```

NOTE

If you link a program with a WORKSTN file and get a message that FDV$ATERM is undefined, you are attempting to link on a system that does not have the VAX Forms Management System (VAX FMS) installed.

This command tells the linker to accept FIRSTTRY.OBJ as input and to produce FIRSTTRY.EXE as output. After the executable file has been created, you run it with the RUN command:

```
$ RUN FIRSTTRY
$
```

NOTE

If you run a program with a WORKSTN file and get a message that there was an error activating FDVSHR, you are attempting to run a WORKSTN program on a system that does not have the VAX FMS Form Driver installed.

3.3 Interpreting Compiler Error Messages

The format of a VAX RPG II compiler error message is as follows:

```
fac-severity-IDENT
```

fac
Represents the facility. The facility is always RPG.

severity
Indicates the severity of the error, which can be I (information), W (warning), E (error), or F (fatal).

IDENT
Represents the IDENT field.

The IDENT field of a VAX RPG II compiler error message designates the error recovery action taken by the VAX RPG II compiler. IDENT fields can have one of the following values:

- SPEC_IGNORED

  The current specification is ignored. The resulting program, if nonfatal, executes as if the specification were not entered.

- ENTRY_IGNORED

  The entry in the current field is ignored. The resulting program, if nonfatal, executes as if the field were blank.

- DEFN_IGNORED

  The current definition of this field is ignored. The resulting program, if nonfatal, uses the previous definition.

- CHAR_IGNORED

  The current character is ignored. The resulting program, if nonfatal, executes as if the column were blank.

- FATAL

  No error recovery action can be taken. The severity level is always fatal.

- ACCEPTED

  The compiler accepts the entry exactly as specified.

- SEE_MESSAGE

  The error text contains the recovery action taken by the VAX RPG II compiler.

- 0_ASSUMED

  The entry in the current field is ignored. The resulting program, if nonfatal, executes as if the field contained 0.

Chapter 4

Interpreting a Compiler Listing

This chapter explains the parts of a full compiler listing. The sample listing in Figure 4–1 is for the program shown under the Source Listing title. The circled numbers on the program listing correspond to the numbered key which follows the figure.

Figure 4–1: Sample Compiler Listing

```
SHIPS (1)                                    (2) 28-Jun-1986 15:58:45    VAX RPG II V2.1    (3)          Page    1
Source Listing                               (4) 28-Jun-1986 15:56:12    RPG$:[HEBERT.RPG]SHIPS.RPG;1 (1)
                                                                                  (5)

     (6)        1         2         3         4         5         6         7         8
    12345678901234567890123456789012345678901234567890123456789012345678901234567890
 (7) 1    (8) H*++
   2      H*  FUNCTIONAL DESCRIPTION:
   3      H*     This program produces a report of shipments for various
   4      H*     products broken down by division and department using an
   5      H*     input file with the shipment data for the past 4 quarters.
   6      H*--
   7      H
   8      FSHIPS   IP  F      41           DISK
   9      FSUMREP  O   F      98           LPRINTER
  10      E                    QTY         4  2 0
  11      LSUMREP   55FL 5OOL
  12      ISHIPS   AA  01
  13      I                                        1   5 DIV   L2
  14      I                                        6   7 DEPT  L1
  15      I                                        8  16 PROD
  16      I                                       17  24 QTY
  17      C*
  18      C   01               XFOOTQTY       PROQTY  30
  19      C   01      PROQTY   ADD  DEPQTY    DEPQTY  30
  20      C*
  21      CL1         DEPQTY   ADD  DIVQTY    DIVQTY  30
  22      CL1                  Z-ADDO         DEPQTY
  23      CL2         DIVQTY   ADD  FINQTY    FINQTY  40
  24      C*
```

```
25       OSUMREP  H  001   1P
26       O                                48 'PRODUCT SHIPMENT REPORT'
27       O        H 02    1P
28       O                         UDATE Y  12
29       O                                48 'PRODUCT SHIPMENT REPORT'
30       O        H  1    1P
31       O                                42 'SHIPMENTS'
32       O        H  2    1P
33       O                                15 'DIVISION  DEPT'
34       O                                24 'PRODUCT'
35       O                                48 'Q1  Q2  Q3  Q4  TOTAL'
36       O        D  1    01
37       O                 L2     DIV       8
38       O                 L1     DEPT     14
39       O                        PROD      25
40       O                        QTY   Z   41
41       O                        PROQTYZ   48
42       O        T  1    L1
43       O        T  0    L2
44       O                        DIV       69
45       O        T  0    L2
46       O                        DIV       69
47       O        T 02    L2
48       O                        DIVQTYZB  48
49       O                                63 '<== Total for'
50       O                        DIV       69
51       O        T  0    LR
52       O                        FINQTY1   48
53       O                                65 '<== GRAND TOTAL'
54
```

(Continued on next page)

Figure 4–1 (Cont.): Sample Compiler Listing

```
SHIPS                                                    28-Jun-1986 15:58:45    VAX RPG II V2.1                    Page    2
Machine Code Listing ❾                                   28-Jun-1986 15:56:12    RPG$:[HEBERT.RPG]SHIPS.RPG;1 (1)
00000000          .BYTE   ^X53,^X48,^X49,^X50,^X53                                                    ; "SHIPS"
00000008          .BYTE   ^X53,^X55,^X4D,^X52,^X45,^X50                                               ; "SUMREP"
00000010          .BYTE   ^X47,^X06,^X02,^X40,^X20,^X91,^X03,^X91,^X44,^X2F,^X92,^X44,^X2F,^X92,^X00  ; "G..@ ...D/.D/.."
00000020          .BYTE   ^X47,^X02,^X02,^X40,^X20,^X92,^X00                                          ; "G..@ .."
00000028          .BYTE   ^X47,^X03,^X02,^X40,^X20,^X93,^X00                                          ; "G..@ .."
00000030          .BYTE   ^X47,^X04,^X02,^X40,^X20,^X91,^X44,^X2C,^X92,^X03,^X91,^X00                 ; "G..@ .D....."
0000003C          .BYTE   ^X00,^X00,^X00,^X0C                                                         ; "...."
00000040          .LONG   ^X00000003
00000044          .ADDRESS  UDAY
00000048          .ADDRESS  UMONTH
0000004C          .ADDRESS  UYEAR
00000050          .LONG   ^X00000001
00000054          .ADDRESS  SHIPS+68
00000058          .LONG   ^X00000002
0000005C          .LONG   ^X00000002
00000060          .ADDRESS  SHIPS
00000064          .LONG   ^X00000001
00000068          .ADDRESS  SHIPS
0000006C          .LONG   ^X00000002
00000070          .LONG   ^X00000001
00000074          .ADDRESS  SHIPS
00000078          .LONG   ^X00000001
0000007C          .ADDRESS  SUMREP+68
00000080          .LONG   ^X00000002
00000084          .LONG   ^X00000002
00000088          .ADDRESS  SUMREP
0000008C          .LONG   ^X00000001
00000090          .ADDRESS  SUMREP
00000094          .LONG   ^X00000002
00000098          .LONG   ^X00000001
0000009C          .ADDRESS  SUMREP
```

```
000000A0          .BYTE   ^X50,^X52,^X4F,^X44,^X55,^X43,^X54,^X20,^X53,^X48,^X49,^X50,^X4D,^X45,^X4E,^X54 ; "PRODUCT SHIPMENT"
000000B0          .BYTE   ^X20,^X52,^X45,^X50,^X4F,^X52,^X54                                             ; " REPORT"
000000B8          .BYTE   ^X53,^X48,^X49,^X50,^X4D,^X45,^X4E,^X54,^X53                                   ; "SHIPMENTS"
000000C4          .BYTE   ^X44,^X49,^X56,^X49,^X53,^X49,^X4F,^X4E,^X20,^X20,^X44,^X45,^X50,^X54         ; "DIVISION  DEPT:
000000D4          .BYTE   ^X51,^X31,^X20,^X20,^X51,^X32,^X20,^X20,^X51,^X33,^X20,^X20,^X51,^X34,^X20,^X20 ; "Q1  Q2  Q3  Q4
000000E4          .BYTE   ^X54,^X4F,^X54,^X41,^X4C                                                       ; "TOTAL"
000000EC          .LONG   ^X00000002
000000F0          .LONG   ^X00000003
000000F4          .ADDRESS  SHIPS
000000F8          .BYTE   ^X3C,^X3D,^X3D,^X20,^X54,^X6F,^X74,^X61,^X6C,^X20,^X66,^X6F,^X72               ; "<== Total for"
00000108          .BYTE   ^X3C,^X3D,^X3D,^X20,^X47,^X52,^X41,^X4E,^X44,^X20,^X54,^X4F,^X54,^X41,^X4C     ; "<== GRAND TOTAL"
                  .PSECT  $CODE
00000000          .ENTRY  SHIPS, ^X0FFC
00000002          MOVAB   G^RPG$HANDLER, (FP)
00000009          MOVAB   $LOCAL+^X26, -(SP)
00000010          SUBL2   #^X0C, SP
00000013          MOVAB   $LOCAL+^X80, R11
0000001A          MOVAB   $PDATA+^X80, R10
00000021          MOVAB   G^RPG$IOEXCEPTION, R9
00000028          MOVAB   G^RPG$PRINT, R8
                    .
                    .
                    .

                ;+
                ;  Program epilogue code
                ;-

00000539          CALLG   $PDATA+^X8C(R10), G^RPG$TERM_PRINT
00000541          BLBS    R0, 72$
00000544          CALLG   $PDATA+^X94(R10), G^RPG$IOEXCEPTION(R9)
00000548          BRB     73$
0000054A  72$:
0000054A  73$:
0000054A          MOVL    #^X01, R0
0000054D          RET
```

(Continued on next page)

Figure 4–1 (Cont.): Sample Compiler Listing

```
SHIPS                                              28-Jun-1986 15:58:45    VAX RPG II V2.1                    Page  11
Cross Reference in Alphabetical Order  ⓾           28-Jun-1986 15:56:12    RPG$:[HEBERT.RPG]SHIPS.RPG;1 (1)

DEPQTY                  19#     19      21      22
DEPT                    14#     38
DIV                     13#     37      44      46      50
DIVQTY                  21#     21      23      48
FINQTY                  23#     23      52
PROD                    15#     39
PROQTY                  18#     19      41
QTY                     10#     16      18      40
SHIPS                    8#     12
SUMREP                   9#     11      25
UDATE                   28


SHIPS                                              28-Jun-1986 15:58:45    VAX RPG II V2.1                    Page  12
Indicator Cross Reference  ⓫                       28-Jun-1986 15:56:12    RPG$:[HEBERT.RPG]SHIPS.RPG;1 (1)
01                      12      18      19      36
L1                      14      21      22      38      42
L2                      13      23      37      43      45      47
LR                      51
1P                      25      27      30      32


SHIPS                                              28-Jun-1986 15:58:45    VAX RPG II V2.1                    Page  13
Compilation Summary                                28-Jun-1986 15:56:12    RPG$:[HEBERT.RPG]SHIPS.RPG;1 (1)

PROGRAM SECTIONS

     Name ⓬                          ⓭ Bytes    Attributes ⓮
  0 $CODE                              1358      PIC   CON   REL   LCL    SHR   EXE    RD NOWRT Align(2)
  1 $LOCAL                             1280      PIC   CON   REL   LCL NOSHR NOEXE    RD   WRT Align(2)
  2 $PDATA                              279      PIC   CON   REL   LCL    SHR NOEXE    RD NOWRT Align(2)
  3 RPG$UDATE                             6      PIC   OVR   REL   GBL NOSHR NOEXE    RD   WRT Align(2)
  4 RPG$HALTS                             9      PIC   OVR   REL   GBL NOSHR NOEXE    RD   WRT Align(2)
```

```
COMMAND QUALIFIERS (15)
    RPG /LIST/MACHINE_CODE/CROSS_REFERENCE/CHECK=ALL/DEBUG/OBJECT/SEQUENCE_CHECK/WARNINGS=ALL SHIPS.RPG
    /CROSS_REFERENCE  /MACHINE_CODE  /SEQUENCE_CHECK
    /CHECK=(RECURSION,BOUNDS,BLANKS_IN_NUMERICS)
    /DEBUG=(SYMBOLS,TRACEBACK)
    /WARNINGS=(OTHER,INFORMATION)

STATISTICS
(16) Run Time:          5.26 seconds
(17) Elapsed Time:      6.58 seconds
(18) Page Faults:       270
(19) Dynamic Memory:    348 pages
```

Key to Figure 4-1:

❶ The program name.

❷ The date and time of compilation.

❸ The name and version number of the compiler.

❹ The creation date and time of the source file.

❺ The complete file specification (device:[directory]filename.type;version) for the source file. The number in parentheses is a text editor page number.

Items 1 through 5 appear at the top of each page in the listing file.

❻ The 80-column ruler.

❼ Source line numbers assigned by the compiler. The VAX/VMS Debugger uses these line numbers as location specifications.

The letter C after the line number indicates that the line was generated by a copy directive.

❽ Source Listing—source code.

❾ Machine Code Listing—the compiler-generated object code for the program you compiled.

❿ Cross-Reference in Alphabetical Order—the user-defined names in alphabetical order and the line numbers on which they are referenced. The first column with the pound sign (#) after the number lists the line number where the data name is defined. For example, DEPQTY is defined on line 19 and referenced on lines 19, 21, and 22.

```
DEPQTY                  19#     19      21      22
```

⓫ Indicator Cross-Reference—the indicators and the line numbers on which they are referenced. For example, indicator 01 is referenced on lines 12, 18, 19, and 36.

```
01                      12      18      19      36
```

⓬ PROGRAM SECTIONS—names the PSECT numbers and names.

⓭ The bytes allocated for each PSECT.

⓮ The PSECT attributes. See the *VAX/VMS Linker Reference Manual* for information on PSECT attributes.

⓯ COMMAND QUALIFIERS—lists the command line you entered and names the compiler defaults that were in effect when the program was compiled.

⓰ The CPU time used to compile the program.

⓱ The time elapsed to compile the program.
⓲ The number of page faults.
⓳ The number of virtual memory pages used to compile the program.

Chapter 5

Debugging Programs

The VAX/VMS Debugger enables you to debug VAX RPG II programs by monitoring the flow of program execution and logic. For a complete description of debugger capabilities, see the *VAX/VMS Debugger Reference Manual.*

The debugger lets you do the following:

- Set breakpoints to stop program execution just before a specified line is executed.
- Set tracepoints to cause the debugger to pause and display a message whenever a specified line is executed.
- Set watchpoints to cause the debugger to stop and display a message whenever a specified variable is modified.
- Examine and modify source code.
- Examine and modify data.
- Evaluate arithmetic expressions.
- Step through a program—single STEP commands cause the debugger to execute one or more lines and then stop program execution.

The debugger needs information generated by both the VAX RPG II compiler and the VAX/VMS Linker. Specifying the /DEBUG qualifier with the RPG command creates the symbolic information for the debugger. Specifying the /DEBUG qualifier with the LINK command makes the information available to the debugger.

At compile time, VAX RPG II supports the following options for the /DEBUG=options qualifier:

- ALL
- NONE
- [NO]TRACEBACK
- [NO]SYMBOLS

Specifying the /DEBUG=SYMBOLS qualifier for the RPG command allows you to examine and change the contents of variables throughout your program. However, file names from File Description specifications are not available as variables.

If you omit the /DEBUG qualifier from the RPG and LINK commands, you can specify the /DEBUG qualifier with the RUN command. In this case, no symbolic information is available to the debugger; you must make every reference to a program variable in terms of its absolute address.

If you do not specify the /DEBUG qualifier with any of the RPG, LINK, or RUN commands and an error occurs, you receive a traceback list (a description of the logic flow up to the point where the error was detected). However, you cannot invoke the debugger. If you compile your program with the /DEBUG=NOTRACEBACK qualifier or link your program with the /NOTRACE qualifier, you do not receive the traceback list. The default options for the /DEBUG qualifier are TRACEBACK and NOSYMBOLS.

If you want to use the source line display while using the COMPILE command, you must inform the debugger where the source file resides. To do this, complete the following steps:

1. Define the symbol RPG to include symbols for the VAX/VMS Debugger. For example:

```
$ RPG :== RPG/DEBUG
```

2. Execute the VAX RPG II editor COMPILE command during the editing session
3. Execute the LINK/DEBUG command after exiting from the VAX RPG II editor
4. Execute the RUN command
5. Enter the debugger command: SET SOURCE source-file-spec

See Chapter 3 for information on compiling and linking VAX RPG II programs and their respective command qualifiers.

If you are using the VAX Performance and Coverage Analyzer (the Analyzer), you must specify the following:

```
/DEBUG=SYS$LIBRARY:PCA$OBJ.OBJ MYPROGRAM.OBJ
```

The VAX Performance and Coverage Analyzer consists of a collector and an analyzer. The collector gathers information (such as execution counts) on your program while it is executing. The analyzer makes it possible to interpret the data gathered by the collector. The analyzer is used to track a performance problem in an entire program down to a certain module, or even down to a certain line of code. See Appendix C for an example of the VAX Performance and Coverage Analyzer applied to a VAX RPG II program.

5.1 Using the Debugger with VAX RPG II

Debugging VAX RPG II programs is somewhat different from debugging programs in other languages. The VAX RPG II program cycle determines the order in which the program lines are processed. See Chapter 1 for a complete discussion of the VAX RPG II program cycle.

You can reference those line numbers VAX RPG II assigns to your program in the listing file. The line numbers you specify in columns 1 through 5 of a specification are not used. The compiler assigns line numbers only to certain specifications at specific points in the logic cycle; therefore, you can specify a breakpoint or tracepoint at these points in the program:

- A break at a File Description specification (see Chapter 15 for more information on specifications) occurs just before an input or update file is opened or just before an output file is created. The line number of this break corresponds to the File Description specification for this file.
- A break at an Input specification occurs before the fields are loaded with data from a record. The line number of this break corresponds to the record definition in an Input specification.
- You can set two breaks for each Calculation specification. The first break occurs just after testing control-level indicators, if used, and just before testing conditioning indicators. The second break occurs just before executing the operation code. For example, if a Calculation specification begins with line number 25, you can specify the line and statement number SET BREAK 25.1 to test indicators.

SET BREAK 25.2 breaks just before executing the operation code. If a particular Calculation specification has no indicators, SET BREAK 25 breaks just before executing the operation code.

- A break at an Output specification occurs after the output buffer has been built but before the record is output. The line number of the break corresponds to the record definition in an Output specification.

5.2 Debugger Commands and Keywords

There are many debugger commands, but not all are appropriate for use in debugging VAX RPG II programs. Table 5–1 lists some debugger commands and keywords (and their abbreviations) that are helpful in debugging VAX RPG II programs.

Table 5–1: Debugger Commands and Keywords

Command Names	(abbrev)	Keywords	(abbrev)
SET	(SE)	LANGUAGE	(LA)
SHOW	(SH)	MODULE	(MODU)
CANCEL	(CAN)	SCOPE	(SC)
EXAMINE	(E)	BREAK	(B)
EVALUATE	(EV)	TRACE	(T)
DEPOSIT	(D)	WATCH	(W)
EXIT	(EXI)		
STEP	(S)		
GO	(G)		
EDIT	(ED)		

The rest of this chapter describes these debugger commands and explains how to use them.

5.3 Preparing to Debug a Program

This section describes the SET LANGUAGE and SHOW LANGUAGE commands used to create the proper environment for debugging a VAX RPG II program.

5.3.1 SET LANGUAGE Command

The SET LANGUAGE command causes the debugger to conduct the debugging dialogue according to the conventions of the specified language. If your program does not call any subprograms written in languages other than RPG II, you do not need to use the SET LANGUAGE command. If your program calls a subprogram written in another language, you can cause the debugger to execute the subprogram by specifying the STEP/INTO command. See Section 5.4.5 for information about the /INTO qualifier. After the debugger has stepped into the subprogram, you must use the SET LANGUAGE command to specify the language of the subprogram. After you have finished executing the subprogram and you have returned to the main program, you must use the SET LANGUAGE command to specify the language of the main program.

The format of the SET LANGUAGE command is as follows:

```
SET LANGUAGE language
```

language
Specifies the language to be used.

5.3.2 SHOW LANGUAGE Command

To determine the language of the program currently being executed, use the SHOW LANGUAGE command. The format of the SHOW LANGUAGE command is as follows:

```
SHOW LANGUAGE
```

The debugger responds by displaying the program's language, as shown in the following example:

```
DBG>SHOW LANGUAGE
language: RPG
```

5.4 Controlling Program Execution

To see what is happening during execution of your program, you must be able to suspend and resume the program at specific points. The following commands are available for these purposes:

- SET BREAK
- SHOW BREAK
- CANCEL BREAK
- SET TRACE
- SHOW TRACE
- CANCEL TRACE
- SET WATCH
- SHOW WATCH
- CANCEL WATCH
- SHOW CALLS
- GO
- STEP
- TYPE
- CTRL/Y
- EXIT

You can specify a VAX RPG II label as a breakpoint or a tracepoint. These labels correspond to specific points in the logic cycle. The following list describes VAX RPG II labels:

- *DETL breaks just before outputting heading and detail lines.
- *GETIN breaks just before reading the next record from the primary or secondary file.
- *TOTC breaks just before performing total-time calculations.
- *TOTL breaks just before performing total-time output.
- *OFL breaks just before performing overflow output.
- *DETC breaks just before performing detail-time calculations.

5.4.1 SET BREAK, SHOW BREAK, and CANCEL BREAK Commands

The BREAK commands allow you to select specific locations for program suspension, so that you can examine or modify the following data:

- Variables
- Table entries
- Array elements

When you specify a table name, you can examine or modify the entry retrieved from the last LOKUP operation.

You can also set a breakpoint at any point listed in Section 5.1.

The BREAK commands perform the following functions:

- SET BREAK defines the line number that will suspend execution.
- SHOW BREAK displays all breakpoints currently set in the program.
- CANCEL BREAK removes selected breakpoints.

The format of the SET BREAK command is as follows:

```
SET BREAK  %LINE lin-num[.stmnt-num]  [DO(command(s))]
```

lin-num
Specifies the line number where the breakpoint will occur. You can also specify a logic cycle label, a TAG name, or a subroutine label.

stmnt-num
Specifies the statement number where the breakpoint will occur. You can use statement numbers only with Calculation specifications that have conditioning indicators.

DO(command(s))
Requests the debugger to perform the specified debugger commands, if specified, when the breakpoint is reached.

In the following example, SET BREAK examines variables TOTAL and AREA when the breakpoint at line 100 is reached:

```
DBG>SET BREAK %LINE 100 DO(EXAMINE TOTAL; EXAMINE AREA)
```

The format of the SHOW BREAK command is as follows:

```
SHOW BREAK
```

SHOW BREAK takes no arguments. The debugger responds by displaying the current breakpoints, as shown in the following example:

```
DBG>SET BREAK LOOP
DBG>SET BREAK %LINE 50
DBG>SHOW BREAK
breakpoint at ARRX37\LOOP
breakpoint at ARRX37\%LINE 50
```

The format of the CANCEL BREAK command is as follows:

```
CANCEL BREAK  %LINE lin-num[.stmnt-num]
              /ALL
```

lin-num[.stmnt-num]

Removes the breakpoint at the specified line and statement number, logic cycle label, TAG name, or subroutine label.

/ALL

Removes all breakpoints in the program.

Normally, the debugger displays the line number when it suspends execution because of a breakpoint or step. There are two exceptions to this behavior:

- When stepping through a subroutine, the debugger displays the subroutine label.

  ```
  DBG>STEP
  stepped to PROG1\SUB1
  ```

- When stepping through a TAG, the debugger displays the TAG name.

  ```
  DBG>STEP
  stepped to PROG1\TAG1
  ```

5.4.2 SET TRACE, SHOW TRACE, and CANCEL TRACE Commands

The TRACE commands let you set, examine, and remove tracepoints in your program. A tracepoint is similar to a breakpoint in that it suspends program execution; however, after displaying the trace variables, program execution resumes immediately. Thus, tracepoints let you follow the sequence of program execution to ensure that execution is occurring in the proper order.

Tracepoints and breakpoints are mutually exclusive. If you set a tracepoint at a current breakpoint, the breakpoint will be canceled. If you set a breakpoint at a current tracepoint, the tracepoint will be canceled.

The TRACE commands perform the following functions:

- SET TRACE establishes points within the program where execution is momentarily suspended.
- SHOW TRACE displays the points in the program where tracepoints are currently set.
- CANCEL TRACE removes one or more tracepoints currently set in the program.

The format of the SET TRACE command is as follows:

```
SET TRACE  %LINE lin-num[.stmnt-num]
```

lin-num[.stmnt-num]
Specifies the line and statement number, logic cycle label, TAG name, or subroutine label where the tracepoint will occur.

The format of the SHOW TRACE command is as follows:

```
SHOW TRACE
```

SHOW TRACE takes no arguments. The debugger responds by displaying the current tracepoints, as shown in the following example:

```
DBG>SET TRACE LOOP2
DBG>SET TRACE %LINE 100
DBG>SHOW TRACE
tracepoint at ARRX37\LOOP2
tracepoint at ARRX37\%LINE 100
```

The format of the CANCEL TRACE command is as follows:

```
CANCEL TRACE  %LINE lin-num[.stmnt-num]
              /ALL
```

lin-num[.stmnt-num]
Removes the tracepoint at the specified line and statement number, logic cycle label, TAG name, or subroutine label.

/ALL
Removes all tracepoints in the program.

5.4.3 SET WATCH, SHOW WATCH, and CANCEL WATCH Commands

The WATCH commands let you monitor the contents of variables. Watchpoints determine when an attempt is made to modify variables. When an attempt is made, the debugger halts program execution and prompts for a debugger command. Watchpoints are monitored continually. Thus, you can determine whether a particular variable is being modified inadvertently during program execution. Watchpoints, tracepoints, and breakpoints are mutually exclusive. The WATCH commands perform the following functions:

- SET WATCH defines the variables to be monitored.
- SHOW WATCH displays the variable currently being monitored.
- CANCEL WATCH disables monitoring of specified variables.

The format of the SET WATCH command is as follows:

```
SET WATCH vbl
```

vbl
Specifies the variable to be monitored. You can monitor variables and array elements.

In the following example, SET WATCH sets a watchpoint for the variable AREA:

```
DBG>SET WATCH AREA
```

The format of the SHOW WATCH command is as follows:

```
SHOW WATCH
```

SHOW WATCH takes no arguments. The debugger responds by displaying the current watchpoints, as shown in the following example:

```
DBG>SET WATCH INDEX2
DBG>SHOW WATCH
watchpoint of ARRX37\INDEX2
```

The format of the CANCEL WATCH command is as follows:

```
CANCEL WATCH   vbl/ALL
```

vbl
Specifies the variable that disables monitoring.

/ALL
Removes all watchpoints from the program.

The following command cancels the watchpoint for the variable AREA:

```
DBG>CANCEL WATCH AREA
```

5.4.4 SHOW CALLS Command

The SHOW CALLS command can be used to produce a traceback of calls to program modules. It is particularly useful when you have returned to the debugger following a CTRL/Y command. The format of the SHOW CALLS command is as follows:

```
SHOW CALLS [n]
```

The debugger displays a traceback list, showing the sequence of calls to program modules leading to the current module.

If you include a value for n, the n most recent calls are displayed.

5.4.5 GO and STEP Commands

The GO and STEP commands let you initiate and resume program execution. The GO command initiates execution from the current line or at a specified point in the program and continues to the end of the program or to the next breakpoint. The STEP command initiates execution from the current line, and continues for a specified number of lines.

The format of the GO command is as follows:

```
GO [%LINE lin-num[.stmnt-num]]
```

lin-num[.stmnt-num]
Specifies the line and statement number, TAG name, or subroutine label where execution will begin.

The normal use of the GO command is to continue execution after a breakpoint or at program initiation. Resuming execution at a point other than the current line can cause unpredictable results because of the nature of the VAX RPG II logic cycle.

Use the STEP command to execute one or more VAX RPG II program lines and immediately return to the debugger. The format of the STEP command is as follows:

```
STEP [/qualifiers] [n]
```

The value specified for n determines the number of statements to be executed. If you specify 0, or omit a value for n, a value of 1 is assumed.

You can specify the following qualifiers with the STEP command:

/[NO]SYSTEM
Causes the debugger to count steps wherever they occur. The /NOSYSTEM qualifier is the default.

/[NO]OVER
Causes the debugger to ignore calls to subprograms as it steps through the program. That is, it steps over each call to a subprogram. The /OVER qualifier is the default.

/[NO]INTO

Causes the debugger to recognize calls to subprograms as it steps through the program. That is, it steps into each subprogram. The /NOINTO qualifier is the default.

/[NO]LINE

Causes the debugger to step through the program on a line by line basis. The /LINE qualifier is the default.

/[NO]SOURCE

Causes the debugger to display the lines of source code that corresponds to the lines being executed with each step. Source lines are also displayed when a breakpoint or watchpoint occurs. When stepping through Input and Output specifications, the debugger displays the first line of a record definition. The /SOURCE qualifier is the default.

You can specify one or more qualifiers each time you issue a STEP command, or you can use a SET STEP command to override the defaults.

The following command specifies that the defaults for the /LINE, /INTO, and /SYSTEM qualifiers are overridden:

```
DBG>SET STEP NOLINE,INTO,SYSTEM
```

When you subsequently issue a STEP command with no qualifiers, the debugger assumes these qualifiers (/NOLINE, /INTO, and /SYSTEM) are in effect. You can, however, supersede the current qualifiers by including a qualifier with a STEP command.

The following command executes 10 lines, regardless of the SET STEP command:

```
DBG>STEP/LINE 10
```

It is advisable to use STEP to execute only one or a few lines at a time. To execute many lines and then stop, use a SET BREAK command to set a breakpoint, then issue a GO command.

5.4.6 TYPE Command

The TYPE command displays the line of source code you specify. The format of the TYPE command is as follows:

```
TYPE [lin-num[:lin-num][,...]]
```

lin-num[:lin-num]
Specifies the lines of source code to be displayed.

The following command displays lines 1 through 30:

```
DBG>TYPE 1:30
```

The following command displays lines 1 and 30:

```
DBG>TYPE 1,30
```

You can display the line after the current line by typing TYPE and by pressing the RETURN key.

5.4.7 EDIT Command

The EDIT command allows you to edit the file you are debugging.

The editing session begins at the current debugging line.

EDIT/EXIT specifies that you want to end the debugging session and begin an editing session.

EDIT/NOEXIT specifies that you want to return to the debugging session after you make your edits. The /NOEXIT qualifier is the default.

5.4.8 CTRL/Y Command

You can use the CTRL/Y command at any time to return to the system command level. You issue this command when you press the CTRL key and the Y key at the same time. The dollar sign ($) prompt will be displayed on the screen. To return to the debugger, type DEBUG. Use the CTRL/Y command if your program goes into an infinite loop or, for some reason, fails to stop at a breakpoint. To find out where you were when CTRL/Y was executed, use the SHOW CALLS command after you have returned to the debugger.

5.4.9 EXIT Command

The EXIT command lets you exit from the debugger when you are ready to terminate a debugging session. The format of the EXIT command is as follows:

```
EXIT
```

The EXIT command uses no arguments. To return to system command level after your program has terminated, use the EXIT command.

5.5 Examining and Modifying Locations

After you have set breakpoints and begun execution, the next step is to see whether correct values are being generated and, if necessary, to change the contents of variables as execution proceeds. You may also want to calculate the value of an expression that appears in your program. The debugger provides the following commands for these purposes: EXAMINE, DEPOSIT, and EVALUATE.

5.5.1 EXAMINE Command

The EXAMINE command lets you look at the contents of the following:

- A variable
- The current table entry
- An array element
- The I/O buffer

The format of the EXAMINE command is as follows:

```
EXAMINE  vbl [,vbl]
```

vbl

Specifies a simple or subscripted variable.

The following command displays the contents of the variable SALES:

```
DBG> EXAMINE SALES
```

The following command displays the contents of the ninth element in array ARRAY:

```
DBG>EXAMINE ARRAY(9)
```

The following command displays the contents of the first through the tenth elements of the array ARRAY:

```
DBG>EXAMINE ARRAY(1:10)
```

You can examine indicators to see whether they are set on or off. Precede the indicator you want to examine with the string *IN. If an indicator is set on, 1 is displayed. If an indicator is set off, 0 is displayed.

The following command displays the current setting for indicator 56:

```
DBG>EXAMINE *IN56
```

The debugger responds by displaying:

```
*IN56: "0"
```

You cannot examine external indicators in this way, but you can do the following. To determine the current value of U5, for example, enter this command:

```
DBG>CALL RPG$EXT_INDS(5)
```

The debugger responds by displaying:

```
value returned is 0
```

The program must have been linked with the /NOSYSSHARE qualifier to do this.

You can also display the current contents of the I/O buffer. To display the I/O buffer, specify the name of the input file, update file, or output file, a dollar sign ($), and the string BUF.

The following command displays the contents of the I/O buffer for the input file INPUT:

```
DBG>EXAMINE INPUT$BUF
```

The following command displays the ASCII equivalent of the string *STRING*, which is *n* characters in length:

```
DBG>EXAMINE/ASCII:n STRING
```

To examine a variable which contains the at sign (@), use %NAME as follows:

```
DBG>EXAMINE %NAME 'ITEM@'
```

5.5.2 DEPOSIT Command

The DEPOSIT command lets you change the contents of specified variables. The format of the DEPOSIT command is as follows:

```
DEPOSIT  vbl=value
```

vbl
Specifies the variable that the value is deposited into.

value
Specifies the value to be deposited.

You can change the contents of a specific variable or of several consecutive variables, as shown in the examples in this section.

Values deposited into numeric fields are aligned on the decimal point. Shorter fields are padded with zeros to the left and right of the decimal point.

The following command places the decimal value 100 into the variable BONUS:

```
DBG>DEPOSIT BONUS=100
```

The following command places the decimal values 100, 150, and 200 into elements 1, 2, and 3 of array ARRAY:

```
DBG>DEPOSIT ARRAY(1)=100, 150, 200
```

The delimiters used to enclose ASCII strings in the DEPOSIT command can be either single (') or double (") quotation marks. Use the keyboard apostrophe for the single quotation mark.

Values deposited into character fields are left justified. If the value contains fewer characters than the character field, the field is padded on the right with spaces.

The following command places the string ACTIVE in the variable STATUS:

```
DBG>DEPOSIT STATUS="ACTIVE"
```

You can also use DEPOSIT to set indicators on or off. Precede the indicator you want to set with the string *IN. To set an indicator on, specify 1 as the variable value. To set an indicator off, specify 0 as the variable value.

The following command sets indicator 56 on:

```
DBG>DEPOSIT *IN56 = "1"
```

5.5.3 EVALUATE Command

The EVALUATE command lets you use the debugger as a calculator to determine the value of arithmetic expressions. The format of the EVALUATE command is as follows:

```
EVALUATE expression
```

expression

Specifies the expression whose value is to be determined.

The following command displays the value of the expression ARRAY(FLD1) * FLD2:

```
DBG>EVALUATE ARRAY(FLD1) * FLD2
```

Using VAX RPG II Features on VMS

This part of the manual provides information on the use of VAX/VMS features in the development of VAX RPG II programs:

- Screen handling
  - Workstation (WORKSTN) files syntax
  - VAX RPG II interface with VAX Forms Management System (VAX FMS)
  - Screen and display forms conversion utility
- Indicators
- Files
- Printer output files
- Tables
- Arrays
- System routines and system services
- Program optimizing

Chapter 6

VAX RPG II Screen Handling

VAX RPG II screen handling is accomplished in two ways: (1) language syntax in the VAX RPG II program, and (2) various screen design activities outside of the VAX RPG II program, including the VAX Forms Management System (VAX FMS).

The general language syntax is based on WORKSTN files used in other vendor RPG II implementations which allow RPG II programs to interact with a terminal. In these implementations, a WORKSTN file interacts with forms that are defined externally to the RPG program.

The VAX RPG II language provides an interface between WORKSTN files and VAX FMS forms. You can create and modify these forms with an interactive forms editor. If you are developing new applications, you should refer to the FMS/EDIT chapter of the *VAX FMS Utilities Reference Manual*. If you are working with existing WORKSTN file programs and forms based on Screen and Display (S and D) specifications, VAX RPG II provides a Conversion Utility that converts the Screen and Display specifications to the VAX FMS form language. You can use the forms on VT100 and VT200 terminals. Additional information on VAX FMS can be found in the *VAX FMS Utilities Reference Manual*.

NOTE

Use of the DSPLY operation code and WORKSTN files in the same program can produce undesirable run-time results on your terminal screen. Displayed information can become mixed and garbled, because neither screen handling mechanism is aware of the other and no coordination takes place. Concurrent use is highly discouraged.

6.1 Creating and Modifying Forms

A VAX FMS form provides the communications bridge between a VAX RPG II program WORKSTN file and a terminal screen. VAX FMS forms include fields and constant information. You can specify various video attributes, such as blink or underline, for parts of the form. You can use fields in the form to display information from the program, or to input information that will be returned to the program. VAX FMS has various methods of validation for each input field.

VAX FMS has two methods for creating forms:

1. The VAX FMS Form Editor Utility allows you to interactively design the layout of a form, showing the position of each field as well as constant information. The form can be modified later. You invoke the VAX FMS Form Editor Utility with the following command:

   ```
   $ FMS/EDIT form-name
   ```
   [1]

2. You can also create a form by editing a text file that contains the VAX FMS form language description for the form. After you create a text file, it can be translated to the VAX FMS binary form representation with the following command:

   ```
   $ FMS/TRANSLATE form-language-file
   ```

   Note that the output from either FMS/EDIT or FMS/TRANSLATE is the binary representation of a form with the default file type FRM. You can use the VAX FMS Form Editor Utility to modify the form after a form language file has been translated. The form language file can be edited again and retranslated. You can also get a file that has the form language from any VAX FMS form using the following command:

   ```
   $ FMS/DESCRIPTION/FULL form-name
   ```

[1] For detailed information on VAX FMS and this command, refer to the *VAX FMS Utilities Reference Manual.*

6.2 Creating Form Libraries

You place the forms used by your program in a VAX FMS form library. For example, an inventory program could have three forms:

1. PART_NUMBER—to display a part number
2. PART_DISPLAY—to display data about a part
3. PART_ERROR—to display a message if a request was made for an invalid part.

You would place these three forms in the PARTS form library as follows:

```
$ FMS/LIBRARY/CREATE PARTS PART_NUMBER,PART_DISPLAY,PART_ERROR
```

The VAX RPG II program would then be able to access the three forms in this library by referencing the indicated names. Note that other programming languages in the VAX/VMS language family can access these forms from the PARTS form library.

6.3 WORKSTN Files

WORKSTN files allow you to access many VAX FMS form capabilities from a VAX RPG II program. However, WORKSTN files provide only a subset of the capabilities available with VAX FMS. The most complete interface between VAX RPG II and VAX FMS is provided by the callable interface to VAX FMS. These features are accessed in VAX RPG II programs by the CALL, PARMx[1], EXTRN, and GIVNG operation codes. For additional information on these operation codes, see Chapters 12 and 16. In most cases, WORKSTN files will provide all the capabilities you need to access VAX FMS. You can use the CALL interface with WORKSTN files to access those VAX FMS features that are not supported directly by WORKSTN files. If you choose to use WORKSTN files *and* the CALL to VAX FMS in the same program, careful attention to Section 6.3.7 will ensure the best results.

WORKSTN files are further described on Control, File Description, Input, Calculation, and Output specifications.

[1] PARMx includes PARM, PARMD, and PARMV.

6.3.1 Control Specifications (H)

The H specification program name is used only as a second choice for the VAX FMS form library name, if an FMTS name is not supplied. See Section 6.3.2 for a detailed description.

6.3.2 File Specifications (F) with WORKSTN Files

WORKSTN files must include the letter C for the file type and WORKSTN for the file device. There can be only one WORKSTN file in a program. A WORKSTN file can have continuation lines that describe the form library name (FMTS), the INFDS data structure, and SLN variable. You indicate continuation lines for the File specification by inserting a K in column 53. The only fields available for use on a continuation line are columns 53 through 65 (or 53 through 67, depending on the option). The following continuation-line option format is accepted by VAX RPG II.

Column	Contents
53	K
54-59	Option name
60-65/67	Value

NOTE

If the F specification is a WORKSTN file, it cannot specify a symbolic device.

If you supply an FMTS name, then that name will be used for the VAX FMS form library name (such as PARTS in the preceding example). If you do not supply an FMTS name, the program name (in columns 75 through 80 of the Control specification) is used for the VAX FMS form library name. If the program name is not supplied, the WORKSTN file name is used as the VAX FMS form library name.

If you supply a start line (SLN) option, the SLN value will offset the form from the start line of the form. The offset form must fit on the screen, or an error status will occur at run time. See Section 6.5.1 for examples.

WORKSTN files must have a file designation of primary (P) or demand (D). If you choose P, all forms display and forms input will be performed at a specific point in the logic cycle. If you choose D, the EXCPT and READ operation codes can be used to determine when forms are displayed and when input from forms is performed. (Note that if you make the WORKSTN file the primary file, no secondary files are allowed in the program.) See Section 6.5.1 for further details.

6.3.3 Input Specifications (I)

VAX RPG II Input specifications are used to describe the declarations and processing to be performed when input is received from the WORKSTN file. The Input specifications extract data from record buffers and set any record identifying indicators on.

You program the Input specifications similarly to an input or update file. The most important part of the Input specifications is ensuring that the start and end positions for each field match the field size and order that was specified in the form. You can help to set up the input field specifications correctly with the following command:

```
$ FMS/DESCRIPTIONS/BRIEF form-name
```

The output from this VAX FMS command is a list of the fields in the form, including the size of each field. If particular fields are display only, they are indicated as such. This information can help you select the proper starting and ending field positions for any input field specifications. Some input constants may cause the VAX FMS record buffer to be adjusted. If you are converting from Screen and Display specifications, see Section 6.7.

6.3.4 Calculation Specifications (C)

If you use a D file designation, making the WORKSTN file a demand (D) file, then you must use the EXCPT and READ operation codes to control both form display and form input. The EXCPT operation code works with a WORKSTN file similarly to any other file. EXCPT will cause output from all exception Output specifications with conditioning indicators set on. Note that you can use EXCPT with factor 2 for finer control over the specific Output specifications.

You use the READ operation code to get input from the last form displayed (that had input fields) since your previous read.

To detect end-of-file (EOF) on the WORKSTN file, you can specify an indicator in columns 58 and 59 with the READ operation code. End-of-file is reached when a READ operation code is executed and the last group of forms displayed since the previous read had no input fields. This read operation will cause the EOF indicator on the READ operation code to be set on, if present. Otherwise, your program terminates.

The READ operation code can also have an indicator to detect errors. See Section 6.5 for further details.

6.3.5 Primary WORKSTN File

A primary WORKSTN file does not require any Calculation specification operation codes. However, form display and form input during the logic cycle are similar to an EXCPT operation code followed by a READ operation code. The LR indicator will be set on when a READ operation to the primary WORKSTN file is executed and the last form displayed has no input fields.

Primary WORKSTN files are processed just like any other primary input file. However, input can become complicated, especially if this is the first time through the logic cycle.

When the logic cycle starts, input to the WORKSTN file can come from one of two places. If no form has been displayed on the first cycle, then a blank record is generated in place of input from the WORKSTN file. If an input form has already been displayed because the first-page (1P) indicator is present in a WORKSTN Output specification, then input is obtained from the WORKSTN file.

When it is not the first logic cycle, all input is assumed to be coming from the WORKSTN file. If no input form was displayed for this logic cycle, then end-of-file will be flagged and your program will terminate.

In any requested READ operation code (for a demand WORKSTN file), a blank record will be returned until a form is output.

6.3.6 Output Specifications (O)

Output specifications for WORKSTN files differ only slightly from Output specifications for other files. Insert a K in column 42 of an Output field specification, indicating that the literal that begins in column 45 on that Output specification is the name of a form to be displayed. The field output specifications that follow the form specification can be data that is to be displayed on the form.

The input and output record buffers for each form have the same field layout, except for an input constant converted by the S and D Converter. See Section 6.7 for further details.

You can use FMS/DESCRIPTIONS/BRIEF (as in Input specifications) to help you specify the ending positions for output fields that contain data to be displayed on a form.

Note that logic cycle output of forms requires you to insert a D in column 15 for each output record for the WORKSTN file that indicates detail output. Each output record corresponds to one form that can be displayed. Insert an E in column 15 for a demand WORKSTN file, indicating exception output that will be controlled by EXCPT operation codes.

To indicate end-of-file on the WORKSTN file, the last-record (LR) indicator can be set on, or a READ operation code can be executed on the WORKSTN file, if the forms displayed since the last READ operation had no input fields. For a primary WORKSTN file, the LR indicator will be set on. For a demand WORKSTN file, any indicator specified on a READ operation code in columns 58 and 59 is set on.

6.3.7 VAX FMS Call Interface Run-Time Support

VAX RPG II provides run-time support to interface WORKSTN files to VAX FMS. If you want to use the WORKSTN files and call VAX FMS (FDV$) routines in the same program, you must understand the built-in WORKSTN run-time support as described in this section. The VAX RPG II sample program SYS$EXAMPLE:RPGFMS.RPG uses VAX FMS to demonstrate the combined use of WORKSTN file syntax and VAX FMS calls. Comments in that program have particularly helpful information on the FMS form 'REGISTER' that is manipulated with VAX FMS (FDV$) routines. Workspace and input/output must be controlled through calls to VAX FMS. See the VAX FMS documentation for detailed information on the VAX FMS call interface.

To use the sample program, enter the following commands:

```
$ RPG SYS$EXAMPLES:RPGFMS
$ LINK RPGFMS,SYS$LIBRARY:RPGSCR
$ DEFINE FMS$EXAMPLES SYS$SYSROOT:[SYSHLP.EXAMPLES.FMS]
$ DEFINE INP FMS$EXAMPLES:SAMP
$ DEFINE SAMP FMS$EXAMPLES:SAMP
$ RUN RPGFMS
```

The VAX RPG II WORKSTN run-time support provides an interface for performing four functions:

- Initialization
- Display a form
- Read from a form
- Termination

Any error returned by a VAX FMS call will cause the program to halt, except as described in the following sections.

6.3.7.1 Initialization

During initialization, the run-time support attaches the terminal to FDV$ATERM using VAX FMS logical channel 255. Then, each form that you reference on the Output specifications is loaded into a separate workspace. Even if the same form name is given in different Output specification records, a separate workspace is created for each Output specification form reference. The form library is opened (on VAX FMS logical channel 254) only if loading a form fails. Thus, some or all of the forms can be linked into the program, and VAX FMS accesses the disk library only if necessary. Also during initialization, VAX RPG II loads RPG-specific named data for handling input constants and selective enabling and disabling of VAX FMS terminators.

6.3.7.2 Displaying a Form

Any program request to display a form (EXCPT operation code or logic cycle output) causes the run-time support to switch to the appropriate workspace, use FDV$PUTAL to output data from the WORKSTN file record buffer, and then use FDV$DISPW to display the loaded form. The run-time support keeps a list of all forms displayed since the previous program request to read from a form. Note that if the FDV$PUTAL returns the FDV$_NOF error, this error will be ignored.

6.3.7.3 Reading from a Form

Any program request to read from a form (READ operation code or logic cycle input) will cause the run-time support to attempt to use FDV$GETAL on the form in the current workspace. If FDV$GETAL fails because the form in the current workspace has no input fields, the current workspace will be switched to the form displayed just prior to the last form. The specific errors which cause FDV$GETAL to try another form are FDV$_NOF and FDV$_DSP. The FDV$_UNF error is ignored. FDV$GETAL is then tried again. This process repeats until the FDV$GETAL succeeds, or until there are no more forms in the list. If the list is exhausted, the LR indicator will be set on in the program (this is because no input-capable forms have been displayed since the last READ operation). One special case is a READ operation before any output has occurred. In this case, a blank record will be returned. After the FDV$GETAL completes, all forms displayed since the last READ operation are marked with FDV$NDISP.

If you wish to be able to read from a form, it must be either the last form displayed prior to the READ operation, or one of the first 100 forms displayed since the last READ operation.

6.3.7.4 Termination

During termination, all workspaces and the terminal are detached. If a VAX FMS form library is open, it is closed.

6.3.7.5 Current Workspace

VAX FMS maintains a current workspace. Therefore, you should be careful when you use VAX FMS calls that modify the current workspace. If you process forms that are not handled by the WORKSTN file, you will not be able to issue a READ operation to the WORKSTN file until you have displayed another form using WORKSTN file output. The output will properly reestablish the current workspace as corresponding to the form displayed with WORKSTN file output.

6.4 Command Keys (K Indicators) and Function Keys

Command keys provide a way for you to interact with the program. When a form is displayed on a screen and the operator is entering input values, there are a number of ways to indicate that the form is complete.

The most common method is to press either the RETURN key or the ENTER key. VAX FMS recognizes these keys as form terminators; when they are received, no further input is allowed to the current form and control returns to the program. The input operation that requested form input is considered to have concluded successfully.

Command keys provide an alternative method to interact with a VAX RPG II program. In addition to the effects just described, one of the K indicators is set on while the others are set off. These input operations are also considered successful. Data from the form is read by the program; the status field in the INFDS indicates that a command key was entered. One of the typical uses for this feature is to provide a way for you to indicate which form to process next. The program can use the K indicators to condition further processing.

All other form terminators are considered to be function keys. Any of these terminate input to the form and control returns to the program with normal status. Data from the form is read by the program; the status field in the INFDS indicates that a function key was entered; no error processing is initiated. In this case, all the K indicators are set off.

In addition to providing a default definition of key sequences associated with command keys, VAX RPG II also allows you to specify a different set of command keys at run time.

When a form is displayed, you may terminate the form by pressing the PF1 key followed by one of 36 characters. Other terminators act as function keys.

6.4.1 K Indicators

VAX RPG II provides 36 K indicators: KA through KZ and K0 through K9. These can be used as general-purpose indicators. In a program containing a WORKSTN file, their values can be modified at the following times:

- If the WORKSTN file is a primary file, K indicators can change during normal logic cycle processing of the primary input file.
- If the WORKSTN file is a demand file, K indicators can change as part of the processing for a READ operation code on the WORKSTN file.

In both cases, the indicators do not change if an error was encountered during the input operation. If there was no error, then all K indicators are set off and the K indicator corresponding to the command key entered is set on.

6.4.2 Command Keys

VAX RPG II provides 36 command keys. Each command key is associated with one K indicator. The default K indicator command key equivalence is described by the following table.

Command Key	K Indicator
PF1 <letter>	K <letter>
PF1 <digit>	K <digit>

The <letter> is an uppercase letter. VAX FMS considers PF1/A or PF1/a to be different terminators. There is no provision for having two different VAX FMS terminators set on the same K indicator.

6.4.3 Function Keys

All form terminators except the ENTER and RETURN keys and the command keys are considered to be function keys.

6.4.4 User-Defined Command Keys

Terminating a form by pressing the PF1A key will result in the KA indicator being set on. The default command keys definitions are given in Section 6.4.2. If you prefer another set of keys to be associated with the K indicators, you can enter a command like the following:

```
$ DEFINE RPG$COMMAND_KEYS "305,306,307"
```

The logical name is translated at run time and should consist of a list of numeric values separated by commas. Each entry is the VAX FMS-defined key sequence that identifies the key or sequence of keys that the operator must enter. The position within the list identifies the associated K indicator. The indicators have the following order: KA through KZ and K0 through K9. The preceding example indicates that the sequence PF1/1 would set on the KA indicator, PF1/2 would set on the KB indicator, and PF1/3 would set on the KC indicator.

You can redefine some or all of the default definitions. A default definition will not be changed if the corresponding entry in the list has no value. A command key will not be associated with a K indicator if the corresponding entry is -1.

Thus, this DCL command line has the meaning described in the following table.

```
$ DEFINE RPG$COMMAND_KEYS "305,,-1,307"
```

Parameter	Keys	Indicator	Explanation
305	PF1 1	KA	Redefines default definition
null	PF1 B	KB	No change to default definition
-1		KC	No keys are associated with KC
307	PF1 3	KD	Redefines default definition

The other keys would retain their default definitions.

Refer to the *VAX FMS Form Driver Reference Manual* for the VAX FMS key codes. To use the keypad keys as terminators, your keypad must be in application mode. You can do this by entering the following DCL command:

```
$ SET TERMINAL/APPLICATION_KEYPAD
```

Or, you can include a call to FDV$SPADA in the Calculation specifications (see the *VAX VMS Form Driver Reference Manual*).

6.4.5 Selective Enabling of Command Keys

VAX RPG II enables all command keys and all function keys. Thus, if the program is trying to read from a WORKSTN file and you enter one of the key sequences that VAX FMS recognizes as a terminator, input to the form will be terminated and control will return to the program.

Pressing the RETURN key or the ENTER key automatically terminates form input, but you can control the treatment of other keys.

You can enable a list of terminators. Entering one of these keys terminates form input. Entering any other terminator does not end form input. VAX FMS instead signals an error and the key is ignored.

As an alternative, you can disable a list of terminators. Entering one of these keys causes VAX FMS to signal an error and ignore the key. Entering any other terminator causes form input to be terminated.

To use selective enabling, follow these steps:

1. Define a function key User Action Routine (UAR) for each form with input fields.
2. Define the named data item RPG$ENABLE_KEYS or RPG$DISABLE_KEYS for each form with input fields.
3. Generate an object file that contains UAR information.
4. Link this object file with your VAX RPG II program.

Note that you can specify the same or different terminators for each form.

6.4.5.1 Defining a Function Key UAR

VAX RPG II provides the run-time routine needed to define a UAR as part of its VAX/VMS Run-Time Library support. You need to modify the form definition so that the VAX FMS Form Driver will invoke this routine when you enter a terminator.

If you are using the VAX FMS Form Editor Utility, invoke the Editor Utility with the following command:

```
$ FMS/EDIT form_library_name/FORM_NAME=formname
```

In the FORM phase (assign form attributes), specify that you want a UAR for this form, then enter RPG$FUN_KEY_UAR in the field labeled Function Key UAR Name.

If you are using the VAX FMS Translate Utility, describe the UAR by adding the following command to the form definition:

```
FUNCTION_KEY_ACTION_ROUTINE = 'RPG$FUN_KEY_UAR'
```

6.4.5.2 Defining a Named Data Item

If you are using the VAX FMS Form Editor Utility, you must first invoke the Editor Utility. Then, in the DATA phase (Enter Named Data Items), enter RPG$ENABLE_KEYS or RPG$DISABLE_KEYS on the first (Name) line of a named data item, and a list of terminators (separated by commas) that are to be enabled or disabled. (If you enter both the RPG$ENABLE_KEYS and RPG$DISABLE_KEYS items for the same form, the disable entry will be ignored.) Terminator values are the same as those described in Section 6.4.4. For example, to enable PF1/A only, you would define a named data item with the name RPG$ENABLE_KEYS and a value of 321.

If you are using the VAX FMS Translate Utility, describe the named data item by adding a named data clause of the form:

```
NAMED_DATA INDEX = n NAME = 'RPG$ENABLE_KEYS' DATA = '321';
```

6.4.5.3 Generating an Object File Containing UAR Information

When all the forms in a form library have been modified, enter a command of the form:

```
$ FMS/VECTOR formlibraryname
```

VAX FMS generates an object file that contains references to all UARs in the specified form files.

6.4.5.4 Linking Your Program

The object file containing UAR information must be linked with your VAX RPG II program.

6.5 INFDS

The WORKSTN file INFDS is a language feature that you can use to handle errors on WORKSTN file operations. The INFDS data structure, if specified, contains status information, including errors that occurred and an identification of the WORKSTN operation that caused the error. The INFDS also contains status information on normal operations. For example, if a command key was pressed, that status is in the INFDS. The information in the INFDS is updated for each WORKSTN operation. If an error condition occurs, you can use the INFDS information to determine the type of error that occurred, then use that information to control the program logic.

6.5.1 File Description Specification (F)

To use the File Description (F) specification, you must specify the INFDS clause on an F specification continuation line for the WORKSTN file. The following program segment shows F specifications for a WORKSTN file and all of the continuation clauses.

The syntax for these specifications is described in the Continuation-line specification. Note that there is a user-defined name (WRKINF) associated with the INFDS clause.

```
FWRK     CD  F      80            WORKSTN
F                                             KSLN    WRKSLN
F                                             KFMTS   WRKFLB
F                                             KINFDS WRKINF
```

6.5.2 Input Specification (I)

You must define a data structure using Input (I) specifications. The following example shows a data structure definition:

```
IWRKINF     DS
I                                        *STATUS  WRKSTA
I                                        *OPCODE  WRKOPC
I                                        *RECORD  WRKREC
I                                        *FMSSTA  WRKFST
I                                        *FMSTER  WRKTER
```

The syntax of the specifications associated with the INFDS data structure is as follows.

The first line of the input specifications includes the following:

- Columns 7 through 12 contain the name of the data structure.
- Column 18 may contain U, or may be blank.
- Columns 19 through 20 must contain DS.
- All other fields that contain information must be blank.

The second and following lines include the following:

- Columns 44 through 51 contain one of these keywords:
  - *STATUS
  - *OPCODE
  - *RECORD
  - *FMSSTA
  - *FMSTER
- Columns 53 through 58 contain a subfield name. These are the names that will be referenced in the program.
- All other fields that contain information must be blank.

The clauses shown can occur in any order, and the same keyword can occur more than once. You do not need to enter all of the clauses.

Note that this is not a general-purpose data structure; you cannot define other subfields in the INFDS data structure.

6.5.3 Calculation Specification (C)

You can refer to the various fields that are defined in a Calculation specification (C) within the body of the VAX RPG II program. This is of particular importance in user-defined error handling.

6.5.4 *STATUS Keyword

The *STATUS keyword identifies a one-digit numeric subfield with zero decimal positions within the INFDS data structure. This subfield contains a code that identifies the status of the last WORKSTN file operation. The codes are as follows:

Code	Definition
0	No error (form input was terminated by pressing the ENTER key or the RETURN key)
1	Command key (form input was terminated by a command key)
2	Function key (form input was terminated by a function key)
3	Error

Any code in *STATUS greater than 2 is considered to be an error condition. If an error occurs on a READ operation code that has an error indicator, the indicator is set on. If in error occurs on the following operations, the program will terminate.

- A READ operation code with no error indicator
- A primary file read
- Normal output
- EXCPT output

6.5.5 *OPCODE Keyword

The *OPCODE keyword identifies a five-character alphanumeric subfield within the INFDS data structure. This subfield contains a value that identifies which WORKSTN operation was executing when the error occurred. The value inserted in the *OPCODE subfield is OPEN, CLOSE, READ, or WRITE. A value is inserted in the *OPCODE subfield on each WORKSTN file operation.

6.5.6 *RECORD Keyword

The *RECORD keyword identifies an eight-character alphanumeric subfield within the INFDS data structure. If *OPCODE contains WRITE, then *RECORD contains the current form name[1]. Otherwise, *RECORD contains blanks.

6.5.7 *FMSSTA Keyword

The *FMSSTA keyword identifies a longword integer (binary) numeric subfield with zero decimal positions within the INFDS data structure. This subfield indicates the general status returned by the last VAX FMS Form Driver call.

6.5.8 *FMSTER Keyword

The *FMSTER keyword identifies a word integer (binary) numeric subfield with zero decimal positions within the INFDS data structure. This subfield indicates the field terminator entered by the operator to terminate input to the form. If the form was terminated by pressing the RETURN key or the ENTER key, this subfield will contain a value of 0. See the *VAX FMS Form Driver Reference Manual* for terminator codes.

6.5.9 Other Subfields

VAX RPG II does not support the following INFDS subfields:

- SIZE
- MODE
- INP
- OUT
- MAJOR/MINOR

[1] Only the first eight characters of the current form name are in *RECORD.

6.6 Example Program Development Cycle

The previous sections of this chapter have described various features of VAX RPG II screen handling. This section will use an example to step through the various program development steps that can be performed to produce a running interactive application using VAX RPG II and VAX FMS.

The following example shows the VAX RPG II program INVENT that includes three VAX FMS forms:

- PART_NUMBER
- PART_UPDATE
- PART_ERROR

The program uses the following indexed file INVENT.DAT:

```
----------
INVENT.DAT
----------
P01NUT     W2RED  0300135
P02BOLT    W2GREEN0880167
P03SCREW   W1BLUE 0180048
P04SCREW   W2RED  0150143
P05CAM     W1BLUE 0130205
P06COG     W3RED  0200215
P07GEAR    W3GREY 0100234
P08BEARINGW2GREY 0260136
P09BOLT    W1WHITE0060015
```

The goal of the program is to allow you to select parts from the INVENT file and display the inventory information for the part, with the possibility of entering updated data. The PART_NUMBER form will be used to retrieve the three-character part number that will be used as the key into the INVENT file to retrieve the inventory information. The PART_UPDATE form will be used to display the inventory information if the part number is located in the inventory. PART_UPDATE will also allow you to update any inventory field except the part number. PART_ERROR will display an error message if an invalid part number is entered. The program will terminate when a part number of P00 is entered.

First, the forms will be defined using the VAX FMS Form Editor Utility to create the PART_NUMBER as follows:

```
$ FMS/EDIT PART_NUMBER
```

The editing session allows any constant text or labels to be placed on the form. A one-character field to contain the form code is defined. This technique is used often in RPG, when multiple input forms are allowed, to distinguish between the various forms in the Input specifications. You will see later in the program how the form code is initialized for each input form.

The input field for the part number is then defined to contain three characters. This field is defined with X99 in VAX FMS to select only part numbers ending with two numbers. As part numbers are entered, VAX FMS will verify that the last two digits are numeric. A prompt for the part number is defined as constant text.

After exiting the VAX FMS Form Editor Utility, you can obtain information about this first form using the /DESCRIPTION qualifier:

```
$ FMS/DESCRIPTION/BRIEF PART_NUMBER

   VAX FMS Form Description Application Aid   - V2.2 -        Brief Description
   ------------------------------------------------------------------------------

   Form Name = PART_NUMBER
   No Help Form
   Area to Clear = 1:1
   Memory Resident Form Size = 262

            Field Name (Max Index)            Pic(Length)   Access    UARs
   ---------------------------------------    -----------   ------   ------

   FORM_CODE                                      X(1)       DISP
   PN                                             X(3)

   Total Length Required = 4
   Longest Field = 3

   No Named Data

   No User Action Routines
```

Two additional VAX FMS editing sessions are used to create PART_UPDATE.FRM and PART_ERROR.FRM. You can then get a brief text description of these forms using this command:

```
$ FMS/DESCRIPTION/BRIEF PART_UPDATE

   FMS Form Description Application Aid   - V2.2 -        Brief Description
   ------------------------------------------------------------------------------

   Form Name = PART_UPDATE
   No Help Form
   Area to Clear = 1:23
   Memory Resident Form Size = 554
```

```
        Field Name (Max Index)            Pic(Length)   Access    UARs
---------------------------------------   -----------   ------   ------

FORM_CODE                                     X(1)       DISP
PN                                            X(3)       DISP
PNAME                                         C(7)
WHOUSE                                        C(2)
COLOR                                         C(5)
WEIGHT                                        9(3)
QTY                                           9(4)

Total Length Required = 25
Longest Field = 7

No Named Data

No User Action Routines

$ FMS/DESCRIPTION/BRIEF PART_ERROR

FMS Form Description Application Aid   - V2.2 -        Brief Description
------------------------------------------------------------------------

Form Name = PART_ERROR
No Help Form

Area to Clear = 2:23
Memory Resident Form Size = 146

        Field Name (Max Index)            Pic(Length)   Access    UARs
---------------------------------------   -----------   ------   ------

F$0002                                        C(22)

Total Length Required = 22
Longest Field = 22

No Named Data

No User Action Routines
```

You can then create the form library as follows:

```
$ FMS/LIBRARY/CREATE PARTS PART_NUMBER,PART_UPDATE,PART_ERROR
```

Next, you develop the VAX RPG II program that will be used to access these forms and the inventory data file:

```
10FPARTS   CD  F      25            WORKSTN
20FINVENT  UC  F      24R 3AI    1 DISK                      S
30IINVENT
40I                                          1  24 DATA
50IPARTS       01   1 CA
60I                                          2   4 PN
70I            02   1 CB
80I                                          2  25 DATA
90IDATA        DS
100I                                         1   3 PN
110I                                         4  10 PNAME
120I                                        11  12 WHOUSE
130I                                        13  17 COLOR
140I                                        18  20OWEIGHT
150I                                        21  24OQTY
160C                    EXCPTP_NUM
170C                    READ PARTS
180C           PN       CHAININVENT              99
190C   99      PN       COMP 'POO'                   LR
200C   99NLR            EXCPTP_ERR
210C  N99                EXCPTP_UPD
220C  N99                READ PARTS
230C  N99                EXCPTINV
2400PARTS  E                P_NUM
2500                                    K  'PART_NUMBER'
2600                        PN           4
2700                                     1 'A'
2800           E            P_UPD
2900                                    K  'PART_UPDATE'
3000                        DATA        25
3100                                     1 'B'
3200           E            P_ERR
3300                                    K  'PART_ERROR'
3400                                    22 'No part by that number'
3500INVENT E                INV
3600                        DATA        24
```

The WORKSTN file is defined on line **10** as a combined demand file. The record length of 25 is chosen based on the longest form buffer required. This value is obtained from the 'Total Length Required' item listed as part of FMS/DESCRIPTION/BRIEF for each form in the library.

Line **20** shows the definition for the inventory file that will be indexed, updated, shared, and accessed using the CHAIN operation.

The data layout for the inventory file is given in the data structure on lines **90** through **150** and referenced in the input record on lines **30** through **40**.

Lines **50** through **80** show the definition of the form that provided the input. The first character in the WORKSTN file record buffer is examined to see if it is 'A' or 'B'. Note that lines **270** and **310** place an 'A' or 'B' into the first position of the WORKSTN file record buffer. This first byte corresponds to the FORM_CODE field, specified as part of the two input forms, but not displayed on these forms. This special FORM_CODE byte is stored to provide a means for the Input specifications to determine the last input form. You can use other methods, such as field record relation indicators on Input specifications, to determine the current input form.

Lines **240** through **340** show the lines that output forms to the screen. The three forms are listed, along with an EXCPT name, to make it easy to select the appropriate form for display. Note that data is placed in the WORKSTN record buffer in lines **260** through **270**, **300** through **310**, and line **340**. The ending positions for these fields match the information obtained with FMS/DESCRIPTION/BRIEF. The ending position for DATA in line **300** is 25 because the initial byte for the FORM_CODE is not included as part of the DATA data structure.

Lines **160** through **230** include the actions performed during each logic cycle. First, the PART_NUMBER form is displayed on line **160**. Then, input is requested from that form on line **170**. The part number obtained is used on line **180** to retrieve the part information from the inventory file. Line **190** determines if the program should terminate, and line **200** displays the error form for any part number that was in error. Lines **210** through **220** display valid part information and allow you to update part information that is returned to the program by the READ operation in line **220**. Finally, the inventory file is updated on line **230**, and the cycle is repeated until a part number of P00 is supplied.

You can then compile, link, and run the program as follows:

```
$ RPG INVENT
$ LINK INVENT,SYS$LIBRARY:RPGSCR
$ RUN INVENT
```

Note that you include SYS$LIBRARY:RPGSCR only when you link programs that use a WORKSTN file. This object module contains the run-time support to interface with VAX FMS.

The program does not need to be run on a VAX/VMS system with VAX RPG II installed.

You may wish to create an object module library by logging in to the SYSTEM account and setting the default to [SYSLIB]. Then use the following command:

```
$ LIBRARIAN/CREATE RPGSCR RPGSCR
```

Then, each VAX RPG II user may set up the following logical:

```
$ DEFINE LNK$LIBRARY SYS$LIBRARY:RPGSCR
```

This logical definition eliminates the need for explicit references to RPGSCR on LINK commands. If the necessary logicals are not defined, or if you do not properly link your program files, you may get an error message that RPG$SCR is undefined. If you link a program with a WORKSTN file and get a message that FDV$ATERM is undefined, you are attempting to link on a system that does not have VAX FMS installed.

The following program shows a WORKSTN primary file. This is an example of a display-only application; you are not allowed to update the inventory information in this case.

```
FPARTS   CP  F      25            WORKSTN
FINVENT  IC  F      24R 3AI     1 DISK                        S
IINVENT      02
I                                          1  24 DATA
IPARTS       01   1 CA
I                                          2   4 PN
IDATA        DS
I                                          1   3 PN
I                                          4  10 PNAME
I                                         11  12 WHOUSE
I                                         13  17 COLOR
I                                         18  20OWEIGHT
I                                         21  24OQTY
C            PN         CHAININVENT                99
C   99       PN         COMP 'POO'                     LR
OPARTS   D         99
O                                      K  'PART_ERROR'
O                 NLR                  22 'No part by that number'
O                  LR                  22 'Exit requested'
O        D         02
O                                      K  'PART_UPDATE'
O                          DATA        25
O                                       1 'B'
O        D         NLR
O                                      K  'PART_NUMBER'
O                          PN           4
O                                       1 'A'
```

6.7 Converting from S and D Specifications

Some RPG implementations provide a method for describing forms with Screen and Display specifications. These specifications have information similar to that available in the VAX FMS Form Language. Many of the fields in the Screen and Display specifications can be converted to the VAX FMS Form Language using the VAX RPG II Conversion Utility as follows:

```
$ RPG/CONVERT=SD_TO_FMS s-and-d-file
```

The input to the Conversion Utility is a file containing Screen and Display specifications for one or more forms. The output is 0 or more VAX FMS Form Language files (with the FLG file type) containing the VAX FMS Form Language. This corresponds closely to the Screen and Display specification description of the forms. Certain Screen and Display specification features cannot be represented in the VAX FMS Form Language, but must be handled separately. These features include:

- Use of screen line 24
- S and D specification RPG indicators
- Multiple input constants
- Output/no input fields

After the Conversion Utility has created the VAX FMS Form Language files, FMS/TRANSLATE and FMS/LIBRARY can be used. You can then modify the forms either by modifying the FLG files, or by using the VAX FMS Form Editor Utility.

6.7.1 S and D Specification Conversion Utility

The input to the SD converter is a single file containing Screen and Display specifications for 0 or more forms. The output is 0 or more FLG files based on the SD specifications. Each group of SD specifications produces one file with the FLG extension.

6.7.1.1 Invoking the Conversion Utility

The Conversion Utility is invoked with the following qualifier:

```
$ RPG/CONVERT=SD_TO_FMS/NODISPLAY file-spec
```

The /CONVERT=SD_TO_FMS qualifier can be abbreviated to /CONVERT because the default is SD_TO_FMS. Also, /NODISPLAY is required only if the Conversion Utility is run in batch mode, or on a terminal with no TPU support. If you omit /NODISPLAY, the screen will be cleared before the Conversion Utility begins.

To save typing, the following symbol is recommended:

```
$ RPGCNV :== RPG/CONVERT=SD_TO_FMS/NODISPLAY
```

If you define this symbol on your system, the Conversion Utility can be called with the following command:

```
$ RPGCNV file-spec
```

The Conversion Utility can also be invoked with the following command:

```
$ EDIT/TPU/NODISPLAY/SECTION=RPGCONVERT file-spec
```

You can get help by typing the following:

```
$ HELP RPG Conversion  (or)
$ HELP RPG /CONVERT
```

To abort the RPG/CONVERT command operation, press CTRL/Y.

6.7.1.2 Overview of Converter Utility Operation

After you invoke the Conversion Utility, a message will be displayed that shows the file specification and the number of lines that were input from the file containing the Screen and Display specifications. The default file type is SD for the input file. Each Screen specification defines a single form that is output as an FLG file. All Display specifications following a Screen specification are output as fields in the FLG. If no Screen specification is seen, no FLG file is created.

If anything other than a comment, or Screen, Display, or Help specification is seen, the input line is output to SYS$OUTPUT, along with the following message:

```
%RPG-W-SPEC_IGNORED, specification was not an S or D spec
```

Any noncontinuation line with an asterisk in column 7 is recognized as a comment. There is no VAX RPG II support for Help specifications.

An input line that cannot be completely translated into the VAX FMS form language is output to SYS$OUTPUT, along with a message in the following form:

```
%RPG-W-COLS_IGNORED, columns a1-a2,a3-a4 have been ignored
```

For example, if columns 37 through 38 and 49 cannot be translated into the VAX FMS Form Language for a Display specification, then the following message is displayed:

```
%RPG-W-COLS_IGNORED, columns 37 to 38,49 have been ignored
```

In certain cases, additional messages beyond the three mentioned previously will be issued.

If you use a field that is not translated to the VAX FMS Form Language, it is possible that the field will be used on many Display specifications. VAX RPG II will report the error on all specifications that have the field that is being ignored, to enable you to check off Screen and Display specifications as the manual conversion (if necessary) is completed.

Note that the output from the converter is a text file. If you wish to use an interpretation for a Screen or Display specification field that is different from that used by the converter, you can make any necessary adjustments to the VAX FMS Form Language text file that is output by the converter. You can also use the VAX FMS Form Editor Utility to make any desired modifications to the form after the form language is translated.

6.7.1.3 Screen Specification (S)

Columns 1 through 5 are not examined in the Screen specification. If column 6 specifies S, then the specification is recognized as a Screen specification, and the processing takes place. If column 7 is an asterisk (*), the Screen specification is ignored.

The following table identifies the significant Screen specification columns and the corresponding VAX FMS Form Language that is output. All other Screen specification columns are ignored. The VAX FMS Form Language used is a close approximation of the Screen specification meaning. In many cases, the exact functionality of a Screen specification feature cannot be duplicated with VAX FMS.

Begin	End	S Spec	VAX FMS Interpretation
7	14	name	FORM NAME
17	18	01-24	FORM START LINE (defaults to 1)
19	20	00-24	AREA_TO_CLEAR (specified on S specification as lines to clear)
21		not Y	UPPERCASE on all fields
28		Y	RPG$ENABLE_KEYS defined for each letter in columns 64-79
		N	RPG$DISABLE_KEYS defined for each letter in columns 64-79

Your FORM START LINE must be constant and less than 24. If you use a FORM START LINE greater than 23, it will be converted to 23. Your AREA_TO_CLEAR must go no higher than 23, otherwise a value of 23 is used. The default value for the START of the AREA_TO_CLEAR is 1, and you cannot use a value less than 1. The AREA_TO_CLEAR_STOP must be at least as large as the AREA_TO_CLEAR_START, or no AREA_TO_CLEAR clause will be output. Note that START line 1 and AREA_TO_CLEAR 23 is the same as leaving AREA_TO_CLEAR blank.

In addition, BACKGROUND=CURRENT is output for each form.

The converter outputs a NAMED_DATA statement when you specify either selective enabling or disabling. For each capital letter that you put in columns 64 through 79 of the Screen specification, a number indicating the key value of the corresponding VAX FMS terminator is output in the data clause. These are the same default encodings used to describe command keys. Note that if you do not use the default command key definitions, you will need to define the logical name RPG$COMMAND_KEYS and modify the RPG$ENABLE_KEYS or RPG$DISABLE_KEYS named data items appropriately.

6.7.1.4 Display Specification (D)

Columns 1 through 5 are not examined.

The letter D in column 6 indicates that this is a Display specification. Display specifications must be preceded by a Screen specification. If no Screen specification is included, the Display specification is ignored, and the following warning is issued:

```
%RPG-W-SPEC_IGNORED, D specifications must follow an S spec
```

If column 7 is an asterisk (*) the Display specification is ignored, unless the Display specification is a continuation line as described below.

If the previous Display specification had any nonblank character in column 80, then the remainder of the Display specification is processed as a continuation of the previous specification. Otherwise, if column 7 is an *, it is recognized as a comment, and the rest of the specification is not examined.

The following table gives the Display specification columns and the corresponding VAX FMS Form Language that is output. All other Display specification columns are ignored. The VAX FMS Form Language used is a close approximation of the Display specification meaning. In many cases, the exact functionality of a Display specification feature cannot be duplicated with VAX FMS.

Begin	End	D spec	VAX FMS Interpretation
7	14	name	FIELD NAME
15	18	> 0	PICTURE (field length)
19	20	> 0	Screen line number (based on Screen specification starting line). Any line number greater than 23 is converted to 23.
21	22	01-99	Screen column number
26		Y	Input field
27			Data validation
		A	A
		N	N
		D	9

Begin	End	D spec	VAX FMS Interpretation
		(other)	X (Note that you must manually convert S on the Display specification. The VAX FMS record buffer will use the length as specified on the Display specification. The sign must be placed in the last byte position for fields to be displayed. There is no automatic data conversion that overpunches the sign on data input or decodes an overpunched sign on data output.)
28		Y	MUST_FILL
29		Y	RESPONSE_REQUIRED
31		Z,B	RIGHT_JUSTIFIED
35		not Y	AUTO_TAB (column 26 on Display specification must also be Y)
39		Y	BOLD
41		Y	BLINKING
43	44	N	NOECHO
45		Y	REVERSE
47		Y	UNDERLINE
49		Y	UNDERLINE (this handles column separators)
51		blank	UPPERCASE

Indicators anywhere on Display specifications are ignored. If column 56 is C or P, or columns 57 through 79 are nonblank, then columns 57 through 79 are interpreted as VAX FMS background text (except in the case where column 26 is Y). Input constants are handled as VAX FMS NOECHO DISPLAY_ONLY fields.

6.7.2 Manual Conversion

If you manually convert from Screen and Display specifications, there are several items to observe that are described in the following sections.

6.7.2.1 Use of Line 24

VAX FMS reserves line 24 on the terminal screen for messages. No field or constant text can be placed on line 24. Because the converter changes line 24 references to line 23, a conflict could arise if line 23 is already being used. The FMS/TRANSLATE operation will issue an error message if any conflicts are detected.

6.7.2.2 Duplicate Field Names

If two Display specifications for the form contain the same field name, you will get the following errors from FMS/TRANSLATE:

```
%FMS-E-DUPFLDNAM, Field name {field-name} was already specified;
  default name has been assigned.
%FMS-E-ENDNOFRM, Translation was completed with errors.
  No binary form was output.
```

This is because VAX FMS requires unique field names. You can correct the error by first making the field names unique in the file containing the Display specifications, and then rerunning the S and D Conversion Utility. Or, you can modify the VAX FMS Form Language file so that all field names in a form are unique.

6.7.2.3 S and D Specification VAX RPG II Indicators

All occurrences of VAX RPG II indicators in Screen and Display specifications are ignored by the converter. A diagnostic will be issued by the converter for each use of VAX RPG II indicators. You can handle many of the uses of VAX RPG II indicators by conditioning indicators in the VAX RPG II program.

6.7.2.4 Record Buffer Layout

Display specifications provide a way for specifying fields that will be input and not output, and fields that will be output but not input. Thus, Display specifications provide you with two alternative record buffer layouts. VAX FMS provides a single consistent record buffer layout for both input and output fields. The messages shown in the next sections will be displayed for those Display specification fields that may require some modification in the program Input or Output specifications.

To verify the correct field layout, use FMS/DESCRIPTION/BRIEF to display the field layout for the form in VAX FMS format and match the layout to the program Input and Output specifications for the form. Note that the first input constant (if any) will be automatically handled at run time. The VAX FMS record buffer layout will not show the first input constant.

6.7.2.5 Multiple Input Constants

If more than one field has been encountered which has a Y in Display specification column 26, and with a letter other than Y in Display specification column 23, the following message is displayed:

```
%RPG-E-MORE_CONVER,   (1) At most one input constant in a form is converted.
```

This type of field is referred to as an input constant. Many programs with multiple input forms use an input constant so that each form will be able to identify itself when input is done. The converter automatically handles one input constant in each form by creating VAX FMS named data:

```
RPG$CONSTANT_TEXT
RPG$CONSTANT_LENGTH
RPG$CONSTANT_START
```

This data will be used at run time to adjust the field layout so that it can accommodate the input constant. No manual conversion is needed for this first input constant. However, some forms might use two input constants. The MORE_CONVER error message will be displayed if more than one input constant is used. The manual conversion involves adjusting the Output specification ending positions for the form to allow space for the input constant. You must list the input constant as an Output specification constant.

6.7.2.6 Output and No Input Fields

If a field has been encountered which has a Y in Display specification column 23 and with a letter other than Y in Display specification column 26 (that is followed by an input field), the following message is displayed:

```
%RPG-E-MORE_CONVER, (1) Output / no input fields may require I specification changes.
```

This type of field receives output data but no input data. VAX RPG II will allow space in the record buffer for these forms. Even though the form will have the field marked as display only, VAX FMS expects space to be allocated in the WORKSTN record buffer for both Input and Output specifications. There is no automatic conversion for any use of output and no input fields.

Chapter 7

Using Indicators

Indicators are two-character alphabetic, numeric, or alphanumeric entries that condition certain operations within the steps of a normal program cycle. Generally, you use indicators to control the following program decisions:

- Under what conditions VAX RPG II uses a file during program execution
- When and under what conditions VAX RPG II performs calculations
- When VAX RPG II can access a field for input
- Under what conditions VAX RPG II writes a field or record to an output file

This chapter describes types of indicators and explains how to use them.

Section 7.1 describes those indicators that are user-defined within a program. Section 7.2 describes internally defined indicators. Section 7.3 describes how to use indicators as fields.

7.1 User-Defined Indicators

Each indicator has a specific function; however, some indicators can be user-defined for more than one function.

To use an indicator to control program operations, you must first define the conditions under which it is set on or off. Then, check the status (on or off) of the indicator to determine what steps your program should perform.

7.1.1 Record-Identifying Indicators

Record-identifying indicators are used to distinguish the types of input records that will be encountered within your program at run time.

You must first define each record type by specifying an identification code in columns 21 through 41 of the Input specification. Then, associate one of the following indicators with each record type in columns 19 and 20 of the Input specification:

- 01 through 99
- L1 through L9
- LR
- H1 through H9

In the following example, VAX RPG II associates the record-identifying indicator 01 with a record type:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                                |      | Match field
                | ||| + Identifying codes + Format   |      | | Fld rec rel
      File      | ||| |                   | | (PB)   |Field | | |
      name      | ||| |    C      C       C| |Field   |name  | | | Field
      |         | ||| |    Z      Z       Z| |location||     | | | indicatrs
     I|         | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
   0  |   1    |   2    |   3    |   4    |   5    |   6    |   7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         * *** *---   *---   *---   .**---*---**     * * * * * * ....
     IINPUT     AA  01  20 C1
```

ZK-4387-85

In this example, if the character in the twentieth position is the number 1, the indicator 01 is set on. Then, you can use the indicator to condition program operations for that particular record type.

The record-identifying indicator for a particular record type is set on when VAX RPG II processes a record of that type. For a primary or secondary file, the record-identifying indicator is set on before total-time calculations. For a chained or demand file, the record-identifying indicator is set on immediately after the record is read. (See Chapter 8 for information on file usage with VAX RPG II.) In either case, it is set off when the program reaches the end of the current program cycle (after detail-time output).

You can use record-identifying indicators to condition both detail-time and total-time calculation and output operations for each record type. The following example shows how record-identifying indicators can be used to condition program operations:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
        .
        .
        .
11 ISALES   AA  01   1 CJ
12 I                                              2   50ITEM
13 I                                             10   16 DESCR
14 I                                             20  242SALES
15 I                                             30  342COST
16 I                                             40  432PROFIT
17 C   01       SALES     SUB  COST       NET      52
18 C   01       TSALES    ADD  SALES      TSALES   62
19 C   01       TCOST     ADD  COST       TCOST    62
20 C            TPROFI    ADD  NET        TPROFI   62
21 OREPORT  H   201   1P
22 O        OR        OF
23 O                          UDATE Y     8
24 O                                     44 'JANUARY SALES REPORT'
25 O                          PAGE       72
26 O                                     68 'PAGE'
27 O         H 22     1P
28 O        OR        OF
29 O                                      5 'ITEM'
30 O                                     23 'DESCRIPTION'
31 O                                     41 'SALES'
32 O                                     56 'COST'
33 O                                     72 'PROFIT'
34 O         D        01
35 O                          ITEM  3     5
36 O                          DESCR      25
37 O                          SALES 1    41
38 O                          COST  1    57
39 O                          PROFIT1    72
40 O         T 1      LR
41 O                                     30 'TOTALS'
42 O                          TSALES1    41 '$'
43 O                          TCOST 1    57 '$'
44 O                          TPROFI1    72 '$'
```

ZK-4388-85

In the preceding example:

- Line 11 causes VAX RPG II to begin reading records from the file SALES. The identification code (columns 21 through 41) groups these records according to a code that represents the month. If the code for the month is J, the record-identifying indicator 01 is set on.
- Lines 17 through 19 use the same record-identifying indicator 01 to condition detail-time calculations. VAX RPG II performs the calculation each time a record of the type described on line 11 is read:
- Line 34 uses the same record-identifying indicator to condition detail-time output. VAX RPG II performs the output operation each time a record of the type described on line 11 is read.

The output file produced by this program might appear as follows:

```
0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
2/4/83                        JANUARY SALES REPORT                      PAGE   1
ITEM          DESCRIPTION          SALES          COST          PROFIT
10005            AMMONIA           60.30          50.00          10.30
10982            MATCHES          295.00         205.00          90.00
22650            NUTMEG           209.00         170.00          39.00
TOTALS                           $564.30        $425.00         $139.30
```

If you use the CHAIN or READ operation to retrieve records, the program does not set the record-identifying indicators off until the beginning of the next program cycle. Be careful when performing more than one CHAIN or READ operation for a file with multiple record types, because more than one indicator can be set on during a single cycle.

7.1.2 Field Indicators

Field indicators test a field in an input record for a positive, negative, zero, or blank value.

You can use any of the following indicators as field indicators:

- 01 through 99
- H1 through H9

Field indicators are used to test the value of a field in the following ways:

- For a positive value, specify a field indicator in columns 65 and 66 of the Input specification.

- For a negative value, specify a field indicator in columns 67 and 68 of the Input specification.
- For a zero or blank value, specify a field indicator in columns 69 and 70 of the Input specification.

Field indicators are set when the data in the field is extracted from the record. After a field indicator is set on, it remains on until the next time the field is extracted, unless it is set off by another use of the same indicator in the program. A field indicator can be used to condition any detail-time or total-time operations. However, at total time, the field indicators assigned to fields from a primary or secondary file retain the setting from the previous detail-time cycle.

The following example shows how field indicators can be used to condition a calculation:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      :
      :
      :
24 IPARTLIS AA  01   1 CF
25 I                                        2  100INVCDE      112233
      :
      :
      :
41 C   11       ITEM       MULT FACT1     ORDER   62H
42 C   22       ITEM       MULT FACT2     ORDER   62H
43 C   33       ITEM       MULT FACT3     ORDER   62H
```

ZK-4389-85

In the preceding example:

- Line 25 tests the value of the field INVCDE to see if it contains a positive value, a negative value, or a zero value. The following is a list of indicators that are set on for each value:
  - If the field contains a positive value, indicator 11 is set on and indicators 22 and 33 are set off.

- If the field contains a negative value, indicator 22 is set on and indicators 11 and 33 are set off.
- If the field contains a zero value, indicator 33 is set on and indicators 11 and 22 are set off.

- Lines 41 through 43 calculate the number of parts to order according to the status of the field indicators.

7.1.3 Resulting Indicators

Resulting indicators condition operations that depend on the result of a calculation. These indicators specify the test (> , <, or =) and indicate the result of the calculation. If the result matches the test, the indicators are set on. The indicators are set off if the next instance of that calculation does not satisfy the indicated test or if the same indicator is used again in the program.

You specify resulting indicators in columns 54 through 59 of the Calculation specification. You can use any of the following indicators as resulting indicators:

- 01 through 99
- L1 through L9
- LR
- H1 through H9
- OA through OG, and OV
- U1-U8
- KA through KZ
- K0 through K9

Resulting indicators in columns 54 and 55 test for the following conditions:

- The result field contains a positive value after an arithmetic operation.
- When comparing two factors, the value in factor 1 is higher than the value in factor 2 in a COMP operation.
- The value of the element found in factor 2 is higher than the value in factor 1 in a LOKUP operation.
- The record is not found in a CHAIN operation.
- Each bit defined in factor 2 is set off in the result field for a TESTB operation.

Resulting indicators in columns 56 and 57 test for the following conditions:

- The result field contains a negative value after an arithmetic operation.
- The value in factor 1 is lower than the value in factor 2 in a COMP operation.
- The value of the element found in factor 2 is lower than the value in factor 1 in a LOKUP operation.
- The defined bits in factor 2 are of mixed status (some on, some off) in the result field for a TESTB operation.
- A subprogram returns with an error status from a CALL operation.
- A READ operation is executed on a WORKSTN file, and an error *other* than EOF occurred.

Resulting indicators in columns 58 and 59 test for the following conditions:

- The result field contains a zero after an arithmetic operation.
- The value in factor 1 is equal to the value in factor 2 in a COMP operation.
- The value of the element found in factor 2 is equal to the value in Factor 1 in a LOKUP operation.
- An end-of-file condition occurs for the demand file in a READ operation.
- A READ operation is executed on a WORKSTN file and the last form displayed has no input fields, or no form has been displayed since the previous READ.
- Each bit defined in factor 2 is set on in the result field for a TESTB operation.

Resulting indicators are also used with the SETON and SETOF operation codes to specify that the indicators are to be set on or off.

The following example shows how resulting indicators can be used to control program operations:

```
                                                    Field length
      Control level                                 | Decimal positions
      |                                             | |Half adjust (H)
      | Indicators          Operation               | ||
      | |                   |                       | ||Resulting
      | |         Factor    |    Factor    Result   | |||indicators
      | |         1         |    2         field    | |||+ - 0
     C| NxxNxxNxx|          |    |         |        | |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *         *    *         *      *--*** * * *
  10 C             SEARCH    LOKUPTAB1               10  11
  20 C             FLD1      COMP 100                222324
  30 C             KEY       CHAINFILE1              32
  40 C                       TESTB'123'    TEST      404142
  50 C                       READ FILE1                  50
  60 C                       SETOF                   1011
  70 C             FLD1      SUB  FLD2     RES       606162
```

ZK-4390-85

In the preceding example:

- Line 10 causes VAX RPG II to search for field SEARCH in table TAB1. If VAX RPG II can find an entry that is equal to the search word, indicator 11 is set on. If VAX RPG II can find an entry that is nearest to and higher in sequence than the search word, indicator 10 is set on.
- Line 20 causes VAX RPG II to compare the contents of FLD1 with the numeric literal 100. If the contents of FLD1 are greater than 100, indicator 22 is set on and indicators 23 and 24 are set off. If the contents of FLD1 are less than 100, indicator 23 is set on and indicators 22 and 24 are set off. If the contents of FLD1 equal 100, indicator 24 is set on and indicators 22 and 23 are set off.
- Because the input file is an indexed file, line 30 instructs VAX RPG II to retrieve a record using the key KEY from the indexed file FILE1. If the record is not found, indicator 32 is set on. Otherwise, indicator 32 is set off.
- Line 40 causes VAX RPG II to test the bits 1, 2, and 3 in TEST. If the bits are all off, indicator 40 is set on and indicators 41 and 42 are set off. If some bits are on and some are off, indicator 41 is set on and indicators 40 and 42 are set off. If the bits are all on, indicator 42 is set on and indicators 40 and 41 are set off.

- Line 50 causes VAX RPG II to read the next record from FILE1. If an end-of-file condition occurs, indicator 50 is set on. Otherwise, indicator 50 is set off.
- Line 60 sets indicators 10 and 11 off.
- Line 70 causes VAX RPG II to evaluate the contents of the result field after the SUB operation. If the result field contains a positive value, indicator 60 is set on and indicators 61 and 62 are set off. If the result field contains a negative value, indicator 61 is set on and indicators 60 and 62 are set off. If the result field contains a zero value, indicator 62 is set on and indicators 60 and 61 are set off.

7.1.4 Control-Level Indicators

Control-level indicators indicate that a particular field in the input record is a control field. Each time VAX RPG II reads a record that contains the control field, it compares the data in the control field with the current value of the control field. If the contents change, a control break occurs, the control-level indicator is set on, and the value in the control field becomes the new current value.

You associate a control-level indicator (L1 through L9) with an input field by specifying the indicator in columns 59 and 60 of the Input specification.

The lowest control level indicator is L1 and the highest is L9. When you use more than one control-level indicator and a higher level control-level indicator is set on because of a control break, VAX RPG II automatically sets on all lower level control-level indicators. When you use a control-level indicator as another type of indicator (for example, as a record-identifying indicator), and that indicator is set on, lower level control-level indicators are not automatically set on.

A control break is likely to occur after the first record with a control field is read. VAX RPG II compares the data in the control field with hexadecimal zeros. Therefore, VAX RPG II bypasses total-time calculation and output operations for the first record containing control fields.

All control-level indicators are set on before total-time calculations when the last-record (LR) indicator is on. All control-level indicators are set off after detail-time output.

The following example shows how to use three different control-level indicators to condition calculation and output operations:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1 H
   2 FSLSCAR  IP  F      14            DISK
   3 FSLSREP  O   F     132      OF     PRINTERTMP
   4 ISLSCAR  AA  01
   5 I                                        1   20BRANCHL3
   6 I                                        3   40SLSPERL2
   7 I                                        5   90CUSTNOL1
   8 I                                       10  142SLSAMT
   9 C   01       SLSAMT    ADD  CUSTOT    CUSTOT  62
  10 CL1          CUSTOT    ADD  SPTOT     SPTOT   72
  11 CL2          SPTOT     ADD  BRTOT     BRTOT   72
  12 CL3          BRTOT     ADD  FINTOT    FINTOT  82
  13 OSLSREP  H  201   1P
  14 O        OR       OF
  15 O                                UDATE Y    9
  16 O                                          25 'SALES REPORT'
  17 O                                          38 'PAGE'
  18 O                                PAGE       43
  19 O        H  1     1P
  20 O        OR       OF
  21 O                                           6 'BRANCH'
  22 O                                          22 'SALESPERSON'
  23 O                                          35 'CUSTOMER'
  24 O                                          46 'SALES'
  25 O        H  2     1P
  26 O        OR       OF
  27 O                                           4 'NO'
  28 O                                          19 'NO'
  29 O                                          32 'NO'
  30 O                                          46 'AMOUNT'
  31 O        D  1     01
  32 O                                BRANCHZ    4
  33 O                                SLSPERZ   16
  34 O                                CUSTNOZ   30
  35 O                                SLSAMT1   45
  36 O        T  2     L1
  37 O                                CUSTOT1B  45
  38 O                                          46 '*'
  39 O        T 12     L2
  40 O                                          42 'TOTAL SALESPERSON'
  41 O                                SLSPERZ   45
  42 O                                SPTOT 1B  54
  43 O                                          56 '**'
  44 O        T  3     L3
  45 O                                          46 'TOTAL BRANCH NO'
  46 O                                BRANCHZ   49
  47 O                                BRTOT 1B  61
  48 O                                          65 '***'
  49 O        T  1     LR
  50 O                                          46 'FINAL TOTAL'
  51 O                                FINTOT1   59 '$'
  52 O                                          64 '****'
```

ZK-4391-85

In the preceding example:

- Lines 5 through 7 assign three control-level indicators, one each to three different control fields. The specification associates the highest control-level indicator (L3) to the most significant input field, BRANCH. The specification associates the next highest control-level indicator to SLSPER and the lowest control-level indicator to CUSTNO.

  If the value of BRANCH changes from the previous record, indicator L3 is set on, which automatically sets on indicators L2 and L1. These three indicators can be used to condition calculation and output operations.
- In line 10, when indicator L1 is on, VAX RPG II adds the amount of the customer sale to the total sales for a particular salesperson. In line 11, when indicator L2 is on, VAX RPG II adds the total sales for the salesperson to the total sales for each branch. In line 12, when indicator L3 is on, VAX RPG II adds the total sales for each branch to compute the final total.
- Line 36 causes VAX RPG II to output the total sales for each customer number when indicator L1 is on.
- Line 39 causes VAX RPG II to output the total sales for each salesperson when indicator L2 is on.
- Line 44 causes VAX RPG II to output total sales for each branch when indicator L3 is on.

You can assign the same control-level indicator to more than one control field. These fields are called split-control fields. The following example shows how to use split-control fields:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)              Decimal positions
                | ||Record identifying indicator      |      Control level
                | |||                                 |      | Match field
                | ||| + Identifying codes + Format    |      | | Fld rec rel
     File       | ||| |                     | | (PB)  |Field | | |
     name       | ||| |    C       C       C| |Field  |name  | | | Field
      |         | ||| |    Z       Z       Z| |location|      | | | indicatrs
     II         | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890
     **        * *** *---   *---   *---   .**---*---**    * * * * * * ....
   5 I                                        1   20BRANCHL1
   6 I                                        3   40SLSPERL1
   7 I                                        5   90CUSTNOL1
```

ZK-4392-85

In this example, the fields BRANCH, SLSPER, and CUSTNO combine to form the control field. When VAX RPG II compares the data in these fields with the same fields in a previous record, indicator L1 is set on when the data changes.

7.1.5 Overflow Indicators

When the printer reaches the overflow line that signals the end of the page, VAX RPG II sets on the overflow indicator assigned to that printer output file. (See Chapter 9 for information on printer output files.)

You define overflow indicators OA, OB, OC, OD, OE, OF, OG, and OV in columns 33 and 34 of the File Description specification.

In the following example, after reaching the overflow line, VAX RPG II sets on the overflow indicator OF. Then, the printer moves to the top of the next page and outputs the heading lines.

```
                Type (HDTE)              Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)    | X                 ------------
                ||Space                  | Y date edit       Y Y 1 A J
                ||| Skip                 | Z zero suppress   Y N 2 B K
                ||| |                    |                   N Y 3 C L
                ||| |   Indicators       |Blank-after (B)    N N 4 D M
     File       ||| |   |        Field    ||End position
     name       ||| |   |        name     |||   Format (PB)
      |         ||| |   |         |       |||   |
     O|         ||BAB A NxxNxxNxx|       |||   |+ Constant or edit word +
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *       ***---**                          ....
  14 OSLSREP    H  201    OF
  15 O                              UDATE Y    9
  16 O                                        25 'SALES REPORT'
  17 O                                        38 'PAGE'
  18 O                              PAGE      43
```

ZK-4393-85

See Chapter 9 for a full description of the overflow routine and overflow indicators.

7.1.6 K Indicators

K indicators (KA through KZ and K0 through K9) can be used to condition calculations, output records, and output fields. They can also be used as resulting indicators. K indicators are set on or off if a WORKSTN file is used. WORKSTN files are used when you want to interact with your VAX RPG II program at a terminal screen. See Chapter 6 for details.

In the following program, a K indicator is set on and is displayed when its associated cursor control key is typed. CTRL/Z is pressed to end the program. Note that this use of K Indicators does not require a WORKSTN file.

```
    0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890
     F*+
     F* File: READ_CURSOR.RPG
     F*
     F* This RPG II program demonstrates the use of the RTL routine
     F* SMG$READ_KEYSTROKE to read a keystroke from the terminal.
     F*
     F* The program takes input from the terminal until CTRL/Z is typed.
     F* If any of the four cursor positioning keys is typed, a string
     F* is displayed corresponding to the key.
     F*
     F* Build this program using the following commands:
     F*
     F* $ RPG READ_CURSOR
     F* $ CREATE SMGDEF.MAR
     F*        .TITLE SMGDEF - Define SMG$ constants
     F*        .Ident /1-000/
     F*        $SMGDEF GLOBAL
     F*        .END
     F* $ MACRO SMGDEF
     F* $ LINK READ_CURSOR,SMGDEF
     F*-
     FTTY     D   V       5            TTY
     C                    CALL REAKEY
     C* External definitions for SMG routines.
     C           CREKB    EXTRN'SMG$CREATE_VIRTUAL_KEYBOARD'
     C           DELKB    EXTRN'SMG$DELETE_VIRTUAL_KEYBOARD'
     C           REAKEY   EXTRN'SMG$READ_KEYSTROKE'
     C* External definitions for SMG terminators.
     C           T_UP     EXTRN'SMG$K_TRM_UP'
     C           T_DOWN   EXTRN'SMG$K_TRM_DOWN'
     C           T_LEFT   EXTRN'SMG$K_TRM_LEFT'
     C           T_RIGHT  EXTRN'SMG$K_TRM_RIGHT'
     C           T_CTRLZ  EXTRN'SMG$K_TRM_CTRLZ'
     C* Create the virtual keyboard.
     C  N99               CALL CREKB
     C                    PARM           KB_ID   90 WL
     C                    SETON                     99
     C* Read a keystroke.
     C                    CALL REAKEY
     C                    PARM           KB_ID   90 RL
     C                    PARM           T_CODE  50 WW
     C* Turn on an indicator if a cursor positioning key was typed.
     C           T_CODE   COMP T_UP                  KA
     C           T_CODE   COMP T_DOWN                KB
     C           T_CODE   COMP T_LEFT                KC
     C           T_CODE   COMP T_RIGHT               KD
     C* Turn on LR to quit if CTRL/Z was typed.
     C           T_CODE   COMP T_CTRLZ               LR
     C* Display a message if a cursor positioning key was typed.
     C   KA      'UP'     DSPLYTTY
     C   KB      'DOWN'   DSPLYTTY
     C   KC      'LEFT'   DSPLYTTY
     C   KD      'RIGHT'  DSPLYTTY
     C* Delete the virtual keyboard.
     CLR                  CALL DELKB
     CLR                  PARM           KB_ID   90 RL
```

.EL

ZK-4661-85

7.2 Internally Defined Indicators

There are some indicators that you do not need to define; VAX RPG II defines them for you. This section describes internally defined indicators and explains how to use them.

7.2.1 First-Page Indicator

When you specify a first-page (1P) indicator, it is set on at the start of the program and set off after detail-time output but before the first record is read. Therefore, you can use the 1P indicator to condition these heading lines you want printed before VAX RPG II processes the first record.

You specify the 1P indicator, which is always represented by 1P, in columns 24 and 25, 27 and 28, or 30 and 31 of the Output specification.

The following example shows how to use the 1P indicator to print a header on the first page of a report:

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)    | X                 ------------
                ||Space                  | Y date edit       Y  Y  1 A J
                ||| Skip                 | Z zero suppress   Y  N  2 B K
                ||| |                    |                   N  Y  3 C L
                ||| |   Indicators       |Blank-after (B)    N  N  4 D M
     File       ||| |   |        Field    ||End position
     name       ||| |   |        name     |||  Format (PB)
     |          ||| |   |        |        |||  |
    0|          ||BAB A NxxNxxNxx|        |||  |+ Constant or edit word +
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *      ***---**                          ....
     OOUTPUT  H  201    1P
     O        OR         OF
     O                           UDATE Y    8
     O                                     43 'SALES REPORT'
     O                           PAGE       72
     O                                     67 'PAGE'
     O         H 22      1P
     O        OR         OF
     O                                      5 'ITEM'
     O                                     23 'DESCRIPTION'
     O                                     41 'SALES'
     O                                     56 'COST'
     O                                     72 'PROFIT'
```

ZK-4385-85

The following heading lines are printed on the first page:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
 5/19/83                          SALES REPORT                          PAGE    1
 ITEM       DESCRIPTION              SALES          COST          PROFIT
```

You can use the 1P indicator to condition only detail or heading output lines. If you have a detail or heading output line conditioned by no indicators or all negative indicators, use a negative 1P (N1P) indicator to prevent this line from being output on the first cycle before the first record is read.

7.2.2 Last-Record Indicator

Like the first cycle in a VAX RPG II program, the last cycle differs from all other program cycles. After VAX RPG II processes the last record in all primary and secondary files for which you specified processing until the end-of-file, the last-record (LR) indicator is set on, along with all the other control-level indicators you specified. The LR indicator causes VAX RPG II to perform all total-time calculations and output operations conditioned by any control-level indicators and by the LR indicator.

The LR indicator is always represented by LR, as shown in the following example:

```
                Type (HDTE)              Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)    | X                 ------------
                ||Space                  | Y date edit       Y Y 1 A J
                ||| Skip                 | Z zero suppress   Y N 2 B K
                ||| |                    |                   N Y 3 C L
                ||| |   Indicators       |Blank-after (B)    N N 4 D M
 File           ||| |   |        Field   ||End position
 name           ||| |   |        name    |||  Format (PB)
  |             ||| |   |        |       |||  |
01              ||BAB A NxxNxxNxx|       |||  |+ Constant or edit word +
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
  **            ***** * *         *      ***---**                          ....
  0             T 1       LR
  0                                          30 'TOTALS'
  0                                 TSALES1  41 '$'
  0                                 TCOST 1  57 '$'
  0                                 TPROFI1  72 '$'
```

ZK-4384-85

The following information is printed only after processing the last record:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
                          TOTALS    $564.30         $425.00         $139.30
```

If your program does not contain a primary input file, you must set on the LR indicator to end the execution of the program. If your program sets on the LR indicator, VAX RPG II automatically sets on all control-level indicators just before total-time calculations. If the LR indicator is set on during total-time calculations, VAX RPG II does not automatically set on all control-level indicators.

7.2.3 Matching-Record Indicator

When you use more than one primary and secondary file, VAX RPG II multifile logic supplies you with a method of selecting the next record to process. You can designate one or more fields in each record to be the matching fields (columns 61 and 62 of the Input specification). When the fields from a primary file and one or more of the secondary files match, the matching-record (MR) indicator is set on. The MR indicator remains set on while processing the records from the primary and secondary files that match. See Chapter 8 for a complete discussion of multifile processing.

At the beginning of detail time, the MR indicator is set on or off, as determined by the matching status of the record to be processed. Therefore, at total time, the MR indicator reflects the matching status of the previous record with the record to be processed.

7.2.4 External Indicators

You can use external indicators to condition any operation in your program. External indicators, which are always represented by U1 through U8, can also appear in columns 71 and 72 of the File Description specification and in columns 54 through 59 of the Calculation specification. External indicators can also be used as resulting indicators.

To use the external indicator, you must also assign the logical name RPG$EXT_INDS to an external indicator using the DEFINE or ASSIGN command, as shown in the following example:

```
$ DEFINE RPG$EXT_INDS "external-indicator-list"
```

An external indicator is set on by specifying it in the external-indicator list. An external indicator is set off by not specifying it in the external-indicator list.

The following example sets on external indicators U1, U5, and U4 and sets off external indicators U2, U3, U6, U7, and U8:

```
$ DEFINE RPG$EXT_INDS "154"
```

When you turn external indicators on or off in a program, the RPG$EXT_INDS is updated automatically for possible subsequent use by another program.

When you use an external indicator to condition a file, the file is opened only when the external indicator is set on. If the external indicator is set off, input files being processed sequentially are treated as if the end-of-file were reached. Use the same external indicator as a conditioning indicator to control calculation and output operations for those files being processed by methods other than sequential processing. Otherwise, a run-time error will occur when you attempt input or output operations on a file that was not opened because the external indicator was set off.

7.2.5 Halt Indicators

You can use halt indicators H1 through H9 as record-identifying indicators, field indicators, or resulting indicators to stop a program when a specific condition occurs.

When you use a halt indicator as a record-identifying indicator, a specific type of record causes the halt.

In the following example, a halt indicator is used as a record-identifying indicator, causing the program to check the character in position 80 of records read from the input file FILEIN. If the eightieth character is not an S, the halt indicator H1 is set on and the program will halt execution. A run-time message will then be displayed saying that this indicator is set on.

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                                |      | Match field
                | ||| + Identifying codes + Format   |      | | Fld rec rel
 File           | ||| |                   | | (PB)   |Field | | |
 name           | ||| |    C      C      C| |Field   |name  | | | Field
 |              | ||| |    Z      Z      Z| |location||     | | | indicatrs
II              | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IFILEIN    AA  H1  80NCS
     I              02  80 CS
     I                                      1  10 FIELD1
```

ZK-4383-85

When a halt indicator is used as a field indicator, a halt occurs because of erroneous input data.

The following program uses a halt indicator as a field indicator. When a record is read from the input file FILEIN, FIELD1 is checked for a negative value. If FIELD1 contains a negative value, the indicator H2 is set on. After this record has been processed, the program will halt.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *         *          *    *          *      *--*** * * *
        .
        .
        .
     IFILEIN  AA  01
     I                                          1   50FIELD1         H2
     C  NH2                 SQRT FIELD1    FDL2    95
```

ZK-4382-85

When a halt indicator is used as a resulting indicator, a halt occurs when calculations produce erroneous results during either detail time or total time.

In the following example, a halt indicator is used as a resulting indicator. If the field FIELD1 is equal to zero, the indicator H4 is set on. After the current record is processed, the program halts.

```
                                                    Field length
      Control level                                 |  Decimal positions
      |                                             |  |Half adjust (H)
      |  Indicators          Operation              |  ||
      |  |                   |                      |  ||Resulting
      |  |         Factor    |    Factor    Result  |  ||indicators
      |  |         1         |    2         field   |  ||+ - 0
     C|  NxxNxxNxx|          |    |         |       |  ||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *         *          *    *          *      *--*** * * *
     C            FIELD1     SUB  59.0       FIELD1         H4
     C  NH4       FIELD2     DIV  FIELD1     FIELD1
```

ZK-4381-85

When a halt indicator is set on, a halt does not occur immediately. Before the program halts, it completes the current cycle and processes the record that caused the error condition.

If any halt indicators are on after detail-time output, a run-time error occurs.

Halt indicators can also be used as field-record-relation indicators and to condition calculation and output operations. See Chapter 15 for more information on using halt indicators as field-record-relation indicators. If you are converting from another RPG II implementation, you might find an incompatibility with the halt indicators H1 through H9. For example, when IBM or Honeywell RPG II encounters a halt indicator, the operator is prompted to respond to a continue message. This allows the program to continue to execute. When VAX RPG II encounters a halt indicator, the program stops executing. The following code can be added to your program as a workaround. This will prompt the operator and provide the option of continuing:

```
F*  File spec for the screen display
F*
FTTYFILE D   F      81            TTY
C*
C*  First time only
C*
C  N99               MOVEL'continue'PROMPT 10
C  N99               MOVE '? '      PROMPT
C*
C*  If halt indicator is on prompt for continue
C*
C   H1      PROMPT   DSPLYTTYFILE   ANSWR   1
C*
C*  Test operator's response if YES then setof HALT indicator
C*  to continue and blank out answer
C*
C   H1      ANSWR    COMP 'Y'                    98
C   98               SETOF                    H198
C                    MOVE ' '       ANSWR
```

The testing of the halt indicator should come after the detail-time calculations.

If this workaround is running from within a command procedure, SYS$INPUT needs to be redefined to your terminal. For example:

```
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ RUN SAMPLE
```

For more information on running the program from a command procedure, see the *Guide to Using DCL and Command Procedures on VAX/VMS.*

7.3 Using Indicators as Fields

The *IN, *IN,n, and *INxx are special words that allow you to use predefined indicators in your program. Sections 7.3.1 and 7.3.2 describe each type of special word.

7.3.1 *IN and *IN,n

The special word *IN is a predefined array with 99 one-position character elements. The elements in this array represent indicators 01 through 99. Use *IN,n, where n is the array index, to reference an indicator. For example, *IN,54 refers to indicator 54.

The elements in this array can assume only two character values—1 and 0. If you reference an indicator using *IN,n and the contents of the element are 0, the indicator is set off. If the contents of the element are 1, the indicator is set on.

You can use either the array or the array element to reference an indicator anywhere any other one-character array or array element can be used. You cannot, however, specify the entire array *IN as the Result field of a PARM operation. To prevent unpredictable results when modifying an element in *IN, assign the character literal 0 or 1 to *IN.

In the following example, the program tests whether the setting for indicator 15 is equal to the setting for indicator 20. In the next line, indicator 20 is set on. Using the MOVE operation to transfer 1 to *IN,20 produces the same result as using the SETON operation code to set on indicator 20.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |       Factor      |    Factor    Result|   |  |||indicators
     | |       1           |    2         field |   |  |||+ - 0
     C| NxxNxxNxx|         |    |         |     |   |  |||> < = +- Comments --+
   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *       *         *    *         *     *--*** * * *
     C          *IN,20    COMP *IN,15                    99
     C                    MOVE '1'       *IN,20
```

ZK-4380-85

7.3.2 *INxx

The special word *INxx is a predefined one-position character field where xx represents any indicator except the first-page (1P), overflow, or external indicators. Like *IN, it can contain only the character 0 or 1.

You can use *INxx anywhere any other one-character field can be used, except as the result field of a PARM operation.

In the following example, the value of the MR indicator is compared to the value of M. If they are the same, indicator 99 is set on. The MR indicator is represented as *INMR.

```
     0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890
       .
       .
       .
     IFILE1       01
     I                                    1  20 TEXT
     I                                   19  20 MATCH   M1
     IFILE2       02
     I                                    2  21 TEXT
     I                                   20  21 MATCH   M1
     C            *INMR     COMP M                          99
```

ZK-4379-85

Chapter 8

Using Files

A file is a collection of information that is organized into groups or sections called records. Each record consists of one or more blocks of characters or numbers called fields.

This chapter explains the VAX RPG II file organizations and record operations that are implemented through VAX Record Management Services (RMS). For additional information on file organization and file and record operations, see the *VAX Record Management Services Reference Manual* and the *Guide to VAX/VMS Disk and Magnetic Tape Operations.*

8.1 File Characteristics

Files are created, modified, and processed according to their respective characteristics, which are chosen to make the most efficient use of both system resources and the data in the files. Significant characteristics of files used in VAX RPG II programs are the file *type, designation,* and *mode of processing.*

The file type used in VAX RPG II programs determines how the records in the files are processed. See Chapter 15 for information on file type. The file types include:

- Input (I)
- Output (O)
- Update (U)
- Display (D)
- Combined (C)

See Chapter 6 for detailed information on Combined (WORKSTN) files.

File designations define the order in which VAX RPG II processes files. See Chapter 15 for information on file designations. The file designations include:

- Primary (P)
- Secondary (S)
- Record-address (R)
- Chained (C)
- Preexecution-time table or array (T), (A)
- Demand (D)
- Full-procedural (F)

8.2 File Names

Columns 7 through 14 (File name) of the File Description specification define the file name. VAX RPG II uses the entry in columns 7 through 14 (File name) and the entry in columns 47 through 52 (Symbolic device) to associate the file name with the VAX/VMS file specification. The default VAX RPG II file type is DAT.

You can use a logical name for the entry in columns 47 through 52 (Symbolic device), and then assign a VAX/VMS file specification to the logical name. If you assign a full file specification to the logical name, VAX RPG II ignores the entry in columns 7 through 14 when determining the file specification. If you do not assign the file-name part of the file specification to the logical name, VAX RPG II uses the entry in columns 7 through 14 when determining the file specification. If you do not assign a file type to the logical name, VAX RPG II uses DAT.

If you do not specify an entry in columns 47 through 52, you can use a logical name as the entry in columns 7 through 14 for the VAX/VMS file specification. If you do not specify a logical name as the entry in columns 7 through 14, the file specification will consist of the file name in columns 7 through 14 and the file type DAT.

The entry in columns 7 through 14 is used as the VAX Run-Time Library default file name string. The entry in columns 47 through 52 (symbolic device) is used as the VAX/VMS Run-Time Library file name string. See the *VAX Record Management Services Reference Manual* for information about file-name strings and default file-name strings.

8.3 Record Formats

The records in a file can all be the same length (fixed) or of different lengths (variable). Variable-length records often use disk storage space more efficiently. The characteristics and requirements of individual applications should be carefully considered when you decide whether to use fixed-length or variable-length records.

8.4 File Types

You can use files in four ways:

- As input to a VAX RPG II program
- As output from a VAX RPG II program
- As an update file in which the records are changed by the program
- To interact with the terminal screen using the VAX Forms Management System (VAX FMS) (see Chapter 6 for details)

8.5 File Organizations

The organization of a file determines how the records in it are arranged. VAX RPG II allows three different file organizations:

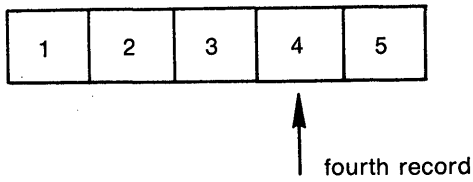
- Sequential
- Direct
- Indexed

Sections 8.5.1 through 8.5.3 describe these file organizations.

8.5.1 Sequential Organization

Sequential file organization is available on all types of devices. Sequential files contain records in the order that they were written. The first logical record in the file is always in the first physical record position, the second logical record in the file is always in the second physical record position, and so on. If you need to access the fourth logical record, you can find it between the third and fifth physical records, as shown in Figure 8–1.

Figure 8–1: Sequential File Organization



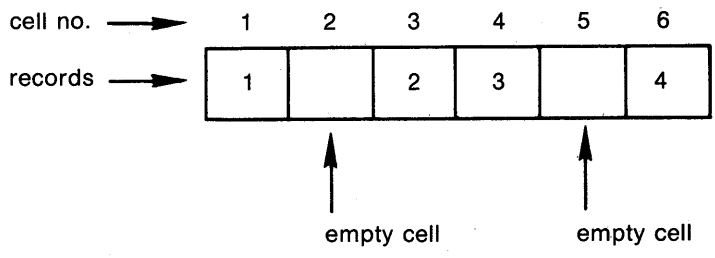
ZK-1462-83

You can retrieve records from a sequential file either sequentially, by reading through the entire file from beginning to end, or randomly, by using relative record numbers or an ADDROUT file.

8.5.2 Direct Organization

Direct file organization is available on disk devices only. The VAX/VMS Run-Time Library handles VAX RPG II direct files as files with relative file organization. A direct file consists of a series of fixed-length positions (or cells) that are numbered consecutively from 1 to n. This number is the relative record number; it indicates the record's position relative to the beginning of the file. (The relative record number of the first cell is always 1.) Each record you write is assigned to a specific cell within the file. For example, you can assign the second record to the fourth cell; its relative record number would be 4. This assignment can result in empty cells; therefore, you must specify a record's relative record number to access it. Figure 8–2 shows that cell numbers 2 and 5 are empty cells.

Figure 8–2: Direct File Organization



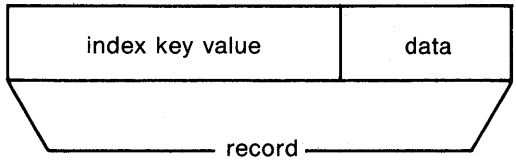
ZK-1463-83

Direct files can be accessed sequentially or randomly by using the CHAIN operation code (see Section 16.7.1 for information on the CHAIN operation code) or by using an ADDROUT file. When you access a direct file sequentially, empty cells are skipped. When you access a direct file randomly using the CHAIN operation, the indicator specified in columns 54 and 55 of the Calculation specification will be set on for an empty cell.

8.5.3 Indexed Organization

Indexed file organization is available on disk devices only. Each record in an indexed file contains an index key value, as shown in Figure 8–3.

Figure 8–3: Indexed File Organization

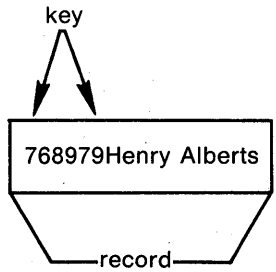


ZK-1464-83

An index key value is a field within each record that is defined by its relative location within the record and by its length. The index key value is the primary means of locating records within the file. For example,

you could use an employee's badge number as the index key value for an employee record. The index key value in Figure 8–4 is the first six characters in the record, or 768979.

Figure 8–4: Index Key Value



ZK-1465-83

You can retrieve a record from an indexed file by specifying its index key value. In fact, you can retrieve records in an indexed file either sequentially or randomly by using index key values, or randomly by using an ADDROUT file.

Another way to access records from an indexed file is sequentially within limits. See Section 8.6.3 for more information on this method of accessing records from indexed files.

8.6 File Access Methods

There are several ways you can access the records in a file, depending on its organization. Table 8–1 lists file organizations and the methods you can use to retrieve records.

Table 8–1: File Access Methods

File Designation	Organization	Access Method
Primary	Sequential	Sequentially
Secondary		Randomly by ADDROUT file[1]
Demand	Direct	Sequentially
Full-procedural	Direct	Randomly by relative record number
	Indexed	Sequentially
		Sequentially by key
		Sequentially within limits
		Randomly by ADDROUT file[1]
Chained	Sequential	Randomly by relative record number[2]
	Indexed	Randomly by key

[1] You cannot process demand or full-procedural files using an ADDROUT file.

[2] You can access the records in a sequential file randomly by relative record number if the records are fixed-length and the file resides on disk.

Although you cannot change the organization of a file after you have created it, you can change the file access method each time you use the file. The method you use depends on how many records your file contains and how often you need to access a record. Use the following guidelines in selecting a file organization and access method:

- If you always process all the records in a file from beginning to end (as in a payroll application), use a sequential file and access the records sequentially.
- If you need to access some or all records under changing or unpredictable conditions (as in a transaction processing system), use an indexed or direct file and access the records randomly.

Sections 8.6.1 through 8.6.5 describe each file access method and provide programming guidelines for each.

8.6.1 Sequential Access

When you access a file sequentially, each input operation retrieves the next record in the file, regardless of the file organization, until either the end-of-file is reached or the program terminates. For an indexed file, records are retrieved in the order of the primary keys.

To specify sequential access for a file, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify I or U to indicate whether the file is to be opened for input or for update.
- Column 16 (file designation)—specify P, S, D, or F to indicate whether the input file is primary, secondary, demand, or full-procedural.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.

The following example specifies the name of a file, INPUT, designated as an input primary file with fixed-length records and a record length of 60 bytes:

```
                                Mode (LR)
                                |Key length
                    Type (IOUDC) || Record address type (APIB)          Addtn(AU)
                    |Des(PSRCTDF)|| |Organization (IT,1-9)              |Expand
                    ||EOF (E)    || ||Overflow indicator  Continue (K) ||Shr(SR)
                    |||Seq (AD)  || ||| Key location      |Opt   Entry |||Rewnd
  File              ||||Fmt (FV) || ||| |   Extension (EL)||     |     |||  |
  name              |||||Blk Rec || ||| |   |Device Symb  Tape   Core  |||  |File
   |                ||||||len len || ||| |   |code   dev   label  index |||  |cond
  F|                ||||||   |   || ||| |   ||     |     |      |     |||  ||
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         ******---*---**-*** *---**      *     *......*-----***.** ..
     FINPUT     IP  F      60           DISK
```

ZK-4394-85

8.6.2 Sequential Access by Key

You can process only indexed primary, secondary, demand, and full-procedural files sequentially by key. VAX RMS reads records in ascending key sequence until it reaches the end of the file or until the program terminates.

To specify sequential access by key for a file, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify I or U to indicate whether the file is to be opened for input or for update.
- Column 16 (file designation)—specify P, S, D, or F to indicate whether the input file is primary, secondary, demand, or full-procedural.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Columns 29 and 30 (key length)—specify the length of the key field. For binary, use 2 for word and 4 for longword.
- Column 31 (record address type)—specify either A, P, or B to indicate that the index keys are in character (A), packed decimal (P), or binary (B) data format.
- Column 32 (file organization)—specify I to indicate that the file is an indexed file.
- Columns 35 through 38 (key location)—specify the starting character position of the key field.

The following example shows how to specify a primary input file, INPUT, with fixed-length records 60 bytes long. The file organization is indexed with its index keys in packed decimal data format.

```
                                Mode (LR)
                                |Key length
                    Type (IOUDC) || Record address type (APIB)          Addtn(AU)
                    |Des(PSRCTDF)|| |Organization (IT,1-9)              |Expand
                    ||EOF (E)    || ||Overflow indicator  Continue (K) ||Shr(SR)
                    |||Seq (AD)  || ||| Key location      |Opt   Entry |||Rewnd
     File           ||||Fmt (FV) || ||| |   Extension (EL)||     |     |||  |
     name           |||||Blk Rec || ||| |   |Device Symb  Tape   Core  |||  |File
      |             ||||||len len || ||| |   |code  dev    label  index |||  |cond
    F|             ||||||   |   || ||| |   ||     |      |      |     |||  ||
   0   |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **       ******---*---**-*** *---**      *     *......*-----***.** ..
     FINPUT   IP  F      60  3PI     1 DISK
```

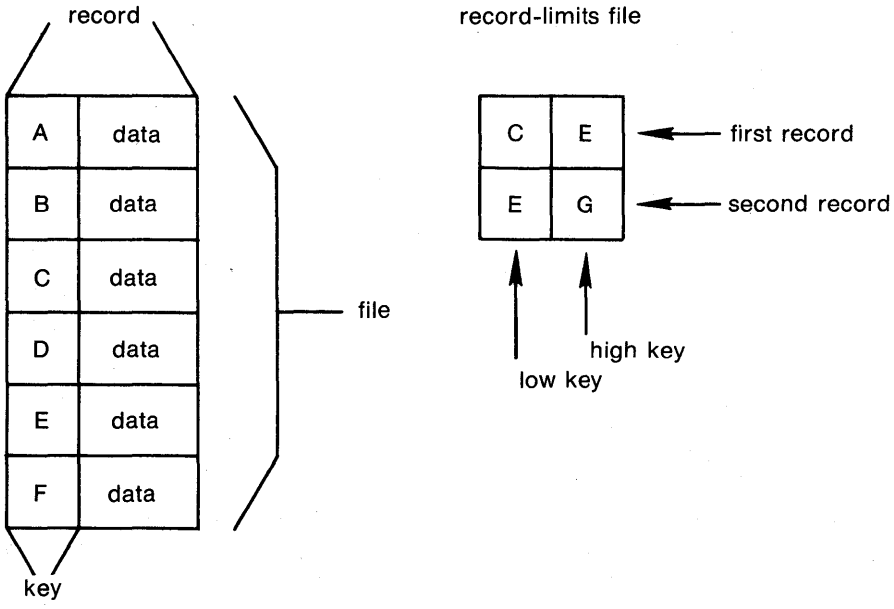
ZK-4395-85

8.6.3 Sequential Access Within Limits

You can process indexed files sequentially within limits by creating a record-limits file that specifies a range of index keys in each record.

In Figure 8–5, the first record in the record-limits file causes VAX RPG II to retrieve those records whose keys are greater than or equal to the low key (C) and less than or equal to the high key (E). When the program reaches a record with a key value greater than E or reaches end-of-file, it reads the next record from the record-limits file to get a new high and low range. The second record in the record-limits file causes the program to retrieve those records whose keys are greater than or equal to the low key (E) and less than or equal to the high key (G). The indexed file is processed until it reaches the end of the record-limits file or the program terminates.

Figure 8–5: Sequential File Access Within Limits



ZK-1466-83

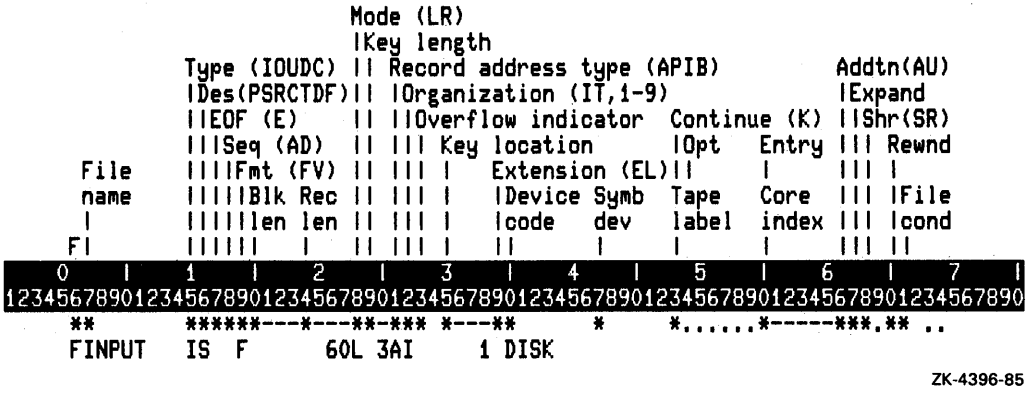
Rules

- In the record-limits file, you can specify only one set of limits for a record.
- The record length must be at least twice the length of the record key.
- The low key must begin in character position 1, and the high key must immediately follow the low key.
- The length of the high and low keys must be the same and must be equal to the length of the key field in the file to be processed.
- Numeric keys can contain leading zeros.
- Alphanumeric keys can contain blanks.

To access a file sequentially within limits, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify I or U to indicate whether the file is to be opened for input or for update.
- Column 16 (file designation)—specify P, S, D, or F to indicate whether the input file is primary, secondary, demand, or full-procedural.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Column 28 (access mode)—specify L to indicate that the indexed file is to be processed sequentially within limits.
- Columns 29 and 30 (key length)—specify the length of the key field.
- Column 31 (record address type)—specify either A, B, or P to indicate that the index keys are in character (A), packed decimal (P), or binary (B) data format.
- Column 32 (file organization)—specify I to indicate that the file is an indexed file.
- Columns 35 through 38 (key location)—specify the starting character position of the key field.

The following example shows how to specify an input secondary file, INPUT, with fixed-length records 60 bytes long. This file is to be processed sequentially within limits. The file organization is indexed, the key field is three bytes long beginning in character position 1, and the keys are in character format.



ZK-4396-85

To access a file sequentially within limits, you must make the following entries for the record-limits file in its File Description specification:

- Column 15 (file type)—specify I to indicate that the file is to be opened for input.
- Column 16 (file designation)—specify R to indicate that the file named in columns 7 through 14 is a record-limits file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Columns 29 and 30 (key length)—specify the length of the key field.
- Column 31 (record address type)—specify either A, P, or blank to indicate that the index keys are in character (A), packed decimal (P), or the same data format as the file being processed by the record-address file (blank).
- Column 39 (extension)—specify E to cause the system to look for an Extension specification.

You must also make the following entries for the record-limits file in its Extension specification:

- Columns 11 through 18 (from file name)—specify the name of the record-limits file.
- Columns 19 through 26 (to file name)—specify the name of the file to be processed by the record-limits file.

The following example shows how to specify the File Description and Extension specifications for processing a file sequentially within limits:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     FIDXA12  IR  F       6  3A       EDISK
     FIDXI12  IP  F      60L 3AI      1 DISK
     E    IDXA12  IDXI12
                  .
                  .
                  .
```

ZK-4397-85

An indexed demand or full-procedural file can also be processed sequentially within limits using the SETLL operation. See Chapter 16 for information on the SETLL operation code.

8.6.4 Random Access

Accessing records randomly allows you to retrieve or write a record anywhere in the file. To do this, you must specify the record location using one of the following methods:

- Relative record numbers
- Keys
- An ADDROUT file

The method you use depends on the organization of the file. Sections 8.6.4.1 through 8.6.4.3 explain these methods.

8.6.4.1 Random Access by Relative Record Number

You can randomly access records in sequential and direct files by specifying relative record numbers that identify records relative to the beginning of the file. For example, the relative record number for the fifth record is 5. Accessing a sequential file using this method requires that the records be of fixed length and that the file reside on disk.

To access a file randomly by relative record number, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify I or U to indicate whether the file is to be opened for input or for update.
- Column 16 (file designation)—specify C or F to indicate whether the file named in columns 7 through 14 is a chained or full-procedural file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Column 28 (mode)—specify R to cause VAX RPG II to access the file randomly, using a relative record number.

You must also make the following entries for the file in its Calculation specification:

- Columns 18 through 27 (factor 1)—specify the relative record number of the record you want to retrieve.

- Columns 28 through 32 (operation code)—specify the CHAIN operation code. Use an indicator in columns 54 and 55 to signal an empty cell condition for a direct file. Otherwise, attempting to use the CHAIN operation code to randomly access an empty cell will cause a run-time error.
- Columns 33 through 42 (factor 2)—specify the name of the file that contains the record you want to retrieve.

The following example shows how to randomly access the direct file RAN07A by relative record number. The primary input file RANI07 provides the record numbers in the field ITEM#.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FRANI07  IP  F      13            DISK
     FRAN07A  UC  F      10R           DISK
     FRAN07B  O   F      30            PRINTER
     IRANI07  AA  01
     I                                      1  11 STORE
     I                                     13  130ITEM#
     IRAN07A  AB  02
     I                                      1   10REC#
     I                                      3   50ACCESS
     I                                      7  10 VALUE
     C           ITEM#      CHAINRAN07A              50
     C   50                 GOTO HANDLR
     C           1          ADD  ACCESS    ACCESS
     C                      SETON                    40
     C                      EXCPT
     C                      SETOF                    40
     C           HANDLR     TAG
     C   50                 SETON                    LR
     ORAN07A  E         02 40
     O                             REC#      1
     O                             ACCESS    5
     O                             VALUE     10
     ORAN07B  H 22      1PN40
     O                                      22 'STORE PURCHASES'
     O        D         01N40
     O                             STORE     14
     O                             ACCESS    20
     O                             VALUE     27
```

ZK-4398-85

8.6.4.2 Random Access by Key

You can randomly retrieve records from an indexed file by specifying their index keys.

To access a file randomly by key, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify I or U to indicate whether the file is to be opened for input or for update.
- Column 16 (file designation)—specify C or F to indicate whether the file named in columns 7 through 14 is a chained or full-procedural file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Column 28 (mode)—specify R to tell VAX RPG II to access records randomly, using index key values.
- Columns 29 and 30 (key length)—specify the length of the key field.
- Column 31 (record address type)—specify either A or P to indicate that the index keys are in character (A) or packed decimal (P) data format.
- Column 32 (file organization)—specify I to indicate that the file is an indexed file.
- Columns 35 through 38 (key location)—specify the starting character position of the key field.

You must also make the following entries for the file in its Calculation specification:

- Columns 18 through 27 (factor 1)—specify the index key of the record you want to retrieve.
- Columns 28 through 32 (operation code)—use the CHAIN operation code. The record you specify can be read from the file either during detail-time or total-time calculations. Specify an indicator in columns 54 and 55 to signal a record-not-found condition. Otherwise, a record-not-found condition will cause a run-time error.
- Columns 33 through 42 (factor 2)—specify the name of the file to be processed.

The following example randomly accesses the indexed file GROCER using keys. The primary input file STORES provides the keys in the field ITEM#.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FSTORES  IP  F      13             DISK
     FGROCER  IC  F      10R 1AI     1 DISK
     FREPORT  O   F      30             PRINTER
     ISTORES  AA  01
     I                                        1  11 STORE
     I                                       13  130ITEM#
     IGROCER  AB  02
     I                                        1   10REC#
     I                                        3   50COUNT
     I                                        7  10 VALUE
     C           ITEM#     CHAINGROCER                50
     C   50                SETON                      LR
     OREPORT  H 22     1PN40
     O                                     22 'STORE PURCHASES'
     O        D        01N40
     O                          STORE      14
     O                          COUNT      20
     O                          VALUE      27
```

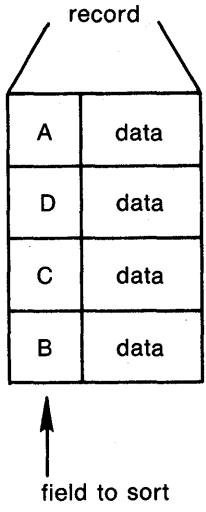
ZK-4399-85

8.6.4.3 Random Access by ADDROUT File

Another way to process files is by using an ADDROUT file. You can use a record-limits file to process only indexed files. You can use an ADDROUT file to process sequential, direct, or indexed files.

ADDROUT files are created by the VAX/VMS Sort/Merge Utility (SORT/MERGE) when you use the /PROCESS=ADDRESS qualifier. You specify a field or fields in the record by which SORT/MERGE sorts the records, as shown in Figure 8–6.

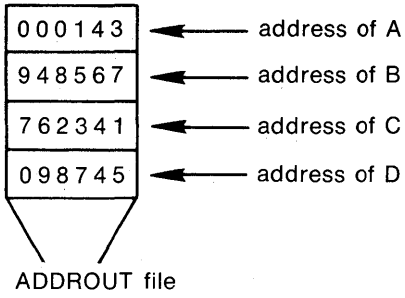
Figure 8–6: Random Access by ADDROUT File



ZK-1467-83

SORT/MERGE sorts the records and places the disk addresses of the sorted records in an ADDROUT file, as shown in Figure 8–7.

Figure 8–7: An ADDROUT File



ZK-1468-83

The program reads the records (addresses) in the ADDROUT file sequentially. Each record in the ADDROUT file corresponds to a record in the original file. The addresses of the records are referred to as Record File Addresses (RFAs) by VAX RMS. For additional information on RFAs, see the *VAX Record Management Services Reference Manual.*

To access a file using an ADDROUT file, you must make the following entries in the File Description specification of the file to be accessed:

- Column 15 (file type)—specify I or U to indicate whether the file is to be opened for input or for update.
- Column 16 (file designation)—specify P or S to indicate that the file named in columns 7 through 14 is primary or secondary.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Column 28 (mode)—specify R to cause VAX RPG II to access records randomly.
- Columns 29 and 30 (key length)—specify the length of the key field if you plan to access an indexed file.
- Column 31 (record address type)—specify I to cause the program to access the file according to the ADDROUT file.

- Column 32 (file organization)—specify I if you plan to access an indexed file.
- Columns 35 through 38 (key location)—specify the starting character position of the key field if you plan to access an indexed file.

To access a file using an ADDROUT file, you must make the following entries for the ADDROUT file in its File Description specification:

- Column 15 (file type)—specify I to indicate that the file is to be open for input.
- Column 16 (file designation)—specify R to indicate that the file named in columns 7 through 14 is an ADDROUT file.
- Column 19 (record format)—specify F to describe the record format.
- Columns 24 through 27 (record length)—specify 6, because record addresses are always 6 bytes in length.
- Columns 29 and 30 (key length)—specify 6, because key addresses are always 6 bytes in length.
- Column 31 (record address type)—specify I to indicate that this is an ADDROUT file.
- Column 32 (file organization)—specify T to indicate an ADDROUT file.
- Column 39 (Extension)—specify E to cause VAX RPG II to look for an Extension specification.

You must also make the following entries for the ADDROUT file in its Extension specification:

- Columns 11 through 18 (from file name)—specify the name of the ADDROUT file.
- Columns 19 through 26 (to file name)—specify the name of the file to be processed by the ADDROUT file.

The following example shows how to specify the ADDROUT file IDXA13 and how to specify the file IDXI13 which will be accessed by the ADDROUT file.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     FIDXA13  IR  F       6  6IT      EDISK
     FIDXI13  IP  F      60R 3II     1 DISK
     FIDX13A  O   F      80     OF     PRINTER
     E    IDXA13  IDXI13
     IIDXI13  AA  02
     I                                          1   3 KEY
     I                                          1  12 TOWN
     I                                         13  14 STATE
     I                                         15  25 COUNTY
     I                                         26  30 ZIP
     I                                         31  350CEN30
     I                                         36  400CEN40
     I                                         41  450CEN50
     I                                         46  500CEN60
     I                                         51  550CEN70
     I                                         56  600CEN80
     OIDX13A  H  202   1P
     O       OR        OF
     O                         UDATE Y   10
     O                                   49 'NEW HAMPSHIRE TOWNS'
     O                         PAGE      77
     O        D 1      02
     O                         TOWN      13
     O                         COUNTY    26
     O                         STATE     30
     O                         CEN80 J   38
     O                         CEN70 J   46
     O                         CEN60 J   54
     O                         CEN50 J   62
     O                         CEN40 J   70
     O                         CEN30 J   78
```

ZK-4400-85

8.6.5 Sequential Access and Random Access by Key

A full-procedural file allows you to read a file both randomly and sequentially. If the full-procedural file is an indexed file, then you can read the file randomly by key using the CHAIN or SETLL operation, and you can read the file sequentially.

To specify an indexed full-procedural file, make the following entries for the file in its File Description specification:

- Columns 7 through 14 must contain the file name.
- Column 15 (file type)—specify I or U to indicate that the file is to be opened for input or for update.
- Column 16 (file designation)—specify F to indicate a full-procedural file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Column 28 (access mode)—specify L to indicate that the indexed file is to be processed sequentially within limits.
- Columns 29 and 30 (key length)—specify the length of the key field.
- Column 31 (record address type)—specify either A, B, or P to indicate that the index keys are in character (A), packed decimal (P), or binary (B) data format.
- Column 32 (file organization)—specify I to indicate that the file is an indexed file.
- Columns 40 through 43 (device code)—specify DISK.

The following example specifies the full-procedural file FPFJ01 to be accessed by a CHAIN operation with the key specified in FPFI01. The file FPFJ01 is then processed sequentially from that point on.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
FTTY      D   F       80             TTY
FFPFI01   ID  F       04             DISK
FFPFJ01   IF  V       47R04AI      1 DISK
FFPF01A   O   V       73           LPRINTER
LFPF01A    55FL 500L
IFPFI01
I                                          1   4 PARTNO
IFPFJ01
I                                          1   4 PARTNO
I                                          5  39 DESCR
I                                         40  43 PRICE
I                                         44  47 AMOUNT
C                      READ FPFI01
C             PARTNO   CHAINFPFJ01              98
C                      EXCPT
C    98       'BAD'    DSPLYTTY
C    98                GOTO END
C             LOOP     TAG
C                      READ FPFJ01                    LR
C   NLR                EXCPT
C   NLR                GOTO LOOP
C             END      TAG
OFPF01A   H  201   1P
O                                       32 'PARTS SUMMARY INVENTORY'
O         H 10     1P
O                                       11 'PART NO'
O                                       30 'DESCRIPTION'
O         H 00     1P
O                                       30 '__________'
O         H 01     1P
O                                       11 '_______'
O         E 01
O                        PARTNO          9
O                        DESCR          47
```

ZK-4662-85

8.7 Creating Files

There are a variety of ways to create files with sequential, direct, and indexed organizations. Sections 8.7.1 through 8.7.3 describe how to create files using a VAX RPG II program.

8.7.1 Creating Sequential Files

You can create sequential files by writing consecutive records to an output file. After a sequential file is created, you can use it as an input file, an update file, or an output file with the ADD option.

To create a sequential file, you must make the following entries in the File Description specification:

- Column 15 (file type)—specify O to indicate the creation of an output file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.

The following example shows how to create the sequential file OUT60A:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FOUTI24  IP  F       24             DISK
     FOUT60A  O   F       24             DISK
     IOUTI24  AA
     I                                             1   3 PN
     I                                             4  10 PNAME
     I                                            11  12 WHOUSE
     I                                            13  17 COLOR
     I                                            18  20 WEIGHT
     I                                            21  24 QTY
     OOUT60A  D        N1P
     O                          PN          3
     O                          PNAME      10
     O                                      4 '1'
     O                          WHOUSE     12
     O                          COLOR      17
     O                          WEIGHT     20
     O                          QTY        24
```

ZK-4401-85

8.7.2 Creating Direct Files

You can create a direct file by specifying a chained output file. To do this, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify O to indicate the creation of an output file.
- Column 16 (file designation)—specify C to indicate that the file named in columns 7 through 14 is a chained file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Column 28 (mode)—specify R to cause VAX RPG II to load a direct file.

The following program shows how to create the direct file OUT60B with variable length records:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FOUTI24  IP  F       24             DISK
     FOUT60B  OC  V       24R            DISK
     IOUTI24  AA
     I                                       1   3 PN
     I                                       4  10 PNAME
     I                                      11  12 WHOUSE
     I                                      13  17 COLOR
     I                                      18  20 WEIGHT
     I                                      21  24 QTY
     C            COUNT      ADD  1         COUNT   10
     C            COUNT      CHAINOUT60B                99
     OOUT60B  D        N1P 25
     O                          PN          3
     O                          PNAME      10
     O                                      4 '3'
     O                          WHOUSE     12
     O                          COLOR      17
     O                          WEIGHT     20
     O                          QTY        24
```

ZK-4402-85

8.7.3 Creating Indexed Files

You can create an indexed file either in unordered key sequence or in ordered key sequence. If you specify unordered, you can write records to an indexed file in any order, regardless of the key sequence. If you specify ordered, you must write records in the order of their key; the order must be ascending. After the file is created, VAX RMS sorts the index keys in ascending order, regardless of the order in which they were written.

To create an indexed file in ordered sequence, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify O to indicate the creation of an output file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Columns 29 and 30 (key length)—specify the length of the key field. For binary, use 2 for word length and 4 for longword length.
- Column 31 (record address type)—specify either A, P, or B to indicate that the index keys are in character (A), packed decimal (P), or binary (B) data format.
- Column 32 (file organization)—specify I to indicate an indexed file.
- Columns 35 through 38 (key location)—specify the starting character position of the key field.

The following program shows how to create an indexed file OUT60A with an alphanumeric key that is three bytes long. The key begins in character position 1 of each record.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     FOUTI24  IP  F       24              DISK
     FOUT60A  O   V       24  3AI     1 DISK
     IOUTI24  AA  01
     I                                             1   3 PN
     I                                             4  10 PNAME
     I                                            11  12 WHOUSE
     I                                            13  17 COLOR
     I                                            18  20 WEIGHT
     I                                            21  24 QTY
     OOUT60A  D           01
     O                         PN          3
     O                         PNAME      10
     O                                     4 '1'
     O                         WHOUSE     12
     O                         COLOR      17
     O                         WEIGHT     20
     O                         QTY        24
```

ZK-4403-85

To create an indexed file in unordered sequence, make the same entries as for an ordered sequence and specify U in column 66 (Unordered).

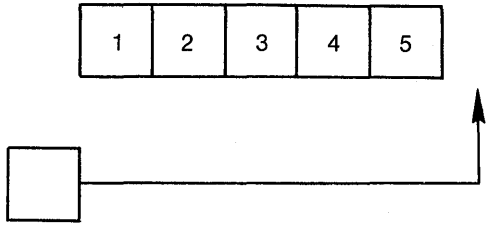
8.8 Adding Records to Files

After you create a file, it may be necessary to add new records to the file. You can add records to a file during detail-time or total-time output, or by using exception output. Sections 8.8.1 through 8.8.3 explain how to add records to files on the basis of their file organization.

8.8.1 Adding Records to a Sequential File

Because the location of each record in a sequential file is fixed in relation to all others, there is no unused space where a new record might be inserted. Therefore, you can add records to a sequential file only at the end of the file, as shown in Figure 8–8.

Figure 8–8: Adding Records to the End of a Sequential File



ZK-1469-83

To add a record to the end of a sequential file, you must make the following entries in its File Description specification:

- Column 15 (file type)—specify O to indicate the creation of a new record.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.
- Column 66 (file addition)—specify A to cause VAX RPG II to add new records to the file.

You must also make the following entries in the file's Output specification:

- Columns 7 through 14 (file name)—define the output file name.
- Columns 16 through 18—specify ADD to identify the record to be added.

The following example accepts input from the terminal and writes records to the end of the file LOG.

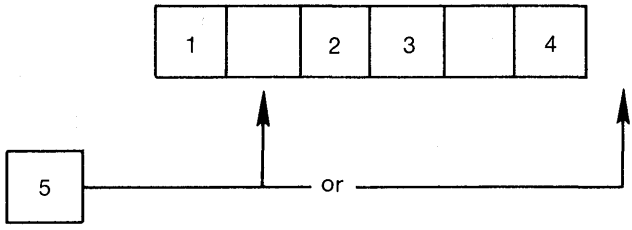
```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FINPUT   IP  F      80              TTY
     FLOG     O   F      80              DISK                        A
     IINPUT       01
     I                                                 1   80 DATA
     OLOG     DADD     01
     O                          DATA      80
```

ZK-4404-85

8.8.2 Adding Records to a Direct File

To add a new record to a direct file, you can either specify the relative record number of an empty cell or add the record at the end of the file, as shown in Figure 8–9.

Figure 8–9: Adding Records to a Direct File



ZK-1470-83

To add records to empty cells in a direct file, you must make the following entries for the file in its File Description specification:

- Column 15 (file type)—specify I or U to indicate that the file is to be opened for input or update.
- Column 16 (file designation)—specify C or F to indicate a chained or full-procedural file.
- Column 19 (record format)—specify F or V to describe the record format.
- Columns 24 through 27 (record length)—specify the length of fixed-length records or the maximum length of variable-length records.

- Column 28 (mode)—specify R to access records randomly, using a relative record number.
- Column 66 (addition)—specify A to add records to the file.

You must also make the following entries in the Calculation specification:

- Columns 18 through 27 (factor 1)—specify the relative record number of the empty cell.
- Columns 28 through 32 (operation code)—specify the CHAIN operation code. Use an indicator in columns 54 and 55 to check whether the cell is empty. The indicator will be set on if the cell is empty. If the cell is empty and an indicator is not specified, a run-time error occurs.
- Columns 33 through 42 (factor 2)—specify the name of the file to which you want to add the record.

Finally, you must make the following entry in the Output specification:

- Columns 7 through 14 (file name)—define the output file name.
- Columns 16 through 18—specify ADD to identify the record to add.

The output operation must follow the CHAIN operation, but before the next CHAIN operation. If not, the output will be to the cell specified by the second CHAIN operation.

The following example reads a primary input file and adds records to the direct file DIRECT. The input field RECNO specifies the record cell to which the field is written.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FINPUT   IP  F       35            DISK
     FDIRECT  IC  F       30R           DISK                          A
     FTTY     D   F       30            TTY
     IINPUT
     I                                              1  30 DATA
     I                                             31  35ORECNO
     IDIRECT
     C           RECNO      CHAINDIRECT                  99
     C  N99       'EXISTS'  DSPLYTTY
     ODIRECT  DADD     99
     O                         DATA       30
```

ZK-4405-85

8.8.3 Adding Records to an Indexed File

If the file is an indexed file, you can add records at any location. The key values for the new records are placed in the index and the entire index is sorted in ascending sequence.

NOTE

When adding records to an indexed file, you cannot specify A in column 66 (file addition) of the File Description specification for indexed files processed sequentially within limits or processed by an ADDROUT file.

You can add new records to an indexed file while processing the file by specifying an A in column 66 (file addition) of the File Description specification. The file can be an input or update file that is processed sequentially or randomly. If you want only to add records, you can specify an output file.

You must also make the following entries in the Output specification:

- Columns 7 through 14 (file name)—define the output file name.
- Columns 16 through 18—specify ADD to identify the records to be added.

The following program adds records to an indexed file using the ADD option on the Output specification:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FIDXIO1  IP  F      24              DISK
     FOUT43A  O   F      24  3AI     1 DISK                        A
     IIDXIO1  AA
     I                                          1  24 PN
     OOUT43A  DADD    N1P
     O                        PN        24
```

ZK-4406-85

8.9 Updating Records in Files

VAX RPG II allows you to update the records in a primary, secondary, demand, full-procedural, or chained file. VAX RPG II allows you to update the records in a sequential file only if the records are of fixed

length. You can update a record in a primary or secondary file only once during the program cycle at detail time. Unlike other types of update files, records in a chained, full-procedural, or demand file can be updated at detail time or at total time.

To update a record, you must retrieve the record you want to change, change the contents, and then write the record back to the file. You need only specify the fields to be changed in a record. The remainder of the record is rewritten, using the data that was read into the input buffer.

You can use a data structure to update a record. See Chapter 12 for an example of updating files with data structures.

VAX RPG II allows you to change the length of a variable-length record being updated. VAX RPG II determines the length of the record by using the highest end position (columns 40 through 43 of the Output specification) of any field in the record. If you need to change the contents of a field in the middle of a variable-length record, but do not want to change the length of the record, you must define the length of the record by defining a one-character field in the last character position of the record.

The following example updates records in the master file MASTER. MASTER contains two different record types of different lengths. Both records contain the field EMP# that must be updated in different character positions. The fields LNGTH1 and LNGTH2 ensure that the records are updated using the correct length. The records of type 01 are 80 characters long. The records of type 02 are 60 characters long.

```
     0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890
     FTRANSACTIP  F      25            DISK
     FMASTER  UC  V      80R21AI     1 DISK
     ITRANSACT
     I                                      1  20 NAME
     I                                     21  25 NEW#
     IMASTER      01   1 CA
     I                                     36  40 EMP#
     I                                     80  80 LNGTH1
     I            02   1 CB
     I                                     46  50 EMP#
     I                                     60  60 LNGTH2
     C* Update record type 01
     C                    MOVEL'A'       KEY    21
     C                    MOVE NAME      KEY
     C           KEY      CHAINMASTER
     C                    EXCPT
     C                    SETOF                    01
     C* Update record type 02
     C                    MOVEL'B'       KEY    21
     C           KEY      CHAINMASTER
     C                    EXCPT
     OMASTER  E       01
     O                        NEW#       40
     O                        LNGTH1     80
     OMASTER  E       02
     O                        NEW#       50
     O                        LNGTH2     60
```

ZK-4407-85

To update the records in a direct or indexed file and simultaneously add new records, you must make the following entries for the file in its File Description specification:

- Column 15 (file type)—specify U to indicate that the file is to be opened for update.
- Column 66 (file addition)—specify A to add new records to the file.

You must also define both Input and Output specifications for the file to be updated. Enter ADD in columns 16 through 18 of those Output specifications that identify the records to be added. The output records without ADD in columns 16 through 18 identify those records to be updated.

You can update records in an indexed file randomly by key, sequentially, or both randomly and sequentially if the file is defined as a full-procedural file. To specify an indexed full-procedural file to be processed in the update mode, you must make the following entries for the file in its File Description specification:

- Column 15 (file type)—specify U to indicate that the file is to be opened for update.
- Column 16 (file designation)—specify F to indicate a full-procedural file.
- Column 32 (file organization)—specify I to indicate an indexed file.

8.10 Deleting Records from Files

You can delete records only from update, direct, and indexed files. To prevent the deletion of needed records, perform the following steps:

- Retrieve the record.
- Evaluate its contents.
- Based on the results of the evaluation, set an indicator to control deletion of the record.

The last record retrieved from the file is the one that is deleted when you specify DEL in columns 16 through 18 of the Output specification. You do not need to describe any fields in the output record, because the operation deletes the entire record.

The following example deletes a record in the master file MASTER, depending on the keys read from the file DELETE:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FDELETE  IP  F       4            DISK
     FMASTER  UC  F      50R 4AI    47 DISK                        A
     FTTY     D   F      80            TTY
     IDELETE
     I                                          1   4 KEY
     IMASTER
     C           KEY       CHAINMASTER               99
     C   99       'NOTFOUND'DSPLYTTY
     OMASTER  DDEL    N99N1P
```

ZK-4408-85

8.11 Processing Files with Matching Records

Matching fields can be used with primary and secondary files to check the sequence of records and to define the order in which records are selected from multiple files.

To use matching fields to verify that the records in the file are in sequence (either ascending or descending), you define one or more fields to be checked by specifying a matching field value (M1 through M9) in columns 61 and 62 in the Input specification. Then, your program checks the sequence by comparing the matching field of one record with the matching field of the previous record. If the field is out of order, a run-time error occurs.

8.11.1 Checking Record Sequence for One Record Type

You designate a record sequence by specifying A or D (ascending or descending) in column 18 of the File Description specification. Assign a matching field value (M1 through M9) to one or more of the fields you want to use as matching fields in columns 61 and 62 (matching field) of the Input specification. When you specify more than one matching field, assign M9 to the most important field. Your program considers all matching fields as one contiguous field with the M9 field in the leftmost position, next to the M8 field, and so on, until you reach M1, even though the fields may not be adjacent in the record or in numeric (M9 through M1) order.

8.11.2 Checking Record Sequence for More Than One Record Type

The fields in a record of one type can be in a different order than the fields in other record types in the same file. For example, in a payroll file consisting of two different record types, one type represents commission payment and the other type represents salary. All employee records are to be in ascending sequence according to district (DSTRCT). Records in a district are to be in ascending sequence according to department and employee number. Therefore, three fields (DSTRCT, DEPT, and EMPNUM) must be checked in each record. The matching field value M3

is assigned to DSTRCT, the most important field; M2 is assigned to DEPT, the next most important field; and M1 is assigned to EMPNUM, the least important field. Refer to the following example:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)              Decimal positions
                | ||Record identifying indicator      |      Control level
                | |||                                 |      | Match field
                | ||| + Identifying codes + Format    |      | | Fld rec rel
     File       | |||  |                    | | (PB)   |Field | | |
     name       | |||  |    C       C       C| |Field   |name  | | | Field
      |         | |||  |    Z       Z       Z| |location||      | | | indicatrs
     ||         | ||| Pos NDcPos NDcPos NDc |Fr  To  ||      | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IPAYROLL AA  01   80 CC
     I                                          1   3 DEPT     M2
     I                                          6   7 DSTRCT   M3
     I                                         14  152COMM
     I                                         25  27 EMPNUM   M1
     I          BB  02   80 CS
     I                                          1   3 DEPT     M2
     I                                          8   9 DSTRCT   M3
     I                                         13  172SALARY
     I                                         25  27 EMPNUM   M1
```

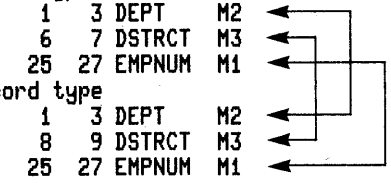
ZK-4409-85

First, the program determines the record type. Then, it looks at the matching fields for the same record type.

In the preceding example, the same three matching fields (DSTRCT, DEPT, and EMPNUM) appear in both record types and are the same length.

The length of matching fields assigned to the same match code must be the same length for each record type. Table 8–2 shows that this is true for the following example:

```
IPAYROLL AA  01  80 CC <-- first record type
I                                    1   3 DEPT    M2
I                                    6   7 DSTRCT  M3
I                                   25  27 EMPNUM  M1
I        BB  02  80 CS <-- second record type
I                                    1   3 DEPT    M2
I                                    8   9 DSTRCT  M3
I                                   25  27 EMPNUM  M1
```



ZK-4410-85

Table 8–2: Matching Field Lengths

Record Type	Matching Field	Field Location	Field Length
first	DSTRCT	6 to 7	2
	DEPT	1 to 3	3
	EMPNUM	25 to 27	3
			8 <— total
second	DSTRCT	8 to 9	2
	DEPT	1 to 3	3
	EMPNUM	25 to 27	3
			8 <— total

Matching fields need not be specified for all the record types in a file.

8.11.3 Using Matching Fields with Field-Record-Relation Indicators

Although there may be different record types in a file, the fields for each record type are often the same. Many fields have the same name, contain the same data, and are in the same character positions for all the record types in a file. When only a few fields differ, you can describe more than one record type in an OR relationship. Refer to the following example:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)              Decimal positions
                | ||Record identifying indicator      |      Control level
                | |||                                 |      | Match field
                | ||| + Identifying codes + Format    |      | | Fld rec rel
 File           | ||| |                     | | (PB)  |Field | | |
 name           | ||| |    C       C       C| |Field  |name  | | | Field
 |              | ||| |    Z       Z       Z| |location||     | | | indicatrs
I|              | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IPAYROLL AA  01  80 CS
     I       OR   02  80 CM
```

ZK-4411-85

You specify common fields only once, because they apply to both record types. The field-record-relation indicators specified in columns 63 and 64 of the Input specification identify the fields unique to a particular record type. Therefore, the COMM field in the following example is associated with record type 01 and the SALARY field is associated with record type 02.

The DSTRCT, DEPT, and EMPNUM matching fields are used in checking the sequence of the records in the PAYROLL file, and the matching-field values M1, M2, and M3 are described only once in columns 61 and 62 without any field-record-relation indicators in columns 63 and 64. Therefore, the matching fields and their values apply to both record types (01 and 02) as shown in the following example.

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                                |      | Match field
                | ||| + Identifying codes + Format   |      | | Fld rec rel
 File           | ||| |                   | | (PB)   |Field | | |
 name           | ||| |    C      C      C| |Field   |name  | | | Field
  |             | ||| |    Z      Z      Z| ||location||     | | | indicatrs
 I|             | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IPAYROLL AA  01   80 CS
     I       OR   02   80 CM
     I                                          1   3 DEPT     M2
     I                                          8   9 DSTRCT   M3
     I                                         25  27 EMPNUM   M1
     I                                         14  152COMM       01
     I                                         13  172SALARY     02
```

ZK-4412-85

If one of the matching fields is in a different record position for each record type, you must assign matching field entries, as shown in the following example:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                                |      | Match field
                | ||| + Identifying codes + Format   |      | | Fld rec rel
 File           | ||| |                   | | (PB)   |Field | | |
 name           | ||| |    C      C      C| |Field   |name  | | | Field
  |             | ||| |    Z      Z      Z| ||location||     | | | indicatrs
 I|             | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IPAYROLL AA  01   80 CS
     I       OR   02   80 CM
     I                                          1   3 EMPNUM   M1
     I                                         20  21 DSTRCT   M3
     I                                          6   72COMM       01
     I                                         10  12 DEPT     M201
     I                                          5   7 DEPT     M202
     I                                         10  142SALARY     02
```

ZK-4413-85

For a 01 record type, matching field DEPT is in field location 10 through 12. For a 02 record type, matching field DEPT is in field location 5 through 7.

8.11.4 Using Matching Fields to Process More Than One File

The processing of a primary file with one or more secondary files is called multifile processing. In multifile processing without matching fields, VAX RPG II first reads all the records from the primary file, then reads all the records from each secondary file in the same order in which they are specified in the File Description specification. By using matching fields, your program can select the records from the secondary file before selecting the records from the primary file, based on the value of their matching fields.

When you use matching fields to process more than one file, the program selects records according to the contents of the matching fields, as follows:

- One record is read from every file and the matching fields are compared. If the records are in ascending sequence, the record with the lowest matching field value is selected for processing. If the records are in descending sequence, the record with the highest matching field value is selected for processing.
- When a record is selected from a file that is then processed, the next record from the file is read. The new record is then compared with the other records not selected in the previous cycle.

You can combine records with and without matching fields in the same file. Records without matching fields are processed before records with matching fields. If two or more of the records being compared have no matching fields, selection of those records is determined by the priority of their files, as follows:

- The records in primary files are processed before the records in secondary files.
- The records in secondary files are processed in order of appearance in the File Description specifications.

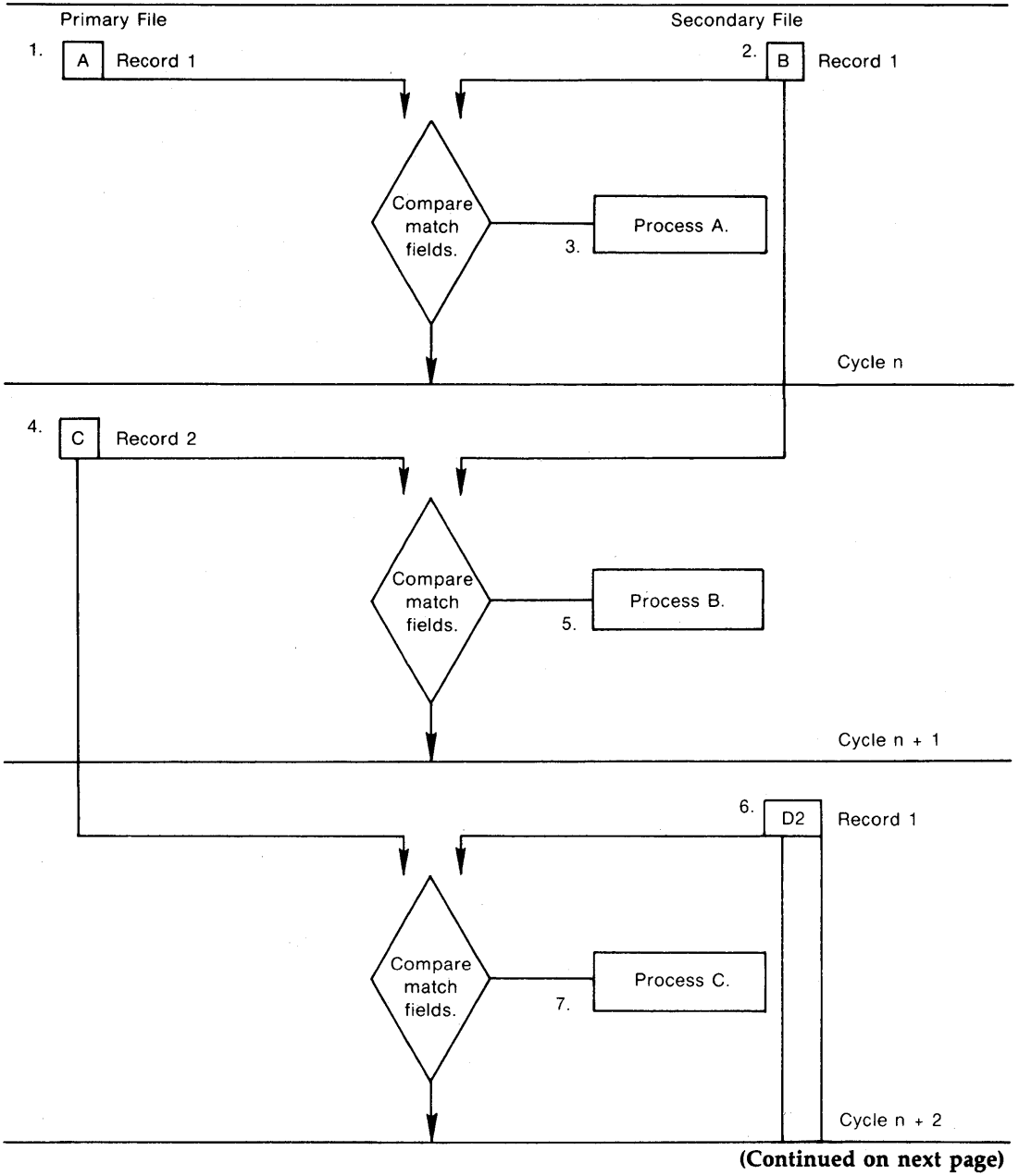
Table 8–3 shows that the matching fields from a primary file are compared with the matching fields from a secondary file to select records in ascending sequence. The letters represent the data in the matching fields.

Table 8–3: Matching Primary and Secondary File Field Values

Record Number	Primary File	Secondary File
1	A	B
2	C	D2
3	D1	X
4	F	Z

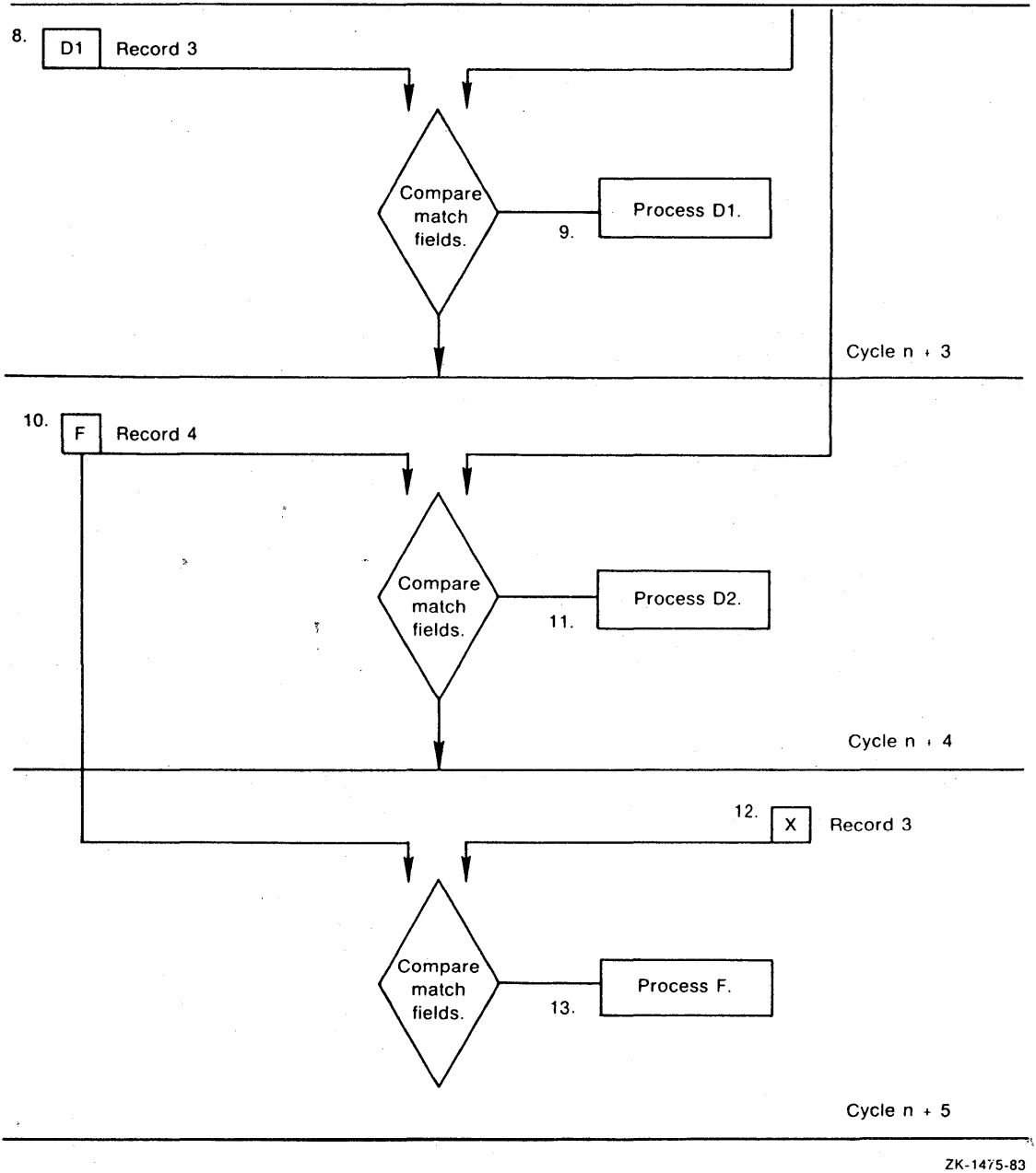
Figure 8–10 shows the logic flow when your program uses matching fields to do multifile processing. A keyed list follows the figure.

Figure 8–10: Using Matching Fields For Multifile Processing



(Continued on next page)

Figure 8–10 (Cont.): Using Matching Fields For Multifile Processing



ZK-1475-83

Key to Figure 8-10:

❶ The first record of the primary file is read and the matching field (A) is located.

❷ The first record of the secondary file is read and the matching field (B) is located.

❸ The contents of the matching field (A) of the first record in the primary file are compared with the contents of the matching field (B) of the first record in the secondary file. A is selected.

❹ The second record of the primary file is read and the matching field (C) is located.

❺ The contents of the matching field (B) of the first record in the secondary file are compared with the contents of the matching field (C) of the second record in the primary file. B is selected.

❻ The second record of the secondary file is read and the matching field (D2) is located.

❼ The contents of the matching field (D2) of the second record in the secondary file are compared with the contents of the matching field (C) of the second record in the primary file. C is selected.

❽ The third record of the primary file is read and the matching field (D1) is located.

❾ The contents of the matching field (D2) of the second record in the secondary file are compared with the contents of the matching field (D1) of the third record in the primary file. D1 is selected.

❿ The fourth record of the primary file is read and the matching field (F) is located.

⓫ The contents of the matching field (D2) of the second record in the secondary file are compared with the contents of the matching field (F) of the fourth record in the primary file. D2 is selected.

⓬ The third record of the secondary file is read and the matching field (X) is located.

⓭ The contents of the matching field (F) of the fourth record in the primary file are compared with the contents of the matching field (X) of the third record in the secondary file. F is selected. Because the primary file is now at its end, the remaining records in the secondary file (X and Z) are processed in order of appearance.

When the matching fields in a primary file match one or more of the matching fields in the secondary files, VAX RPG II sets the matching-record (MR) indicator on before detail-time calculations. You can use the MR indicator to condition calculation and output operations for the record just selected. The indicator remains on for one complete program cycle. It is set off if the record selected for processing contains no matching fields. A record selected using the FORCE operation code causes the MR indicator to remain off for one program cycle while the forced record is processed.

VAX RPG II processes matching records for two or more files in the following ways:

- When a record from the primary file matches a record from the secondary file, the record from the primary file is processed before the record from the secondary file is processed. The record-identifying indicator that identifies the record type just selected is on at the time the record is processed.
- When records from ascending files do not match, your program processes the record with the lowest matching field content first.
- When records from descending files do not match, your program processes the record with the highest matching field content first.
- A record type that has no matching field specification is processed immediately after the previous record is processed. In this case, the MR indicator is set off. If this record type is the first in the file, your program processes this record first, even when it is not in the primary file.
- The matching of records makes it possible to enter data from primary records into their secondary records because your program processes the record from the primary file before matching the record from the secondary file. However, the transfer of data from the secondary record to matching primary records can be done only when look-ahead fields are specified.

In the following example, matching fields are used to combine a primary file with two secondary files in ascending sequence. Record-identifying indicators are assigned in the following way:

- 01—Records from the primary file with matching fields
- 02—Records from the primary file without matching fields
- 03—Records from the first secondary file with matching fields

- 04—Records from the first secondary file without matching fields
- 05—Records from the second secondary file with matching fields
- 06—Records from the second secondary file without matching fields

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     FRECI99A IP AF      80            DISK
     FRECI99B IS  F      80            DISK
     FRECI99C IS  F      80            DISK
     FOUTPUT  O   F      80            DISK
     IRECI99A     01  80 C1
     I                                      1  80 TEXT
     I                                      1   2 MATCH   M1
     I            02  80 C2
     I                                      1  80 TEXT
     IRECI99B     03  80 C3
     I                                      1  80 TEXT
     I                                      1   2 MATCH   M1
     I            04  80 C4
     I                                      1  80 TEXT
     IRECI99C     05  80 C5
     I                                      1  80 TEXT
     I                                      1   2 MATCH   M1
     I            06  80 C6
     I                                      1  80 TEXT
     OOUTPUT  D       N1P
     O                        TEXT     80
```

ZK-4414-85

Table 8–4 lists the contents of the matching fields for all three files: primary, first secondary, and second secondary. Field values with A after the value represent values from the primary file. Field values with B after the value represent values from the first secondary file. Field values with C after the value represent values from the second secondary file.

Table 8–4: Matching Field Values

Record Number	Primary File	First Secondary File	Second Secondary File
1	none	none	10C
2	none	20B	30C
3	20A	30B	50C
4	20A	30B	50C
5	40A	60B	none
6	50A	none	60C
7	none	70B	80C
8	60A	80B	80C
9	80A	80B	none

Table 8–5 lists the steps involved in processing the files in Table 8–4 and those indicators that must be set on for the operation to occur.

Table 8–5: Processing Records with Matching Fields

Step	Record Type	Matching Field Value	Indicators for Processing
1	02	none	Not MR and 02
2	02	none	Not MR and 02
3	04	none	Not MR and 04
4	05	10C	Not MR and 05
5	01	20A	MR and 01
6	01	20A	MR and 01
7	03	20B	MR and 03
8	03	30B	Not MR and 03
9	03	30B	Not MR and 03
10	05	30C	Not MR and 05
11	01	40A	Not MR and 01
12	01	50A	MR and 01
13	02	none	Not MR and 02
14	05	50C	MR and 05

Table 8–5 (Cont.): Processing Records with Matching Fields

Step	Record Type	Matching Field Value	Indicators for Processing
15	05	50C	MR and 05
16	06	none	Not MR and 06
17	01	60A	MR and 01
18	03	60B	MR and 03
19	04	none	Not MR and 04
20	05	60C	MR and 05
21	03	70B	Not MR and 03
22	01	80A	MR and 01
23	03	80B	MR and 03
24	03	80B	MR and 03
25	05	80C	MR and 05
26	05	80C	MR and 05
27	06	none	Not MR and 06

8.12 Processing Files with Multiple Keys

The following program reads one input file with three keys. It uses three different file specifications to gather the three keys. Note that the three file names use identical fields, and that each file name uses a different key to point to the same file. Also note the use of the same fields by a data structure.

```
     0    I    1    I    2    I    3    I    4    I    5    I    6    I    7    I
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FIDXI01  IP  F      24  4AI    21 DISK
     FIDXJ01  IS  F      24  3AI     1 DISK   IDXI01
     FIDXK01  IS  F      24  2AI    11 DISK   IDXI01
     FIDX03A  O   F      24             DISK
     IIDXI01  AA
     I                                        1   3 PN
     I                                        4  10 PNAME
     I                                       11  12 WHOUSE
     I                                       13  17 COLOR
     I                                       18  20 WEIGHT
     I                                       21  24 QTY
     IIDXJ01  BB
     I                                        1   3 PN
     I                                        4  10 PNAME
     I                                       11  12 WHOUSE
     I                                       13  17 COLOR
     I                                       18  20 WEIGHT
     I                                       21  24 QTY
     IIDXK01  CC
     I                                        1  24 FIELDS
     IFIELDS      DS
     I                                        1   3 PN
     I                                        4  10 PNAME
     I                                       11  12 WHOUSE
     I                                       13  17 COLOR
     I                                       18  20 WEIGHT
     I                                       21  24 QTY
     OIDX03A  D       N1P
     O                       FIELDS    24
```

ZK-4667-85

8.13 File Buffers

The VAX RPG II compiler creates one buffer for each file in a VAX RPG II program. However, there are some programs in which input and output buffers for the same file should be distinct.

As an example, consider a program that is sequentially reading an indexed file and occasionally executing the ADD operation code in the file using EXCPT processing at total time. At the beginning of the logic cycle, a record is selected for input. (For this example, use the FOO record.) When a control break occurs, the program does some total-time processing and ends with an EXCPT operation on the same file. This results in a new record being added to the file, for example the BAR record. Continuing with the normal logic cycle, the program enters detail time and the record selected for input has become the BAR record which was just written instead of the expected FOO record. The same buffer is used for both input and output. When the BAR record is written, the record buffer is overwritten and its previous contents lost, so that when it is time for field extraction to occur, the wrong record is found.

There are a number of workarounds that you can use. The ADD records could be written to a distinct output file and merged with the master files outside of the application. Or, you can use the total-time processing to save the record to be written, and set an indicator on. Then, during detail-time processing, use the indicator to trigger the EXCPT operation. Because the fields of the input record have been populated by this time, no conflict occurs with a single record buffer.

The best alternative is to open two streams to the same file and have two F specifications which effectively refer to the same file. Normal input processing takes place using one stream and all ADD processing occurs on the other stream. Because a record buffer is allocated for each file, two buffers are created in this scheme, and no conflict occurs. Both files need to use the file sharing option (S in column 68 of the F specifications) in this case.

Only one update is allowed for each logic cycle for update files other than demand and chained files.

Chapter 9

Using Printer Output Files

If you want to create a formatted report by printing an output file, you must decide what the report will look like before you write your program. You must know what information is to be printed on each heading, detail, and total line, and where the individual fields are to appear.

Designing the physical layout of your report is an important part of the work necessary to produce a formatted report. VAX RPG II provides several features you can use to print certain information automatically and to control the printing of other information. Sections 9.1 and 9.2 describe these features and explain how to use them.

NOTE

Printer output files cause a file to be in VAX/VMS print-file format. The default PRINT command causes the insertion of a form-feed character when the form nears the end of a page. To suppress the insertion of form-feed characters, use the /NOFEED qualifier on the PRINT command when printing printer output files created by VAX RPG II programs.

9.1 Editing Output

VAX RPG II has several means of structuring and editing your program output:

- Edit codes are one-letter codes you can specify in column 38 of your Output specification to edit the data in a numeric output field.
- Constants are single-quoted literals (for example, 'foobar') you can specify in columns 45 through 70 of the Output specification to describe or label printer output columns, rows, and other items of importance to the program output.
- Edit words are special instructions placed in columns 45 through 70 of the Output specification that affect the sign of numeric data, the body of the field to be edited, and the characters that will be printed regardless of the sign. Certain digit selection, zero suppression, blank insertion, and currency symbol manipulation are all controlled by edit words.

You can use predefined edit codes and edit words to format numeric data for your report. Edit codes and words supply additional information about the output, thus increasing your report's usefulness to the end user. Section 9.1.2 describes constants and explains how to use them. Section 9.1.1 describes edit codes and their modifiers and explains how to use them. See Chapter 15 for detailed information on edit words.

9.1.1 Using Edit Codes and Edit Code Modifiers

You can specify specialized editing for numeric data by entering one of the one-character edit codes in column 38 of the Output specification. Edit codes consist of (1) simple edit codes (X, Y, and Z) that perform one predefined function, and (2) combined edit codes (1, 2, 3, 4, A, B, C, D, J, K, L, and M) that perform a combination of predefined functions. See Chapter 15 for detailed information on edit codes.

In most cases, using one or more edit codes to format numeric data is sufficient. However, in some cases you might want to use an edit code modifier for special formatting such as replacing leading zeros with asterisks (*) or putting a dollar sign ($) immediately to the left of the leftmost character.

See Chapter 15 for detailed information on edit code modifiers.

9.1.2 Using Constants

Constants are used to specify headings that describe the contents of a particular column. To specify a constant, enter the constant string, enclosed in apostrophes, in columns 45 through 70 (constant or edit word). In the following example, SAMPLES REPORT appears in character positions 24 through 35 of the printed output file:

```
                Type (HDTE)              Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)    | X                 ------------
                ||Space                  | Y date edit       Y Y 1 A J
                ||| Skip                 | Z zero suppress   Y N 2 B K
                ||| |                    |                   N Y 3 C L
                ||| |   Indicators       |Blank-after (B)    N N 4 D M
 File           ||| |   |        Field    ||End position
 name           ||| |   |        name     |||  Format (PB)
  |             ||| |   |          |      |||   |
 0|             ||BAB A NxxNxxNxx|      |||   |+ Constant or edit word  +
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
    **          ***** * *         *      ***---**                        ....
    0                                        35 'SALES REPORT'
```

ZK-4415-85

Rules

- Constants can contain from 1 to 24 characters.
- Enclose constants in apostrophes (for example, 'EMPLOYEE NAME'). The apostrophes are not printed.
- When using constants, leave columns 32 through 39 and column 44 blank.

9.2 Using Special Words

VAX RPG II provides special words that enable you to perform the following kinds of formatting:

- Printing the date on every page
- Printing a page number and incrementing the page number by one for each new page
- Repeating data fields in an output record

This section describes special words and explains how to use them.

9.2.1 Printing the Date: UDATE, UDAY, UMONTH, UYEAR

UDATE automatically prints the date in the format mm-dd-yyyy. To put slashes (/) between the fields (for example, 05/27/1986), specify Y in column 38 of the Output specification.

The default date is the system date. To change the default date, define the logical name RPG$UDATE to the date format you want. The following example changes the date to June 12, 1986, using the format dd-mmm-yyyy:

```
$ DEFINE RPG$UDATE "12-JUN-1986"
```

You can change the UDATE output format by specifying D, I, or J in column 21 of the Control specification. Specifying D changes the UDATE format to dd-mm-yyyy. Specifying I or J changes the UDATE format to dd.mm.yyyy.

You can use UDAY, UMONTH, and UYEAR to print each component of the date in the format you need, as shown in the following example:

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)  | X                   ----------
                ||Space                | Y date edit         Y Y 1 A J
                ||| Skip               | Z zero suppress     Y N 2 B K
                ||| |                  |                     N Y 3 C L
                ||| |   Indicators     |Blank-after (B)      N N 4 D M
     File       ||| |   |        Field ||End position
     name       ||| |   |        name  |||   Format (PB)
     |          ||| |   |        |     |||   |
    O|          ||BAB A NxxNxxNxx|     |||   |+ Constant or edit word  +
     0    |     1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *        *      ***---**                          ....
     O          H        1P
     O                                UYEAR      8
     O                                           9 '-'
     O                                UMONTH    11
     O                                          12 '-'
     O                                UDAY      14
```

ZK-4416-85

In this example, the special words UYEAR, UMONTH, and UDAY in the Output specification change the date format to yy-mm-dd. The output might look like this:

```
83-05-16
```

Rules

- You cannot specify Y in column 38 (edit code) of the Output specification for UDAY, UMONTH, or UYEAR. Instead, specify a constant in columns 45 through 70 (constant or edit word) to separate the day, month, and year.
- You can use these special words in factor 1 or 2 of the Calculation specification.
- You cannot use these special words in the result field of the Calculation specification.
- You cannot use the blank after option (column 39 of the Output specification) with these special words.

9.2.2 Printing the Time

The TIME operation code on a Calculation specification prints the system time of day on your program output.

See Section 16.11 for more information on using the TIME operation code.

9.2.3 Numbering Pages: PAGE and PAGE1 through PAGE7

VAX RPG II provides eight special words, PAGE and PAGE1 through PAGE7, for numbering pages in printed output files. VAX RPG II automatically increments the page number by one for each new page. You can use more than one special word to page number several different output files. For example, if your program produces four different reports, each of the four can have its own page numbering sequence by using PAGE1, PAGE2, PAGE3, and PAGE4. A total of eight concurrent page sequences is possible (using PAGE through PAGE7).

To use one of the special words for page numering, specify it as a field in the Input, Calculation, or Output specification. When you use a special word for page numbering as an input field or as the result field of a calculation, you can use any field length up to 15 digits, but you must specify zero decimal positions. VAX RPG II suppresses leading zeros and signs on output unless you use an edit word or an edit code, or specify a packed decimal or binary data format.

If you do not define the length of a special word for page numbering elsewhere (for example, defining a field to represent the page number as a result of a calculation), the page number is output as a four-digit numeric field with zero decimal positions. Page numbering begins with 1.

To change the beginning page number, enter the page number you want to use as an input record and name the field PAGE or use a PAGE field as the result of a calculation. Enter the field in columns 53 through 58 of the Input specification, as shown in the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      .
      .
      .
IINPUT    AA  99
I                                              1    40PAGE
      .
      .
      .
OOUTPUT   H  2      1P
O         OR        OF
O                             UDATE Y     8
O                                        36 'D E P O S I T '
O                                        49 ' R E P O R T'
O                                        68 'PAGE'
O                             PAGE       72
```

ZK-4417-85

In the preceding example, the contents of the field PAGE appear in character positions 69 through 72. If the value of the field is 0032, the page numbering begins with 33, because VAX RPG II adds 1 to the number. The output appears as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
5/16/83                 D E P O S I T  R E P O R T                  PAGE  33
```

Another way to change the page number is to assign the page number you want minus one to a PAGE field in the result field, as shown in the following example:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      .
      .
      .
C  N99                Z-ADD89         PAGE
C                     SETON                      99
      .
      .
      .
OOUTPUT  H  2     1P
O        OR       OF
O                          UDATE Y    8
O                                    36 'D E P O S I T '
O                                    49 ' R E P O R T'
O                                    68 'PAGE'
O                          PAGE       72
```

ZK-4418-85

The output appears as follows:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
 5/16/83                D E P O S I T  R E P O R T                PAGE  90
```

In the preceding example, Z-ADD assigns 89 to PAGE. VAX RPG II adds 1 to this number and begins numbering pages with 90. The assignment occurs when indicator 99 is set off so that VAX RPG II makes the initial page number assignment only once and not every time a record is read.

You can restart page numbering at any point in the program. Use any one of these methods to reset the value of a PAGE field:

- Specify the blank after option (column 39 of the Output specification) for a PAGE field to reset the page number to 1 after the current record is output.
- Use a PAGE field as the result of an operation in the Calculation specification or as an Input field.
- Use output indicators in the Output specification to condition the value of a PAGE field. When the indicator is set on, the value of the PAGE field is reset to 1 before the current record is output. You cannot use these indicators to control the printing of a PAGE field, because a PAGE field is always printed.

9.2.4 Saving Time by Repeating Data: *PLACE

You can use the special word *PLACE to repeat data in an output record. The fields or constants you want to repeat must have been previously defined. Then, you can use the same fields or constants without having to specify their field names (columns 32 through 37 of the Output specification) and end positions (columns 40 through 43 of the Output specification). When you specify *PLACE in columns 32 through 37, VAX RPG II repeats all the data between the beginning position and the highest end position for any previously defined field in the output record. To prevent overlapping, the end position on the same Output specification as *PLACE must be at least twice as high as the highest end position of the group of fields you want to repeat.

When using *PLACE, the following columns in the Output specification that contain *PLACE must be left blank:

- Column 38 (edit code)
- Column 39 (blank after)
- Column 44 (data format)
- Columns 45 through 70 (constant or edit word)

In the following example, *PLACE specifies these fields again:

- LIST#
- DESCR
- STOCK#
- ONHAND
- PRICE

```
     0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890
     FOUTI91  IP  F      26            DISK
     FOUT91A  O   F      80            PRINTER
     I*
     IOUTI91  AA  01
     I                                          1   6 STOCK#
     I                                          7  18 DESCR
     I                                         19  210ONHAND
     I                                         22  262PRICE
     C*
     C   01       LIST#      ADD  1          LIST#   30
     O*
     OOUT91A  D         N1P
     O                            LIST# Z     4
     O                            DESCR       18
     O                            STOCK#      26
     O                            ONHANDZ     31
     O                            PRICE K     39 '$'
     O                            *PLACE      79
```

ZK-4419-85

Sample output from this example might look like the following:

```
1  PARSNIPS       VEG1PQ   17   $.89     1  PARSNIPS       VEG1PQ   17   $.89
2  SKIM MILK      DAROSK  134  $1.70     2  SKIM MILK      DAROSK  134  $1.70
3  POTATO CHIPS   SNK945  100  $1.19     3  POTATO CHIPS   SNK945  100  $1.19
4  2 QRT PEPSI    DRNK1A   87  $1.29     4  2 QRT PEPSI    DRNK1A   87  $1.29
5  BAKED BEANS    CANFOD   90   $.65     5  BAKED BEANS    CANFOD   90   $.65
```

9.3 Conditioning Output Lines

Although you can use any type of indicator to condition output, the first-page (1P) and overflow indicators specifically condition output. Sections 9.3.1 and 9.3.2 describe how these indicators condition output.

9.3.1 Printing Lines Before Reading the First Record: First-Page Indicator

You can use the 1P indicator to condition those heading lines you want printed before VAX RPG II processes the first record.

You specify the 1P indicator in columns 24 and 25, 27 and 28, or 30 and 31 of the Output specification.

See Section 7.2.1 for complete information on the 1P indicator, including a program example to print heading lines before reading the first record.

9.3.2 Specifying Page Breaks: Overflow Indicator

You use overflow indicators to specify when a page break should occur before certain lines are printed. These indicators are used primarily to condition the printing of heading lines, but can also be used to condition calculation operations and other types of output lines.

You can use overflow indicators only for output files going to the printer. You define the indicator in columns 33 and 34 of the File Description specification. The same overflow indicator must be used to condition the overflow lines for that same file. If no indicator is specified for that file, VAX RPG II automatically handles overflow. See Section 9.4 for information on automatic overflow.

VAX RPG II sets on an overflow indicator only the first time an overflow condition occurs for the current page. An overflow condition exists whenever one of the following occurs:

- A line is printed on the overflow line.
- A line is printed past the overflow line.
- The overflow line is passed during a space operation.
- The overflow line is passed during a skip operation.

Rules

- Spacing past the overflow line sets the overflow indicator on.
- Skipping past the overflow line to any line on the same page sets the overflow indicator on.
- Skipping past the overflow line to any line on the new page does not set the overflow indicator on unless the skip is specified past the overflow line on the new page.

- A skip to a new page specified on a line not conditioned by an overflow indicator sets the overflow indicator off before the form advances to a new page.
- If you specify a skip to a new line and the printer is currently on that line, a skip does not occur.
- When an OR line is specified for an output print record, the space and skip entries of the preceding line are used. If space and skip requirements differ from the preceding line, enter space and skip entries on the OR line.
- An overflow indicator can appear on either line of an AND or an OR relationship. In an AND relationship, the overflow indicator must appear on the main specification line for that line to be considered an overflow line. In an OR relationship, the overflow indicator can be specified on either the main specification line or the OR line. However, only one overflow indicator can be associated with one group of output indicators.
- If an overflow indicator is used on an AND line, the line is not an overflow line. In this case, the overflow indicator is treated like any other output indicator.
- An overflow indicator cannot condition an exception line (E in column 15 of the Output specification), but can condition fields within the exception record.

During a normal program cycle, VAX RPG II checks whether the overflow indicator is set on only once (immediately after total-time output). Detection of the overflow indicator causes the following operations:

- VAX RPG II prints all total lines conditioned by an overflow indicator when the indicator is set on.
- VAX RPG II prints those heading and detail lines conditioned by an overflow indicator when the indicator is set on.
- Advancement to a new page does not happen automatically. Normally, one of the overflow lines specifies a skip to the top of a new page.

If the overflow indicator is set on, you can fetch the overflow routine before printing any total or detail line by specifying F (fetch overflow) in column 16 of the Output specification. Fetch overflow alters the

VAX RPG II logic cycle to prevent printing detail, total, and exception lines on or over the perforation between pages. When you fetch the overflow routine, VAX RPG II performs the following operations:

- When an output line specifies fetch overflow, VAX RPG II checks whether the overflow indicator for that file is set on. If it is, VAX RPG II calls the overflow routine and prints only those overflow lines associated with the file described on the Output specification.
- After VAX RPG II prints the overflow lines, it prints the line that specified the fetch overflow.
- VAX RPG II prints any detail-time and total-time lines left for that program cycle.

Rules

- If you want to fetch the overflow line for each record in an OR relationship, you must specify F (fetch overflow) in column 16 for each line.
- You cannot specify an overflow indicator in columns 23 through 31 on the same line with F (fetch overflow) in column 16.

To decide when to fetch the overflow routine, study all possible overflow situations and count lines, spaces, and skips to determine what happens when an overflow occurs.

In the following example, the length of a page is 15 lines. The overflow line is line 12. When the overflow line is reached, the overflow indicator is set on, which conditions the heading line that prints the date, report title, and page number at the top of each page.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FOUTI93  IP  F       74              DISK
     FOUT93A  O   F       80      OG      LPRINTER
     LOUT93A   15FL 120L
     IOUTI93  AA  01
     I                                          1   5 ZIP
     I                                         10  150CEN30
     I                                         16  210CEN40
     I                                         22  270CEN50
     I                                         28  330CEN60
     I                                         34  390CEN70
     I                                         40  450CEN80
     I                                         46  47 STATE
     I                                         48  59 COUNTY
     I                                         63  74 TOWN
     OOUT93A  H  102   1P
     O        OR       OG
     O                          UDATE Y   10
     O                                    47 'SOUTHERN NEW HAMPSHIRE'
     O                                    53 'TOWNS'
     O                          PAGE      77
     O         D 1      01
     O                          TOWN      13
     O                          COUNTY    26
     O                          STATE     30
     O                          CEN80 J   38
     O                          CEN70 J   46
     O                          CEN60 J   54
     O                          CEN40 J   62
     O                          CEN40 J   70
     O                          CEN30 J   78
```

ZK-4421-85

A sample of the output from this example might look like the following:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
  12/14/83                SOUTHERN NEW HAMPSHIRE TOWNS                        1
Hampstead    Rockingham    NH  3,785   2,401   1,261     823     823     775
Kingston     Rockingham    NH  4,111   2,882   1,672   1,002   1,002   1,017
Litchfield   Hillsborough  NH  4,150   1,420     721     341     341     286
Newmarket    Rockingham    NH  4,290   3,361   3,153   2,640   2,640   2,511
Atkinson     Rockingham    NH  4,397   2,291   1,017     434     434     407
Rye          Rockingham    NH  4,508   4,083   3,244   1,246   1,246   1,081
Hollis       Hillsborough  NH  4,679   2,616   1,720     996     996     879
Peterborough Hillsborough  NH  4,895   3,807   2,963   2,470   2,470   2,521
Raymond      Rockingham    NH  5,453   3,003   1,867   1,340   1,340   1,165
```

12/14/83		SOUTHERN NEW HAMPSHIRE TOWNS						2
Plaistow	Rockingham	NH	5,609	4,712	2,915	1,414	1,414	1,366
Windham	Rockingham	NH	5,664	3,008	1,317	630	630	538
Seabrook	Rockingham	NH	5,917	3,053	2,209	1,782	1,782	1,666
Pelham	Hillsborough	NH	8,090	5,408	2,605	979	979	814
Amherst	Hillsborough	NH	8,243	4,605	2,051	1,174	1,174	1,115
Milford	Hillsborough	NH	8,685	6,622	4,863	3,927	3,927	4,068
Bedford	Hillsborough	NH	9,481	5,859	3,636	1,561	1,561	1,326
Hampton	Rockingham	NH	10,493	8,011	5,379	2,137	2,137	1,507
Exeter	Rockingham	NH	11,024	8,892	7,243	5,398	5,398	4,872

12/14/83		SOUTHERN NEW HAMPSHIRE TOWNS						3
Goffstown	Hillsborough	NH	11,315	9,284	7,230	4,247	4,247	3,839
Londonderry	Rockingham	NH	13,598	5,346	2,457	1,429	1,429	1,373
Hudson	Hillsborough	NH	14,022	10,638	5,876	3,409	3,409	2,702
Merrimack	Hillsborough	NH	15,406	8,595	2,989	1,253	1,253	1,084
Derry	Rockingham	NH	18,875	11,712	6,987	5,400	5,400	5,131
Salem	Rockingham	NH	24,124	20,142	9,210	3,267	3,267	2,751
Portsmouth	Rockingham	NH	26,254	25,717	26,900	14,821	14,821	14,495
Nashua	Hillsborough	NH	67,865	55,820	39,096	32,927	32,927	31,463
Manchester	Hillsborough	NH	90,936	87,754	88,282	77,685	77,685	76,834

9.4 Automatic Overflow

When an overflow indicator is not assigned to an output file going to the printer, the compiler assigns the first unused indicator to the file. This causes a skip to line 1 whenever an overflow occurs, and the overflow routine executes for this file.

You can override the printing of overflow lines by specifying an overflow indicator on the File Description specification. However, do not use the same indicator to condition any output line. This causes continuous printing of lines, regardless of page boundaries.

9.5 Defining the Page Size

The Line Counter specification allows you to alter the default format of a printed output file[1]. You can use this specification to change the number of lines on a page and to change the overflow line.

[1] The default format is 66 lines, with the overflow line at 60.

To define the page size, you must make the following entries in the Line Counter specification:

- Columns 7 through 14 (file name)—specify the name of the output file. This file must have been previously defined on the File Description specification with PRINTER in columns 40 through 46 (device code) and L in column 39 (extension).
- Columns 15 through 17 (form length)—specify the number of lines printed on a page.
- Columns 18 and 19 (FL)—if you specify an entry in columns 15 through 17 (form length), you must enter FL in columns 18 and 19. This entry indicates to the compiler that columns 15 through 17 define the form length.

If you do not specify an entry for form length, the default is 66 lines.

To define the overflow line, you must make the following entries in the Line Counter specification:

- Columns 20 through 22 (overflow line number)—specify the line number where an overflow occurs.
- Columns 23 and 24 (OL)—if you specify an overflow line number in columns 20 through 22, you must enter OL in columns 23 and 24. This entry indicates to the compiler that columns 20 through 22 define the overflow line number.

If you do not specify an entry for the overflow line, the default is line 60.

9.6 Spacing and Skipping Lines

You can define how your printed output file will look by specifying the number of lines to space or skip. Spacing is relative to the line currently being printed; therefore, use spacing between detail lines in a report. Skipping lines repositions the printer to an absolute line number; therefore, specify skipping for the column headers of a report. For example, if you specify skip to line number 2, the output line associated with that specification will be printed only on the second line of each page.

To specify the number of lines to space, you must make the following entries in the Output specification:

- Column 17 (space before)—specifies the number of lines to be spaced before printing a line.
- Column 18 (space after)—specifies the number of lines to be spaced after printing a line.

To specify the number of lines to skip, you must make the following entries in the Output specification:

- Columns 19 and 20 (skip before)—specifies the line number to skip to before printing a line.
- Columns 21 and 22 (skip after)—specifies the line number to skip to after printing a line.

If you make entries in both spacing and skipping columns for the same line, VAX RPG II formats the output in the following order:

1. Skip before
2. Space before
3. Print the output line
4. Skip after
5. Space after

You can specify entries in columns 17 through 22 (space and skip) for the second line in an OR relationship; otherwise, the preceding line specifies the entries for spacing and skipping.

NOTE

If the line printer listing of a printer output file includes an unexpected blank page at the end of the file, use the DCL PRINT/NOFEED/PASSALL command.

The following example prints TOP on line 1, TEST LINE on line 7, PRINT TWICE FOR BOLDING on line 13, and the fields beginning on line 16.

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)  | X                   -----------
                ||Space                | Y date edit         Y Y 1 A J
                ||| Skip               | Z zero suppress     Y N 2 B K
                ||| |                  |                     N Y 3 C L
                ||| |   Indicators     |Blank-after (B)      N N 4 D M
     File       ||| |   |        Field  ||End position
     name       ||| |   |        name   |||   Format (PB)
      |         ||| |   |        |      |||   |
     O|         ||BAB A NxxNxxNxx|      |||   |+ Constant or edit word  +
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *      ***---**                          ....
     OOUT92A    H         1P
     O                                        41 'TOP'
     O          H 320411 1P
     O                                        44 'TEST LINE'
     O          H  0     1P
     O                                        30 'PRINT TWICE FOR BOLDING'
     O          H     15 1P
     O                                        30 'PRINT TWICE FOR BOLDING'
     O          D 1     N1P
     O                          DESCR         18
     O                          STOCK#        26
     O                          ONHANDZ       31
     O                          PRICE K       39 '$'
```

ZK-4422-85

Sample output from this example might look like the following:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
1                                        TOP
2
3
4
5
6
7                                   TEST LINE
8
9
10
11
12
13     PRINT TWICE FOR BOLDING
14
15
16     1 LB CARROTS   VEG1MQ    47    $.79
17     6 PACK SODA    DRNK2A    40   $1.48
18     1 LB BUTTER    DAROBT    38   $1.59
19     STEAK          METO      22   $3.15
20     HEAD LETTUCE   VEG1WQ    63    $.35
```

Chapter 10
Using Tables

In VAX RPG II, a table is a collection of similar data items arranged in a specific order. Each entry in a table must have the same length and the same data type (either character or numeric). There are two types of tables you can use to locate a specific data item quickly and easily.

- Single tables—consist of one group of similar entries. When you search a single table, the result of the search is whether the item you are searching for is present in the table. If the searched-for item is found, that entry becomes the current entry.
- Related tables—are two associated tables that can be entered in alternating format. You search the first table to find out if the entry is present. If the entry is found, VAX RPG II retrieves the corresponding entry from the second table. Related tables need not have the same number of entries unless they are described in alternating format in the same Extension specification.

If you describe a table in alternating format, the first entry from the first table is read first; then, the first entry from the second table is read. This alternate reading continues until all entries from both tables are read. Together, the corresponding entries from each table form one record. For an example of alternating format using arrays, see Section 11.5.4.

Types of tables are differentiated by whether they are loaded at compile time or preexecution time. Loading is the process by which the program assigns the data you supply to the elements in the table.

The following characteristics determine when a table should be loaded:

- The contents of a table
- The frequency with which the entries in the table require changing
- The way the table is to be used

10.1 Compile-Time Tables

Compile-time tables are part of the source program. They are compiled with the source program and become a permanent part of the object program. The following example shows a source program and a compile-time table:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890

     FINPUT   IPE F      30            DISK
     FREPORT  O          40            DISK
     E                    TABA   10  50  5
     IINPUT   AA  01
     I                                        1   5 ITEM
     I                                        6  102FLD1
     I                                       15  30 FLD2
     C   01       ITEM      LOKUPTABA                    11
     C  N11                 SETON                     H1
     C   11       100       ADD  FLD1      NEW     62
     OREPORT  D         01 11
     O                        NEW    B  20
//
10001100021000310004100051000610007100081000910010
20001200022000320004100052000610007200082000920010
30001300023000330004100053000610007300083000930010
40001400024000340004100054000610007400084000940010
50001500025000350004100055000610007500085000950010
/*
```

ZK-4431-85

One advantage of compile-time tables is that they do not need to be loaded separately each time the program is run. However, if you need to change any of the entries in a compile-time table, you must revise the table and then recompile the program with the revised table. You can, however, make temporary changes in the table during calculations. To make these temporary changes permanent, you must output the table. See Section 10.8 for information about outputting tables.

The data in a compile-time table must follow the source program and any alternate sequence records.

10.2 Preexecution-Time Tables

Preexecution-time tables are not part of the object program; each table is loaded separately from an input data file. One advantage of preexecution-time tables is that you can make frequent changes to the table without recompiling the program.

Preexecution-time tables are loaded before the first program cycle begins.

10.3 Creating Table Input Records

Table input records are the values for the entries in a table. When creating table input records, observe the following rules.

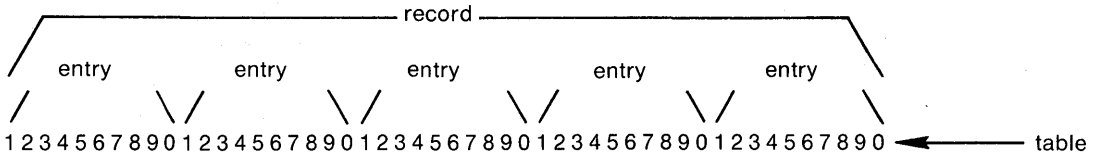
General Rules

- The first entry must begin in character position 1.
- All entries must be contiguous, with no space between entries, as shown in Figure 10–1.
- You cannot span an entry across two records. Therefore, the length of a record is limited to the device's maximum record length. If you use related tables in alternating format, corresponding records cannot exceed the maximum record length.
- Each input record, except the last, must have the same number of entries. This record can be shorter to accommodate an uneven number of entries.

Compile-Time Rules

- The first record must be preceded by a record containing either double slashes (//) and a blank or double asterisks (**) and a blank in character positions 1 through 3. Because these strings are delimiters, records in a compile-time table cannot contain either of these three characters in positions 1 through 3.
- The last record of the last compile-time table or array can be followed with a record containing a slash and an asterisk (/*) in the first two character positions. This record is optional and must be the *last* record in the source program, if used.

Figure 10–1: Table Input Record

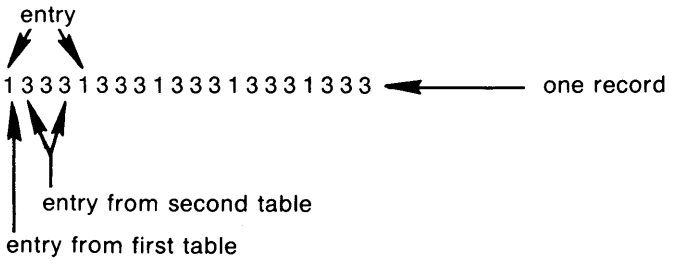


The table in Figure 10–1 consists of five entries in a record, and each entry is 10 characters long.

When creating table input records for related preexecution-time and compile-time tables in alternating format, you must enter an entry from the first table and then follow with the corresponding entry from the second table.

If you define each entry from the first table to be one character long and each entry from the second table to be three characters long, your table input record might appear as in Figure 10–2.

Figure 10–2: Related Tables



The table in Figure 10–2 consists of fives entries in a record, and each entry consists of two related entries. The first entry is one character long. The second entry is three characters long.

10.4 Defining Tables

To define a single table, you must make the following entries in the Extension specification:

- Columns 27 through 32 (table name)—specify the name of the table. Table names can be up to six characters long, but the first three characters must be TAB.
- Columns 33 through 35 (entries per record)—specify the number of entries in a record. Because tables can have one or more entries for each record, calculate the maximum number of entries in a record by dividing the record length by the length of an entry.
- Columns 36 through 39 (number of entries per table)—specify the number of entries in the table.
- Columns 40 through 42 (length of entry)—specify the length of each entry.
- Column 43 (data format)—if the table contains numeric data, you must specify its format. Specify P (packed decimal format), B (binary format), or leave the entry blank (overpunched decimal format). When you specify packed decimal format, make sure the length of entry represents the length of the numeric data in unpacked form. When you specify binary format, the length of entry you specify must indicate the number of bytes required to store the binary field. (Use 4 for two-byte signed binary numbers or 9 for four-byte signed binary numbers.)
- Column 44 (decimal positions)—for numeric data, specify the number of positions to the right of the decimal point. You must specify 0 for no Decimal positions.
- Column 45 (sequence)—specify ascending (A) or descending (D) to indicate that the entries in a table are in the specified sequence, or leave this column blank to specify an unsequenced table.

There are additional considerations for compile-time tables and preexecution-time tables, which are discussed in Sections 10.4.1 and 10.4.2.

10.4.1 Defining a Compile-Time Table

To define a compile-time table, you must make the entries shown for any single table. There is one notable exception to the requirements shown in Section 10.4—columns 43 and 55 must be blank for a compile-time table because packed and binary data formats are not allowed in VAX RPG II program source code.

In the following example, the table name is TABLE1. There are 10 entries in the table, with one entry in each record. The length of each entry is five digits, with two decimal positions. The data type of the entry in each record is overpunched numeric by default.

```
                                          ------F = Format (PB)
                                         | -----D = Decimal positions
                                         || ----S = Sequence (AD)
                                         |||
                                         |||Alternating table or array
      From     To       Table EntEnt Len|||name  Len
      file     file     or    perin  of F|||     of F
      name     name     array RecTbl Ent|D||     Ent|D
      |        |        name  |   |   | ||S|     |  ||S
E     |        |        |     |   |   | ||||     |  |||+-- Comments ---+
  0   |   1    |    2   |   3   |   4   |   5   |   6   |   7   |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *....*    *        *     *--*---*--****     *--****
E                       TABLE1  1  10  5 2
```

ZK-4423-85

You can define one or two tables either individually, or as a main table with an alternate table defined in alternating format. To define an alternate table, you must make the following entries for the alternate table in the same Extension specification you used to describe the main table:

- Columns 46 through 51 (table name)—specify the name of the alternate table. Table names can be up to six characters long, but the first three characters must be TAB.
- Columns 52 through 54 (length of entry)—specify the length of each entry in the alternate table.
- Column 55 (data format)—if the alternate table contains numeric data, you must specify its format. Specify packed decimal format (P), binary format (B), or leave the entry blank (overpunched decimal format). When you specify packed decimal format, make sure the length of entry represents the length of the numeric data in unpacked form.

When you specify binary format, the length of entry you specify must indicate the number of bytes required to store the binary field. (Use 4 for two-byte signed binary numbers or 9 for four-byte signed binary numbers.)

- Column 56 (decimal positions)—for numeric data, specify the number of positions to the right of the decimal point. You must specify 0 for no decimal positions.
- Column 57 (sequence)—specify ascending (A) or descending (D) to indicate that the entries in a table are in the specified sequence, or leave this column blank to specify an unsequenced table.

The main table's values for entries per table (columns 36 through 39), from file name (columns 11 through 18), and entries per record (columns 33 through 35) are also used for the alternate table.

In the following example, two related tables are loaded from the input file INPUT. The second table, TAB2, is the alternate table.

```
                                          ------F = Format (PB)
                                         | -----D = Decimal positions
                                         || ----S = Sequence (AD)
                                         |||
                                         |||Alternating table or array
          From     To       Table EntEnt Len|||name  Len
          file     file     or    perin  of F|||     of F
          name     name     array RecTbl Ent|D||     Ent|D
          |        |        name  |   |   |  ||S|     |  ||S
     E    |        |        |     |   |   |  ||||     |  |||+-- Comments ---+
      0   |    1   |    2   |    3   |    4   |    5   |    6   |    7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     *....*        *        *      *--*---*--*****    *--****
     E    INPUT             TAB1     2   4  5 0ATAB2    5 0A
```

ZK-4424-85

When defining compile-time tables, observe the following rules.

Rules

- If the compile-time table contains numeric data, it must be in over-punched format. Therefore, leave column 43 (data format) blank or leave column 55 (data format) blank, if you are using related tables in alternating format.
- The input records for compile-time tables must be in the same order in which the tables appear in the Extension specification.

10.4.2 Defining a Preexecution-Time Table

To define a preexecution-time table, make the same entries you made for a single table as in Section 10.4. In addition, in columns 11 through 18 (from file name), enter the name of the input file that contains the data for the table, as shown in the following example:

```
                                          ------F = Format (PB)
                                         | -----D = Decimal positions
                                         || ----S = Sequence (AD)
                                         |||
                                         |||Alternating table or array
          From     To       Table EntEnt Len|||name  Len
          file     file     or    perin  of F|||     of F
          name     name     array RecTbl Ent|D||     Ent|D
          |        |        name  |  |   |  ||S|     |  ||S
     E    |        |        |     |  |   |  ||||     |  |||+-- Comments ---+
    0     |    1   |    2   |    3   |    4   |    5   |    6   |    7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     *....*        *        *     *--*---*--*****    *--****
     E    INPUTFIL         TABLEA 10  50  5
```

ZK-4425-85

The table input file must be defined in a File Description specification by specifying T in column 16 (file designation).

When using preexecution-time tables, observe the following rules.

Rules

- The input file cannot contain more entries than are defined for the table. If it does, a run-time error occurs.
- The input file can contain fewer entries than are defined for the table, only if you do not specify a sequence. When you do not specify a sequence and the table contains fewer entries than are defined, the remaining entries are automatically filled with blanks for character data or zeros for numeric data.

10.5 Referencing Table Entries

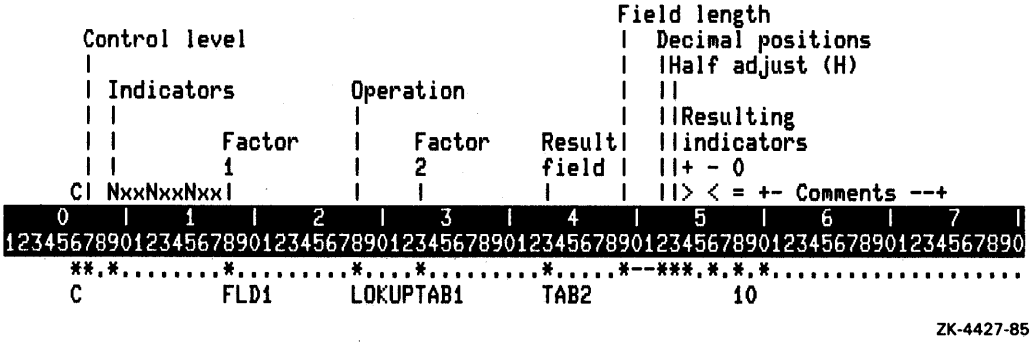
When you use a table name as an operand other than as factor 2 in an operation or other than as the result field in a LOKUP operation, the table name refers to the data retrieved by the last successful search. You can then use the entry as an operand in a calculation or modify the contents of the entry when the table name is used as the result field in a calculation.

In the following example, FLD1 is the search argument in the LOKUP operation. If the program can locate FLD1 in TAB1, indicator 10 is set on. Then, the result of the calculation on the next line replaces the current contents of the located entry in TAB1 because the table entry is used as the result field.

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |         Factor    |    Factor    Result|  |||indicators
     | |         1         |    2         field |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |  |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    **.*........*.........*....*.........*.....*--***.*.*.*.....................
    C           FLD1      LOKUPTAB1                      10
    C   10      TAB1      MULT 100       TAB1
```

ZK-4426-85

You can specify which entry is the current entry for related tables and then reference the current entry in subsequent calculations. In the following example, FLD1 is the search argument in the LOKUP operation. If the program locates FLD1 in TAB1, that entry becomes the current entry. Then, VAX RPG II locates the corresponding entry in TAB2 and it then becomes the current entry for TAB2. When you reference these entries in subsequent calculations, VAX RPG II uses the current entry in both tables.



ZK-4427-85

10.6 Searching Tables

The LOKUP operation code searches for an entry in a table. This operation starts with the first entry and searches each element for a match with the search argument. Specifying a table sequence is not necessary when performing LOKUP operations for an equal match. However, if you specify a sequence, the table can be searched faster. To save time in searching an unsequenced table, place the more frequently referenced entries at the beginning of the table.

To search a table for an entry, you must make the following entries in the Calculation specification:

- Columns 18 through 27 (factor 1)—specify a field or literal representing the entry you want to locate. Make sure the search argument has the same length and data format as the entries of the table being searched.
- Columns 28 through 32 (operation code)—specify the LOKUP operation code.
- Columns 33 through 42 (factor 2)—specify the name of the table to be searched.
- Columns 54 through 59 (resulting indicator)—specify one or more indicators to condition the search and to indicate whether the search has been successful. You can use this indicator to condition subsequent calculation and output operations.

In the following example, the program tries to match the search argument ITEM with an entry in table TABA. If a matching entry is found, indicator 11 is set on. If no matching entry is found, the halt indicator H1 is set on and the program terminates.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789

     FINPUT   IPE F      30            DISK
     FREPORT  O          40            DISK
     E                    TABA   10  50  5
     IINPUT   AA  01
     I                                        1   5 ITEM
     I                                        6  102FLD1
     I                                       15  30 FLD2
     C   01       ITEM      LOKUPTABA                      11
     C  N11                 SETON                        H1
     C   11       100       ADD  FLD1      NEW     62
     OREPORT  D         01 11
     O                         NEW    B  20
//
10001100021000310004100051000610007100081000910010
20001200022000320004100052000610007200082000920010
30001300023000330004100053000610007300083000930010
40001400024000340004100054000610007400084000940010
50001500025000350004100055000610007500085000950010
/*
```

ZK-4431-85

In this compile-time table, there are 10 entries in a record and 50 entries in a table. Each entry is five characters long.

When you specify a sequence (either ascending or descending), you can use resulting indicators (EQUAL, HIGH, and LOW) in the Calculation specification to indicate the condition to search for and the result of the search. You can specify one of the following search conditions:

- Columns 54 and 55 (HIGH)—nearest to but greater than value only
- Columns 56 and 57 (LOW)—nearest to but less than value only
- Columns 54 and 55, and 58 and 59 (EQUAL or HIGH)—equal or nearest to but greater than value
- Columns 56 and 57, and 58 and 59 (EQUAL or LOW)—equal or nearest to but less than value

The following program searches the unsequenced table TABLE2 for the value LENGTH and searches the sequenced table TABLE1 to check for a value greater than or equal to COST. If both conditions are satisfied, the subroutine PROCES is called to process the entry.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
.....**.*........*.........*....*.........*.....*--***.*.*.*
     FFILE1   IT  F  80  80           EDISK
     FFILE2   IT  F  80  80           EDISK
     FINFILE  IP  F  80  80            DISK
     E    FILE1           TABLE1  1   6  3 2A
     E    FILE2           TABLE2  1   6  3 0
     IINFILE  AA  11
     I                                        1   32COST
     I                                        4   60LENGTH
     I                                        7  100NUMBER
     C   11       LENGTH    LOKUPTABLE2                 20
     C  N20 11              GOTO NOPROC
     C   11       COST      LOKUPTABLE1              26  26
     C  N26                 GOTO NOPROC
     C                      EXSR PROCES
     C            NOPROC    TAG
```

ZK-4430-85

You can also specify a table in the result field to retrieve the entry that corresponds to the entry located in a LOKUP operation. See the example in Section 10.6 for an example of the LOKUP operation.

10.7 Updating Tables

To change the contents of an entry in, or add new entries to, a preexecution table, edit the input file that contains the table. You can also use a program to modify a table and output the new entries.

The following example searches related tables in alternating format. The first table, TABA, consists of a list of numbers of items in stock. The second table, TABB, consists of a list of unit prices corresponding to the item numbers. We want to raise the unit price of each item by 5% and output the updated table.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        ***** * *          *      ***---**
     FMASTER   IPE F       30              DISK
     FTABLE1   IT  F       22             EDISK
     FTABLE2   O   F       22              DISK
     FREPORT   O   F       60              DISK
     E     TABLE1   TABLE2  TABA     2  10  5    TABB     6 2
     IMASTER   AA   01
     I                                              1    5 ITEM
     C    01         ITEM        LOKUPTABA        TABB            11
     C   N11                     SETON                          H1
     C    11         1.05        MULT TABB        TABB     62H
     OREPORT   D           11
     O                               TABB       20
```

ZK-4428-85

The related tables, TABA and TABB, are preexecution-time tables. They are loaded from the input table file TABLE1. In the Extension specification, the output file TABLE2 is automatically created. (Automatic creation means that the output file does not require an Output specification.)

When the program executes, it reads the first record from the primary input file MASTER. If the search argument ITEM is matched, indicator 11 is set on and the corresponding entry from TABB is made available for processing. If no match is found, the halt indicator H1 is set on and the program terminates without creating the output file TABLE2.

When the program ends, the tables TABA and TABB are written to file TABLE2 with the same number of entries per record as the table input file TABLE1.

10.8 Outputting Tables

When you specify the name of an output file in columns 19 through 26 (to file name) of the Extension specification, your program creates the file automatically, as shown in the example in Section 10.7.

When you specify a table as a field on an Output specification, you can output only the entry found by the last LOKUP operation.

In the following example, the table TABSH is read from the file TABFILE. For this example, the table is short; that is, not all 80 entries contain data. The LOKUP operation searches the table for the first entry containing zeros. This entry is replaced with a field read from the input file IFILE. The EXCPT operation code outputs the entry TABSH with the new data. Remember, the entry that is updated and then output by the Output specification is the entry found by the last LOKUP operation. When the last cycle occurs, the entire updated table will be written to the file TABFILE2.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

     FIFILE   IP  F      80            DISK
     FTABFILE IT  F      80           EDISK
     FTABFILE2O   F      80            DISK
     FOFILE   O   F      80            DISK
     E    TABFILE TABFILE2TABSH  10  80  4 0
     IIFILE   AA  01
     I                                          1   40ENTRY
     C   01      0000      LOKUPTABSH                    20
     C   01 20             Z-ADDENTRY     TABSH
     C   01 20             EXCPT
     OOFILE   E
     O                         TABSH     10
```

ZK-4429-85

Chapter 11

Using Arrays

In VAX RPG II, an array, like a table, is a collection of similar data items arranged in a specific order. You can reference individual array elements (entries) by specifying an array index, or process an entire array by specifying the array name during calculation operations.

You use an array instead of a table when you want to affect all the elements in the array with a single reference or be able to reference a specified number of separate elements at the same time. For example, when you want to compute a 5% sales tax for each element in an array, you use a single specification to perform the operation for every element.

Types of arrays are differentiated by whether they are loaded at compile time, preexecution time, or execution (run) time. Loading is the process by which the program assigns the data you specify to the elements in an array.

The following characteristics determine when an array should be loaded:

- The contents of an array
- The frequency with which the elements in the array require changing
- The way the array is to be used

11.1 Compile-Time Arrays

Compile-time arrays are part of the source program. They are compiled with the source program and become a permanent part of the object program. One advantage of compile-time arrays is that they do not need to be loaded separately each time the program is run. However, if you need to change any of the entries in a compile-time array, you must revise the array and then recompile the program with the revised array. You can, however, make temporary changes in the array during calculation operations. To make these temporary changes permanent, you must output the array and then, using the output file as input, recompile the program. See Section 11.10 for information about outputting arrays.

When you use a compile-time array, the array input data must follow the source program and any alternate sequence (ALTSEQ) records. If you use more than one array, the data for each array must follow in the same sequence as specified on the Extension specifications.

The following example shows a source program with the input data for two compile-time arrays and their alternate compile-time arrays.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
01010H                                                                          NOPRIN
01040FPROCD   IP  F      80            DISK   NOPDAT                            NOPRIN
01050FINLIST  O   F     132     OF     PRINTER                                  NOPRIN
02010E                  AR1     1   5  5 0AALT    20                           NOPRIN
02020E                  AR2     4   4  5 0AALT     4 2
03010IPROCD   AA  01
03020I                                      1   50PRODNO
03030I                                      6   80QUAN
04010C                    Z-ADD1          I       20
04020C            PRODNO  LOKUPAR1,I                      20
04030C                    Z-ADD1          T       20
04040C            PRODNO  LOKUPAR2,T                      21
04050C   21       QUAN    MULT ALT2,T     AMT     72
05010OINLIST  H   201   1P
050200        OR        OF
020300                        UDATE       18 '  /  /  '
050400                                    47 'INVENTORY PARTS LIST'
050500                        PAGE        65 '  0 '
050600         H  1     1P
050700        OR        OF
050800                                    32 'PRODUCT        PRODUCT'
050900                                    53 'UNIT'
051000         H  2     1P
051100        OR        OF
051200                                    17 'NUMBER'
051300                                    45 'DESCRIPTION        QTY'
051400                                    64 'PRICE     AMOUNT'
060100         D  1     01
060200                        PRODNO      16 '    0'
060300                   20   ALT1,I      39
060400                  N20               39 '***NO DESCRIPTION***'
060450                   21   ALT2,T      53 ' 0.  '
060500                        QUAN        45 ' 0 '
060700                  N21               53 '*NONE'
060800                   21   AMT         65 '  ,  0.  '
060900         T 1      LR
061000                                    27 'END OF PRICE LIST'
**                                                  )
17526BOLT                                           |
18171SCREW                                          |  compile-time array AR1
19226NAIL                                           |  and the alternate compile-time
25116NUT                                            |  array ALT1
29258MAGNESIUM COVER                                |
**                                                  )
1752601261817100591922601732925858 43               }  compile-time array AR2 and
/*                                                  )  the alternate compile-time
                                                       array ALT2
```

ZK-4448-85

11.2 Preexecution-Time Arrays

Preexecution-time arrays are not part of the object program; each array is loaded separately and is used like an input data file. One advantage of preexecution-time arrays is that you can make frequent changes to the array without recompiling the program.

11.3 Execution-Time Arrays

Execution-time arrays are created by using Input or Calculation specifications. These arrays are loaded either from input data or as the result of calculation operations after program execution begins.

11.4 Creating Array Input Records

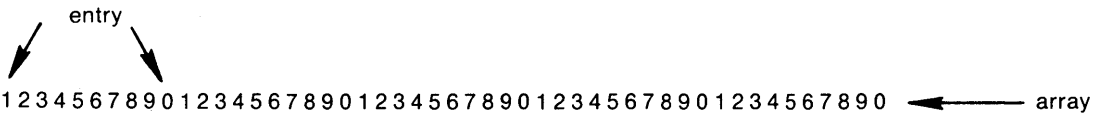
When creating array input records for compile-time and preexecution-time arrays, observe the following rules.

General Rules

- The first entry must begin in character position 1; all entries must be contiguous, with no space between entries, as shown in Figure 11–1.
- You cannot span an entry across two records. Therefore, the length of a record is limited to the device's maximum record length. If you use related arrays in alternating format, corresponding entries cannot exceed the maximum record length.
- Each array input record, except the last, must have the same number of entries. This record can be shorter to accommodate an uneven number of entries.

  The array in Figure 11–1 consists of five entries, and each entry is 10 characters long.

Figure 11–1: Array Input Record



ZK-1473-83

When creating compile-time array input records, observe the following rules.

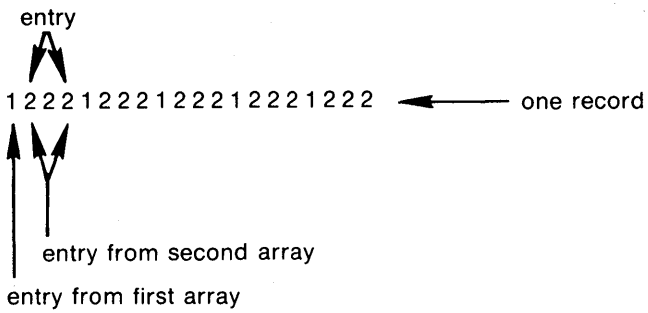
Compile-Time Rules

- The first record must be preceded by a record containing either double slashes (//) and a blank or double asterisks (**) and a blank in character positions 1 through 3. Because these strings are delimiters, compile-time array records cannot contain either of these characters in positions 1 through 3.
- The last record of the last compile-time table or array can be followed by a record containing a slash and an asterisk (/*) in the first two character positions. This record is optional and must be the last record in the source program, if used.

When creating array input records for related preexecution-time and compile-time arrays in alternating format, you must enter an entry from the first array and then follow with the corresponding entry from the second array.

If you define each entry from the first array to be one character long and each entry from the second array to be three characters long, your array input record might appear as in Figure 11–2.

Figure 11–2: Related Arrays



The array in Figure 11–2 consists of five entries in a record, and each entry consists of two related entries. The first entry is one character long. The second entry is three characters long.

11.5 Defining Arrays

To define an array, you must make the following entries in the Extension specification:

- Columns 27 through 32 (array name)—specify the name of the array. You cannot use TAB as the first three letters of an array name.
- Columns 36 through 39 (number of entries per array)—specify the number of entries in the array.
- Columns 40 through 42 (length of entry)—specify the length of each entry.
- Column 44 (decimal positions)—for numeric data, specify the number of positions to the right of the decimal point. You must specify 0 for no decimal positions.
- Column 45 (sequence)—specify ascending (A) or descending (D) to indicate that the entries in an array are in the specified sequence, or leave this column blank to specify an unsequenced array.

11.5.1 Defining a Compile-Time Array

To define a compile-time array, you must make the following entry in the Extension specification in addition to the entries required for all arrays:

- Columns 33 through 35 (entries for each record)—specify the number of entries in a record. Arrays can have one or more entries per record. The length of all entries in a compile-time array cannot exceed 96 characters. All records, except the last, must contain the same number of entries; each entry must be the same length.

The following example describes the compile-time array A1. The array has eight entries with four entries in each record. Each entry is a character field that is six bytes long. The array records are located at the end of the program.

```
                                          ------F = Format (PB)
                                         | -----D = Decimal positions
                                         || ----S = Sequence (AD)
                                         |||
                                         |||Alternating table or array
         From    To      Table EntEnt Len|||name  Len
         file    file    or    perin  of F|||     of F
         name    name    array RecTbl Ent|D||     Ent|D
         |       |       name  |  |   |  ||S|     |  ||S
    E    |       |       |     |  |   |  ||||     |  |||+-- Comments ---+
    0    |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    *....*       *       *     *--*---*--*****    *--****
    E                    A1      4   8  6
    .
    .
    .
**
KAUNISKAUPPANAINENKAIKKI
MUKAVAPALJONJUUSTOOSOITE
/*
```

ZK-4434-85

11.5.2 Defining a Preexecution-Time Array

To define a preexecution-time array, you must make the following entries in the Extension specification in addition to the entries required for all arrays:

- Columns 11 through 18 (from file name)—specify the name of the input file that contains the data for the array. This input file is called a table input file. It must be defined in a File Description specification by specifying T in column 16 (File designation); the T associates the file with the array.
- Columns 33 through 35 (entries per record)—specify the number of entries in a record. Arrays can have one or more entries per record. The length of all entries in a preexecution-time array cannot exceed the maximum number of characters for the device from which the array is loaded. All records except the last must contain the same number of entries; each entry must be the same length.

If your preexecution-time array contains numeric data, you can indicate the data format by specifying packed decimal format (P) or binary format (B), or by leaving the column blank (overpunched decimal format). When you specify packed decimal format, make sure the length of entry represents the length of the numeric data in unpacked form. When you specify binary format, the length of entry you specify must indicate the number of bytes required to store the binary field. (Use 4 for two-byte signed binary numbers or 9 for four-byte signed binary numbers.)

When using preexecution-time arrays, observe the following rules.

Rules

- The input file cannot contain more entries than are defined for the array. If it does, a run-time error occurs.
- The input file can contain fewer entries than are defined for the array, but only if you do not specify a sequence. When you do not specify a sequence and the array contains fewer entries than are defined, the remaining entries are automatically filled, either with blanks for alphanumeric data or with zeros for numeric data.

11.5.3 Defining an Execution-Time Array

To define an execution-time array, the number of entries that must be made in the Extension specification is the same as that required for all arrays.

If you want to load an execution-time array from an input file, you must make the following entries for the array input file in its Input specification:

- Column 43 (data format)—if the array contains numeric data, indicate the data format by specifying packed decimal format (P) or binary format (B), or by leaving the entry blank (overpunched decimal format).
- Columns 44 through 51 (field location)—specify the beginning and ending character positions of the entire array, partial array, or array element being loaded. If the data format is packed decimal or binary, the field location must represent the actual size of an array element in bytes.

The following example shows how to use the Input specification to load an entire execution-time array containing packed decimal numbers as a single field. Array ARR contains seven elements, and each element is four bytes long. The execution-time array is loaded from the input file ARRIN as a single field in packed decimal format.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
          .
          .
          .
E                        ARR         7  7 0
IARRIN    AA  03
I                                      P   1  280ARR
```

ZK-4435-85

You can load part of an execution-time array using one input field. The length of the field must be a multiple of the length of one entry. The array is loaded beginning with the first element and entries continue to be loaded until the end of the input field is reached.

In the following example, ARR contains 25 entries. Each entry is one character long. VAX RPG II loads the first 10 elements of the array ARR.

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      .
      .
      .
E                      ARR      25  1
IARRIN    AA  03
I                                          1  100ARR
```

ZK-4436-85

11.5.4 Defining Related Arrays in Alternating Format

You can define related arrays either individually or in alternating format. To define arrays in alternating format, you must make the following entries for the second (alternate) array in the same Extension specification you used to describe the first (main) array:

- Columns 46 through 51 (array name)—specify the name of the alternate array.
- Columns 52 through 54 (length of entry)—specify the length of an entry in the alternate array.
- Column 55 (data format)—you need only specify the data format for alternate preexecution-time arrays that contain numeric data. Specify packed decimal format (P) or binary format (B), or leave the entry blank (overpunched decimal format). When you specify packed decimal format, make sure the length of entry represents the length of the numeric data in unpacked form. When you specify binary format, the length of entry you specify indicates the number of bytes required to store the binary field. (Use 4 for two-byte signed binary numbers or 9 for four-byte signed binary numbers.)
- Column 56 (decimal positions)—for numeric data, specify the number of positions to the right of the decimal point. You must specify 0 for no decimal positions.
- Column 57 (sequence)—to indicate that the order of entries in an alternate array are in the specified sequence, specify either ascending (A) or descending (D) or leave this column blank to specify an unsequenced array.

The entries made in the following columns for the main array also apply to the alternate array:

- Columns 11 through 18 (from file name)
- Columns 33 through 35 (entries per record)
- Columns 36 through 39 (entries in array)

The following example describes the preexecution-time array A1 with 6 entries in each record and 24 entries in the array. The entries for array A1 are alternated with entries for array B1. A1 contains overpunched numeric data that is six digits long with no decimal places. B1 contains overpunched numeric data that is six digits long with two decimal places. Each record in the data file contains six entries for A1 and six entries for B1. The arrays are loaded from the file ARRFIL.

```
                                          ------F = Format (PB)
                                         | -----D = Decimal positions
                                         || ----S = Sequence (AD)
                                         |||
                                         |||Alternating table or array
      From     To       Table EntEnt Len|||name  Len
      file     file     or    perin  of F|||     of F
      name     name     array RecTbl Ent|D||     Ent|D
      |        |        name  |  |   |  ||S|     |  ||S
E     |        |        |     |  |   |  ||||     |  |||+-- Comments ---+
  0   |   1    |   2    |   3    |   4    |   5    |   6    |   7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
*....*        *        *      *--*---*--*****     *--****
E     ARRFIL           A1       6  24  6 0 B1       4 2
```

ZK-4437-85

A sample record from ARRFIL might look like this:

```
000001245000000216240000034520000004799000000577770000066550
\   /\  /\   /\  /    . . .
 ^   ^  ^   ^  ^
 |   |  |   |
A1,1 B1,1 A1,2 B1,2 . . .
```

11.6 Referencing Arrays

With tables, you can reference only the entry retrieved by the last LOKUP operation. With arrays, you can refer to either an entire array or an individual array element. One advantage of referencing an entire array is that a single operation can affect all the elements in the array.

You can specify an array name, a comma, and an index up to 10 characters long for factor 1 or factor 2 in a Calculation specification. You can specify an array element up to six characters long for the result field.

You can use an entire array as factor 1, factor 2, or the result field in the following operations:

- ADD
- Z-ADD
- SUB
- Z-SUB
- MULT
- DIV
- SQRT
- MOVE
- MOVEL
- MOVEA
- XFOOT
- LOKUP
- PARM

When you specify an array name in the following calculations, VAX RPG II repeats the operation for each element in the array:

- ADD
- Z-ADD
- SUB
- Z-SUB
- MULT
- DIV
- SQRT

- MOVE
- MOVEL

When using entire arrays (nonindexed) in any calculations, observe the following rules.

Rules

- When you specify arrays with the same number of elements for factor 1, factor 2, and the result field, VAX RPG II performs the operation on the first element, then on the second element, and so on, until all the elements in the array have been processed.

  If the arrays do not have the same number of elements, VAX RPG II ends the operation when the last element of the array with the fewest elements is processed.
- When one factor is a field or constant and the other factor or result field is an entire array, VAX RPG II performs the operation once for every element in the array.
- If the operation requires factor 2 only and the result field is an array, VAX RPG II performs the operation once for every element in the array.
- You must specify an array for the result field.
- You cannot use resulting indicators to condition calculations with arrays.

If you use an array for the result field and an element as one of the factors in a calculation, VAX RPG II alters the value of the element as a result of the calculation. When this occurs, VAX RPG II uses the new value in all subsequent operations that reference that element. For example, two numeric arrays contain the data shown in Table 11–1.

Table 11–1: Array Element Values

Array Element	Value
ARR1,1	4
ARR1,2	3
ARR1,3	1
ARR1,4	5
ARR2,1	2
ARR2,2	7
ARR2,3	5
ARR2,4	9

If every element of ARR1 is added to element ARR2,3 and the result is placed in ARR2, the elements of the resulting array are as shown in Table 11–2.

Table 11–2: Array Elements in Calculations

Array Element	Expression	Resulting Value
ARR2,1	(4 + 5)	9
ARR2,2	(3 + 5)	8
ARR2,3	(1 + 5)	6
ARR2,4	(5 + 5)	10

You can specify an array element in most operations that take a character or numeric field as factor 1, factor 2, or the result field. To specify an individual array element, enter the array name, a comma, and the index. For example, ARR,12 specifies the twelfth element of array ARR. You can also use a field name to represent the index. For example, if you specify ARR,FLD, the index value is determined by the value of the field FLD.

An array index, whether it is a literal or a field, must always be greater than or equal to 1 and less than or equal to the number of elements in the array. If it is not and you specify the /CHECK=(BOUNDS) qualifier to the RPG command, a run-time error will occur. If it is not and you do not specify the /CHECK=(BOUNDS) qualifier to the RPG command, unpredictable results will occur.

If you plan to use the same array element in a calculation for every program cycle, use a constant number as the index. If, however, you want to reference different array elements, use a field name as the index.

When array elements are scattered throughout an input record, each field must be described individually on the Input specification. A field description indicates the position of an element in the array. In such cases, there are two ways to load the data into the array:

- Assign a unique field name to each field of array data on the input record and then enter calculations to move individually each data field into the appropriate array element.
- Assign the array name with the proper index to each field of array data in the input record. The array is loaded automatically as the data is read.

The following example shows how to load individually each element of an execution-time array:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      .
      .
      .
E                    ARR         7  7 0
IARRIN   AA  03
I                                     P     1   40ARR,1
I                                     P     5   80ARR,2
I                                     P     9  120ARR,3
I                                     P    13  160ARR,4
I                                     P    17  200ARR,5
I                                     P    21  240ARR,6
I                                     P    25  280ARR,7
```

ZK-4438-85

In the following example, a company employs eight sales people whose weekly sales amounts are recorded in an input file. Each record of the file contains the weekly sales amounts; one new record is recorded in the file each week. At the end of the year, the company generates a report listing the sales totals for each week and the grand total for the entire year.

```
    0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890

FINPUT1  IPE F      60            DISK
FREPORT  O   F      60            DISK
E                    WEEK        8  6  2
E                    YEAR        8  8  2
IINPUT1  AA  01
I                                        1  482WEEK
C   01                XFOOTWEEK       TOTAL   82
C   01      WEEK      ADD  YEAR       YEAR
CLR                   XFOOTYEAR       GRAND  102
OREPORT  D        01
O                                     20 'WEEKLY TOTAL='
O                          TOTAL      35 '$   ,   .  '
O        T        LR
O                                     20 'YEARLY TOTAL='
O                          GRAND      35 '$  ,   ,   .  '
```

ZK-4439-85

Two execution-time arrays, WEEK and YEAR, are defined in the Extension specification. The Input specification instructs the program to load the array WEEK after reading each sales record from the input file INPUT1.

The input file for the execution-time array is not like a table input file with a corresponding File Description specification. Therefore, data is not automatically loaded into the array at the beginning of execution. Instead, you must describe on Input specifications the input data to be loaded into the array.

The array elements are in contiguous positions in the input record. Therefore, when the name of the array is specified as the field name, the data is automatically loaded into the appropriate elements of the array as the input record is read. In this case, only one Input specification is necessary to describe an input record of array data.

The XFOOT operation calculates the sum of all the elements in the array WEEK and puts the sum in the result field TOTAL. The next calculation adds one array to the other. Adding arrays involves adding each element of one array to the corresponding element of the other array. Normally, when you use an array name in a calculation, the operation is performed on each element of the array; then, an array of the results is created. Therefore, you cannot use resulting indicators to indicate the result of the operation.

These arrays have the same number of elements; therefore, any specified operation is performed until all elements have been processed. In the case of two arrays containing different numbers of elements, the specified operation is performed only until the last element in the shorter array is processed.

In the following example, the program produces results identical to those of the previous example. However, here the array elements are scattered throughout the input record.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789

FINPUT2  IPE F      60            DISK
FREPORT  O   F      60            DISK
E                    WEEK        8  6  2
E                    YEAR        8  8  2
IINPUT   AA  01
I                                        1   62WEEK,1
I                                        8  132WEEK,2
I                                       15  202WEEK,3
I                                       22  272WEEK,4
I                                       29  342WEEK,5
I                                       36  412WEEK,6
I                                       43  482WEEK,7
I                                       50  552WEEK,8
C   01                 XFOOTWEEK      TOTAL   82
C   01       WEEK      ADD  YEAR      YEAR
CLR                    XFOOTYEAR      GRAND  102
OREPORT  D        01
O                                     20 'WEEKLY TOTAL='
O                          TOTAL      35 '$   ,   .  '
O        T        LR
O                                     20 'YEARLY TOTAL='
O                          GRAND      35 '$  ,   ,   .  '
```

ZK-4440-85

11.7 Searching Arrays

The LOKUP operation code searches for an element in an array. To determine whether a particular element exists, you specify a search argument and define the conditions under which the LOKUP operation will succeed. You must also use a resulting indicator that specifies the condition and that will indicate the result of the LOKUP operation. The indicator is set on only if the search is successful; otherwise, the indicator

is set off. When searching for a HIGH or LOW condition, you must specify a sequence for the array in column 45 (sequence) of the Extension specification. Enter an indicator in these columns to test for the following conditions:

- Columns 58 and 59 (EQUAL)—equal
- Columns 54 and 55 (HIGH)—nearest to but greater than value
- Columns 56 and 57 (LOW)—nearest to but less than value
- Columns 54 and 55, and 58 and 59 (EQUAL or HIGH)—equal or nearest to but greater than value
- Columns 56 and 57, and 58 and 59 (EQUAL or LOW)—equal or nearest to but less than value

If you specify both EQUAL and HIGH or EQUAL and LOW, the EQUAL condition takes precedence if entries satisfy both conditions.

To search an array for an element, you must make the following entries in the Calculation specification:

- Columns 18 through 27 (factor 1)—specify a field, literal, array element, or table representing the element you want to locate. Make sure the search argument has the same length and data format as the elements in the array being searched.
- Columns 28 through 32 (operation code)—specify the LOKUP operation code.
- Columns 33 through 42 (factor 2)—specify the name of the array to be searched.
- Columns 54 through 59 (resulting indicator)—specify one or more indicators to test for a condition and to indicate whether the search has been successful. You can use these indicators to condition subsequent calculation and output operations.

In the following example, the program tries to match the search argument QTY with an entry in the array ARR. If a matching entry is found, indicator 11 is set on. If the entry is not found, indicator 11 is set off.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  |||indicators
     | |         1         |    2         field |   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
    ** *           *         *    *         *      *--*** * * *
    C   01         QTY       LOKUPARR                     11
```

ZK-4441-85

If you want to start searching an array at some point other than at the beginning, specify the array and its index where you begin the search. The index can be a literal or a field name. In the following example, the search begins with the seventh element of array ARR:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  |||indicators
     | |         1         |    2         field |   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
    ** *           *         *    *         *      *--*** * * *
    C   01         QTY       LOKUPARR,7                   11
```

ZK-4442-85

If you want to reference the element found in the last LOKUP operation, specify the array name and an index field in factor 2 of the LOKUP operation. If the search is successful, the index value of the array element that satisfied the condition is stored in the index field and the resulting indicator is set on. If the search is unsuccessful, the value 1 is placed in the index field and the resulting indicator is set off. If you do not specify the index field, a successful LOKUP operation indicates whether an element contains the data for which you are searching, but does not return the element's index value.

If you want to begin the search with the first element, you must initialize the index field to 1 before the LOKUP operation occurs.

You can also search for more than one array element by locating all the elements in an array that satisfy a certain condition. When the condition is satisfied, the program adds 1 to the value in the index field to continue the search with the next element.

In the following example:

- The program loads a preexecution-time array from the file INPUT1.
- The search argument SEARCH contains the value 50000; the LOKUP operation searches for any array element containing a value lower than the search argument.
- If the search is successful, indicator 56 is set on. This indicator causes the EXCPT operation to print the contents of each array element (and its index) that satisfies the search condition.
- After the program prints the array element, it sets indicator 56 off and adds 1 to the field containing the array index. While the index field remains below 11, the search continues by setting indicator 54 on; this causes the program to loop back to line 01090. This process continues until all 10 elements are searched.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890

01020FINPUT1  IT  F      50           EDISK
01030FINPUT2  IPE F      10            DISK
02040FOUTPUT  O   F      60            DISK
01050E    INPUT1          ARY1   10  10  5 0D
01060IINPUT2  AA  01
01070I                                        1   50SEARCH
01080C   01                Z-ADD1         I      20
01090C           LOOP      TAG
01100C   01      SEARCH    LOKUPARY1,I                 56
01105C   56                EXCPT
01107C                     SETOF                     56
01110C   01      1         ADD  I         I
01120C   01      11        COMP I                    54
01130C   01 54             GOTO LOOP
011400OUTPUT  E        56
011500                                   7 'INDEX='
011600                         I         9
011700                                   18 'VALUE='
011800                         ARY1,I    23
```

ZK-4443-85

An example of the output file might appear as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
 INDEX=06    VALUE=40000
 INDEX=07    VALUE=30000
 INDEX=08    VALUE=20000
 INDEX=09    VALUE=10000
 INDEX=10    VALUE=00000
```

The column numbers in this example are for reference and do not appear in the output.

11.8 Moving Array Data

You can use the MOVEA operation code to move the following array data:

- Contiguous array elements to a field
- A field or literal to contiguous array elements
- Contiguous elements of one array to contiguous elements of another array

If the array is not indexed, data movement starts with the first element of an array or field. If the array is indexed, the move starts with the element you specify. Data movement stops when either of the following conditions is met:

- The last array element is moved or filled.
- The number of characters moved equals the length of the shorter field, as specified either in columns 33 through 42 (factor 2) or in columns 43 through 48 (result field) of the Calculation specification.

See Chapter 16 for more information on the MOVEA operation code.

The following example shows a preexecution-time array ARR20 being loaded from the file ARRFILE. A copy of ARR20 is moved into the execution-time array ARR15 using the MOVEA operation code.

```
     0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890

     FARRFILE IT  F      80           EDISK
     E    ARRFILE         ARR20   5  50  4
     E                    ARR15      50  4
     C                     MOVEAARR20     ARR15
```

ZK-4444-85

11.9 Updating Arrays

To change the contents of an element in a compile-time array, or to add new elements to a compile-time array, edit the source program containing the array data, and then recompile the program.

To change the contents of an element in a preexecution-time array, or to add new elements to such an array, edit the table input file that contains the array.

You can make temporary changes in arrays during program execution by using the array name as a result field. You can make these temporary changes permanent by writing the array to an output file that you can use later as an input file.

The following example describes the array COSTL, which consists of six-digit overpunched numeric data with two decimal places. This array is read from the file ARRAYIN. During program execution, changes can be made to this array. At the completion of the program, the array will be written to the output file ARRAYOUT. The format in which it is written is the same as that in which it was read, that is, eight entries in each record with each entry being a six-digit overpunched numeric data type with two decimal positions. The files ARRAYIN and ARRAYOUT must also be described on File Description specifications as an input table file (ARRAYIN) and an output table file (ARRAYOUT).

```
                                          ------F = Format (PB)
                                         | -----D = Decimal positions
                                         || ----S = Sequence (AD)
                                         |||
                                         |||Alternating table or array
          From     To       Table EntEnt Len|||name  Len
          file     file     or    perin  of F|||     of F
          name     name     array RecTbl Ent|D||     Ent|D
          |        |        name  |  |   |  ||S|     |  ||S
     E    |        |        |     |  |   |  ||||     |  |||+-- Comments ---+
   0    |   1    |   2    |   3    |   4    |   5    |   6    |   7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     *....*        *        *      *--*---*--****     *--****
     E    ARRAYIN  ARRAYOUTCOSTL    8 100  6 2
```

ZK-4445-85

11.10 Outputting Arrays

You can output either an entire array or individual array elements. To output entire arrays, you can make entries either in an Extension specification or in an Output specification.

To write a compile-time or preexecution-time array using an Extension specification, you must make the following entry:

- Columns 19 through 26 (to file name)—specify the name of a sequential output file. This file must have been previously defined in a File Description specification. The program automatically writes the compile-time or preexecution-time array you specified in the Extension specification to this output file after reaching the end of the program.

To write a compile-time, preexecution-time, or execution-time array using an Output specification, you must make the following entries:

- Columns 32 through 37 (field name)—specify the name of the array you want to write. The array is written every time the program processes a record unless you specify indicators in columns 23 through 31 of the Output specification.
- Columns 40 through 43 (end position)—specify the character position where the last entry of the array ends.

In the following example, for each record read from FILEA, the execution-time array DISCNT is written out to the file FILEB using Output specifications:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      .
      .
      .
E    COSTLIST        PRICE   5  10  5 2
E                    DISCNT     10  5 2
IFILEA   AA  01
I                                          1   22PERCNT
C   01       PRICE      MULT PERCNT    DISCNT
OFILEB   D  1     01
O                                      60 'COST WITH DISCOUNT OF '
O                           PERCNT3    72
O                                      74 '%'
O        D  1     01
O                           DISCNT    120 '  $0.  '
```

ZK-4446-85

To output an individual array element, specify the array and the index of the desired element (in the form ARR,n, where n is either a constant or a field name) in columns 32 through 37 (field name).

The following example outputs only the first and second elements of array DSCT:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      .
      .
      .
E    COSTLIST        PRICE   5  10  5 2
E                    DSCT       10  5 2
IFILEA   AA  01
I                                          1   22PERCNT
C   01       PRICE      MULT PERCNT    DSCT
OFILEB   D  1     01
O                                      20 'ITEM 1 COST: '
O                           DSCT,1     32 '  $0.  '
O                                      50 'ITEM 2 COST: '
O                           DSCT,2     62 '  $0.  '
```

ZK-4447-85

If you want to output numeric array elements, you can use edit codes or edit words to add commas or dollar signs, or to suppress leading zeros. Do not use edit codes or edit words to modify array data if you are going to use the data as input to subsequent programs.

When you specify an edit code with an entire array (nonindexed), VAX RPG II automatically inserts two spaces between elements of the array in the output record.

Chapter 12

Calling System Routines from VAX RPG II

This chapter describes the use of VAX RPG II operation codes to access VAX/VMS Run-Time Library (RTL) routines, system services, utilities (such as the VAX Forms Management System (VAX FMS) and the VAX Terminal Data Management System (TDMS)), and subprograms written in languages other than VAX RPG II[1]. You can access these routines by using the following VAX RPG II operation codes:

- CALL operation code—invokes the routine
- PLIST operation code—defines the parameter list, if used
- PARM, PARMD, and PARMV operation codes—determine the parameter-passing mechanism.
- GIVNG operation code—receives a function value or return status
- EXTRN operation code—defines a VAX RPG II name for an external symbol name

See Chapter 16 for more information on these operation codes.

Although calling VAX/VMS Run-Time Library routines, system services, utilities, and subprograms can provide many advantages, you should note the following:

- Do not call these routines if you can perform the task using VAX RPG II.
- Do not mix VAX/VMS Run-Time Library and VAX RPG II output routines.

[1] There are no VAX RPG II subprograms. VAX RPG II modules cannot be called from VAX RPG II or any other language.

- If a VAX/VMS Run-Time Library routine and a system service perform the same task, use the VAX/VMS Run-Time Library routine.

System routines are subroutines and functions provided by the VAX/VMS operating system. Each system routine has an entry point (the routine or service name) and an argument list. Each system routine may also return a function value or condition value to the program that calls it.

System routines perform common tasks, such as finding the square root of a number or allocating virtual memory. If you use system routines, you will not have to rewrite code every time you want to perform a common task. Using system routines allows you to concentrate on application-specific tasks, not utility tasks. Some system routines even help independent parts of programs allocate resources cooperatively.

A system routine can be called from any VAX/VMS language if that language supports the data structures required by the particular routine. The results of a system routine will be the same, no matter what language you use.

The system routines that are most commonly called from user programs are VAX/VMS Run-Time Library routines and system services. These system routines are documented in the *VAX/VMS Run-Time Library Routines Reference Manual* and the *VAX/VMS System Services Reference Manual.*

12.1 Run-Time Library Routines

The VAX/VMS Run-Time Library routines are assigned facility names that represent specific types of common tasks. These facilities and the types of tasks they perform are shown in Table 12–1.

Table 12–1: VAX/VMS Run-Time Library Facilities

Facility	Tasks Performed
LIB$	General purpose procedures that obtain records from devices, manipulate strings, convert data types for I/O, allocate resources, obtain the system date or time, signal exceptions, establish condition handlers, enable detection of hardware exceptions, and process cross-reference data.
MTH$	Mathematics procedures that perform arithmetic, algebraic, and trigonometric calculations.
OTS$	Language-independent support procedures that perform tasks such as data type conversions as part of a compiler's generated code.
SMG$	Screen management procedures that assist you in designing, composing, and keeping track of complex images on a video screen and that provide terminal-independent tasks.
STR$	String manipulation procedures that perform tasks such as searching for substrings, concatenating strings, and prefixing and appending strings.

12.2 System Services Routines

The VAX/VMS system services are routines that perform various tasks such as controlling processes, communicating among processes, and coordinating I/O.

Unlike VAX/VMS Run-Time Library routines, which are grouped by facility name, all system services share the same facility prefix (SYS$). However, these services are logically divided into groups of services that perform similar tasks. Table 12–2 describes these groups.

Table 12–2: Groups of VAX/VMS System Services

Group	Tasks Performed
AST	Allows processes to control the handling of ASTs
Change Mode	Changes the access mode of particular routines
Condition Handling	Designates condition handlers for special purposes

Table 12–2 (Cont.): Groups of VAX/VMS System Services

Group	Tasks Performed
Event Flag	Clears, sets, reads, and waits for event flags, and associates with event flag clusters
Information	Returns information about the system, queues, jobs, processes, locks, and devices
Input/Output	Performs I/O directly, without using VAX RMS
Lock Management	Enables processes to coordinate access to shareable system resources
Logical Names	Provides methods of accessing and maintaining pairs of character string logical names and equivalence names
Memory Management	Increases or decreases available virtual memory, controls paging and swapping, and creates and accesses shareable files of code or data
Process Control	Creates, deletes, and controls execution of processes
Security	Enhances the security of VAX/VMS systems
Timer and Time Conversion	Schedules events; obtains and formats binary time values

12.3 Procedure for Calling System Routines

Seven steps are required to call any system routine:

1. Declare the system routine
2. Determine the type of call (function or procedure)
3. Declare the arguments
4. Include symbol definitions (if applicable)
5. Call the routine or service
6. Check the condition value (if applicable)
7. Locate the result

In the following sections, you can follow these steps in writing a program to call the VAX/VMS Run-time Library routine LIB$STAT_TIMER. This system routine returns one of five statistics: (1) elapsed time, (2) CPU time, (3) buffered I/O count, (4) direct I/O count, or (5) page fault count.

12.3.1 Declare the System Routine

Declare a system routine in your program as you declare any other external routine. The declaration statement will vary depending on whether the system routine is being called as a function or procedure.

The routine declaration statement for calling LIB$STAT_TIMER as a function should appear as follows:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |       Factor      |    Factor    Result|   |  ||indicators
     | |       1           |    2         field |   |  ||+ - 0
    C| NxxNxxNxx|           |    |         |     |   |  ||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *          *    *         *     *--*** * * *
     C             STATIM     EXTRN'LIB$STAT_TIMER'
     C                        CALL STATIM
     C                        PARM           CODE    90 RL
     C                        PARM           VALUE   90 WL
     C                        GIVNG          RETVAL
```

ZK-4638-85

The routine declaration statement for calling LIB$STAT_TIMER as a procedure should appear as follows:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  |||indicators
     | |         1         |    2         field |   |  |||+ - 0
     C| NxxNxxNxx|         |    |         |     |   |  |||> < = +- Comments --+
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
     1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *          *         *    *         *     *--*** * * *
     C             STATIM    EXTRN'LIB$STAT_TIMER'
     C                       CALL STATIM
     C                       PARM           CODE    90 RL
     C                       PARM           VALUE   90 WL
```

ZK-4639-85

12.3.2 Determine the Type of Call (Function or Procedure)

Before you can call a system routine or service, you must determine whether the call should be a function call or a procedure call by referring to the Returns section of the system routine documentation.

A system routine must be called as a function if it returns a condition value or a function value. For example, the Returns section of the LIB$STAT_TIMER system routine documentation contains the following description:

```
RETURNS
        VMS Usage: cond_value
        type:      longword (unsigned)
        access:    write only
        mechanism: by value
```

Because the LIB$STAT_TIMER system routine returns a condition value, you must call it as a function.

NOTE

To call a system routine as a function in VAX RPG II, you must use the GIVNG operation code.

You can call many system routines as procedures, if you choose not to refer to the condition code. This is highly discouraged because it can lead to many undiscovered errors. (Checking condition values is described in Section 12.3.6.) DIGITAL recommends that you call a system routine as a procedure only if it does not return a condition value or a function value. In this case, the Returns section of the system routine documentation contains the following description:

```
RETURNS
        None
```

12.3.3 Declare the Arguments

Most system routines have one or more arguments that you can use to pass information to the system routine and to obtain information from the system routine. Arguments can be required or optional.

For example, consider the arguments for the VAX/VMS Run-Time Library routine LIB$STAT_TIMER. This routine has three arguments; two are required and one is optional. You can determine which arguments are required by looking at the Format section of the system routine documentation. In the case of LIB$STAT_TIMER, the format is as follows:

```
LIB$STAT_TIMER code ,value [,handle-adr]
```

The **handle-adr** argument appears in brackets ([]), indicating that it is an optional argument. Optional arguments to a system routine appear in brackets in that routine's Format section. For this example, you want to declare only the two required arguments, code and value.

To declare an argument for a system routine, first check that argument's description in the system routine documentation. The argument description for the code argument is as follows:

```
code

VMS Usage:  function_code
type:       longword integer (signed)
access:     read only
mechanism:  by reference
```

The code argument contains the address of a signed longword, and that is the statistic returned by LIB$STAT_TIMER. The signed longword must be an integer from one to five.

The VMS Usage function_code indicates that the data type returned by the routine is dependent on other factors. Table 12–3 lists the VAX RPG II equivalent for each of the VAX/VMS Usages. You can declare the argument using the code provided in Table 12–3.

The following example shows that the parameter contained in the field CODE of the code argument is passed by reference:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |        Factor     |    Factor    Result|   |  ||indicators
     | |        1          |    2         field |   |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  ||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *      *--*** * * *
     C                       PARM           CODE     90 RL
```

ZK-4630-85

For more information on parameter-passing mechanisms, see Section 12.3.3.1.

The procedure used in declaring the code argument is also used in declaring the value argument. First, check the argument description for the value argument:

```
value

VMS Usage: varying_arg
type:      unspecified
access:    write only
mechanism: by reference
```

The value argument contains the address of a longword or quadword, and that is the statistic returned by LIB$STAT_TIMER. All statistics are longword integers except elapsed time, which is a quadword.

The VMS Usage varying_arg indicates that the data type returned by the routine is dependent on other factors. In this case, the data type returned is dependent on the statistic you want to return. For this example, the statistic that you want to return is code 5, page fault count. This statistic is returned in a signed longword integer. Therefore, you need to check Table 12–3 to find the VAX RPG II statements that are used to declare a longword_signed data structure.

The declaration statements for all VAX/VMS routines and system services arguments can be found by looking up the VAX/VMS Usage in Table 12–3.

Table 12–3: VAX/VMS Data Structures

VAX/VMS Data Structure	VAX RPG II Implementation
access_bit_names	NA
access_mode	Declare as text string of one byte. When using this data structure, you must interpret the ASCII contents of the string to determine the access_mode.
address	L[1]
address_range	Q[1]
arg_list	NA
ast_procedure	L[1]
boolean	NA
byte_signed	Declare as text string of one byte. When using this data structure, you must interpret the ASCII contents of the string.
byte_unsigned	Same as for byte_signed.[1]
channel	W[1]
char_string	TEXT STRING
complex_number	DATA STRUCTURE
cond_value	condvalue GIVNG OPCODE Columns 43 through 58
context	L[1]
date_time	Q[1]
device_name	TEXT STRING
ef_cluster_name	TEXT STRING
ef_number	L[1]
exit_handler_block	DATA STRUCTURE

[1] VAX RPG II does not typically support unsigned data structures. However, unsigned information may be passed using the signed equivalent, if the contents do not exceed the range of the signed data type.

Table 12–3 (Cont.): VAX/VMS Data Structures

VAX/VMS Data Structure	VAX RPG II Implementation
fab	Generated implicitly by the compiler on your behalf. It is not possible for a user to access the fab data structure from a VAX RPG II program.
file_protection	W[1]
floating_point	F or D Column 55
function_code	F
io_status_block	Q
item_list_2	DATA STRUCTURE
item_list_3	DATA STRUCTURE
item_quota_list	NA
lock_id	L[1]
lock_status_block	DATA STRUCTURE
lock_value_block	DATA STRUCTURE
logical_name	TEXT STRING
longword_signed	L
longword_unsigned	L[1]
mask_byte	NA
mask_longword	L[1]
mask_quadword	Q[1]
mask_word	W[1]
null_arg	NA
octaword_signed	DATA STRUCTURE
octaword_unsigned	DATA STRUCTURE
page_protection	L[1]
procedure	L[1]
process_id	L[1]

[1] VAX RPG II does not typically support unsigned data structures. However, unsigned information may be passed using the signed equivalent, if the contents do not exceed the range of the signed data type.

Table 12–3 (Cont.): VAX/VMS Data Structures

VAX/VMS Data Structure	VAX RPG II Implementation
process_name	TEXT STRING
quadword_signed	Q
quadword_unsigned	Q[1]
rights_holder	Q[1]
rights_id	L[1]
rab	NA
section_id	Q[1]
section_name	TEXT STRING
system_access_id	Q[1]
time_name	TEXT STRING
uic	L[1]
user_arg	L[1]
varying_arg	Dependent upon application.
vector_byte_signed	ARRAY OF CHARACTER STRING
vector_byte_unsigned	ARRAY OF CHARACTER STRING[1]
vector_longword_signed	ARRAY OF LONGWORD INTEGER (SIGNED) L
vector_longword_unsigned	ARRAY OF LONGWORD INTEGER L[1]
vector_quadword_signed	NA
vector_quadword_unsigned	NA
vector_word_signed	ARRAY OF WORD INTEGER (SIGNED) W
vector_word_unsigned	ARRAY OF WORD INTEGER W[1]
word_signed	W
word_unsigned	W[1]

[1]VAX RPG II does not typically support unsigned data structures. However, unsigned information may be passed using the signed equivalent, if the contents do not exceed the range of the signed data type.

12.3.3.1 Parameter-Passing Mechanisms

This section describes conventions for passing arguments in VAX RPG II programs.

A calling program can pass a parameter in one of three ways:

- By value—the PARMV operation code passes a parameter by value.
- By reference—the PARM operation code passes a parameter by reference.
- By descriptor—the PARMD operation code passes a parameter by descriptor.

When the PARMV operation code passes a parameter by value, the parameter list contains the actual, uninterpreted 32-bit value of the parameter.

In the following example, the constant 0 is passed by value:

```
                                                    Field length
     Control level                                  | Decimal positions
     |                                              | |Half adjust (H)
     | Indicators          Operation                | | |
     | |                   |                        | | |Resulting
     | |        Factor     |    Factor    Result    | | |indicators
     | |        1          |    2         field     | | |+ - 0
    C| NxxNxxNxx|          |    |         |         | | |> < = +- Comments --+
  0  |   1  |   2  |   3  |   4  |   5  |   6  |   7  |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *       *          *   *          *     *--*** * * *
     C                      PARMV          0
```

ZK-4632-85

When the PARM operation code passes a parameter by reference, the parameter list contains the address of the location that contains the value of the parameter. Most languages pass scalar data by reference.

Character data is always passed as a fixed-length string. Numeric data, by default, is passed as a packed decimal string. When passing a parameter by reference, you may specify an access type and a data type for numeric data in columns 54 through 57 of the Calculation specification. Sections 12.3.3.2 and 12.3.3.3 describe access and data types in detail.

In the following example, the parameter contained in the field TIMLEN is passed by reference.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |        Factor     |    Factor    Result|   |  ||indicators
     | |        1          |    2         field |   |  ||+ - 0
     C| NxxNxxNxx|         |    |         |     |   |  ||> < = +- Comments --+
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *      *--*** * * *
     C                       PARM           TIMLEN  90 WL
```

ZK-4633-85

When the PARMD operation code passes a parameter by descriptor, the parameter list entry contains the address of a descriptor for the parameter.

In the following example, the field TIMBUF containing the parameter (fixed-length string) is passed by descriptor:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |        Factor     |    Factor    Result|   |  ||indicators
     | |        1          |    2         field |   |  ||+ - 0
     C| NxxNxxNxx|         |    |         |     |   |  ||> < = +- Comments --+
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *      *--*** * * *
     C                       PARMD          TIMBUF 23
```

ZK-4634-85

In passing the arguments to the procedure, you must declare the passing mechanism. When passing parameters by descriptor (using the PARMD operation code), VAX RPG II uses:

- An array descriptor for entire arrays
- A scalar decimal descriptor for numeric data with positions to the right of the decimal point
- A scalar descriptor for all other data types

VAX RPG II passes parameters in a scalar format, unless the parameter is an entire array.

The passing mechanism required for a system routine argument is indicated in the argument description. This is shown in the following description of the one-char-str argument to LIB$CHAR:

```
one-char-str

VMS Usage: char_string
type:      character string
access:    write only
mechanism: by descriptor
```

In this case, the required passing mechanism is by descriptor. The passing mechanisms allowed in system routines are those listed in the VAX Procedure Calling and Condition Handling Standard section of the *Introduction to VAX/VMS System Routines.*

NOTE

Any passing mechanisms not described in this section are unsupported in VAX RPG II. If a system routine requires a passing mechanism not described in this section, it is not possible to call that routine directly from VAX RPG II.

You are required to specify the passing mechanism, as shown in the following example, where the PARM operation codes indicate that both CODE and VALUE are being passed by reference:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | ||                  |                        |  ||Resulting
     | ||        Factor    |    Factor    Result|   |  ||indicators
     | ||        1         |    2         field |   |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  ||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *           *         *    *         *     *--*** * * *
     C              STATIM    EXTRN'LIB$STAT_TIMER'
     C                        CALL STATIM
     C                        PARM           CODE    90 RL
     C                        PARM           VALUE   90 WL
     C                        GIVNG          RETSTA
```

ZK-4643-85

12.3.3.2 Parameter Access Types (Column 54)

The parameter access type indicates the actions that the VAX/VMS Run-Time Library routine is permitted to perform on the parameter. Access types that you can use in VAX RPG II are as follows:

- Read-only (R)—the parameter can only be read.
- Write-only (W)—the parameter can only be written.
- Modify (M)—the parameter can be modified (read and written).

You can specify the parameter access type and data type with the PARM operation code. If you specify a parameter access type, you must also specify its data type.

In the following example, the TIMLEN field is a longword integer (column 55) with write-only access (column 54):

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |        Factor     |    Factor    Result|  |||indicators
     | |        1          |    2         field |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |  |||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *     *--*** * * *
     C                       PARM           TIMLEN  90 WL
```

ZK-4635-85

12.3.3.3 Parameter Data Types (Columns 55 through 57)

If you specify a parameter access type, you must also specify its data type. The following parameter data types can be passed from a VAX RPG II program to a VAX/VMS Run-Time Library routine:

- Word integer (W)—signed
- Longword integer (L)—signed
- Quadword integer (Q)—signed
- F_floating single-precision (F)
- D_floating double precision (D)

- Numeric string, right overpunched sign (NRO)
- Packed decimal string—default data type for numeric data
- Character string—default data type for character data

Define the parameter data type in columns 55 through 57 of the Calculation specification. You can specify a data type only for numeric fields passed by reference.

In the following example, the data type of the numeric field TIMLEN is a right overpunched sign (NRO):

```
                                                    Field length
  Control level                                     | Decimal positions
  |                                                 | |Half adjust (H)
  | Indicators          Operation                   | ||
  | |                   |                           | ||Resulting
  | |        Factor     |    Factor     Result|     | ||indicators
  | |        1          |    2          field |     | ||+ - 0
  C| NxxNxxNxx|         |    |          |     |     | ||> < = +- Comments --+
   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890
  ** *        *         *   *         *     *--*** * * *
  C                     PARM            TIMLEN     MNRO
```

ZK-4637-85

12.3.4 Include Symbol Definitions

Many system routines require values that are defined in separate symbol definition files. For example, when you call any VAX/VMS Run-Time Library routine in the VAX Screen Management (VAX SMG$) facility, you must include the file SMGDEF.

VAX/VMS Run-Time Library routines require you to include symbol definitions when you call a VAX Screen Management (VAX SMG$) routine, or a routine that is a jacket to a system service. (A jacket routine provides a simpler, more easily used interface to a system service.)

All system services routines require you to include the file System Services Definition File (SSDEF) to check status. Many system services require other symbol definitions as well. To determine whether you need to include other symbol definitions for the system service you wish to call, refer to the documentation for that service. For example, if the

documentation states that values are defined in the *foobar* macro, you must include those symbol definitions in your program.

In VAX RPG II, a definition macro is included as follows:

```
$ CREATE SMGDEF.MAR
        .TITLE SMGDEF - Define SMG$ constants
        $SMGDEF GLOBAL
        .END
$ MACRO SMGDEF
$ LINK RPGPROG,SMGDEF
```

The LIB$STAT_TIMER system routine does not use any included definition files, so this step is not applicable for this example.

12.3.5 Call the Routine or Service

The call to a VAX/VMS Run-Time Library routine or system service is set up as an external call in VAX RPG II. The syntax of the CALL statement will depend on whether the call is a function call or a procedure call.

12.3.5.1 Calling a System Routine as a Function Call

Call a system routine as a function call according to the Format section in the routine or service description. For example, the format for LIB$STAT_TIMER is as follows:

```
LIB$STAT_TIMER code ,value [,handle-adr]
```

In a format statement, an optional argument can appear in one of two ways:

- [,optional-argument]
- ,[optional-argument]

In general, VAX/VMS Run-Time Library routines use the format [,optional-argument]. If the comma appears *inside* the brackets ([,optional-argument]), you can omit the optional argument if it is the last argument in the list. For example, look at the optional arguments of an imaginary routine, LIB$EXAMPLE_ROUTINE:

```
LIB$EXAMPLE_ROUTINE arg1 [,arg2] [,arg3] [,arg4]
```

You can omit all of the optional arguments without using a placeholder, as shown in the following example:

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |         Factor    |    Factor    Result|  |||indicators
     | |         1         |    2         field |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |  ||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *         *    *         *     *--*** * * *
     C           LIBEXA     EXTRN'LIB$EXAMPLE_ROUTINE'
     C                      CALL LIBEXA
     C                      PARM           ARG1
     C                      GIVNG          RETSTA
```

ZK-4640-85

However, if you omit an optional argument in the middle of the argument list, you must insert a placeholder, as shown in the following example:

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |         Factor    |    Factor    Result|  |||indicators
     | |         1         |    2         field |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |  ||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *         *    *         *     *--*** * * *
     C           LIBEXA     EXTRN'LIB$EXAMPLE_ROUTINE'
     C                      CALL LIBEXA
     C                      PARM           ARG1
     C                      PARMV          0
     C                      PARM           ARG3
     C                      GIVNG          RETSTA
```

ZK-4641-85

Calling the system routine LIB$EXAMPLE_ROUTINE as a function call using all optional arguments would appear as in the following example:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |        Factor     |    Factor    Result|   |  |||indicators
     | |        1          |    2         field |   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  |||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *          *    *         *     *--*** * * *
     C             LIBEXA     EXTRN'LIB$EXAMPLE_ROUTINE'
     C                        CALL LIBEXA
     C                        PARM           ARG1
     C                        PARM           ARG2
     C                        PARM           ARG3
     C                        GIVNG          RETSTA
```

ZK-4642-85

In general, VAX/VMS system services use the format ,[optional-argument]. If the comma appears *outside* the brackets (,[optional-argument]), you must pass a zero by value. In the following example, the constant 0 is passed by value:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |        Factor     |    Factor    Result|   |  |||indicators
     | |        1          |    2         field |   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  |||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *          *    *         *     *--*** * * *
     C                        PARMV          0
```

ZK-4636-85

12.3.5.2 Calling a System Routine as a Procedure Call

If the routine or service you are calling does not return a function value or condition value, you may call the system routine as a procedure. The same rules apply to optional arguments, and you should still follow the calling sequence presented in the Format section of the routine or service description.

One system routine that does not return a condition value or function value is the VAX/VMS Run-Time Library routine LIB$SIGNAL. LIB$SIGNAL should always be called as a procedure, as shown in the following example:

```
                                                    Field length
      Control level                                 |  Decimal positions
      |                                             |  |Half adjust (H)
      | Indicators          Operation               |  ||
      | |                   |                       |  ||Resulting
      | |        Factor     |    Factor    Result   |  |||indicators
      | |        1          |    2         field    |  |||+ - 0
     C| NxxNxxNxx|          |    |         |        |  |||> < = +- Comments --+
    0   |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *      *--*** * * *
     C           SIGNAL      EXTRN'LIB$SIGNAL'
     C                       CALL SIGNAL
     C                       PARMV           CODE    90
```

ZK-4644-85

12.3.6 Check the Condition Value

After you call the system routine and control is returned to your program, you should check the condition value returned (if there is one). In general, all system routines return a condition value with the following exceptions:

- The system routine returns a function value.
- The system routine has no condition values.
- The system routine has no condition values returned, but rather, has condition values signaled. (Success conditions are not signaled.)
- The call to the routine was made as a procedure call.

These exceptions are described in the Returns, Condition Values Returned, or Condition Values Signaled section of the system routine documentation.

If there is a condition value, you must check this value to make sure that it indicates success. All success condition values are described in the Condition Values Returned section of the system routine documentation. Success condition values always appear first in this list.

Many system routines return the condition value SS$_NORMAL as a success value. If this is the only possible success condition, you can test for its presence, as shown in the following example:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |        Factor     |    Factor    Result|   |  ||indicators
     | |        1          |    2         field |   |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  ||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *          *    *         *     *--*** * * *
     C             SSNORM     EXTRN'SS$_NORMAL'
     C             STATIM     EXTRN'LIB$STAT_TIMER'
     C                        CALL STATIM
     C                        PARM           CODE    90 RL
     C                        PARM           VALUE   90 WL
     C                        GIVNG          RETSTA
     C             SSNORM     COMP RETSTA                    01
     C             STOP       EXTRN'LIB$STOP'
     C  N01                   CALL STOP
     C                        PARMV          RETSTA  90 RL
```

ZK-4645-85

It is also possible to check for any success code because all success codes have an odd value (not evenly divisible by 2). The following example program segment will continue execution if any success code is returned. The call to STATIM is conditioned by an indicator which will be set off if any success code is returned.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  |||indicators
     | |         1         |    2         field |   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  |||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *      *--*** * * *
     C            STATIM     EXTRN'LIB$STAT_TIMER'
     C                       CALL STATIM                    01
     C                       PARM            CODE    90 RL
     C                       PARM            VALUE   90 WL
     C                       GIVNG           RETSTA
     C            STOP       EXTRN'LIB$STOP'
     C   01                  CALL STOP
     C                       PARMV           RETSTA  90 RL
```

ZK-4646-85

When several success condition values are possible, you can continue execution on specific success codes. For example, the system service $SETEF returns one of two success values, SS$_WASSET or SS$_WASCLR. If you want to continue when the success code SS$_WASSET is returned, you can check for this condition value as follows:

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  ||indicators
     | |         1         |    2         field |   |  ||+ - 0
    C| NxxNxxNxx|          |    |         |      |   |  ||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *         *    *         *      *--*** * * *
     C           SETEF     EXTRN'SYS$SETEF'
     C                     CALL SETEF
     C                     PARM           EFN         RL
     C                     GIVNG          RETSTA
     C           WASSET    EXTRN'SS$_WASSET'
     C           WASSET    COMP RETSTA                    01
```

ZK-4647-85

If indicator 01 is on, then SS$_WASSET was returned by the call.

If the condition value returned is not a success condition, then the routine did not complete normally and the information it was to return may be missing, incomplete, or incorrect.

If the condition value returned was not a success code, you can check for a particular error condition, as shown in the following example:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *          *          *    *          *     *--*** * * *
     C                                MOVE 'Input: ' PRMSTR  7
     C                RMSEOF      EXTRN'RMS$_EOF'
     C                GETINP      EXTRN'LIB$GET_INPUT'
     C                            CALL GETINP                    02
     C                            PARMD           INPSTR255
     C                            PARMD           PRMSTR
     C                            PARM            INPLEN      WW
     C                            GIVNG           RETVAL
     C   02           'Error'     DSPLYTTY
     C   02           RMSEOF      COMP RETVAL                 03
     C   03           ' EOF'      DSPLYTTY
```

ZK-4648-85

12.3.7 Locate the Result

After you have declared the arguments, called the routine, and checked the condition value, you are ready to use the result. To find out where the result is returned, look at the description of the system routine you are calling.

12.3.7.1 Function Results

If the system routine is called as a function, the result is written into the variable in factor 2 of the GIVNG operation code.

For example, in the call to MTH$ACOS in the following example, the result is written into the variable RESULT:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *          *          *    *          *     *--*** * * *
     C                ACOS        EXTRN'MTH$ACOS'
     C                            CALL ACOS
     C                            PARM            COS         RF
     C                            GIVNG           RESULT
```

ZK-4649-85

This result is described in the Returns section of the system routine description.

12.3.7.2 Procedure Results

If the system routine is called as a procedure, the result is written into one or more of the arguments. To determine which argument holds the result, examine the access entry in each of the argument descriptions. If the access entry in an argument description is "write-only" or "modify", that argument contains output information written by the procedure.

For example, LIB$CURRENCY returns the default system currency symbol ($). The following argument description shows that the currency string is returned in the currency_str argument:

```
currency_str

VMS Usage: char_string
type:      character string
access:    write only
mechanism: by descriptor
```

In all system routines, the output information returned by the routine or service has an access of "write-only" or "modify".

12.4 Examples of Calling VAX/VMS Run-Time Library Routines

The following examples demonstrate calls to VAX/VMS Run-Time Library routines from VAX RPG II programs.

You cannot call all VAX/VMS Run-Time Library routines because VAX RPG II cannot supply some types of parameters, such as addresses. See the *VAX/VMS Run-Time Library Routines Reference Manual* for information on all VAX/VMS Run-Time Library routines and the parameters they require.

The following example shows a call to the STR$UPCASE routine to change the lowercase string to uppercase letters. This routine requires two parameters: (1) the source string, and (2) the destination string. Both the source string HEAD and the destination string RESULT must be passed by descriptor, so the PARMD operation code is used.

Because the name of this VAX/VMS Run-Time Library routine is longer than eight characters, the EXTRN operation code is used to refer to STR$UPCASE as UPCASE.

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |         Factor    |    Factor    Result| |  ||indicators
     | |         1         |    2         field | |  ||+ - 0
     C| NxxNxxNxx|         |    |         |     |  ||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *         *    *         *     *--*** * * *
     C                                   MOVE 'rep head'HEAD    8
     C             UPCASE                EXTRN'STR$UPCASE'
     C                                   CALL UPCASE
     C                                   PARMD          RESULT  8
     C                                   PARMD          HEAD
```

ZK-4650-85

The following example calls the LIB$SET_SYMBOL routine to redefine the Command Language Interpreter (CLI) symbol MY_PARAMETER to be the string OFF. This routine requires two parameters to be passed by descriptor: (1) the symbol to be defined, and (2) the value to be given

to the symbol. Line 220 moves the string OFF (the value to be given to the symbol) to the field SETVAL. Line 230 assigns a five-character name, STSYM, to this routine name. Lines 240 and 250 assign the 12-character string MY_PARAMETER to the field SYMBL, the symbol to be defined. Line 260 invokes this routine. Lines 270 and 280 pass the two parameters to the routine.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  |||indicators
     | |         1         |    2         field |   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  |||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
    ** *          *         *    *         *     *--*** * * *
 220C                       MOVE 'OFF'     SETVAL  3
 230C               STSYM   EXTRN'LIB$SET_SYMBOL'
 240C                       MOVE 'METER'   SYMBL  12
 250C                       MOVEL'MY_PARA' SYMBL
 260C                       CALL STSYM
 270C                       PARMD          SYMBL
 280C                       PARMD          SETVAL
```

ZK-4651-85

The following example calls the LIB$GET_INPUT routine to ask the user for input from the terminal screen. This procedure requires three parameters: (1) the input text INPSTR (passed by descriptor) from the screen, (2) the prompt string PRMSTR (passed by descriptor) that is displayed before accepting input, and (3) the number of characters INPLEN (passed by reference) that are written to the input text. Also, this example designates the field RETVAL to accept the return status (RMS$_EOF is the VAX/VMS Run-Time Library symbolic constant representing one possible return status) of the operation. The program uses the EXTRN operation code to retrieve the value of the symbolic constant representing a return

status. If the operation is unsuccessful, indicator 02 is set on and the string Error is displayed on the screen. If the operation is unsuccessful because the file is at the end-of-file, the string EOF is also displayed.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     |  Indicators            Operation             |  ||
     |  |                     |                     |  ||Resulting
     |  |          Factor     |     Factor    Result|  ||indicators
     |  |          1          |     2         field |  ||+ - 0
     C| NxxNxxNxx|            |     |         |     |  ||> < = +- Comments --+
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *           *          *    *          *     *--*** * * *
     C                         MOVE 'Input: ' PRMSTR  7
     C                RMSEOF   EXTRN'RMS$_EOF'
     C                GETINP   EXTRN'LIB$GET_INPUT'
     C                         CALL GETINP                 02
     C                         PARMD          INPSTR255
     C                         PARMD          PRMSTR
     C                         PARM           INPLEN     WW
     C                         GIVNG          RETVAL
     C   02           'Error'  DSPLYTTY
     C   02           RMSEOF   COMP RETVAL             03
     C   03           ' EOF'   DSPLYTTY
```

ZK-4652-85

The information provided in this chapter is general to all VAX/VMS Run-Time Library routines and system services. For specific information on these routines, refer to the *VAX/VMS Run-Time Library Routines Reference Manual* and the *VAX/VMS System Services Reference Manual.*

12.5 Examples of Calling VAX/VMS System Services

Most system services are used primarily by the VAX/VMS operating system on behalf of users. However, many system services are useful for application programming.

The use of some system services is restricted to protect system performance and the integrity of user processes. The privileges and quotas assigned in the User Authorization File (UAF) determine whether you can use a restricted system service. These privileges and quotas apply to every image that your process executes.

The following example calls the SYS$ASCTIM system service to obtain the time. The time is converted from 64-bit system time format to an ASCII string. This service requires three parameters: (1) the length of the returned output string TIMLEN (passed by reference), (2) the character string TIMBUF, to receive the converted time (passed by descriptor), and (3) the conversion value 0 (passed by value). A conversion value of 1 causes only the hour, minute, second, and hundredth of a second fields to be returned. A value of 0 causes the full date and time to be returned. Note that the length of the returned output string must be long enough to accommodate the data to be returned. Because the TIMLEN parameter must be a longword, the access type (write-only) and data type (longword integer) are specified in columns 54 and 55.

If the operation is successful, the date and time (TIMBUF) are displayed on the screen. If the operation is unsuccessful, indicator 02 is set on.

```
                                                    Field length
      Control level                                 |  Decimal positions
      |                                             |  |Half adjust (H)
      | Indicators          Operation               |  ||
      | |                   |                       |  ||Resulting
      | |         Factor    |    Factor    Result|  ||indicators
      | |         1         |    2         field |  ||+ - 0
     C| NxxNxxNxx|          |    |         |     |  ||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *     *--*** * * *
     C            ASCTIM     EXTRN'SYS$ASCTIM'
     C                       CALL ASCTIM                 02
     C                       PARM           TIMLEN     WL
     C                       PARMD          TIMFUF 23
     C                       PARMV          0
     C   N02       TIMBUF    DSPLYTTY
[EOB]
```

ZK-4654-85

The following example calls two VAX/VMS System Services—SYS$CRELOG and SYS$GETMSG. The VAX/VMS System Service SYS$CRELOG sets on external indicators 3 and 7 to control the opening of files in a VAX RPG II program by calling SYS$CRELOG to define the logical name RPG$EXT_INDS. If the operation is unsuccessful, BUFFER receives the error message which SYS$GETMSG returns, and the program displays the error message.

This example also demonstrates a method for modifying the external indicators logical. The effect is that subsequent program runs will have the appropriate external indicators set on, depending on the value of the RPG$EXT_INDS logical. The external indicators in the program example below are not modified in the currently running program. See Chapter 7 for information on modifying the external indicators in a currently running program.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

FERROR   D   F      80            TTY
C*++
C* Call SYS$CRELOG to set on the external indicators 3 and 7.
C*--
C                     MOVEL'RPG$EXT_'LOGNAM 12
C                     MOVE 'INDS'    LOGNAM
C                     MOVE '3,7'     STRING  3
C*
C           CRELOG    EXTRN'SYS$CRELOG'
C                     CALL CRELOG                  99
C                     PARMV          1
C                     PARMD          LOGNAM
C                     PARMD          STRING
C                     PARMV          0
C                     GIVNG          RETVAL
C*++
C* If the call was not successful,
C* call SYS$GETMSG to get the error text
C*--
C   99                CALL GETMSG
C                     PARMV          RETVAL 100
C                     PARM           LENGTH  90 WL
C                     PARMD          BUFFER 80
C                     PARMV          0
C                     PARMV          0
C           GETMSG    EXTRN'SYS$GETMSG'
C*++
C* Display the error text
C*--
C   99      BUFFER    DSPLYERROR
C*++
C* Set on an indicator to end the program
C*--
C                     SETON                     LR
```

ZK-4655-85

The following example calls a VAX/VMS Run-Time Library routine and a VAX/VMS system service. The VAX/VMS Run-Time Library routine LIB$CVT_HTB accepts as input an eight-digit hexadecimal value. The program calls the system service SYS$GETMSG to retrieve the error message text associated with the condition.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FERROR   D   F      80            TTY
     C*++
     C* Prompt message
     C*--
     C                    MOVE 'x value:'MESSAG 16
     C                    MOVEL'Enter '  MESSAG
     C           MESSAG   DSPLYERROR     HEX     8
     C*++
     C* Call LIB$CVT_HTB to convert to binary
     C*--
     C                    CALL CVTHTB
     C                    PARMV          8
     C                    PARM           HEX
     C                    PARM           VALUE      WL
     C           CVTHTB   EXTRN'LIB$CVT_HTB'
     C*++
     C* Call SYS$GETMSG to get the error text
     C*--
     C                    CALL GETMSG
     C                    PARMV          VALUE   90
     C                    PARM           LENGTH  90 WL
     C                    PARMD          BUFFER 80
     C                    PARMV          0
     C                    PARMV          0
     C           GETMSG   EXTRN'SYS$GETMSG'
     C*++
     C* Display the error text
     C*--
     C           BUFFER   DSPLYERROR
     C*++
     C* Set on an indicator to end the program
     C*--
     C                    SETON                     LR
```

ZK-4656-85

For additional information on coding considerations when using external routines, see the *Introduction to VAX/VMS System Routines* and the *Guide to Creating Modular Procedures on VAX/VMS.*

The *Introduction to VAX/VMS System Routines* contains the VAX Procedure Calling and Condition Handling Standard. The VAX/VMS Modular Programming Standard can be found in Appendix A of the *Guide to Creating Modular Procedures on VAX/VMS.*

12.6 Examples of Calling Subprograms

In addition to calling VAX/VMS Run-Time Library routines and system services, VAX RPG II programs can also call subprograms written in languages other than VAX RPG II.

The following program calls a VAX COBOL subprogram and a VAX BASIC subprogram:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *         *         *    *         *     *--*** * * *
     C*--
     C*++
     C* The same parameter list is used by both calls
     C*--
     C           PARAM     PLIST
     C                     PARM           MESSAG 16
     C*++
     C* Call the VAX COBOL program
     C*--
     C                     MOVEL'RPG call'MESSAG
     C                     MOVE 'ed COBOL'MESSAG
     C                     CALL 'COBOL1'  PARAM
     C*++
     C* Call the VAX BASIC program
     C*--
     C                     MOVE 'BASIC'   MESSAG
     C                     CALL 'BASIC1'  PARAM
     C*++
     C* Set on an indicator to end the program
     C*--
     C                     SETON                     LR
```

ZK-4657-85

The following example is the VAX COBOL subprogram:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBOL1.
DATA DIVISION.
LINKAGE SECTION.
01  MESSAGE-1 PIC X(16).
PROCEDURE DIVISION USING MESSAGE-1.
P0.
        DISPLAY MESSAGE-1.
        EXIT PROGRAM.
```

The following example is the VAX BASIC subprogram.

```
100 SUB BASIC1 (STRING MESSAGE = 16 BY REF)
200 PRINT MESSAGE
300 END SUB
```

12.7 Examples of Screen Handling with System Calls

This section provides examples of VAX RPG II program fragments that perform screen handling using the VAX Terminal Data Management System (TDMS), the VAX Forms Management System (FMS), the VAX Screen Management (SMG$), and the CALL statement. Note that access to a subset of VAX FMS is integrated into the language using WORKSTN files, making it unnecessary to use the CALL statement. See Chapter 6 for details.

VAX TDMS, VAX FMS, and VAX SMG$ are designed to make it easier to develop interactive applications. Both VAX TDMS and VAX FMS provide utilities that let you define all the screen forms outside the VAX RPG II program. They also let you design forms by typing them directly onto the terminal screen. An example of a TDMS program is provided in SYS$EXAMPLES:RPGTDMS.RPG.

The following TDMS examples are part of the complete program example provided in SYS$EXAMPLES.

NOTE

If you use SYS$EXAMPLES:RPGTDMS.COM and get a TDMS TSS error that the form display failed, use the DCL command SET TERMINAL/DEVICE to set the device type (if possible) to a device supported by TDMS.

The following example demonstrates the use of data structures and the COPY_CDD directive in a VAX RPG II program that calls TDMS. See Chapter 15 for more information on data structures and the COPY_CDD directive.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IEMPLOYEE                                  1  88  EMPREC
     I
     IEMPREC      DS
     I/COPY_CDD 'CDD$TOP TDMS$EXAMPLES.EMPLOYEE.EMPLOYEE_RECORD'

[EOB]


Press the PF2 key to get help information
```

ZK-4669-85

The following example demonstrates the use of long character literals in an VAX RPG II program that calls TDMS. See Chapter 15 for more information on long character literals.

```
                                                          Field length
     Control level                                        |  Decimal positions
     |                                                    |  |Half adjust (H)
     | Indicators          Operation                      |  ||
     | |                   |                            |  ||Resulting
     | |        Factor     |    Factor    Result|       |  |||indicators
     | |        1          |    2         field |       |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |       |  |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    ** *         *          *    *         *     *--*** * * *
    C            REQUES     EXTRN'TSS$REQUEST'
    C                       CALL REQUES                    99
    C                       PARM           CHAN    90 WL
    C                       PARM           LIBID   90 WL
    C                       PARMD          "
    C"                                     'EMPLOYEE_INITIAL_REQUEST'
```

ZK-4658-85

For further information on VAX TDMS, see the following related documents:

- *VAX TDMS Forms Manual*
- *VAX TDMS Request and Programming Manual*
- *VAX TDMS Application Programming Manual*
- *VAX TDMS Sample Application Manual*

The following program segment is from a VAX RPG II program that calls VAX FMS to display a form:

```
                                                    Field length
 Control level                                      |  Decimal positions
 |                                                  |  |Half adjust (H)
 | Indicators          Operation                    |  ||
 | |                   |                            |  ||Resulting
 | |         Factor    |    Factor    Result|       |  |||indicators
 | |         1         |    2         field |       |  |||+ - 0
C| NxxNxxNxx|         |    |         |     |       |  |||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *         *    *         *     *--*** * * *
     C             FCLRSH    EXTRN'FDV$CLRSH'
     C                       MOVE 'FIRST '  FORM1   6
     C                       CALL FCLRSH
     C                       PARMD          FORM1
```

ZK-4659-85

For further information on VAX FMS, see Chapter 6 and the *VAX FMS Reference Manual.*

Following is a VAX RPG II program that calls VAX SMG$ routines. This program displays the word "Menu" beginning on line 2, column 5.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  ||indicators
     | |         1         |    2         field |   |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  ||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *         *    *         *     *--*** * * *
     C           CREPAS    EXTRN'SMG$CREATE_PASTEBOARD'
     C           CREDIS    EXTRN'SMG$CREATE_VIRTUAL_DISPLAY'
     C           PUTCHA    EXTRN'SMG$PUT_CHARS'
     C           PASDIS    EXTRN'SMG$PASTE_VIRTUAL_DISPLAY'
     C                     Z-ADD0          ZERO    90
     C                     Z-ADD1          LINCOL  90
     C                     Z-ADD2          LINE    90
     C                     Z-ADD5          COLUMN  90
     C                     MOVE 'Menu'     OUT     4
     C* Create the pasteboard.
     C                     CALL CREPAS
     C                     PARM            PASTID  90 WL
     C                     PARMV           ZERO
     C                     PARM            HEIGHT  90 WL
     C                     PARM            WIDTH   90 WL
     C* Create the virtual display.
     C                     CALL CREDIS
     C                     PARM            HEIGHT     RL
     C                     PARM            WIDTH      RL
     C                     PARM            DISPID  90 WL
     C* Output the 'Menu'.
     C                     CALL PUTCHA
     C                     PARM            DISPID     RL
     C                     PARMD           OUT
     C                     PARM            LINE       RL
     C                     PARM            COLUMN     RL
     C* Paste the virtual display.
     C                     CALL PASDIS
     C                     PARM            DISPID     RL
     C                     PARM            PASTID     RL
     C                     PARM            LINCOL     RL
     C                     PARM            LINCOL     RL
     C                     SETON                      LR
```

ZK-4668-85

For further information on VAX SMG$ routines, see the *VAX/VMS Run-Time Library Routines Reference Manual.*

Chapter 13

Optimizing Your Programs

The word "optimization" as used in this chapter, refers to the process of improving the efficiency of programs. The objective of optimization is to produce programs that achieve the greatest amount of processing with the least amount of time, memory, and secondary storage.

13.1 Optimizing with Data Structures

Using data structures to update files can improve the run-time performance of your programs. The following example updates a file with a data structure defined in an Input specification and used in an Output specification.

```
     0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890

FOUT94A  UD  F      24            DISK
FOUT94B  UD  F      24            DISK
IOUT94A  AA
I                                        1   3 PN
I                                        4  10 PNAME
I                                       11  12 WHOUSE
I                                       13  17 COLOR
I                                       18  20 WEIGHT
I                                       22  240QTY
IOUT94B  AA
I                                        1  24 DS94B
IDS94B       DS
I                                        1   3 PN2
I                                        4  10 PNAME2
I                                       11  12 WHOUS2
I                                       13  17 COLOR2
I                                       18  20 WEIGH2
I                                       22  240QTY2
.
.
.
OOUT94A  E
O                        PN          3
O                        PNAME      10
O                        WHOUSE     12
O                        COLOR      17
O                        WEIGHT     20
O                        QTY        24
OOUT94B  E
O                        DS94B      24
```

ZK-4432-85

Note that the fields to be updated in the Output specification for file OUT94B are not listed a second time, as they would have to be without use of a data structure. Because the layout of the fields is described only *once* in the data structure, the results are shorter code and a program less prone to error. Without a data structure, the fields must be described on both the Input and Output specifications.

13.2 Optimizing with Adjacent Fields in Records

VAX RPG II extracts adjacent fields from the record buffer with a single MOVE instruction and writes them back in the same way to save time. This optimization is performed only if data conversion is unnecessary. Therefore, you should keep the fields contiguous to avoid requiring multiple MOVE instructions.

13.3 Optimizing with Blank Factor 1

If you use blank factor 1, you will have less code to write and your program will be less prone to error because you are not writing the same factor twice. The following example, which is part of the preceding example program, demonstrates this technique:

```
    0    1    2    3    4    5    6    7
1234567890123456789012345678901234567890123456789012345678901234567890

I                                          22  240QTY2
C* Read records from update files.
C                     READ OUT94A                    LR
C  NLR               READ OUT94B
C* Update quantity to reflect the fact that 100 of each part came in.
C  NLR               ADD  100       QTY
C  NLR               ADD  100       QTY2
C* Write the updated records.
C  NLR               EXCPT
```

ZK-4433-85

13.4 Optimizing with the Asterisk Indicator

The asterisk indicator (*) is used when you have multiple calculation specifications that are conditioned by the same indicators. There are two advantages to using the asterisk indicator. First, it saves you coding time and reduces errors, because multiple indicators do not have to be repeated in groups of Calculation specifications that are to be conditioned by the same indicator. Second, it improves run-time performance.

If you do not use the asterisk indicator, groups of indicators must be tested on each Calculation specification, even if the same indicators are used to condition a group of Calculation specifications. With an * in column 11, only a single test for each Calculation specification is needed to test whether the previous specification was executed.

The * can be used in a similar fashion to condition output records and output fields.

13.5 Optimizing File Performance

You can control file access and improve file performance through optimizing techniques discussed in this manual and in the *Guide to VAX/VMS File Applications*. The following optimizing techniques are discussed in Chapter 15 of this manual:

- For information on the use of expansion factors to prevent bucket splitting and to improve search efficiency, see Section 15.3.24.
- For information on file sharing, see Section 15.3.25.
- For information on multibuffer count, see Section 15.3.21.
- For information on longer block length for decreasing I/O processing time, see Section 15.3.9.
- For information on multiblock count, see Section 15.3.14.

For more information on optimizing techniques, the *Guide to VAX/VMS File Applications* provides pertinent information on tuning sequential, relative, and indexed files. That manual also describes optimizing file performance and processing in a VAXcluster, and offers performance recommendations.

Language Reference

This part of the manual provides reference information on the primitives and constructs of VAX RPG II:

- Language elements
- VAX RPG II specifications
- Operation codes

Chapter 14

VAX RPG II Language Elements

This chapter describes the primitives or *elements* of a VAX RPG II program. These elements include the character set, the various data types, and the names that are defined and used to identify program constructs and certain parts of those constructs.

14.1 VAX RPG II Character Set

VAX RPG II uses the full ASCII character set, including:

- Uppercase letters A through Z, except for character literals and comment fields
- Digits 0 through 9
- Special characters (such as #, $, @)

Appendix A contains the complete ASCII character set and character values.

14.2 VAX RPG II Data Types

VAX RPG II input and output operations use the following data types that determine how many bits of storage compose a unit of data in a program, and how that unit is to be interpreted and manipulated at program execution time.

VAX RPG II supports five different data types for input and output operations:

- Character
- Word binary numeric
- Longword binary numeric
- Packed decimal
- Overpunched decimal

This section describes each data type.

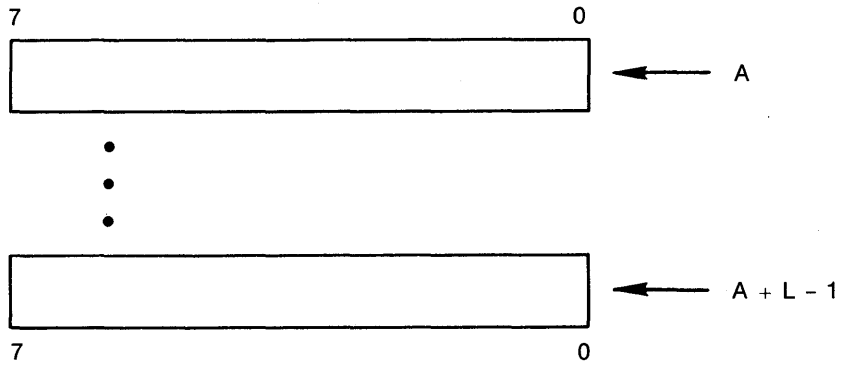
14.2.1 Character Data Type

Character data is a string of bytes containing ASCII codes as binary data. The length can be from 1 to 9999 bytes. The format of a character string is shown in Figure 14–1.

NOTE

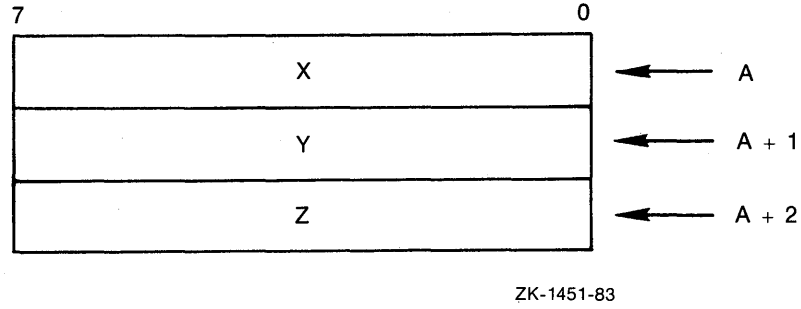
In all subsequent diagrams, A represents the address of the first byte of the string and L represents the length of the string in bytes.

Figure 14–1: Format of a Character String



The address of a string specifies the first character of a string. The address XYZ is shown in Figure 14–2.

Figure 14–2: Address of a String

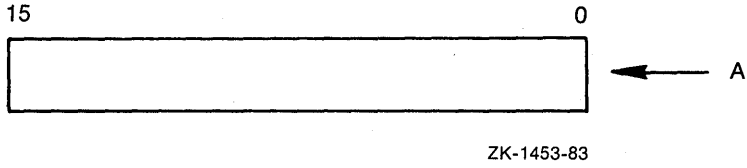


14.2.2 Binary Data Type

Binary data is stored as binary values in a word or longword. A word is two contiguous bytes, starting on an arbitrary byte boundary. The bits are numbered from the right (0 through 15). When interpreted as a signed

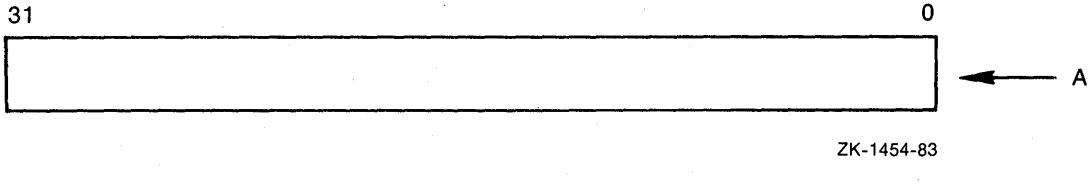
quantity, a word is a twos complement number with bits increasing in significance from bit 0 through bit 14, and with bit 15 designating the sign. A two-byte word supports up to four decimal digits. The largest number that can be represented by a word in VAX RPG II is 9999. A word data type is shown in Figure 14–3.

Figure 14–3: Word Data Type



A longword is four bytes, starting on an arbitrary byte boundary. The bits are numbered from the right (0 through 31). When interpreted as a signed quantity, a word is a twos complement number with bits increasing in significance from bit 0 through bit 30, and with bit 31 designating the sign. A four-byte longword supports up to 9 decimal digits. The largest number that can be represented by a longword in VAX RPG II is 999,999,999. A longword datatype is represented in Figure 14–4.

Figure 14–4: Longword Data Type

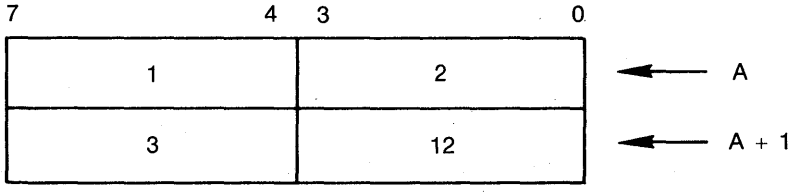


14.2.3 Packed Decimal Data Type

Packed decimal data is stored as a string of bytes. Each byte is divided into two 4-bit half bytes (nibbles), with one decimal digit stored in each half byte. The first, or most significant digit is stored in the high-order half byte of the first byte; the second digit is stored in the low-order half

byte of the first byte; the third digit is stored in the high-order half byte of the second byte; and so on. The sign of the number is stored in the low-order half byte of the last byte of the string. The number +123, in packed decimal format, is shown in Figure 14–5.

Figure 14–5: Packed Decimal Data Type



ZK-1455-83

A decimal 10, 12, 14, or 15 represents a plus sign, although 12 is used when the number is created as a result of a VAX arithmetic instruction. A decimal 11 or 13 represents a minus sign, although 13 is used when the number is created as a result of a VAX arithmetic instruction.

The following formula can be used to determine the length in digits of a packed decimal field:

```
number of digits = 2n - 1
where n = number of bytes used
```

See Section 15.6.15.4 for examples of selecting numeric data types in a VAX RPG II program.

14.2.4 Overpunched Decimal Data Type

Overpunched decimal data is a contiguous sequence of bytes in memory, with one decimal digit in a byte. Digits of decreasing significance are assigned to increasing addresses. The sign is superimposed on the last digit (trailing numeric string).

All bytes of overpunched decimal data, except the least significant digit, must contain ASCII decimal digits (0 through 9). Table 14–1 lists the representation for all digits but the least significant digits.

Table 14–1: Overpunched Decimal Representation of All but the Least Significant Digits

Sign	Decimal	Hexadecimal	ASCII Character
0	48	30	0
1	49	31	1
2	50	32	2
3	51	33	3
4	52	34	4
5	53	35	5
6	54	36	6
7	55	37	7
8	56	38	8
9	57	39	9

There are several variations of overpunched decimal format. Alternate forms of overpunched decimal format are accepted on input. The normal form of overpunched decimal format is generated on output. Valid representations of the digits and signs in each of the latter two formats (input and output) are shown in Table 14–2.

Table 14–2: Overpunched Decimal Representations of Least Significant Digits and Signs

	Overpunched Decimal Format		ASCII Characters	
Digit	Decimal	Hexadecimal	Normal	Alternate
0	48	30	0	{ [?
1	49	31	1	A
2	50	32	2	B
3	51	33	3	C
4	52	34	4	D
5	53	35	5	E
6	54	36	6	F
7	55	37	7	G
8	56	38	8	H

Table 14–2 (Cont.): Overpunched Decimal Representations of Least Significant Digits and Signs

	Overpunched Decimal Format		ASCII Characters	
Digit	Decimal	Hexadecimal	Normal	Alternate
9	57	39	9	I
-0	125	7D	}	]:!
-1	74	4A	J	
-2	75	4B	K	
-3	76	4C	L	
-4	77	4D	M	
-5	78	4E	N	
-6	79	4F	O	
-7	80	50	P	
-8	81	51	Q	
-9	82	52	R	

Figure 14–6 shows 123 in trailing numeric string format.

Figure 14–6: Overpunched Decimal Data Type (123)

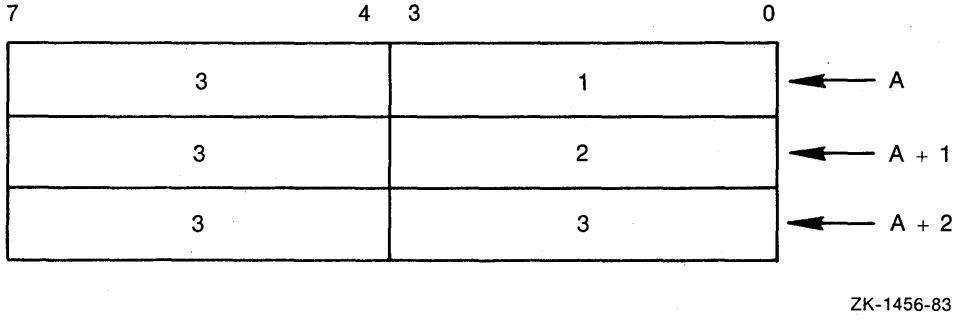
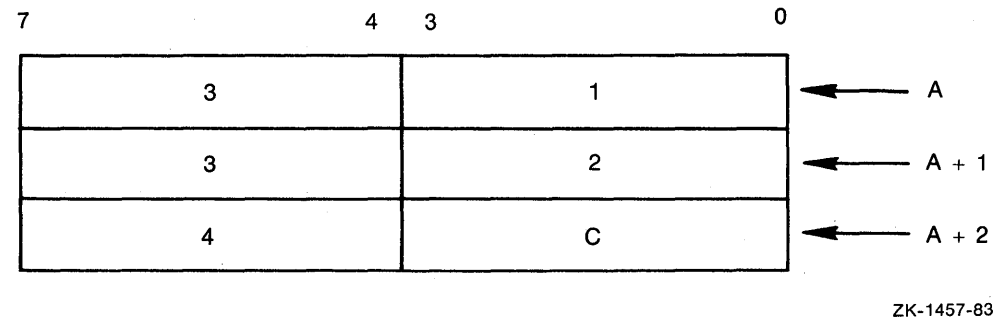


Figure 14–7 shows -123 in trailing numeric string format.

Figure 14–7: Overpunched Decimal Data Type (-123)



14.3 User-Defined Names

A user-defined name is a named quantity that identifies items in a VAX RPG II program. These items include:

- Files—a file name is assigned to a file.
- Fields—a field name is assigned to a field in a program. You can use a field name in more than one field definition if each definition using that name has the same data type, the same length, and the same number of decimal positions.
- Arrays—an array name is assigned to an array. The first three characters cannot be TAB.
- Tables—a table name is assigned to a table. The first three characters must be TAB.
- Labels—a label identifies the destination of a GOTO operation code.
- Subroutines—a subroutine name is assigned to a subroutine.
- PLIST—a PLIST name is assigned to a list of parameters to be passed to a subprogram.
- EXCPT—an EXCPT name can be used in factor 2 of the EXCPT operation code and in the Field name field of exception record Output specifications. See Section 15.8.5 for information on exception record types.

When defining a name, observe the following rules:

Rules

- The first character of a name must be one of the following:
  - Uppercase letters A through Z
  - An underscore (_)
  - A pound sign (#)
  - A dollar sign ($)
  - An at sign (@)
- The remaining characters of a name can be the uppercase letters A through Z, the digits 0 through 9, an underscore (_), a pound sign (#), a dollar sign ($), or an at sign (@).
- You must left justify names.
- You cannot embed blanks in a name.
- You cannot use a VAX RPG II special word as a name. See Chapter 9 for information on special words.
- The maximum length of a name is six characters, except for a file name, which can be up to eight characters long.
- Every user-defined name must be unique. For example, a name assigned to a file cannot be used as a field name.

Chapter 15

VAX RPG II Specifications

VAX RPG II specifications are the major program element of the language and tell the computer what data to use, how to process it, and what to do with the results. This chapter provides general VAX RPG II information and a description of each VAX RPG II specification.

The VAX RPG II specifications described in this chapter are as follows:

- Control
- File Description
- Extension
- Line Counter
- Input
- Calculation
- Output

Each specification description includes the following information:

- A brief explanation of the specification's purpose
- The specification's format
- A detailed explanation of each column
  - A brief explanation of the column's purpose
  - A table listing valid entries for the column
  - An example of a typical use

Use the information in this chapter for quick reference. For information on topics requiring a more detailed explanation, see the Table of Contents or Index.

Language Conventions and Format

15.1 General Language Information

This section provides a description of the notation conventions for VAX RPG II specifications, data fields that are common to all specifications, and special instructions to the language compiler.

15.1.1 Notation Conventions

This section describes the notation conventions of the specifications described in this chapter.

Two rows of digits identify column numbers, as shown in the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
```

ZK-4454-85

The arrow in the following example points to column 43. The vertical line above the first zero separates columns 30 through 39 from columns 40 through 49; the vertical line above the second zero separates columns 40 through 49 from columns 50 through 59.

```
    |    4    |
01234567890
           ↑
```

ZK-4530-85

Asterisks (*) in the dotted line below the two rows of column numbers indicate the beginning of fields that have values for the specification being described. Each field is terminated by another asterisk, by a dot, or by column 75.

The positions of the asterisks are different for each specification.

The asterisk in the following example indicates that you can enter a value in column 6.

The dots in the line below the two rows of column numbers identify fields that must be blank.

```
  0    I    1    I    2    I    3    I    4    I    5    I    6    I    7    I
12345678901234567890123456789012345678901234567890123456789012345678901234567890
    *.....................................................................
```

ZK-4455-85

In the following example, dashes after an asterisk indicate a field that must contain numeric data. Numeric data must be right justified in the field. Blanks after an asterisk indicate a field that must contain alphanumeric data. Alphanumeric data must be left justified in the field.

```
  0    I    1    I    2    I    3    I    4    I    5    I    6    I    7    I
12345678901234567890123456789012345678901234567890123456789012345678901234567890
    **      ******---*---**-*** *---**     *     *......*-----**..** ..
```

ZK-4456-85

In the following example, the value 32 has been entered in columns 20 through 23; the value 41 has been entered in columns 24 through 27; no value has been entered in column 28; the value 9 has been entered in columns 29 and 30; no value has been entered in column 31.

```
I   2   I
012345678901
*---*---**-*
  32  41  9
```

ZK-4457-85

15.1.2 Common Fields

This section describes fields that are common to all specifications.

15.1.2.1 Line Number

You can associate a line number in columns 1 through 5 on each line of your program. Line numbers are optional, but can be useful in checking the sequence of lines. To check whether line numbers are in the proper sequence, you must specify the /SEQUENCE_CHECK qualifier to the RPG command. If you do not, VAX RPG II will ignore all line numbers. The absence of line numbers does not affect your program.

Column Number	Allowable Values	Explanation
1-5	Any number	Associates a line number with the program line

Additional Information

If you specify the /SEQUENCE_CHECK qualifier to the RPG command and the line numbers are out of sequence, VAX RPG II will issue a warning compile-time error.

15.1.2.2 Specification Type

You must identify the type of specification in column 6 on each line of your program.

Column Number	Allowable Values	Explanation
6	H	Control specification
	F	File Description specification
	E	Extension specification
	L	Line Counter specification
	I	Input specification
	C	Calculation specification
	O	Output specification

15.1.2.3 Comments

You can use columns 75 through 80 to write comments about the program line. VAX RPG II ignores entries in columns 75 through 80. In other implementations of VAX RPG II, these columns are used for a program name. VAX RPG II uses the source file name as the name of the program.

Column Number	Allowable Values	Explanation
75-80	Any character	Documents the program line

Rules

- Blank lines can appear between any two specifications. VAX RPG II ignores blank lines.
- A specification containing only a form feed can appear between any two specifications. Specifications containing only a form feed are treated like blank lines except in the listing file, where they cause the listing to skip to the top of the next page.

Additional Information

You can also use an entire specification to write a comment when you precede the comment with an asterisk (*) in column 7. You can do this with any type of specification. Any line with an asterisk (*) in column 7 before the first double slash (//) or double asterisk (**) is considered a comment.

15.1.3 Compiler Directing Statements

You can include compiler directing statements in your program source file to access other files as information resources. You do this with the COPY and COPY_CDD directives as described in the following subsections.

15.1.3.1 COPY Directive

The COPY directive allows you to copy one or more source files into the main source file during compilation. This feature can be used for copying in common subroutines, record definitions, or other useful information.

The COPY directive can be followed by a modifier statement which supplies additional information on a preceding Input specification field. See Section 15.1.3.3 for the syntax and rules of modifier statements.

If you specify the /SEQUENCE_CHECK qualifier, then the copied lines are sequence checked separately from the main source file.

The COPY and COPY_CDD directive lines are always listed in the listing file. The copied lines immediately follow the COPY directive line in the listing. Each line is given a unique listing line number. A "C" is placed after the line number in the listing record to indicate that the line was generated by a COPY directive. There is still a single set of line numbers to mark the entire source file, after allowing for text that has been copied into the main source file.

If a COMPILE command is issued from the VAX RPG II editor and an error is flagged within the compilation of a COPY or COPY_CDD directive, the cursor is placed at the COPY directive, and the message is displayed on the message line. You must leave the VAX RPG II editor if you want to see the compiler source listing.

The file SO_MUCH_TIME_AND_SO_LITTLE_TO_DO.RPG is copied in as a result of the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *          *     *--*** * * *
     C/COPY 'SO_MUCH_TIME_AND_SO_LITTLE_TO_DO'
```

ZK-4458-85

If multiple COPY directives are used in a program to define compile-time tables or arrays, the second and subsequent entries beyond the first COPY directive will be flagged as errors. For example:

```
O/COPY 'ARRAY1'
O/COPY 'ARRAY2'
```

The second COPY directive is flagged as an error because a ** was recognized from the first COPY directive, and COPY directives cannot follow a ** or //. For detailed information, see Section 15.1.3.

The workaround for this is to use append or plus lists on the RPG compiler command. For detailed information on the RPG command, see Section 3.1.2.

The COPY directive adheres to the following rules:

Rules

- The COPY directive may appear anywhere within the source file before the first double slash (//) or double asterisk (**) line. It cannot appear after that, because the remaining lines do not contain a specification type. All lines after the first // or ** are treated as nonsource lines even if the // or ** occurs in a COPYgfile.
- The COPY directive cannot appear within a long character literal.
- There is no limit on the number of copy directives in a program.
- Copy directives cannot be nested. A file copied in cannot contain a COPY or COPY_CDD directive. See Section 15.1.3.2 for information on the COPY_CDD directive.
- A COPY directive must appear on a line of its own and have the following syntax:
  - Column 6—must contain any valid specification type (not checked for sequence)
  - Columns 7 through 12—must contain COPY (note the blank space in column 12)
  - Columns 13 through 74—must contain the file specification enclosed in single quotes; the file specification does not need to start in column 13
- A default file type of RPG is used

15.1.3.2 COPY_CDD Directive

Record definitions can be stored in the VAX Common Data Dictionary (CDD) and shared among VAX RPG II programs. You can extract these data definitions from the CDD and use them as field definitions on Input or Output specifications. If you change a definition in the CDD, you do not need to rewrite the program using the new definition; however, you must recompile the program to obtain the latest definition in the CDD.

The following example shows how to use the COPY_CDD directive to copy field information from the CDD on an Input specification:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       * *** *---    *---    *---    .**---*---**      * * * * * * ....
     I/COPY_CDD 'RECORD.IN'
```

ZK-4459-85

RECORD.IN is the CDD path name for a record description stored in the CDD. A path name must be enclosed in single quotation marks; note that the path name does not need to start in column 17.

For information on CDD path names, see the *VAX Common Data Dictionary User's Guide.*

Any CDD name larger than six characters is truncated to the first six characters of the CDD name. If this truncated name has been used previously in a data structure, the last six characters of the CDD name are used.

The CDD provides these RPG-specific features: NAME FOR RPG, EDIT_WORD FOR RPG, and EDIT_CODE FOR RPG. For more information on attributes, see the *VAX Common Data Dictionary Utilities Reference Manual.*

The following example and figure show field definitions entered in the CDD and the Input specifications that must be entered to extract the information.

Using the DCL command CDDL, these field definitions are entered:

```
DEFINE RECORD CDD$TOP.EXAMPLE.ADDRESS_RECORD
   ADDR STRUCTURE.
       STREET              DATATYPE IS TEXT
                           SIZE IS 30 CHARACTERS
       CITY                DATATYPE IS TEXT
                           SIZE IS 30 CHARACTERS
       STATE               DATATYPE IS TEXT
                           SIZE IS 2 CHARACTERS
       PHONE STRUCTURE.
            AREA           DATATYPE IS NUMERIC RIGHT OVERPUNCHED
                           SIZE IS 3 DIGITS
            NUMBER         DATATYPE IS NUMERIC RIGHT OVERPUNCHED
                           SIZE IS 7 DIGITS
                               NAME FOR RPG IS
                               P#
         END PHONE STRUCTURE.
     END ADDR STRUCTURE.
END ADDRESS_RECORD.
```

In the VAX RPG II program, these Input specifications are entered:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        * *** *---    *---    *---    .**---*---**      * * * * * * ....
     IINFO     AA
     I/COPY_CDD 'EXAMPLE.ADDRESS_RECORD'
```

ZK-4460-85

The information is extracted from the CDD and parsed as though the user had entered the following:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        * *** *---    *---    *---    .**---*---**      * * * * * * ....
     IINFO     AA
     I                                          1  30 STREET
     I                                         31  60 CITY
     I                                         61  62 STATE
     I                                         63  72 PHONE
     I                                         63  650AREA
     I                                         66  720P#
```

ZK-4461-85

15.1.3.3 COPY_CDD Directive Modifiers

To include indicators on input fields copied from the CDD you must enter a modifier statement after the COPY_CDD directive. You can modify any field in the current record including those copied from the CDD. The following fields can be modified:

- Control-level indicator
- Matching fields
- Field-record-relation
- Field indicators

Rules

- A modifying statement is distinguished from other specifications by an ampersand (&) in column 7.
- Only specifications that define fields can be modified.
- As many modifiers as desired can be specified in one modifier specification.
- The same field can be modified by multiple modifiers.
- A field specification must be syntactically valid before and after a modifier is applied.

A modifier can be used to add an indicator where there was none in the original specification. The following example shows a control-level indicator set on the AREA field in the program:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        * *** *---    *---    *---    ,**---*---**     * * * * * * ....
     IINFO     AA
     I/COPY_CDD 'EXAMPLE.ADDRESS_RECORD'
     I&                                                  AREA  L1
```

ZK-4462-85

The field AREA is treated as if it had been specified with an L1 indicator in the control-level indicator field, that is, as if the specification had been as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     I                                          63  650AREA  L1
```

ZK-4463-85

A modifier can be used to supersede a previously specified value. You can also blank a field by entering an ampersand (&) as the first character of the desired field and leaving the rest of the field blank, as in the following example:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         * *** *---    *---    *---    .**---*---**      * * * * * * ....
     IMSLI27       01    1 CA
     I/COPY 'MSL27A'
     I&                                                  FLDA       &
```

ZK-4465-85

The previous example assumes that the file to be copied (MSL27A.RPG) contains the following specification:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         * *** *---    *---    *---    .**---*---**      * * * * * * ....
     I                                             6    6 FLDA       01
```

ZK-4464-85

In the example, FLDA is treated as if no field-record-relation indicator is given, that is, as if the Input specification is as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         * *** *---    *---    *---    .**---*---**      * * * * * * ....
     I                                             6    6 FLDA
```

ZK-4466-85

Modifiers can apply to all field definitions that follow the record containing the last COPY or COPY_CDD directive. In the preceding example, if FLDA occurred several times in the copied file, each occurrence would be modified.

If the source program looked like the following example, then only occurrences of the FLDA field associated with the second COPY directive would be modified.

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       * *** *---    *---    *---    .**---*---**      * * * * * * ....
     IMSLI27  AA  01
     I/COPY 'MSL27A'
     I        BB  02
     I/COPY 'MSL27A'
     I&                                                FLDA       &
```

ZK-4467-85

When a modifier specification is not preceded by a COPY directive, all previous input fields can be modified.

Control Specification (H)

15.2 Description

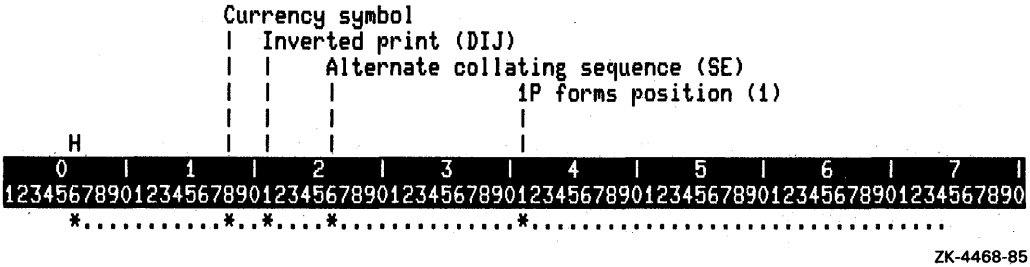
The Control specification allows you to do the following:

- Assign a character other than the default (a dollar sign ($)) as the currency symbol
- Specify the notation for numeric fields, edit codes, and UDATE
- Specify an alternate collating sequence
- Check the alignment of printed forms
- Specify the program name that can be used with WORKSTN files to determine which forms library to use

A Control specification is not required and should be used only when you need to define the use of one or more items described in the previous list. Use only one Control specification in a program.

15.2.1 Control Specification Format

The format of the Control specification is as follows:



ZK-4468-85

15.2.2 Specification Type

Use column 6 to identify the type of specification for every program line.

Column Number	Allowable Values	Explanation
6	H	Indicates that this program line is a Control specification

15.2.3 Currency Symbol

Use column 18 to specify a character other than the dollar sign ($) to represent the currency symbol.

Column Number	Allowable Values	Explanation
18	Blank	Uses the dollar sign ($) as the currency symbol. The dollar sign is the default.
	Character	Uses the character you specify instead of the dollar sign ($) for the currency symbol.

You can use any character for the currency symbol except zero (0), asterisk (*), comma (,), ampersand (&), decimal point (.), minus sign (-), C (C), and R (R).

15.2.4 Inverted Print

The inverted print feature allows you to format your program output to conform with a variety of notation conventions. For example, using the United States convention for expressing a number with thousands and decimals, a number could be 12,345.67; the same value could be correctly expressed as 12.345,67 using other conventions. Use column 21 to specify the notation the printer uses for numeric fields, edit codes, and UDATE.

Column Number	Allowable Values	Explanation
21	Blank	Uses a period as the decimal notation and a comma as the thousands separator in numeric literals and edit codes (for example, 1,234.56 and .56). Uses a slash (/) as the separator for the Y edit code; uses the mm/dd/yy format for UDATE (for example, 07/04/76). See Section 15.8.14 for information on edit codes. See Chapter 9 for information on UDATE.
	D	Uses the same format as the BLANK option for numeric literals and edit codes. Uses a slash (/) as the separator for the Y edit code; uses dd/mm/yy format for UDATE.
	I	Uses a comma as the decimal notation and a period as the thousands separator in numeric literals (for example, 1.234,56) and edit codes. Uses dd/mm/yy format for UDATE; decimal points separate the day, month, and year elements (for example, 24.03.83).
	J	Uses the same format as the I option for UDATE, numeric literals, and edit codes with the following exception: writes a zero to the left of the comma when the field contains only a fraction (for example, 0,56).

15.2.5 Alternate Collating Sequence

Use column 26 to specify the collating sequence you want VAX RPG II to use when comparing character fields using the COMP operation code and when checking the sequence of matching fields.

Column Number	Allowable Values	Explanation
26	Blank	Uses the ASCII collating sequence. See Appendix A for the ASCII character set.
	E	Uses the EBCDIC collating sequence. See Appendix A for the EBCDIC character set.
	S	Uses a user-defined collating sequence. See Appendix A for the ASCII characters and their hexadecimal values.

To define a collating sequence that is different from the standard ASCII or EBCDIC sequences, you must specify the hexadecimal value of each character whose position in the sequence you want to change, as follows:

- Specify S in column 26 of the Control specification.
- Include the specification for ALTSEQ records after the Output specification, but before any compile-time table and arrays, if used.
- Precede the ALTSEQ records with double slashes (//) and a blank or double asterisks (**) and a blank in columns 1 through 3.
- Specify the entries in the following table.

Column Number	Allowable Values	Explanation
1-8	ALTSEQbb	Indicates that you are specifying an alternate collating sequence. Note that bb represents two blanks.
9,10	Hexadecimal value	Specifies the hexadecimal value of the character you want to change.
11,12	Hexadecimal value	Specifies the new hexadecimal value of the character whose position in the collating sequence you want to change.

Repeat this sequence of hexadecimal numbers up to column 80 for additional changes. The first blank space in an ALTSEQ record terminates the ALTSEQ entries for that record. The rest of the line can be used for comments.

In the following example, columns 9 and 10 and 13 and 14 contain the hexadecimal value of the character to be changed, and columns 11 and 12 and 15 and 16 contain the new hexadecimal value of the character. In the following collating sequence, VAX RPG II changes the uppercase Z (5A) to an uppercase A (41) and the lowercase w (77) to an uppercase J (4A).

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **
//
ALTSEQ  5A41774A
```

ZK-4469-85

15.2.6 Forms Position

Use column 41 to check the alignment of printed output on a nonstandard form.

Column Number	Allowable Values	Explanation
41	Blank	Specifies no forms-positioning check
	1	Checks the forms positioning by printing the first output line

This entry is optional and valid only for a nonspooled device.

Additional Information

When you specify forms positioning, the printer outputs the first line. Then, VAX RPG II asks the following question for each printer output file in the order that the first lines are output:

```
Is forms positioning correct?
Yes, type Continue, No, type Retry:
```

If you type CONTINUE, the program will print the second output line, and so on, until all lines are output. If the forms are not correctly positioned, realign the form, and then type RETRY. VAX RPG II will print the first line again so that you can determine whether the form is positioned correctly.

15.2.7 Forms Library

Use columns 75 through 80 to indicate the VAX Forms Management System (VAX FMS) forms library if the continuation line option FMTS in the F specification is not to be included in the WORKSTN file specification.

15.2.8 Example

In the following example, these control characteristics are defined:

- Program line is a Control specification
- Currency symbol is a pound sign (#)
- VAX RPG II uses
  - A comma (,) as the decimal notation for numeric literals and edit codes
  - A period (.) as the thousands separator for numeric literals and edit codes
  - The dd.mm.yy format for UDATE
- Collating sequence is EBCDIC
- Forms-positioning check is not required

```
                  Currency symbol
                  |  Inverted print (DIJ)
                  |  |    Alternate collating sequence (SE)
                  |  |    |              1P forms position (1)
                  |  |    |              |
     H            |  |    |              |
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     *...........*..*....*.............*.................................
     H            #  I    E
```

ZK-4470-85

File Description Specification (F)

15.3 Description

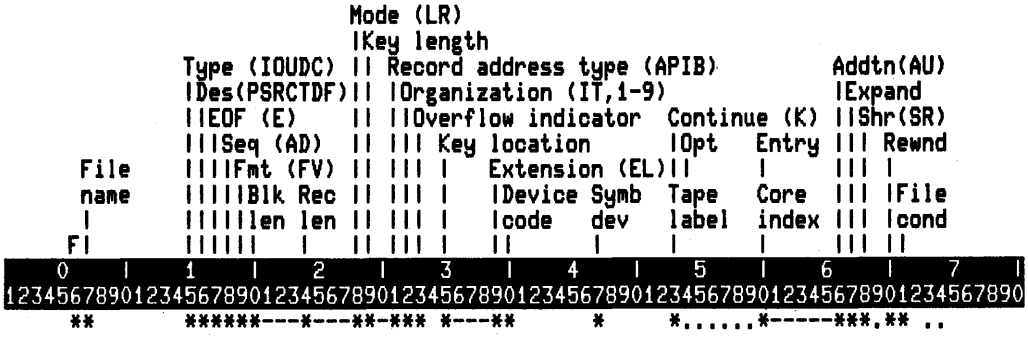
The File Description specification describes the following attributes of each file you use in your program:

- File Description specification format
- Specification type
- File name
- File type
- File designation
- End-of-file
- Sequence
- File format
- Block length
- Record length
- Mode of processing
- Key length
- Record address type
- File organization or additional input/output area
- Overflow indicators
- Key location
- Extension code
- Device code
- Symbolic device
- Tape label
- File Description specification continuation lines
- Core index
- File addition and unordered output
- Expansion factor
- File sharing
- Tape rewind

- File condition
- Example

15.3.1 File Description Specification Format

The format of the File Description specification is as follows:



ZK-4471-85

15.3.2 Specification Type

Use column 6 to identify the type of specification for every program line. Note that F specification continuation lines contain blanks in columns 7 through 52.

Column Number	Allowable Values	Explanation
6	F	Indicates that this program line is a File Description specification

15.3.3 File Name

Use columns 7 through 14 to identify the files you use in a VAX RPG II program.

Column Number	Allowable Values	Explanation
7-14	File name	Identifies the file

Rules

- The number of files you can open depends on the open file quota set by your system manager. To determine the number of files you can open in a VAX RPG II program, type the SHOW PROCESS/QUOTA command and look for the number to the right of open file quota :.
- The specification options explained in Sections 15.3.4 through 15.3.26 apply to the file you identify in columns 7 through 14.
- VAX RMS uses the file name as the default file specification string. See Chapter 8 for information about how VAX RPG II uses the file name field and the symbolic device field to generate the VAX/VMS file specification.
- VAX FMS uses the file name as part of determining the form library name to be used for WORKSTN files. See Chapter 6 for additional information.

15.3.4 File Type

Use column 15 to specify the file type which defines how VAX RPG II processes the records in the file.

Column Number	Allowable Values	Explanation
15	I	Designates an input file. VAX RPG II reads the records from an input file and uses these records as data. These records must be defined in Input specifications unless column 16 contains R or T. Input files can reside on disk or tape, or can be read from a terminal or from cards.
	O	Designates an output file. The program writes or prints the records of an output file. These records must be defined in Output specifications unless this is a table output file. Output files can be written to a printer, disk, tape, or terminal.

Column Number	Allowable Values	Explanation
	U	Designates an update file. Update files must reside on disk. VAX RPG II can read, change, and write records in an update file. The records in these files must be defined in the Input and Output specifications.
	D	Designates a display input or output file. Use display files to accept input from a terminal or to display output on a terminal. You must complete a Calculation specification to define the fields you want to display using the DSPLY operation code. See Chapter 16 for information on the DSPLY operation code.
	C	Designates a combined input/output file. A combined file must have a device code of WORKSTN.

15.3.5 File Designation

Use column 16 to specify file designation which defines the order of processing files. See Chapter 8 for information on processing files.

Column Number	Allowable Values	Explanation
16	Blank	Specifies a nonchained output file or a display file.
	P	Specifies a primary file. You can use only one file as the primary file. It can be an input or an update file. In multifile processing, the primary file determines the order of record selection; in single file processing, the primary file provides all input records. If a primary file is not specified and one or more secondary files are specified, the first secondary file is assigned as the primary file. If no primary or secondary files are specified, you must provide an exit for your program by setting on the last-record (LR) indicator.

Column Number	Allowable Values	Explanation
	S	Specifies a secondary file. A secondary file can be an input or an update file. See Chapter 8 for information on processing secondary files.
	R	Specifies a record-address file. A record-address file indicates which records to process and the order in which they are to be processed. This file must be associated with a file defined in an Extension specification. See Chapter 8 for information on record-address files.
	C	Specifies a chained file. A chained file resides on disk and can be used as an input, output, or update file. You use the CHAIN operation code in the Calculation specification to randomly read records from a chained file. You use an output chained file to add records to a direct file. The CHAIN operation positions the file before VAX RPG II writes each record. See Chapter 16 for more information on chained files.
	T	Specifies a preexecution-time table or array. You must enter, in columns 11 through 18 of the Extension specification, the name of the file that contains the data from which you want to load the table or array. See Chapter 10 for information on tables. See Chapter 11 for information on arrays.
	D	Specifies a demand input or update file. You can use the READ operation code in the Calculation specification to sequentially access the records in a demand file. See Chapter 16 for more information on using the READ operation code to access records from demand files.
	F	Specifies a full-procedural input or update file. You can use the READ and/or CHAIN operation code in the Calculation specification to sequentially and/or randomly access the records.

15.3.6 End-of-File

Use column 17 to specify end-of-file that indicates whether the program can end before VAX RPG II processes all the records in the file.

Column Number	Allowable Values	Explanation
17	Blank	Causes the program to finish reading all the records from every primary and secondary file before ending, if column 17 is blank for all primary and secondary files. If column 17 is not blank for all primary and secondary files, VAX RPG II may or may not process all the records in this file (the file described in this specification). If column 17 is blank for all primary and secondary files, VAX RPG II processes all the records in this file (the file described in this specification).
	E	Causes the program to finish reading the records in the file before ending the program, regardless of the presence of other files. You can use this option for input or update files as primary, secondary, or record-address files. You cannot use this option on a file being processed by a record-address file. When you specify E in column 17 for the primary file, and there are matching records in the primary and secondary files, VAX RPG II reads and processes any records in the secondary file that match the last record of the primary file before ending the program.

15.3.7 Sequence

Use column 18 to specify ascending or descending sequence to check matching fields. See Chapter 8 for information on matching fields.

Column Number	Allowable Values	Explanation
18	Blank	Indicates that the program contains no matching fields or, if it does, assumes the same value as specified for a previous primary or secondary file. If the program contains matching fields and you do not specify a sequence for any file containing matching fields, VAX RPG II assumes an ascending order.
	A	Checks matching records for ascending order.
	D	Checks matching records for descending order.

Rules

- This entry applies only to primary or secondary files with matching fields. See Chapter 8 for more information on primary and secondary files.
- This entry must be the same for each file you process with matching fields.

15.3.8 File Format

Use column 19 to specify file format. File format specifies whether the records in the file are all the same length, or whether they can be of different lengths. You can save processing time if all the records are the same length and each record is completely filled with data. If the records are not completely filled with data, you waste space. Variable-length records use space more efficiently, but take longer to process.

Column Number	Allowable Values	Explanation
19	F	Indicates that all records are the same (fixed) length
	V	Indicates that records can be of different (variable) lengths

Rules

- If you specify variable-length records, VAX RPG II uses the highest value in columns 40 through 43 in the Output specification as the length for that record.
- You must specify fixed-length records for sequential files being processed as update files.

Additional Information

When a variable-length record is read, VAX RPG II fills the unused portion of the input buffer with spaces. Character fields with characters beyond the end of the record will be filled with spaces. Numeric fields with digits beyond the record will be filled with spaces. This condition will cause a run-time error, unless you use the /CHECK:BLANKS_IN_NUMERICS qualifier with the RPG command. A numeric field in packed decimal or binary format cannot extend beyond the end of the input record. If it does, unpredictable results will occur.

15.3.9 Block Length

Use columns 20 through 23 to specify the length of a block. Data is stored in physical records called blocks. A block is the smallest number of bytes VAX RPG II transfers in a physical READ or WRITE operation.

In general, by specifying a longer block length, you decrease I/O processing time because more records will be available at any given time. For example, a program that loads a single key indexed file with approximately 1700 80-byte records could result in an approximately 60% decrease in direct I/Os and a 40 to 50% decrease in elapsed time. This occurs when the block length is increased from 512 bytes (VAX RMS bucket size of 1 byte) to 4096 bytes (VAX RMS bucket size of 8). However, do not specify a block length that exceeds the Working Set Quota. To display the Working Set Quota, type the following:

```
$ SHOW WORKING_SET
```

Column Number	Allowable Values	Explanation
20-23	Blank	Uses the same entry for block length as the record length (columns 24 through 27)
	1-9999	Specifies the bucket size (in bytes) for those direct and indexed files being created on disk, or specifies the block length (in bytes) for files on tape

Rules

- For disk files, the block length you specify sets the VAX RMS bucket size parameter. VAX RPG II divides the block length you specify by 512 and rounds the result to the next highest integer, if necessary. For example, if you specify a block length of 2048 bytes, the VAX RMS bucket size is 4 bytes.
- For disk files, the minimum block length is 512 bytes.
- For output tape files, the block length you specify sets the VAX RMS block size parameter. The block length must be either (1) equal to the entry in columns 24 through 27 (record length) of the File Description specification, or (2) an integer multiple of the record length. In either case, the block length cannot be greater than the maximum record length for the device. See your System Manager for the maximum record length. Block length is not needed for an input tape. If specified, it is ignored by VAX RMS.

  To make your tape compatible with non-DIGITAL systems, use the ANSI standard block length: less than or equal to 2048 bytes.
- Right justify this entry.
- Leading zeros can be omitted.

Additional Information

For additional information on quotas, see the *VAX/VMS Guide to File Applications*.

15.3.10 Record Length

Record length specifies the length of each fixed-length record in a file, or the maximum length for variable-length records.

Column Number	Allowable Values	Explanation
24-27	1-9999	Specifies the number of characters in each record

Rules

- The record length for fixed-length records in an update file must be the same value you used to write the records.
- Right justify this entry.
- Leading zeros can be omitted.

15.3.11 Mode of Processing

Use column 28 to specify the method VAX RPG II uses to access records in a file. Your choice of processing method depends on the entries for file designation and file organization. Your choice of processing method for input and update files depends on the entries for the file type, the mode of processing, the record address type, and file organization. See Tables 15–1 through 15–3 to select the correct value.

Column Number	Allowable Values	Explanation
28	Blank	Accesses records sequentially, or accesses records sequentially by key
	L	Accesses records sequentially within limits
	R	Accesses records randomly using a relative record number or an index key, or using an ADDROUT file, or tells the program to load a direct file

Additional Information

Sequential processing reads the records in the order in which they were written. Sequential-by-key processing reads records from indexed files that are used as primary, secondary, or demand files. The key refers to

the index, which is read in ascending order. Sequential-within-limits processing reads records in one of two ways:

- Specifying a range of records to be read.
- Using the SETLL operation code in the Calculation specification to set the lowest key for the records in a demand file. The program reads records with keys equal to or higher than the key you specify.

Random processing reads records from chained files in one of the following two ways:

- For sequential or direct files, records are accessed by their relative record number. A relative record number identifies the position of a record relative to the beginning of the file.
- For indexed files, records are accessed by their index key values.

ADDROUT file processing uses the ADDROUT file generated by SORT/MERGE. The ADDROUT file contains binary record numbers that correspond to the addresses of records; therefore, the records to be read are located by their addresses.

Files on devices other than disk can be accessed only sequentially.

For input or update primary, secondary, or demand files that reside on disk, you can use the entries listed in Table 15–1.

Table 15–1: Modes of Processing for Primary, Secondary, and Demand Files

File Organization	Allowable Access Modes	Allowable Entries				
		Column:				
		16	28	31	32	
Sequential	Sequential	P	b	b	b	(b or 1-9)
		S	b	b	b	(b or 1-9)
		D	b	b	b	(b or 1-9)
		F	b	b	b	(b or 1-9)
Direct	By ADDROUT file	P	R	I	b	(b or 1-9)
		S	R	I	b	(b or 1-9)
Indexed	Sequential	P	b	b	b	
		S	b	b	b	
		D	b	b	b	
		F	b	b	b	
	By ADDROUT file	P	R	I	I	
		S	R	I	I	

Legend

P—Primary file
S—Secondary file
I—Indexed file
D—Demand file
F—Full-Procedural file
b—Blank
R—Random
L—Sequential within limits
A/P—Alphabetic or Packed keys
A/P/B—Alphabetic, Packed, or Binary keys
1-9—Additional areas

Table 15–1 (Cont.): Modes of Processing for Primary, Secondary, and Demand Files

File Organization	Allowable Access Modes	Allowable Entries			
		Column:			
		16	28	31	32
	Sequential by key	P	b	A/P/B	I
		S	b	A/P/B	I
		D	b	A/P/B	I
		F	b	A/P/B	I
	Sequential within limits	P	L	A/P	I
		S	L	A/P	I
		D	L	A/P	I
		F	L	A/P	I

Legend

P—Primary file
S—Secondary file
I—Indexed file
D—Demand file
F—Full-Procedural file
b—Blank
R—Random
L—Sequential within limits
A/P—Alphabetic or Packed keys
A/P/B—Alphabetic, Packed, or Binary keys
1-9—Additional areas

For record-address files, you can use the entries listed in Table 15–2.

Table 15–2: Modes of Processing for Record-Address Files

File Organization	Allowable Access Modes	Allowable Entries			
		Column:			
		16	28	31	32
Sequential	Sequential[1]	R	b	b	b
Direct	Sequential[2]	R	b	I	T

[1] Indicates a record-limits file

[2] Indicates an ADDROUT file

Legend

I—Indexed file
T—Table
b—Blank
R—Random

For input or update chained files, you can use the entries listed in Table 15–3.

Table 15–3: Modes of Processing for Input or Update Chained Files

File Organization	Allowable Access Modes	Allowable Entries			
		Column:			
		16	28	31	32
Sequential or Direct	Random, by relative record number	C F	R R	b b	b b
Indexed	Random, by key	C F	R R	A/P/B A/P/B	I I

Legend

I—Indexed file
C—Chained
F—Full-Procedural file
b—Blank
R—Random
A/P/B—Alphabetic, Packed, or Binary keys

In the following example, the program accesses an indexed file as a demand file and three update chained files.

```
                        Mode (LR)
                        |Key length
                Type (IOUDC) || Record address type (APIB)          Addtn(AU)
                |Des(PSRCTDF)|| |Organization (IT,1-9)               |Expand
                ||EOF (E)    || ||Overflow indicator  Continue (K) ||Shr(SR)
                |||Seq (AD)  || ||| Key location      |Opt   Entry |||Rewnd
 File           ||||Fmt (FV) || ||| |   Extension (EL)||     |     |||  |
 name           |||||Blk Rec || ||| |   |Device Symb  Tape   Core  |||  |File
  |             ||||||len len || ||| |   |code   dev   label  index |||  |cond
FI              ||||||   |   || ||| |   ||     |     |      |     |||  ||
  0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890
     **       ******---*---**-*** *---**      *     *......*-----***.** ..
     FINDEX1  ID  F      18  5AI     5 DISK
     FCHAIN1  UC  F      24R           DISK
     FINDEX2  UC  F      24R 3AI     1 DISK
     FCHAIN2  UC  F      24R           DISK
```

ZK-4472-85

15.3.12 Key Length

Use columns 29 and 30 to specify key length. Key length indicates the length in bytes of one of the following:

- The index keys in an indexed file
- The index keys in a record-limits file
- The addresses in an ADDROUT file

Column Number	Allowable Values	Explanation
29,30	Blank	Indicates a sequential, direct, or display file
	1-99	Specifies the length of the record key for an indexed file or a record-address file

Rules

- You must use a value of 6 for the length of the record addresses in an ADDROUT file.
- Right justify this entry.
- Leading zeros can be omitted.

In the following example, the program reads a chained indexed file. The length of the record key is three bytes.

```
                          Mode (LR)
                          |Key length
                Type (IOUDC) || Record address type (APIB)          Addtn(AU)
                |Des(PSRCTDF)|| |Organization (IT,1-9)              |Expand
                ||EOF (E)    || ||Overflow indicator  Continue (K) ||Shr(SR)
                |||Seq (AD)  || ||| Key location      |Opt   Entry |||Rewnd
     File       ||||Fmt (FV) || ||| |   Extension (EL)||     |     |||  |
     name       |||||Blk Rec || ||| |   |Device Symb  Tape   Core  |||  |File
      |         ||||||len len || ||| |   |code   dev    label  index |||  |cond
    F|         ||||||   |   || ||| |   ||     |      |      |     |||  ||
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        ******---*---**-*** *---**      *     *......*-----***.** ..
     FINDEX    IC  F      24R 3AI     1 DISK
```

ZK-4473-85

15.3.13 Record Address Type

Use column 31 to specify record address type. Record address type helps define the mode of processing. See Table 15–2 to select the correct value.

Column Number	Allowable Values	Explanation
31	Blank	Uses relative record numbers to process sequential and direct chained files, or reads records sequentially from an input or update file, or creates or adds records to a sequential output file
	A	Processes or loads indexed files according to the record keys in alphanumeric format
	P	Processes or loads indexed files according to record keys in packed format
	B	Processes or loads indexed files according to record keys in binary (word or longword) format
	I	Processes the file according to an ADDROUT file, or identifies an ADDROUT file

15.3.14 File Organization or Additional Input/Output Area

File organization specifies how records are arranged in a file. Additional I/O allows you to specify the number of I/O areas. Both attributes work in conjunction with the mode of processing. See Tables 15–2 and 15–3 to select the correct value.

Column Number	Allowable Values	Explanation
32	Blank	Indicates a sequential or direct file, using one I/O area
	I	Indicates an indexed file
	T	Indicates an ADDROUT file
	1-9	Indicates the number of I/O areas for a sequential or direct file

Additional Information

For sequential files, VAX RPG II adds 1 to the Additional I/O area value you specify in column 32. VAX RPG II uses the value for Additional I/O to set the VAX RMS multiblock count in the Record Access Block (RAB).

See the *Guide to VAX/VMS File Applications* for information about multiblock count and for optimizing file performance.

15.3.15 Overflow Indicators

Use columns 33 and 34 to specify an overflow indicator. You can use an overflow indicator to specify a page break before or after certain lines are printed. See Chapter 9 for information on overflow indicators.

Column Number	Allowable Values	Explanation
33,34	Blank	Specifies no overflow indicator
	OA-OG, OV	Specifies an overflow indicator to condition output lines when an overflow occurs

Rules

- You can use an overflow indicator to condition only printer output files.
- You can assign only one overflow indicator to a file. If you have more than one printer output file and you want to use an overflow indicator to condition each file, you must specify a unique overflow indicator for each file.

15.3.16 Key Location

Use columns 35 through 38 to specify key location. Each record in an indexed file has a key field that VAX RMS uses to locate records. This key field can be anywhere in the record, but must be in the same location for each record in the file. Key location specifies where to find the key field in the record.

Column Number	Allowable Values	Explanation
35-38	Blank	Indicates that the file is not indexed
	1-9999	Specifies the location of the key field

Rules

- Right justify this entry.
- Leading zeros can be omitted.

15.3.17 Extension Code

Use column 39 to specify the extension code that causes VAX RPG II to read either the Extension or the Line Counter specification for more information about the file. You must complete an Extension specification for tables, arrays, and record-address files. You can complete a Line Counter specification for a printer output file. The Line Counter specification specifies the length of the printed page and defines the overflow line number.

Column Number	Allowable Values	Explanation
39	Blank	Specifies no Extension or Line Counter specification
	E	Causes VAX RPG II to read the Extension specification for the file
	L	Causes VAX RPG II to read the Line Counter specification for the file

15.3.18 Device Code

Use columns 40 through 46 to specify the device code that indicates on what type of device the file is stored.

Column Number	Allowable Values	Explanation
40-46	Blank	Specifies the default disk device.
	DISKXXX	Specifies a disk device where X is any character. Disk can be specified for sequential files but is required for indexed and direct files. Disk is the default device.
	TAPEXXX	Specifies a tape device for sequential files only. X can be any character.
	PRINTXX	Specifies a print device for an output file. X can be any character.
	TTYXXXX	Specifies a terminal device for a display file or a sequentially processed input or update file. X can be any character.
	READXXX	Specifies a card reader for sequentially processed input files. X can be any character. To use a card reader, you must specify I in column 15 and P, S, T, or D in column 16. Also, leave columns 28 through 38 blank.

Column Number	Allowable Values	Explanation
	WORKSTN	Specifies the mechanism by which many VAX FMS form capabilities can be accessed from a VAX RPG II program. You must include a C for the file type when you specify WORKSTN as the file device. There can only be one WORKSTN file in a program. See Chapter 6 for additional information.

Rules

Left justify this entry.

Additional Information

If you specify a device name other than one of the allowable values, VAX RPG II accepts it, but issues a warning message at compile time. VAX RPG II assumes a disk device, unless you specify D in column 15, in which case VAX RPG II assumes a terminal (TTY) device.

For additional information on tape and disk operations, see the *Guide to VAX/VMS Disk and Magnetic Tape Operations.*

15.3.19 Symbolic Device

Use columns 47 through 52 to specify the symbolic device. The symbolic device can be a logical name for any device. VAX RPG II uses the symbolic device as the file name string. VAX RMS uses the file name string and the default file name string (the file name that appears in columns 7 through 14) as the default name of a file being processed for input or output operations. See Chapter 8 for information on how VAX RPG II uses the symbolic device field (columns 47 through 52) and the file name field (columns 7 through 14) to generate the VAX/VMS file specification.

Column Number	Allowable Values	Explanation
47-52	Any character	Represents the symbolic device

The symbolic device name can contain up to six characters. It must not be specified for display or WORKSTN files.

15.3.20 Tape Label

Use column 53 to identify the label for a magnetic tape.

Column Number	Allowable Values	Explanation
53	Blank	Indicates that the magnetic tape has a standard VAX/VMS ANSI label
	S	Indicates that the magnetic tape has a standard VAX/VMS ANSI label

VAX/VMS can process only magnetic tapes with VAX/VMS ANSI labels.

15.3.21 F Specification Continuation Lines

F specification continuation lines can optionally follow a WORKSTN file specification. You can use continuation lines to indicate:

- A VAX FMS forms library
- A WORKSTN file information data structure
- An offset of forms on the screen

Column Number	Allowable Values	Explanation
7-52	Blank	These columns must be blank.
53-59	FMTS	A VAX FMS library option.
	INFDS	A WORKSTN file information data structure option.
	SLN	Offset of forms on screen option.
60-65	Field name or data structure name	Specifies the variable containing the number of lines to offset the forms on the screen for the SLN option. Or, specifies the WORKSTN file information data structure name.
60-67	VAX FMS form library name	Specifies the VAX FMS forms library name for the FMTS option.

15.3.22 Core Index

Use columns 60 through 65 to set the VAX RMS multibuffer count in the Record Access Block (RAB). See the *Guide to VAX/VMS File Applications* manual for information about multibuffer count and for optimizing file performance.

Column Number	Allowable Values	Explanation
60-65	Blank	Specifies that the file is not indexed or that an indexed file has no core index
	1-9999	Specifies the number of bytes to reserve for the core index

Rules

- VAX RPG II divides the core index value by 512 and rounds the value to the next highest integer, if necessary. For example, if the core index is 513, the VAX RMS multibuffer count is 2.

If the operation results in an integer that is greater than 127, VAX RPG II uses 127 as the VAX RMS multibuffer count.

- Right justify this entry.
- Leading zeros can be omitted.

15.3.23 File Addition and Unordered Output

Use column 66 to specify file addition and unordered output. File addition and unordered output determine how new records are added to a file. You can add records to sequential, direct, and indexed files that reside on disk. On tape, you must go to the logical end of the tape before adding records to a file; otherwise, new records would overwrite existing records.

Column Number	Allowable Values
66	Blank, A, U

For output files, you can choose one of the following entries.

Entry	Explanation
Blank	Creates an indexed file and adds records by primary key, or creates a sequential or direct file.
A	Adds records to an existing indexed or direct file or to the end of an existing sequential file. When you use this option, you must also specify ADD in columns 16 through 18 of the Output specification.
U	Creates an indexed file and adds records in an unordered sequence.

For input files, you can choose one of the following entries.

Entry	Explanation
Blank	Reads records from a file without adding new records or changing existing records.
A	Reads records from an indexed or direct file and allows you to add new records. When you use this option, you must also specify ADD in columns 16 through 18 of the Output specification.

For update files, you can choose one of the following entries.

Entry	Explanation
Blank	Allows you to update the records in a file.
A	Allows you to update the records in, and add records to, an indexed or direct file. When you use this option, you must also specify ADD in columns 16 through 18 of the Output specification.

You cannot add records to an indexed file that is being processed by a record-address file.

15.3.24 Expansion Factor

When records are added to indexed files, they are placed in buckets. (Buckets hold the contents of records.) If you attempt to randomly add a record to a full bucket, VAX RMS causes the bucket to split. VAX RMS tries to keep half of the records in the original bucket and moves the other records to a newly created bucket. Each split record leaves behind a pointer to the new bucket. When the system searches for one of the records in the newly created bucket, it must first go to the bucket where the record previously resided, read the pointer, and then move to the bucket indicated by the pointer. This pointer manipulation overhead takes time and wastes disk space.

To prevent bucket splitting and to improve search efficiency, use an Expansion factor when creating an indexed file to reserve bucket space for the records you write to an indexed file. Also, specify a bucket size that is a multiple of the disk cluster size. To show the disk cluster size, type the following:

```
$ SHOW DEVICE device/FULL
```

Use column 67 to specify the expansion factor.

Column Number	Allowable Values	Explanation
67	Blank or 0	Completely fills a bucket
	1 (minimum)	Sets indexed bucket fill size to 50% and sets data fill size to 100%
	2 (average)	Sets indexed bucket fill size to 50% and sets data fill size to 75%
	3 (above average)	Sets indexed bucket fill size to 50% and sets data fill size to 60%
	4 (maximum)	Sets indexed bucket fill size to 50% and sets data fill size to 50%

Additional Information

If the records you want to add are distributed unevenly by their key values, then VAX RMS must split the buckets. In this case, use an expansion factor of zero. Records with key values that are close in sequence and records added to the end of the file cause VAX RMS to split the buckets anyway. For these kinds of records, use an expansion factor of zero.

For output or update indexed files that are being created, VAX RMS uses the expansion factor to set the data bucket fill size and indexed bucket fill size in the key Extended Attribute Block (XAB). VAX RPG II multiplies the bucket size value by 512 and adjusts the result based on the percentages listed.

Table 15–4 shows how the values for expansion factor and block length set the values for the following VAX RMS parameters:

- FAB$B_BKS (bucket size)
- XAB$W_IFL (indexed bucket fill size)
- XAB$W_DFL (data bucket fill size)

See the *VAX Record Management Services Reference Manual* for information on these parameters. See the *Guide to VAX/VMS File Applications* for information on indexed bucket fill size (index_fill) and data bucket fill size (data_fill).

Table 15–4: Expansion Factor and Block Length Values

File Expansion	Block Length	FAB$B_BKS	XAB$W_IFL	XAB$W_DFL
1 (minimal)	1536	3	768	1536
2 (average)	2048	4	1024	1536
3 (above average)	1024	2	512	614
3 (above average)	2048	4	1024	1228
4 (maximum)	2000	4	1024	1024

15.3.25 File Sharing

Use column 68 to specify the file sharing requirements of the file. File sharing allows more than one program to access the records in a file at the same time. If more than one program tries to access the same record, the first program that accessed the record will be allowed to change it and then one of the following options will take effect:

- S option—the record will be locked, preventing access by other programs until the first program is finished with the record
- R option—the record will be locked, preventing update access by other programs, but will not be locked from programs attempting to read the record

However, on a CHAIN operation code, you can specify an indicator to be set on when a record is locked, allowing the program to proceed while the record is still locked. See Section 16.7.1 for information on CHAIN indicators for locked records.

Column Number	Allowable Values	Explanation
68	Blank	Uses VAX RMS default file sharing
	S	Specifies file sharing
	R	Specifies file sharing with the lock for writing option

Rules

- Column 68 must be blank for a display file (D in column 15 of the File Description specification) and for an ADDROUT file (T in column 32 of the File Description specification).
- Specifying S or R in column 68 is valid for a sequential file only if the sequential file has fixed-format records (F in column 19 of the File Description specification) and with a record length of 512.

Additional Information

Table 15–5 shows file sharing that is inherently specified as a result of the combination of the entries in columns 15, 66, and 68 of the File Description specification. The File Description specification that specifies these entries is assumed to be the first to open the file.

Table 15–5: File Sharing

Columns (b = blank)			
15	66	68	
File Type	File Addition	Share	Explanation
I	b	b	Any number of programs with the same entries in these three columns can read the file simultaneously. Any program with a different entry in file type or File Addition for this file will receive a file-locked error.
I O U	A b,A,U b,A	b b, b	No other program is allowed simultaneous access to the file. Any other program will receive a file-locked error.
I, O, U	b,A,U	S, R	Any other program with an S or R in Share can access the file simultaneously, unless the file is for output and the file does not specify A for File Addition, in which case a new version of the file is created. Any other program with a blank in Share will receive a file-locked error.

VAX RPG II does not set the SHR field of the File Access Block (FAB) for the file when Share is left blank. When you specify S in column 68,

VAX RPG II sets the SHR field to allow GET, PUT, DEL, and UPD access. When you specify R in column 68, VAX RPG II also sets the RLK option. See the *Guide to VAX/VMS File Applications* for more information on file sharing.

15.3.26 Tape Rewind

Use column 70 to specify tape rewind that positions a tape according to the current conditions.

Column Number	Allowable Values	Explanation
70	Blank	Indicates either that the file does not reside on tape, or, if the file does reside on tape, that the tape will rewind when the file is opened and closed
	U or R	Rewinds the tape when the file is opened and when the file is closed
	N	Does not rewind the tape
	K	Rewinds the tape when the file is opened
	L	Rewinds the tape when the file is closed

15.3.27 File Condition

Use columns 71 and 72 to specify the file condition. File condition associates an external indicator with a file. External indicators control file access at run time. When you condition the file with an external indicator, VAX RPG II opens the file only when the external indicator is set on. You can use external indicators to condition primary and secondary input and update files, record-address files, and output files. You can condition a record-address file by using an external indicator only if the following conditions are met:

- The record-address file is associated with a primary or secondary input or update file.
- The same indicator (or no indicator) is used to condition the associated file.

See Chapter 7 for more information on external indicators.

Column Number	Allowable Values	Explanation
71,72	Blank	Indicates no external indicator for this file
	U1-U8	Names the external indicator that controls file access at run time

When you condition a file with an external indicator, use the same indicator to condition calculations and output operations for the same file.

15.3.28 Example

In the following example:

- Line 1020 describes a primary input file with fixed-record format. Each record is 96 bytes long. The file CICWMS resides on disk with a symbolic device of XX1.
- Line 1030 describes a demand file for input with fixed-record format. Each record is 96 bytes long. The file CARD58 resides on disk with a symbolic device of XX2. The external indicator U1 must be set on if the file is to be opened.
- Line 1040 describes a demand file for update with fixed-record format. Each record is 190 bytes long. The file ICM resides on disk with a symbolic device of XX4.
- Line 1110 describes an output file with fixed-length records. Each record is 132 bytes long. The output file CONTRL will be written to the printer whose symbolic device is X11. The overflow indicator is specified to condition output lines when an overflow occurs.

```
                                  Mode (LR)
                                  |Key length
                Type (IOUDC)      || Record address type (APIB)          Addtn(AU)
                |Des(PSRCTDF)     || |Organization (IT,1-9)              |Expand
                ||EOF (E)         || ||Overflow indicator  Continue (K) ||Shr(SR)
                |||Seq (AD)       || ||| Key location       |Opt   Entry ||| Rewnd
      File      ||||Fmt (FV)      || ||| |   Extension (EL)||     |     ||| |
      name      |||||Blk Rec      || ||| |   |Device Symb   Tape   Core  ||| |File
      |         ||||||len len     || ||| |   |code  dev    label  index ||| |cond
     Fl         ||||||   |       || ||| |   ||      |      |      |     ||| ||
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         ******---*---**-*** *---**      *     *......*-----***.** ..
01020FCICWMS     IP   F      96            DISK   XX1
01030FCARD58     ID   F      96            DISK   XX2                      U1
01040FICM        UD   F     190            DISK   XX4
01110FCONTRL     O    F     132      OF    PRINTR X11
```

ZK-4474-85

Extension Specification (E)

15.4 Description

The Extension specification allows you to give VAX RPG II additional information about record-address files, tables, and arrays. See Chapter 8 for information about record-address files. See Chapter 10 for information about tables. See Chapter 11 for information about arrays.

In general, use the Extension specification to describe the following information about a table or array:

- Name of the table or array
- Number of entries per record
- Number of entries per table or array
- Length of each table entry or array element
- Format of numeric data
- Decimal position
- Sequence of entries
- Related tables or related arrays in alternating format

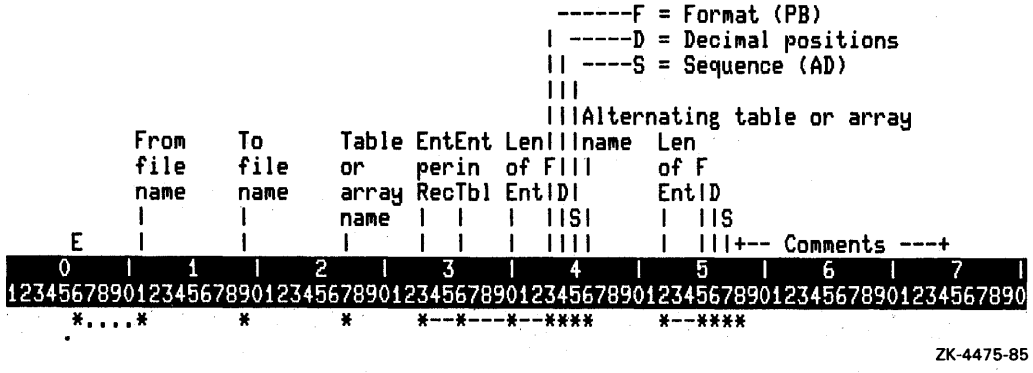
If your program uses a record-address file, complete columns 11 through 26 to provide the following information:

- Name of the record-address file
- Data file associated with the record-address file

If your program uses tables or arrays, the time at which you load the tables or arrays determines the columns you must complete. For compile-time tables and arrays, you must complete columns 19 through 57; for preexecution-time tables and arrays, complete columns 11 through 57; and for execution-time arrays, complete columns 27 through 32 and columns 36 through 45.

15.4.1 Extension Specification Format

The format of the Extension specification is as follows:



ZK-4475-85

15.4.2 Specification Type

Use column 6 to identify the type of specification for every program line.

Column Number	Allowable Values	Explanation
6	E	Indicates that this program line is an Extension specification

15.4.3 From File Name

Use columns 11 through 18 to specify the from file name that identifies the name of the record-address file or the input file used to load a preexecution-time table or array.

Column Number	Allowable Values	Explanation
11-18	Blank	Loads the table or array named in columns 27 through 32 (table or array name) at compile time, if columns 33 through 35 (entry per record) are completed. Loads the array at execution time if specified by the Input and/or Calculation specification and if columns 33 through 35 are left blank.
	Name	Names a record-address file, if specified. Otherwise, names the table or array input file. VAX RPG II uses this file to load the table or array at preexecution time.

Rules

- This file name must be the same file name you used in the File Description specification.
- Left justify this entry.

15.4.4 To File Name

Use columns 19 through 26 to specify the to file name in connection with your entry in columns 11 through 18 (from file name). If you name a record-address file in columns 11 through 18, specify the name of the input or update file it processes in columns 19 through 26.

Column Number	Allowable Values	Explanation
19-26	Blank	Does not write the table or array file at the end of the program.
	File name	Identifies the data file associated with the record-address file, if you use a record-address file. If you do not use a record-address file, this entry names the output file that receives the data from the table or array at the end of the program.

Rules

- This file name must be the same file name you used in the File Description specification.
- If you want to write a table or an array to an output file at the end of the program, enter the file name in columns 19 through 26.
- You cannot write an execution-time array to an output file.
- Left justify this entry.

15.4.5 Table or Array Name

Use columns 27 through 32 to name the table or array you want to use.

Column Number	Allowable Values	Explanation
27-32	Blank	Indicates that the file named in columns 11 through 18 (from file name) is a record-address file
	Name	Identifies the name of the table or array

Rules

- A table name can be any string of three to six alphanumeric characters, beginning with TAB. Table names cannot contain embedded blanks.
- An array name can be any string of one to six alphanumeric characters, beginning with an alphabetic character. Array names cannot begin with TAB and cannot contain embedded blanks.
- If you use tables or arrays in alternating format, this entry describes the name of the main table or array. Identify the alternate table or array name in columns 46 through 51.
- Left justify this entry.

15.4.6 Number of Entries in a Record

Use columns 33 through 35 to specify the number of entries in a table or array input record. Complete this entry for compile-time and preexecution-time tables and arrays.

Column Number	Allowable Values	Explanation
33-35	Blank	Indicates a record-address file or an execution-time array
	1-999	Specifies the number of entries in a table or array input record

Rules

- All records except the last must have the same number of entries. The last record can have fewer entries to accommodate a number of entries that is not an even multiple of the defined number of entries in the record.
- The first entry must begin in the first position of the record.
- Leave no spaces between entries in a record.
- Entries cannot span two records.
- If you use tables or arrays in alternating format, each record must contain a corresponding entry. The entries from the main table or array and the corresponding entries from an alternate table or array are treated as one entry.
- Right justify this entry.
- Leading zeros can be omitted.

15.4.7 Number of Entries in a Table or Array

Use columns 36 through 39 to specify the number of entries in a table or array and in an alternate table or array, if an alternate table or array is used.

Column Number	Allowable Values	Explanation
36-39	Blank	Indicates that the file named in columns 11 through 18 (from file name) is a record-address file
	1-9999	Specifies the number of entries in a table or array

Rules

- If a compile-time or preexecution-time table or array is not completely full, VAX RPG II fills the unused entries with blanks for alphanumeric data or zeros for numeric data. If you specify an entry in column 45 (sequence) of the Extension specification, preexecution-time and compile-time tables and arrays must be full (VAX RPG II does not fill the short entries).
- Right justify this entry.
- Leading zeros can be omitted.

15.4.8 Length of Entry

Use columns 40 through 42 to specify the length of entry that defines the number of character (alphanumeric or numeric) positions in each table or array entry.

Column Number	Allowable Values	Explanation
40-42	Blank	Indicates that the file named in columns 11 through 18 (from file name) is a record-address file
	1-999	Specifies the number of character positions (both alphanumeric and numeric) in each table or array entry

Rules

- For an alphanumeric entry, the maximum number of characters is 999.
- For a numeric entry, the maximum number of digits is 15.
- For numeric data, the maximum number of digits in binary format is 9.
- For compile-time arrays, the maximum length of an entry is 96 characters, because this is the largest record that can be entered in the source program.
- When you use table and arrays in alternating format, this entry specifies the length of the entry in the main table or array.
- Because all entries in a table or array must be the same length, fill unused alphanumeric character positions with blanks and fill numeric entries with zeros.
- Right justify this entry.
- Leading zeros can be omitted.

15.4.9 Format

Use column 43 to specify how numeric data is stored. Data can be stored in one of the following three formats:

- Overpunched decimal
- Packed decimal
- Binary

Select a format based on the storage space available and the frequency of use. See Chapter 14 for more information on data formats.

Column Number	Allowable Values	Explanation
43	Blank	Specifies that numeric data is in overpunched decimal format, or that the table or array contains alphanumeric data. If you do not specify a table or an array, a blank indicates that the file named in columns 11 through 18 (from file name) is a record-address file.
	P	Specifies that numeric data is in packed decimal format. This format is valid only for preexecution-time tables or arrays.
	B	Specifies that numeric data is in binary format. This format is valid only for preexecution-time tables or arrays.

15.4.10 Decimal Positions

Use column 44 to specify the number of positions to the right of the decimal point for numeric data in a table or array.

Column Number	Allowable Values	Explanation
44	Blank	Specifies a record-address file or indicates that the table or array, if used, contains alphanumeric data
	0-9	Specifies the number of positions to the right of the decimal point for numeric data in a table or array

You must specify zero for numeric data with no decimal positions.

15.4.11 Sequence

Use column 45 to specify the sequence that defines the order of entries in a table or array. VAX RPG II checks each entry for the order you specify.

Column Number	Allowable Values	Explanation
45	Blank	Specifies a record-address file, or indicates that the entries in a table or an array are unordered
	A	Specifies that the entries in a table or array are in ascending order
	D	Specifies that the entries in a table or array are in descending order

Rules

- Consecutive entries that are equal in value are considered to be in sequence.
- When you use tables or arrays in alternating format, this entry specifies the sequence of the main table or array.
- When you specify a sequence for a compile-time or preexecution-time table or array, VAX RPG II checks the sequence of the entries in a table or an array. If an entry in a compile-time table or array is out of sequence, VAX RPG II reports a fatal error during compilation. If a preexecution-time table or array is out of sequence, a run-time error occurs.
- You must specify a sequence if you use an indicator to test for a HIGH or LOW condition in a LOKUP operation associated with the table or array. See Chapter 16 for information on LOKUP.
- You can specify a sequence for an execution-time array, but VAX RPG II does not check the sequence. However, if the execution-time array is not in correct sequence and you specify a LOKUP operation with a HIGH or LOW condition, unpredictable results will occur.

15.4.12 Alternate Table or Array

Use columns 46 through 57 to define the name, entry length, data format, number of decimal positions, and sequence for an alternate table or array. If you specify a table, you must use another table as its alternate. If you specify an array, you must use another array as its alternate. The same rules for columns 27 through 45 apply to the entries in columns 46 through 57.

15.4.13 Comments

Use columns 58 through 74 to document the program line.

Column Number	Allowable Values	Explanation
58-74	Any character	Documents the program line

15.4.14 Example

The following example specifies these operations:

- A preexecution-time array to be loaded from the file TABLEF at the start of program execution (line 40)
- A compile-time table with an alternate table (line 50)
- A compile-time array (line 60)
- An execution-time array (line 70)

Extension Specification (E)

```
                                                 ------F = Format (PB)
                                                | -----D = Decimal positions
                                                || ----S = Sequence (AD)
                                                |||
                                                |||Alternating table or array
          From      To       Table EntEnt Len|||name  Len
          file      file     or    perin  of F|||     of F
          name      name     array RecTbl Ent|D||     Ent|D
          |         |        name  |   |   | ||S|     |  ||S
     E    |         |        |     |   |   | ||||     |  |||+-- Comments ---+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     *....*        *        *      *--*---*--****     *--****
  40E    TABLE1             ARRAY1  4    8   5 0
  50E                       TABLE2  4    8   1  TABLE3   3 0
  60E                       ARRAY2 30  100   1
  70E                       ARRAY3     30   1
```

ZK-4476-85

Line Counter Specification (L)

15.5 Description

The default length for a page of printer output is 66 lines; the default overflow line is line 60. When the printer reaches the overflow line, VAX RPG II sets the overflow indicator on.

The Line Counter specification allows you to alter the default page format of a printer output file. You can use this specification to change both the number of lines on a page and the overflow line.

15.5.1 Line Counter Specification Format

The format of the Line Counter specification is as follows:

```
                Form length (1-112)
      File      |  FL (if Form length used)
      name      |  | Overflow line number (1-112)
      |         |  | |  OL (if Overflow line used)
     L|         |  | |  |
   0  |  1   |  2   |  3   |  4   |  5   |  6   |  7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         *--* *--* ..........................................
```

ZK-4477-85

15.5.2 Specification Type

Use column 6 to identify the type of specification for every program line.

Column Number	Allowable Values	Explanation
6	L	Indicates that this program line is a Line Counter specification

15.5.3 File Name

Use columns 7 through 14 to name the output file.

Column Number	Allowable Values	Explanation
7-14	File name	Identifies the name of the output file

Rules

- The output file must be described on the File Description specification with PRINTER in columns 40 through 46 (device code) and L in column 39 (extension).
- Left justify this entry.

See Chapter 14 for information on naming files.

15.5.4 Form Length

Use columns 15 through 17 to define the number of lines printed on a page. When the printer reaches the last specified line, it skips to the next page and resumes printing.

Column Number	Allowable Values	Explanation
15-17	1-112	Defines the maximum number of lines that can be printed on a page

Rules

- This entry must be a numeric value.
- Right justify this entry.
- Leading zeros can be omitted.

15.5.5 FL

If you specify an entry in columns 15 through 17 (form length), you must enter FL in columns 18 and 19. This entry specifies that columns 15 through 17 define the form length.

Column Number	Allowable Values	Explanation
18,19	FL	Causes VAX RPG II to use the form length defined in columns 15 through 17

15.5.6 Overflow Line Number

Use columns 20 through 22 to specify the overflow line number. When the page reaches the overflow line, VAX RPG II sets the overflow indicator on.

Column Number	Allowable Values	Explanation
20-22	1-112	Specifies the overflow line number

Rules

- This entry must be equal to or less than the entry in columns 15 through 17 (form length).
- This entry must be a numeric value.
- Right justify this entry.
- Leading zeros can be omitted.

15.5.7 OL

If you specify an overflow line number in columns 20 through 22 (overflow line number), you must enter OL in columns 23 and 24. This entry specifies that columns 20 through 22 define the overflow line number.

Column Number	Allowable Values	Explanation
23,24	OL	Causes VAX RPG II to use the overflow line number defined in columns 20 through 22.

15.5.8 Example

In the following example, the form length is 100 lines and the overflow line number is line 96:

```
                Form length (1-112)
      File      |  FL (if Form length used)
      name      |  | Overflow line number (1-112)
      |         |  | |  OL (if Overflow line used)
     L|         |  | |  |
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         *--* *--* ..............................................
     LINPUT     100FL 960L
```

ZK-4478-85

Input Specification (I)

15.6 Description

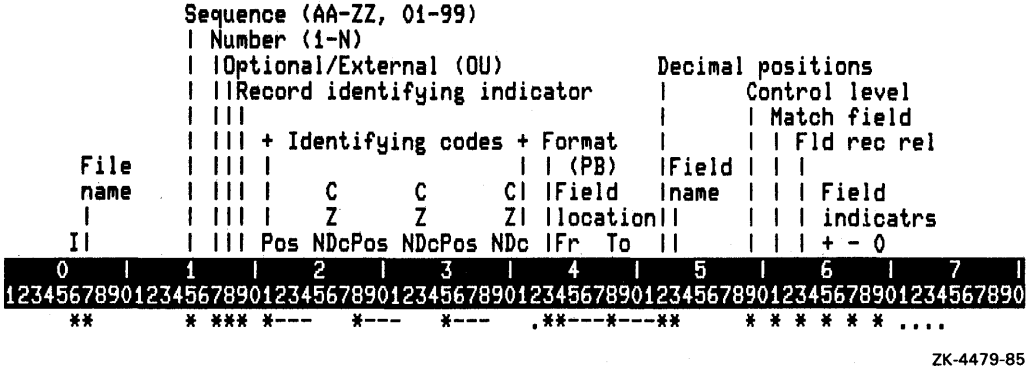
The Input specification describes the records in input and update files. Each record is further divided into fields. Columns 7 through 42 describe the file and its records. Columns 43 through 74 describe the fields in each record.

The Input specification also describes data structure statements and data structure subfields. See Section 15.6.4 for information on data structures.

You must use an Input specification to describe each input or update file except for table input files and record-address files.

15.6.1 Input Specification Format

The format of the Input specification is as follows:



ZK-4479-85

15.6.2 Specification Type

Use column 6 to identify the type of specification for every program line.

Column Number	Allowable Values	Explanation
6	I	Indicates that this program line is an Input specification

15.6.3 File Name

Use columns 7 through 14 to name the input or update file.

Column Number	Allowable Values	Explanation
7-14	File name	Identifies the name of the input or update file

Rules

- Use the same name you specified in the File Description specification.
- If this column is blank, VAX RPG II assumes that the information in this program line describes a field or record from the file named last.
- Describe all the records and fields for one file before describing another file.
- Left justify this entry.

In the following example, EMPLOYEE is the name of the input file, and the fields NAME and ADDRES, each containing 20 characters, belong to each record in the file:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                                |      | Match field
                | ||| + Identifying codes + Format   |      | | Fld rec rel
      File      | ||| |                   | | (PB)   |Field | | |
      name      | ||| |    C      C      C| |Field   |name  | | | Field
       |        | ||| |    Z      Z      Z| |location|      | | | indicatrs
     II         | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IEMPLOYEEAA
     I                                          1  20 NAME
     I                                         21  41 ADDRES
```

ZK-4480-85

15.6.4 Data Structures

Data structures consist of a data structure statement and zero or more data structure subfields. You must describe data structures on Input specifications. Data structures follow all Input specifications (including all modifier specifications) for records.

You can use data structures to do the following:

- Define one area of storage more than one way
- Define subfields within a field or subfield
- Reorganize fields in an input record
- Select the internal numeric data type for fields
- Communicate information between programs with a local data area

15.6.4.1 Data Structure Statement

The format of the data structure statement is as follows:

Column Number	Allowable Values	Explanation
7-12	Data structure name	Identifies the data structure. The data structure name, which is not required, can be a field defined on Input specifications or Calculation specifications or defined nowhere else in the program.
18	U	Optional. Identifies this data structure as the local data area.
19-20	DS	Specifies a data structure.
48-51	1-9999	Columns 48 through 51 may optionally contain the data structure length.

Rules

- The data structure name can appear on only one data structure specification, cannot be a look-ahead field, and can be specified anywhere a character field is allowed.
- The length of the data structure is one of the following:
  - The length specified in the Input specification, if the data structure name is an input field
  - The length specified in columns 48 through 51 of the data structure statement
  - The highest to position (end position) of a subfield within a data structure, if the data structure name is not an input field
- The length computed by the to position must be less than or equal to the length specified in the Input specification or the data structure statement.
- Any length on a Calculation specification must match the largest value specified in the Input specification or the data structure statement.
- Because it is possible to specify the length of a single data structure in all of the preceding ways in a single VAX RPG II program, a compiler diagnostic will be given for any conflicts. This will not occur if the length in columns 48 through 51 exceeds the highest to position for any subfield in the data structure.

15.6.4.2 Data Structure Subfields

Data structure subfields are specified in columns 43 through 58. They are defined the same as for any other input field specification. See Sections 15.6.11 through 15.6.14 for those field specification requirements.

The field location's start and end positions are relative to the beginning of the data structure, not to the beginning of the data record.

Rules

- All columns except columns 43 through 58 must be left blank.
- The subfield name can be the same as a field defined on an Input specification or a Calculation specification.
- Subfields can be used as factor 1, factor 2, or the result field of a Calculation specification or as output fields.
- The same subfield name cannot be used in more than one data structure.
- A data structure name cannot be used as a subfield name in another data structure.
- Numeric subfields must contain numeric data when used in CHAIN, LOKUP, COMP, editing, or arithmetic operations.
- If arrays are specified as subfields, the length specified must equal the amount of storage required to store the entire array.
- A data structure subfield cannot be an indicator (*IN field) or a UDATE field.
- Overlapping subfields cannot be used in the same calculation in such a way that the result field overlaps either factor 1 or factor 2. If either factor 1, factor 2, or the result field references a subfield in a data structure that is an entire array or an array with a variable index, then that array is used to determine whether overlap exists. The same array name can be referenced in the appropriate factors of a Calculation specification without violating the overlap rule.
- Any subfield previously defined in an input record must be the same in length (in digits) and in decimal positions. If the numeric data type is different from what was specified in an input record, the length (in digits) must still be the same as previously defined.
- Any subfield defined more than once in the same data structure must be defined with the same data type and start position, the same length, and the same decimal positions, in the data structure.

- Neither data structures nor data structure subfields can be individual array elements.
- All entries for a data structure statement and its data structure subfields must appear together; they cannot be mixed with entries for other data structures.
- A data structure statement and a data structure subfield cannot have the same name.

See Section 15.6.15 for examples of using data structures.

15.6.4.3 Local Data Area

The VAX RPG II local data area is a data structure of up to 512 bytes used to communicate information from one VAX RPG II program to another. In addition, the VAX RPG II local data area can be manipulated (read or written) at DCL command level or from a program written in another language.

To specify a local data area, a data structure must have a U in column 18 on the Input specifications. The data structure does not need a name. Only the first 512 bytes of the data structure are loaded at program start and written at program exit. Only one data structure may have a U specified in column 18.

The VAX RPG II local data area is implemented with VAX/VMS DCL symbols (see the *VAX/VMS DCL Dictionary* for examples of manipulating DCL symbols). The following four symbols correspond to the indicated bytes within a data structure which has a U specified in column 18:

RPG$LDA1	1-128
RPG$LDA2	129-256
RPG$LDA3	257-384
RPG$LDA4	385-512

15.6.5 Sequence

Use columns 15 and 16 to specify the sequence that defines the ordering sequence of the record types in a file (for example, distinguishing employee name records from employee badge number records). VAX RPG II does not order records according to sequence; it checks the sequence of records in the input or update file instead.

Column Number	Allowable Values	Explanation
15,16	Any two alphabetic characters	Performs no sequence checking for this record. You can use any two letters from AA through ZZ, for example, BB, ZA, or DE. You must specify an alphabetic sequence for chained and demand files and look-ahead fields.
	Blanks	Specifies no sequence checking for this record.
	Any two-digit number	Assigns a sequence number to a record. You can use any two numbers from 01 to 99; however, you must use sequence codes in ascending order, beginning with 01.

VAX RPG II does not require that all Input specifications in alphabetic sequence appear before those Input specifications in numeric sequence.

15.6.6 Number

If you assigned a numeric sequence code in columns 15 and 16, use column 17 to indicate the number of records in a record type.

Column Number	Allowable Values	Explanation
17	1	Specifies that there is only one record of this type
	N	Specifies that there can be more than one record of this type

Leave this column blank if you specified an alphabetic sequence in columns 15 and 16.

15.6.7 Option

If you assigned a numeric sequence code in columns 15 and 16, you can use column 18 to specify whether a record of that type must be present to continue processing records.

Column Number	Allowable Values	Explanation
18	Blank	Specifies that a record of that type must be present
	O	Specifies that a record of that type is optional

Leave this column blank, if you specified an alphabetic sequence in columns 15 and 16.

15.6.8 Record-Identifying Indicator

Specifying an indicator in columns 19 and 20 associates the indicator with a particular record type. When VAX RPG II processes a record of the type you specify for this program line, it also sets on the indicator, which remains on until after detail-time output. Then, VAX RPG II sets off all indicators used as record-identifying indicators. See Chapter 7 for more information.

Column Number	Allowable Values	Explanation
19,20	Blank	Specifies not to set on an indicator when VAX RPG II processes a record of the type you specify.
	01-99	Specifies a record-identifying indicator.
	L1-L9	Specifies a control-level indicator. When VAX RPG II sets on this type of indicator, it does not automatically set on lower-level control-level indicators.
	H1-H9	Specifies a halt indicator.
	LR	Specifies a last-record indicator.
	**	Specifies that the fields described on the subsequent program lines are look-ahead fields.
	DS	Specifies a data structure.

Look-ahead fields allow you to do the following:

- Determine when the last record of a control group is processed
- Extend the matching-field processing capability

VAX RPG II typically processes one record at a time. The data from the record currently being processed is available. With look-ahead fields, you can evaluate the data from the next record to be processed and then determine which operation to perform.

Any or all of the fields in a file can be specified as look-ahead fields. The description applies to all records regardless of their record type.

To specify a field as a look-ahead field, you must enter a double asterisk (**) in columns 19 and 20 of the Input specification. Also, you must specify an alphabetic sequence in columns 15 and 16 of the Input specification. All other columns must be blank. Then, on the next program line, enter the field name, as shown in the following example:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)              Decimal positions
                | ||Record identifying indicator      |      Control level
                | |||                                 |      | Match field
                | ||| + Identifying codes + Format    |      | | Fld rec rel
     File       | ||| |                     | | (PB)   |Field | | |
     name       | ||| |    C      C      C| |Field   |name  | | | Field
      |         | ||| |    Z      Z      Z| |location||     | | | indicatrs
    I|          | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    **         * *** *---    *---    *---    .**---*---**     * * * * * * ....
    IEMPLOYEEAA  01
    I                                          1  20 NAME
    I          BB  **
    I                                         21  25 BADGE
```

ZK-4481-85

Rules

- Look-ahead fields can be used only with input or update primary or secondary files.
- For input files, look-ahead fields apply to the next record in the file.
- For update files, look-ahead fields apply only to the next record in the file if the current record being processed was read from another file. Therefore, if you are using only one file, the look-ahead field is the current record being processed.
- Look-ahead fields can be specified only once in a file.
- Look-ahead fields cannot be the only record in the file.
- As VAX RPG II processes the last record, it fills any look-ahead fields with 9s. In this case, if the field is 10 characters long, it will contain the data 9999999999.
- Columns 59 through 70 must be blank on Input specifications describing look-ahead fields.
- You cannot specify blank after (B in column 39 of the Output specification) for look-ahead fields.
- A look-ahead field cannot be used as a result field in a Calculation specification.

15.6.9 Record Identification Codes

Use columns 21 through 41 to define a record type and to specify the code that indicates how to identify it. You can subdivide these columns into three subsets (columns 21 through 27, 28 through 34, and 35 through 41), each defining a different code.

If you use more than one subset, the record must satisfy all record identification codes. Used in this way, the codes form an AND relationship. If VAX RPG II cannot identify a record according to the identification codes of all the records in a file, it issues a run-time error.

If there is only one record type for a file, you can leave these columns blank. Also, you can leave these columns blank when describing the last record type in a file. This defines a record type to catch all records that do not fall into any of the record types you previously described.

VAX RPG II checks records for a record type in the order in which you specify them on the Input specification.

15.6.9.1 Position

Use columns 21 through 24, 28 through 31, and 35 through 38 to define the position that specifies where to look for the identification code in the input record.

Column Number	Allowable Values	Explanation
21-24 28-31 35-38	Blank	Indicates that there is no record identification code. In this case, make sure that the corresponding not, character (C), zone (Z), or digit (D) portion and the character columns are blank.
	1-9999	Defines the position of the character you specify in columns 27, 34, and 41. For example, the number in columns 28 through 31 specifies the position of the character in column 34.

Rules

- Right justify this entry.
- Leading zeros can be omitted.

15.6.9.2 Not

Use columns 25, 32, and 39 to specify whether an identification code must be present in the input record.

Column Number	Allowable Values	Explanation
25,32,39	Blank	Indicates that the identification code you specify in the next two columns (26 and 27, 33 and 34, and 40 and 41) must be present to identify a record type. For example, if column 32 is left blank, the identification code in columns 33 and 34 must be present.
	N	Indicates that the identification code must not be present to identify a record type. For example, if you specify N in column 39, the identification code in columns 40 and 41 must not be present.

15.6.9.3 CZD Portion

Use columns 26, 33, and 40 to specify what portion of the character to use when identifying a record code. You can use the character (C), zone (Z), or digit (D) portion of the character. Many characters have either the same zone or digit portion. To distinguish between zone and digit portions, you must use their EBCDIC equivalent. See Appendix A for the ASCII character set and their corresponding EBCDIC zone and digit codes.

Column Number	Allowable Values	Explanation
26,33,40	Blank	Indicates that there is no record identification code. Its corresponding position, not, and character columns must be left blank.
	C	Causes VAX RPG II to use the entire character to identify the record.
	Z	Causes VAX RPG II to use the EBCDIC zone portion to identify the record.
	D	Causes VAX RPG II to use the EBCDIC digit portion to identify the record.

15.6.9.4 Character

Use columns 27, 34, and 41 to specify the identification character for the input record.

Column Number	Allowable Values	Explanation
27,34,41	Any character	Specifies the character part of the identification code

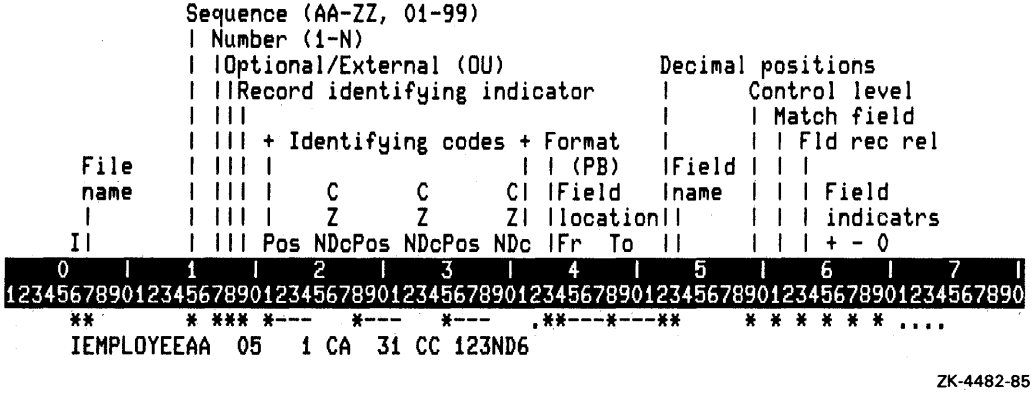
In the following example:

- I in column 6 specifies that this program line is an Input specification.
- EMPLOYEE in columns 7 through 14 names the input file. This file contains the name, address, and telephone number for each employee.
- The characters AA in columns 15 and 16 specify no sequence checking.

- VAX RPG II sets on the indicator you specified in columns 19 and 20 (05) after it reads a record that matches the identification code defined in columns 21 through 41.

  There are three parts to the code that identifies this record type:

  1. Position 1 must contain the character A.
  2. Position 31 must contain the character C.
  3. Position 123 must not contain a character with an EBCDIC digit portion of the number 6. This includes the characters F, O, W, 6, f, o, and w.



ZK-4482-85

15.6.10 AND and OR

Entering AND in columns 14 through 16 or entering OR in columns 14 and 15 associates two program lines that specify an identification code.

AND specifies that the identification codes in two program lines must be matched in order to identify the record and to set on the record-identifying indicator, if one is specified.

OR specifies that the identification code in either program line must be matched in order to identify the record and to set on the record-identifying indicator, if one is specified.

Column Number	Allowable Values	Explanation
14-16	AND	Specifies an AND relationship between the identification codes on this program line and the previous program line
14,15	OR	Specifies an OR relationship between the record identification codes on this program line and the previous program line

Rules

- If you use AND, columns 7 through 13 and 17 through 20 must be left blank.
- If you use OR, columns 7 through 13 and 16 through 18 must be left blank.
- You can enter a record-identifying indicator in columns 19 and 20 in an OR line. If you leave columns 19 and 20 blank, the record-identifying indicator in the preceding program line also applies to this program line.

In the following example, there are four characters that identify a record type in the file EMPLOYEE:

1. Position 1 must contain the character A.
2. Position 31 must contain the character C.
3. Position 1111 must contain the character zero.
4. Position 123 must not contain the digit 6.

The record must meet all the conditions in both program lines before VAX RPG II sets on the indicator (05).

VAX RPG II identifies a record type in the file RETIRED if position 1 contains the character I and position 31 contains the character D, or if position 123 does not contain the digit 6. The record must meet the conditions defined in either program line before VAX RPG II sets on the indicator (06).

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                                |      | Match field
                | ||| + Identifying codes + Format   |      | | Fld rec rel
      File      | ||| |                     | | (PB)   |Field | | |
      name      | ||| |    C       C       C| |Field   |name  | | | Field
       |        | ||| |    Z       Z       Z| ||location||      | | | indicatrs
     I|        | ||| Pos NDcPos NDcPos NDc |Fr  To  ||      | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IEMPLOYEEAA  05   1 CA  31 CC 123NC6
     I       AND    1111 C0
     IRETIRED AA  06   1 CI  31 CD
     I       OR      123NC6
```

ZK-4483-85

15.6.11 Format

Use columns 7 through 41 to describe the file and the records in the file. Use columns 43 through 75 to describe the fields of each record. Field descriptions must begin one line below the file and record description. Use a separate line to describe each field you plan to use.

If a field contains numeric data, use column 43 to specify its format. You can specify overpunched decimal, packed decimal, or binary format. See Chapter 14 for information on data formats.

Column Number	Allowable Values	Explanation
43	Blank	Indicates that the field contains either alphanumeric characters or numeric data in overpunched decimal format
	P	Indicates that numeric data is in packed decimal format
	B	Indicates that numeric data is in binary format

See Chapter 14 for information on numeric data types.

15.6.12 Field Locations From and To

You define the fields of a record by specifying their location. Use columns 44 through 47 to specify the beginning character position of the field. Use columns 48 through 51 to specify the ending character position of the field.

Column Number	Allowable Values	Explanation
44-47	1-9999	Specifies the beginning character position of the field
48-51	1-9999	Specifies the ending character position of the field

Rules

- The maximum length of a field depends on the type of data it contains. The maximum field length of overpunched decimal data is 15. The field length of binary data can be 2 or 4. The maximum field length of packed decimal data is 8. To determine the field length of packed decimal data, divide the number of digits by 2 and add 1, ignoring the remainder. For example, if the number of digits in packed decimal data is 9, the length is 5. The maximum field length of alphanumeric data is 9999.
- Fields can overlap if you give each field a different name.
- Right justify this entry.
- Leading zeros can be omitted.

15.6.13 Decimal Positions

If a field contains numeric data, use column 52 to specify the number of digits to the right of the decimal point.

Column Number	Allowable Values	Explanation
52	Blank	Indicates that this field contains alphanumeric data
	0-9	Specifies the number of positions to the right of the decimal point

Rules

- You must specify a value in this column even if the numeric data has no decimal points. In this case, use zero.
- The number of decimal positions must be less than or equal to the number of digits in the numeric field.

If you specify 2 in this column and the field contains the data 12345, the field's value is interpreted as 123.45. If you specify 4 in this column and the field contains the data 12345, the field's value is interpreted as 1.2345.

15.6.14 Field Name

Use columns 53 through 58 to assign a name to the field you defined in columns 43 through 52, or to specify the page number for PAGE.

Column Number	Allowable Values	Explanation
53-58	Name	Specifies the name of the field. The name can be a field name, array name, or array element.
	PAGE PAGE1-PAGE7	Specifies a page number. See Chapter 9 for more information on paging special words.
	*IN,*INxx	Sets the specified indicator. See Chapter 7 for more information on indicators.

Rules

- The field name can be any combination of six characters except for blanks or special characters, as long as the first character is a letter. See Chapter 14 for more information on user-defined names.
- You cannot use the reserved words UDATE, UDAY, UMONTH, and UYEAR as a field name.
- Use a unique name for each field. If you use the same name to describe more than one field in the same record type, VAX RPG II uses the field described last.
- You can use the same name for fields of different record types as long as both fields are numeric with the same number of digits, or both fields are alphanumeric with the same length.
- You can load an entire array from an input record by entering the array name in columns 53 through 58. If you do, columns 59 through 62 and 65 through 70 must be blank.
- You can load an array element by entering the array name followed by a comma and an array index.

To eliminate duplicate coding, use OR in columns 15 and 16 to define the same field names for different record types, as shown in the following example:

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)              Decimal positions
                | ||Record identifying indicator      |      Control level
                | |||                                 |      | Match field
                | ||| + Identifying codes + Format    |      | | Fld rec rel
 File           | ||| |                     | | (PB)   |Field | | |
 name           | ||| |    C      C      C| |Field   |name  | | | Field
  |             | ||| |    Z      Z      Z| |location||     | | | indicatrs
 II             | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **       * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IEMPLOYEEAA  05   1 CA
     I         OR  06  51 D6
     I                                        1  30 NAME
     I                                       31  50 ADDRES
     I                                       51  60OPHONE
```

ZK-4484-85

In this example, there are the same three fields (NAME, ADDRES, and PHONE) in two different types of records. In the first record type, the character A is in position 1. In the second record type, the number 6 is in position 51.

The NAME field contains alphanumeric data; it begins in position 1 and ends in position 30. The ADDRES field contains alphanumeric data; it begins in position 31 and ends in position 50. The PHONE field contains numeric data with no decimal positions; it begins in position 51 and ends in position 60.

15.6.15 Examples of Using Data Structures

This section provides examples of using data structures in a VAX RPG II program.

15.6.15.1 Multiple Definitions of Storage Area

The following example shows two fields that would normally require 1550 and 2400 bytes of storage without data structures. With data structures, however, these two fields are allocated using the same 2400 bytes of storage. In addition, several subfields within these fields are defined. The byte locations for each data structure subfield identify the locations, in a single data structure, where each data structure subfield is allocated.

This example also demonstrates the optional length specification of the data structure on the data structure statement. If you omit the length of the data structure, VAX RPG II computes it as described in Section 15.6.4.1.

Input Specification (I)

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                               |      | Match field
                | ||| + Identifying codes + Format  |      | | Fld rec rel
     File       | ||| |                     | | (PB)  |Field | | |
     name       | ||| |    C      C      C| |Field  |name  | | | Field
      |         | ||| |    Z      Z      Z| ||location||     | | | indicatrs
    I|         | ||| Pos NDcPos NDcPos NDc |Fr  To  ||     | | | + - 0
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    **        * *** *---   *---   *---   .**---*---**     * * * * * * ....
    IPERSNELL     01
    I                                          11550 PREC
    IMEDICAL      02
    I                                          12400 MREC
    I             DS                            2400
    I*PERSONNEL RECORDS FIELDS
    I                                          11550 PREC
    I                                           1  50 CTGRYA
    I                                          50 100 CTGRYB
    I                                         100 150 CTGRYC
    I                                         150 800 BKGRND
    I                                        8001500 FTRUSE
    I*MEDICAL RECORDS FIELDS
    I                                          12400 MREC
    I                                           1 550 IMNLGY
    I                                         550 950 HMTLGY
    I                                        9501550 RADLGY
    I                                       15501950 XRAY
    I                                       19502400 OPROOM
```

ZK-4485-85

15.6.15.2 Defining Subfields Within a Field or Subfield

The following example shows how to divide a field into subfields. To do this, you must specify the name of the field to be divided on the Input specification data structure statement.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        * *** *---    *---    *---    .**---*---**      * * * * * * ....
     ISURPLUS      01
     I                                                 1   12 ITEM
     IITEM         DS
     I                                                 1    4 WHOSE#
     I                                                 5    8 AREA#
     I                                                 9   12 YEAR
     I                                                11   11 DECADE
```

ZK-4486-85

15.6.15.3 Reorganizing Fields in an Input Record

In the following example, a data structure is used to reorganize fields from an input record. The first collection of fields describes the input record field layout. The second collection of fields (in the data structure) describes how the fields are organized in memory when the program runs.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        * *** *---    *---    *---    .**---*---**      * * * * * * ....
     IPART     AA
     I                                                 1    3 PN
     I                                                 4   10 PNAME
     I                                                11   12 WHOUSE
     I                                                13   20 COLWEI
     I                                                13   17 COLOR
     I                                                18   20 WEIGHT
     I                                                19   20ONWEIGH
     I                                                21   24 QTY
     IREGROP       DS
     I                                                19   21 PN
     I                                                 4   10 PNAME
     I                                                11   17 WHOCOL
     I                                                11   12 WHOUSE
     I                                                13   17 COLOR
     I                                                 1    3 WEIGHT
     I                                                23   26 QTY
```

ZK-4487-85

This example shows the difference between an input field (the PART record) and a data structure subfield (the REGROP data structure). If either of the fields COLOR or WEIGHT is changed in a Calculation specification, no change will be reflected in the field COLWEI because COLOR and WEIGHT are not redefinitions of that field. In contrast, if either COLOR or WHOUSE is changed, WHOCOL will also change because COLOR is a redefinition of one portion of that field, and WHOUSE is a redefinition of another portion of that field. Changing WHOCOL changes COLOR and WHOUSE. Changing the value of COLWEI in a Calculation specification will not change COLOR or WEIGHT.

15.6.15.4 Selecting the Internal Numeric Data Type for Fields

The following example shows how to use a data structure to select a numeric data type that will be used internally. Choosing specific numeric data types can improve performance where numeric fields are passed as parameters in a CALL statement, because numeric data type conversion is then not needed at run time. In this example, the numeric data type is indicated by the field name.

```
     0    I    1    I    2    I    3    I    4    I    5    I    6    I    7    I
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        * *** *---   *---   *---   .**---*---**     * * * * * * ....
     I           DS
     I                                    P   1   30PACKED
     I                                    B   4   50WORD
     I                                    B   6   90LONGWO
     I                                        10  140OVERPU
```

ZK-4488-85

If you specify a numeric data type for a data structure subfield, VAX RPG II does not automatically convert numerics to packed decimal format. A numeric conversion is performed if you define a subfield with a numeric data type that is different from the input field declaration. Arithmetic comparisons are done with the field maintaining the declared data type. Note that arithmetic operations (ADD/SUB/MULT/DIV) are still performed in packed decimal format, and a conversion is made before performing any of the arithmetic operations.

Where VAX/VMS Run-Time Library routines are called with various numeric data types, you can use data structures to declare the numeric data type so a conversion is not needed for the CALL statement.

Note that data structures do not support floating point numeric data.

The following example shows several numeric fields defined in an input record, then redefined with different numeric data types in a data structure. Each field redefinition must have the same number of digits as any previous field definition.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **        * *** *---    *---    *---    .**---*---**     * * * * * * ....
     IFILE     BB
     I                                        B   1   20IWORD4
     I                                        B  11  140ILONG9
     I                                        P  21  250IPACK9
     I                                           31  340IOVER4
     I                                           41  470IOVER7
     I                                           51  590IOVER9
     I               DS
     I                                           1   40IWORD4
     I                                        P  11  150ILONG9
     I                                        B  21  240IPACK9
     I                                        B  31  320IOVER4
     I                                        P  41  440IOVER7
     I                                        B  51  540IOVER9
```

ZK-4489-85

Each field is named to highlight the number of digits assigned to it, as defined in the input record. For example, PACK9 is defined to be a 5-byte (9-digit) field in the input record. The data structure indicates that PACK9 will be stored internally in the data structure as a longword (4 bytes). OVER7 is defined to be a 7-byte (7-digit) field in the input record. The data structure indicates that OVER7 will be stored internally in the data structure as a packed field (4 bytes). In all cases, the number of digits for the field as defined on the input record must be the same as the number of digits in any subsequent field redefinition.

15.6.15.5 Examples of Using Local Data Area

The following example demonstrates use of a local data area. The program LDA is as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

FTTY     D   V      80             TTY
I            UDS
I                                            1  15 NAME
I                                           16  17 0AGE
C            NAME       DSPLYTTY
C            AGE        DSPLYTTY
C                       MOVEL'S. Jones'NAME
C                       Z-ADD29         AGE
C                       SETON                      LR
```

ZK-4665-85

The following commands load the local data area with a name and age. The name and age are modified in the program, and this information is written back to the local data area upon exiting the program.

```
$ RPG$LDA1 = "K. Smith       "
$ RPG$LDA1[15,2] := "45"
$ RUN LDA
K. Smith
45
$ SHOW SYMBOL RPG$LDA1
RPG$LDA1 = "S. Jones       29"
$ RUN LDA
S. Jones
29
```

The following example demonstrates use of a local data area which contains binary data. The program LDA_BINARY is as follows:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

FTTY     D   V      80             TTY
I            UDS
I                                          1  15 NAME
I                                      B  16  17 0AGE
C            NAME       DSPLYTTY
C            AGE        DSPLYTTY
C                       MOVEL'S. Jones'NAME
C                       Z-ADD29         AGE
C                       SETON                      LR
```

ZK-4664-85

The following commands load the local data area with a name and age. The name and age are modified in the program, and this information is written back to the local data area upon exiting the program.

```
$ RPG$LDA1 = "K. Smith       "
$ RPG$LDA1[15*8,16] = 45
$ RUN LDA_BINARY
K. Smith
45
$ RUN LDA_BINARY
S. Jones
29
```

The following example demonstrates use of a local data area with 386 bytes of information. The program LDA_386 is as follows:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

IDS386       UDS
I                                        1 383 DATA1
I                                      384 386 DATA2
C                       MOVE 'abc'     DATA2
C                       SETON                      LR
```

ZK-4663-85

The following commands display the information written to the local data area by the program:

```
$ RUN LDA_386
$ CHAR_384 = F$EXTRACT(127,1,RPG$LDA3)
$ SHOW SYMBOL CHAR_384
CHAR_384 = "a"
$ SHOW SYMBOL RPG$LDA4
RPG$LDA4 = "bc"
```

Note that in this example, the field DATA1 overlaps RPG$LDA1, RPG$LDA2, and RPG$LDA3. The field DATA2 is written to the last byte of RPG$LDA3 and the first two bytes of RPG$LDA4.

15.6.16 Control-Level Indicator

Use columns 59 and 60 to specify control-level indicators. Control-level indicators cause VAX RPG II to compare the contents of a field with the contents of the same field from a previous record. If the fields are not equal, a control break occurs and VAX RPG II sets on the control-level indicator assigned to that field.

You can use this type of indicator to condition input, calculation, and output operations.

Column Number	Allowable Values	Explanation
59,60	Blank	Indicates that this field is not a control field
	L1-L9	Associates a control-level indicator with the field you specify in columns 53 through 58

Rules

- You can specify control-level indicators for primary and secondary files only.
- You can assign control-level indicators in any order.
- Control-level indicators are ranked from highest (L9) to lowest (L1). When a control break causes VAX RPG II to set on a control-level indicator, all lower control-level indicators are set on. All control-level indicators are set off after detail-time output.

- When you assign the same control-level indicator to more than one field, the fields are referred to as split-control fields. In this case, fields must be either all numeric or all alphanumeric and described on adjacent lines. Split-control fields do not need to be the same length.
- Fields with different control-level indicators can overlap in a record.
- You do not need to specify the same number of control fields for all record types.
- VAX RPG II initializes control fields to hexadecimal zeros. This usually causes a control break to occur on the first record with a control field. Because of this, VAX RPG II bypasses total-time calculation and output operations for this cycle.
- You cannot specify control-level indicators for binary data or look-ahead fields. Also, you cannot specify a control-level indicator when you specify an array name in columns 53 through 58.
- VAX RPG II ignores decimal positions and signs (positive and negative) when determining a control break.
- Because field names are ignored, you can assign the same control-level indicator to multiple fields with the same name.
- If you assign the same control-level indicator to more than one field in different types of records, the fields must be either all numeric with the same number of digits or all alphanumeric with the same number of characters.
- The total length of a split-control field must be the same length as other uses of the same control-level indicator.
- If a control field contains packed decimal data the zoned decimal length, which is two times the packed decimal length minus one, is considered the length of the field.

See Section 15.6.18 for information about using a field-record-relation indicator with control fields.

In the following example, each record in the file EMPLOYEE contains the same three fields: NAME, ADDRES, and DEPTNO. The length of NAME is 30 characters; the blank in column 52 indicates that the contents of the field are alphanumeric. The length of ADDRES is 20 characters. Both fields are assigned the same control-level indicator (L8), so they are split-control fields. DEPTNO contains more significant data and is assigned a higher-level control-level indicator. When the contents of DEPTNO change, VAX RPG II sets on both control-level indicators (L9 and L8).

Input Specification (I)

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)              Decimal positions
                | ||Record identifying indicator      |      Control level
                | |||                                 |      | Match field
                | ||| + Identifying codes + Format    |      | | Fld rec rel
      File      | ||| |                     | | (PB)   |Field | | |
      name      | ||| |    C       C       CI |Field   |name  | | | Field
       |        | ||| |    Z       Z       ZI ||location||      | | | indicatrs
     I|        | ||| Pos NDcPos NDcPos NDc |Fr  To  ||      | | | + - 0
   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890
     **        * *** *---   *---   *---   .**---*---**     * * * * * * ....
     IEMPLOYEEAA  05   1 CA  31 CC
     I         OR  99 123ND6
     I                                        1  30 NAME  L8
     I                                       31  51 ADDRESL8
     I                                      123 1230DEPTNOL9
```

ZK-4490-85

15.6.17 Matching Fields

Use columns 61 and 62 to specify matching fields. Matching fields instruct VAX RPG II to compare the fields in records from one or more files. When the contents of a field from a primary file match the contents of a field from a secondary file, VAX RPG II sets on the matching-record indicator (MR).

You can use the matching-record indicator to condition calculation and output operations.

Column Number	Allowable Values	Explanation
61,62	Blank	Indicates that this field is not a matching field.
	M1-M9	Identifies a matching field. See Chapter 8 for information about matching fields and multifile processing.

Rules

- You can use matching fields with one file to perform sequence checking or with multiple files to control the order of processing records. See Chapter 8 for information on multifile processing.
- You can compare only those fields from records in primary and secondary input and update files.
- You can compare up to nine different fields in a single record.
- If you specify more than one matching field for a record type, all the fields are logically concatenated and treated as one continuous field. The fields are combined according to descending sequence (M9 through M1) of matching field values.
- The program performs sequence checking for all record types with matching field specifications. An error in sequence causes a run-time error and terminates the program.
- You must define the same number of matching fields and the same matching field values (M1 through M9) for all those records that contain matching fields.
- You can overlap matching fields in a single record.
- Whenever you use more than one matching code, all matching fields must match before VAX RPG II sets on the matching-record indicator (MR).
- Matching fields assigned the same matching code (M1 through M9) must be either numeric with the same number of digits, or alphanumeric with the same length.
- Not all files or all record types within one program must have matching fields. However, at least one record type from each of two files must have matching fields if the files are to be matched.
- If the matching field contains packed data the zoned decimal length, which is two times the packed length minus one, is considered the length of the matching field. It is valid to match a packed field in one record against a zoned decimal field in another if the digit lengths are identical. The length must always be odd because the length of a packed field is always odd.
- The file sequence you specify in column 18 of the File Description specification must be the same for the files you compare—all ascending or descending.
- You can check the sequence of a single sequential file using M1 through M9 codes to designate the sequence. If the file is out of sequence, a run-time error occurs.

- You cannot specify matching values for binary data and look-ahead fields. You cannot specify matching values when you specify an array name in columns 53 through 58.
- If you specify an alternate collating sequence, VAX RPG II uses the alternate sequence when comparing the values in matching fields containing alphanumeric data.
- VAX RPG II ignores field names, so fields from different record types can have the same name and match code.
- When you specify an ascending sequence check, VAX RPG II initializes the matching value to hexadecimal zeros. When you specify a descending sequence check, VAX RPG II initializes the matching value to hexadecimal FFs. VAX RPG II initializes the matching value of a numeric field to zero.
- VAX RPG II compares matching fields containing numeric data based on their absolute values because decimal positions and signs are ignored.
- Matching fields cannot be split; the same matching field value cannot be used more than once for one type of record.
- When you specify a matching field value for a field without a field-record-relation indicator, you must specify all matching field values once without a field-record-relation indicator. If all matching fields are not common to all records, use a dummy matching field. See Section 15.6.19 for information on using a field-record-relation indicator with matching fields.
- Matching fields are independent of control-level indicators.

See Chapter 8 for examples of matching fields.

15.6.18 Field-Record-Relation Indicator

Use columns 63 and 64 to specify field-record-relation indicators that control the conditions under which VAX RPG II extracts data from the input buffer into a field. These conditions include control breaks, matching records, halts, and external indicators.

The most common use of a field-record-relation indicator is as a record-identifying indicator to group several different record types in an OR relationship and associate fields with a particular record type. You can also use field-record-relation indicators to extract data if a particular external indicator is on.

Column Number	Allowable Values	Explanation
63,64	Blank	Indicates no field-record-relation indicator
	01-99	Indicates that the field-record-relation indicator is a record-identifying indicator
	L1-L9	Indicates that the field-record-relation indicator is a control-level indicator
	MR	Indicates that the field-record-relation indicator is the matching-record indicator
	U1-U8	Indicates that the field-record-relation indicator is an external indicator
	H1-H9	Indicates that the field-record-relation indicator is a halt indicator

Rules

The following rules apply to field-record-relation indicators used with control and matching fields:

- You must specify control fields and matching fields without field-record-relation indicators *before* you specify those fields with them.
- When the field-record-relation indicator associated with a matching or control field is on, VAX RPG II uses that field as the control or matching field for the record rather than the same control or matching field specified without a field-record-relation indicator. Otherwise, VAX RPG II uses the control or matching field without the field-record-relation indicator.
- When you have not defined an entire set of matching fields without a field-record-relation indicator, a full set of matching fields must be assigned to each field-record-relation indicator used with a matching field.
- You must use the same field-record-relation indicator for split-control fields. You must describe the split-control fields on consecutive lines.
- You must group control and matching fields that use field-record-relation indicators according to indicator.
- Field-record-relation indicators for control and matching fields can be only 01 through 99 or H1 through H9 indicators. Also, the field-record-relation indicator for control and matching fields must be a record-identifying indicator specified on either the preceding record definition line, or in one of the lines in an OR relationship.

Input Specification (I)

If you have two records with eight fields each, and the first seven fields are the same but the last field is different, you can use the record-identifying indicator as the field-record-relation indicator to condition the field that is different, rather than defining all eight fields for both records.

In the following example, the last two fields were conditioned with the same record-identifying indicators in lines 1 and 2. If the job code is 2, VAX RPG II uses the bonus value from positions 72 through 74. If the job code is not 2, VAX RPG II will use the bonus value from positions 75 through 77.

```
                Sequence (AA-ZZ, 01-99)
                | Number (1-N)
                | |Optional/External (OU)            Decimal positions
                | ||Record identifying indicator     |      Control level
                | |||                                |      | Match field
                | ||| + Identifying codes + Format   |      | | Fld rec rel
 File           | ||| |                     | | (PB)   |Field | | | |
 name           | ||| |    C       C      CI |Field   |name  | | | | Field
  |             | ||| |    Z       Z      ZI ||location||      | | | | indicatrs
 II             | ||| Pos NDcPos NDcPos NDc |Fr  To  ||      | | | | + - 0
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        * *** *---    *---    *---    ,**---*---**     * * * * * * ....
     IEMPLOYEEAA  01  71 C2
     I         OR  02  71NC2
     I                                        1  10 FNAME
     I                                       11  12 MINIT
     I                                       13  33 LNAME
     I                                       34  54 ADDRES
     I                                       55  60 STATE
     I                                       61  700PHONE
     I                                       71  710JOBCDE
     I                                       72  742BONUS   01
     I                                       75  772BONUS   02
```

ZK-4491-85

15.6.19 Field Indicators

Use columns 65 through 70 to specify field indicators. Field indicators check the condition of numeric or alphanumeric fields when they are extracted from the input record. Once checked, the field can be in one of three conditions:

1. If the numeric field in columns 53 through 58 is greater than zero, the condition is positive and VAX RPG II sets on the field indicator in columns 65 and 66. Otherwise, VAX RPG II sets off the indicator.
2. If the numeric field in columns 53 through 58 is less than zero, the condition is negative and VAX RPG II sets on the field indicator in columns 67 and 68. Otherwise, VAX RPG II sets off the indicator.
3. If the numeric field in columns 53 through 58 is equal to zero, or if the alphanumeric field in columns 53 through 58 contains blanks, the condition is null and VAX RPG II sets on the field indicator in columns 69 and 70. Otherwise, VAX RPG II sets off the indicator.

Column Number	Allowable Values	Explanation
65-70	Blank	Indicates no field indicators.
	01-99	Associates a field indicator with a field.
	H1-H9	Indicates that the field indicator is a halt indicator. Halt indicators check for errors in data. For example, you can specify a halt indicator to check for zeros in a numeric field. If VAX RPG II processes the record and finds a zero in the field, it sets on the halt indicator that results in a run-time error.

Rules

- Use columns 65 through 70 to check numeric fields.
- Use columns 69 and 70 to check alphanumeric fields.
- You can use the same field indicator for more than one field in different record types. The status of the indicator depends on the record type last read.
- Columns 65 through 70 must be blank when columns 53 through 58 contain an array without an index or look-ahead fields.
- You can assign one or more field indicators to a numeric field.

Calculation Specification (C)

15.7 Description

The Calculation specification allows you to describe the calculations you want to perform and to define their order in the following ways:

- Entries in columns 7 through 17 control when a calculation is to be performed.
- Entries in columns 18 through 53 describe the type of calculation to be performed.
- Entries in columns 54 through 59 specify which indicators the program sets on or off as a result of the calculation.

There are two general rules:

1. Specify each calculation on a single line; arrange the calculations in the order you want them executed.
2. Specify detail-time calculations first, then total-time calculations and, finally, calculations in subroutines.

15.7.1 Calculation Specification Format

The format of the Calculation specification is as follows:

```
                                                    Field length
      Control level                                 |  Decimal positions
      |                                             |  |Half adjust (H)
      | Indicators          Operation               |  ||
      | |                   |                       |  ||Resulting
      | |         Factor    |    Factor    Result   |  ||indicators
      | |         1         |    2         field    |  ||+ - 0
    C| NxxNxxNxx|           |    |         |        |  ||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *      *--*** * * *
```

ZK-4492-85

15.7.2 Specification Type

Use column 6 to identify the type of specification for every program line.

Column Number	Allowable Values	Explanation
6	C	Indicates that this program line is a Calculation specification.

15.7.3 Control Level

Use columns 7 and 8 to indicate whether the calculation is performed at detail time, at total time, or is part of a subroutine.

Column Number	Allowable Values	Explanation
7,8	Blank	Performs the calculation at detail time, or indicates that the program line is part of a subroutine.
	L0	Performs the calculation at total time for each program cycle.
	L1-L9	Performs the calculation at total time after a control break occurs, or when you use the SETON operation to set on the control-level indicator, or when the indicator is set on as a record-identifying indicator, or when the indicator is set on as a resulting indicator in a calculation.
	LR	Performs the calculation at total time after the program processes the last record, or when you use the SETON operation to set on the last-record (LR) indicator, or when the indicator is set on as a record-identifying indicator, or when the indicator is set on as a resulting indicator in a calculation.
	SR	Indicates that the calculation is part of a subroutine.

Column Number	Allowable Values	Explanation
	AN or OR	Establishes a relationship between two program lines. If you use AN, the conditions for the indicators in both program lines must be satisfied before VAX RPG II executes the calculation. If you use OR, the conditions for the indicators in one program line or the other must be satisfied before VAX RPG II executes the calculation. You can use an unlimited number of AN or OR program lines with up to three indicators on each line to condition a single calculation. The last line in an AN or OR relationship specifies the calculation.

Additional Information

You can specify the following declarative statements in total-time calculations and optionally leave columns 7 and 8 blank:

- EXTRN
- GIVNG
- PARM
- PARMD
- PARMV
- PLIST
- TAG

In the following example, the L1, L2, L3, and LR control-level indicators perform calculations at total time after a control break occurs or when the SETON operation code sets on the indicator.

```
                                                          Field length
     Control level                                        |  Decimal positions
     |                                                    |  |Half adjust (H)
     | Indicators          Operation                      |  ||
     | | |                 |                            |  ||Resulting
     | | |       Factor    |    Factor    Result|        |  |||indicators
     | | |       1         |    2         field |        |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |        |  |||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *         *    *         *     *--*** * * *
     C*            Total calculations:
     C*
     CL1                     SETOF                        LRL9L8
     CL1                     SETOF                        L7L6L5
     CL1                     SETOF                        L4L3L2
     CL1           'L 1'     DSPLYS
     CL1                     SETOF                        L2L3LR
     C*
     CL1                     EXSR HILLS
     CL1           PELHAM    TAG
     C*
     CL1           'L 2'     DSPLYS
     CL3           'L 3'     DSPLYS
     C*
     CL3                     EXSR CARROL
     CL1                     GOTO PETERB
     C*
     CL1           'L 4'     DSPLYS
     CL2           6         COMP BERRY                   313233
     CLR                     SETOF                        24
```

ZK-4493-85

15.7.4 Indicators

Use indicators in columns 10, 11, 13, 14, 16, and 17 to condition the calculations you specify in columns 28 and 32. You can specify up to three indicators on a program line; precede the indicator with N to cause VAX RPG II to perform the calculation only when the indicator is not on. Use columns 9 through 11 to describe the first indicator, columns 12

through 14 to describe the second, and columns 15 through 17 to describe the third. Using the indicators in this way forms an AND relationship.

Column Number	Allowable Values	Explanation
10 13-14 16-17	Blank	Performs the calculation whenever the conditions specified in columns 7 and 8 are satisfied.
11	*	Repeat line. If the preceding line was performed, then the * line will be performed.
	Indicator	Performs the calculation when the conditions for the indicator are met.
9,12,15	N	Causes VAX RPG II to perform the calculation only when the indicators associated with N are not set on. N in column 9 conditions the indicator in columns 10 and 11. N in column 12 conditions the indicator in columns 13 and 14. N in column 15 conditions the indicator in columns 16 and 17.

Additional Information

You can use one of the following indicators in columns 10 and 11, 11, 13 and 14, and 16 and 17:

- Asterisk (*)
- Record-identifying (01-99)
- Control-level (L1-L9)
- Last-record (LR)
- Matching-record (MR)
- Halt (H1-H9)
- External (U1-U8)
- Overflow (OA-OG, and OV)
- KA-KZ and K0-K9

VAX RPG II performs total calculations for a control break before performing detail-time calculations for the record that causes the control break.

When multiple calculation lines are to be performed for the same set of conditions, you must specify the conditions only on the first line; indicate the same conditions for the successive lines with an asterisk indicator (*) in column 11. The result is that if the preceding line is performed, the * line will be performed. If there are additional conditions that must be met before the * line is to be performed, those conditions may be stated in columns 12 through 17.

Halt indicators in columns 10, 11, 13, 14, 16, and 17 cause VAX RPG II to bypass the operation when it finds an error in the input data or in a previous calculation. VAX RPG II processes the record that causes the error before stopping your program. In this case, the record in error could cause an error in calculation before your program terminates.

Depending on the relationship between indicators in columns 7 and 8 and columns 9 through 17, the actions VAX RPG II takes will vary as follows:

- When you specify a control-level indicator in columns 7 and 8 and a matching-record indicator in columns 9 through 17, MR indicates the result of matching the previous record rather than the record just read that caused a control break. VAX RPG II executes all the operations conditioned by control-level indicators before determining the matching condition of the record just read.
- When you use a control-level indicator in columns 10, 11, 13, 14, 16, and 17 instead of in columns 7 and 8 of the Calculation specification, VAX RPG II performs the calculation on the first record of a new control group at detail time.
- In a single program cycle, VAX RPG II performs all operations conditioned by the control-level indicators in columns 7 and 8 before it performs the operations conditioned by the control-level indicators in columns 9 through 17.
- If you condition a calculation with a last-record indicator in columns 9 through 17 when columns 7 and 8 are blank, the calculation is performed only if the last-record indicator is set on during detail-time calculations. If the last-record indicator is set on when VAX RPG II reaches the end-of-file or during total-time calculations, VAX RPG II does not perform detail-time calculations.

Calculation Specification (C)

In the following example, the record-identifying indicators 01, 02, and 03 must be on to perform the calculation SALARY * BONUS1 = GROSS. In the second program line, the indicator 04 must be off and indicator 05 must be on to perform the calculation SALARY * BONUS2 = GROSS.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  ||indicators
     | |         1         |    2         field |   |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  ||> < = +- Comments --+
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *        *         *    *         *     *--*** * * *
     C   01 02 03SALARY    MULT BONUS1    GROSS
     C  N04 05   SALARY    MULT BONUS2    GROSS
```

ZK-4494-85

15.7.5 Factors 1 and 2

Use columns 18 through 27 and 33 through 42 to provide the operands for the calculation you specify in columns 28 through 32. Use columns 18 through 27 for factor 1 and columns 33 through 42 for factor 2. The operands you use depend on the operation you specify. See Chapter 16 for information on operations and the operands they require.

Column Number	Allowable Values	Explanation
18-27 or 33-42	Field name	Names the field that contains data. These are the same fields you defined in columns 53 through 58 of the Input specification or in columns 43 through 48 of the Calculation specification.
18-27 or 33-42	Literal	Specifies an alphanumeric or numeric constant. Numeric literals can consist of the digits 0 through 9, one decimal point, and one arithmetic sign. Numeric literals cannot exceed 10 characters and cannot contain blanks. You must specify the sign in the leftmost character position. Alphanumeric literals can be up to eight characters including blanks. You must enclose alphanumeric literals in single quotation marks (for example, 'NH'). Use the keyboard apostrophe mark for the single quotation mark. If you want to use an apostrophe in a literal, you must enter two consecutive apostrophes (for example, it''s).
8-74	Long character literal	Specifies an alphanumeric constant that contains 1 to 460 characters. A double quotation mark (") is placed in the first character of the field on the specification. The rest of the field is left blank. On the next line, a double quotation mark (") is placed in column 7. Columns 8 through 74 contain the character literal, which must be enclosed within single quotation marks ('). The character literal can be anywhere on the line. If you wish the character literal to continue on the next line, follow the ending single quotation mark with a plus sign (+) and continue the literal in the same manner on the next specification. All the rules for "normal" character literals apply to the long character literal placed in columns 8 through 74. If more than one long character literal is entered on a Calculation specification, the character literal for the first (leftmost) entry is on the next specification, followed by the character literal for the second entry on the specification after that.

Column Number	Allowable Values	Explanation
	Table or array	Specifies the table name, array name, or array element you specified previously in an Extension specification.
	Subroutine name	Specifies one of the following components of a subroutine: BEGSR (marks the beginning of a subroutine) and EXSR (executes a subroutine).
	Special words	Specifies one of the following special words: UDATE, UMONTH, UDAY, UYEAR, PAGE, PAGE1 through PAGE7, *IN, and *INxx. See Chapter 9 for information on special words. See Chapter 7 for information on *IN and *INxx.
	Label	Specifies the label for TAG, GOTO, and ENDSR operations. See Chapter 16 for information on TAG, GOTO, and ENDSR operation codes.
	File name	Specifies the file name for CHAIN, DSPLY, READ, SETLL, or FORCE operations. See Chapter 16 for information on specifying files for these operations.

Note that you must left justify the entries in Factors 1 and 2 unless they are numeric literals, which must be right justified.

In the following example, the literal 'All work and no play makes Jack a dull boy.' is moved to the field SHINE:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *         *          *    *          *     *--*** * * *
     C                       MOVE "          SHINE  80
     C" 'All work and no play make Jack a dull boy.'
```

ZK-4495-85

This example shows a long character literal continuing on another line:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *         *          *    *          *     *--*** * * *
     C                       MOVE "          SHINE  80
     C"                            'All work and no play make Jack '+
     C" 'a dull boy.'
```

ZK-4496-85

15.7.6 Operation Code

Use columns 28 through 32 to specify the operation code that indicates which calculation to perform on the operands you specified in columns 18 through 27 and 33 through 42. See Chapter 16 for more information on operation codes.

Column Number	Allowable Values	Explanation
28-32	Operation code	Performs the action specified by the operation code. See Chapter 16 for information on operation codes.

15.7.7 Result Field

Use columns 43 through 48 to provide the result field that will contain the outcome of the calculation you specified in columns 18 through 42. You can use a field you specified previously in an Input, Calculation, or Extension specification or use columns 49 through 52 to define its length and decimal positions.

Column Number	Allowable Values	Explanation
43-48	Name	Identifies the result field. The result field can contain a field name, table or array name, array element, or one of the following special words: PAGE, PAGE1-7, *IN, or *1Nxx.

Rules

- The result field name can be any combination of alphanumeric characters; the first character must be alphabetic. Embedded blanks are not allowed.
- You cannot use a look-ahead field, a field defined by an EXTRN operation, UDATE, UDAY, UMONTH, or UYEAR as a result field.

15.7.8 Field Length

If you use the Calculation specification to define a result field, use columns 49 through 51 to define the length of the result field you specified in columns 43 through 48. Otherwise, you can leave columns 49 through 51 blank. To prevent undefined or truncated results, make sure the length of the result field is long enough to hold the largest possible result.

Column Number	Allowable Values	Explanation
49-51	1-999	Specifies the length of the result field

Rules

- The maximum length for numeric data is 15 digits.
- The maximum length for alphanumeric data is 999 characters.
- If the field is described elsewhere in the program and an entry is made in columns 49 through 51, both entries must specify the same length.
- Right justify this entry.
- Leading zeros can be omitted.

15.7.9 Decimal Positions

If you use the Calculation specification to define the result field and the result field contains numeric data, use column 52 to specify the number of positions to the right of the decimal point.

Column Number	Allowable Values	Explanation
52	Blank	Indicates that this field contains alphanumeric data or that the result field has been defined elsewhere
	0-9	Specifies the number of positions to the right of the implied decimal point

Rules

- If the field has been described previously in the program and an entry is made in column 52, both entries for decimal positions must be the same.
- The number you specify in this column must be smaller than the number in columns 49 through 51.
- If the result field contains alphanumeric data, leave this column blank.
- When the result is numeric, but has no decimal positions, you must specify zero.

15.7.10 Half Adjust

Use column 53 to specify whether VAX RPG II is to round the numeric data in the result field. VAX RPG II adds five to the position immediately to the right of the last digit and puts the new value in the result field. VAX RPG II performs the addition on the absolute value of the number. For example, if the result of an arithmetic operation is 123.456 and the result field specifies two decimal positions, VAX RPG II half adjusts the value in the result field to 123.46.

Column Number	Allowable Values	Explanation
53	Blank	Performs no half adjusting
	H	Half adjusts the numeric data in the result field

Rules

- You cannot half adjust the result field of an MVR operation or a DIV operation that is followed immediately by an MVR operation.
- You cannot half adjust alphanumeric data.

Additional Information

VAX RPG II sets resulting indicators according to the value of the result field after half adjusting. See Table 16-1 in Chapter 16 for a list of operation codes that allow you to specify half adjust.

15.7.11 Resulting Indicators

Use columns 54 through 59 to enter resulting indicators that test the outcome of a calculation. You can use these resulting indicators to condition other calculation or output operations, or to establish field-record relations.

Column Number	Allowable Values	Explanation
54-59	01-99	Uses a record-identifying indicator as the resulting indicator.
	H1-H9	Uses a halt indicator as the resulting indicator.
	K0-K9 KA-KZ	Uses a K indicator as the resulting indicator.
	L1-L9	Uses a control-level indicator as the resulting indicator.
	LR	Uses a last-record indicator as the resulting indicator.
	OA-OG, OV	Uses an overflow indicator as the resulting indicator.
	U1-U8	Uses an external indicator as the resulting indicator.

Rules

- A resulting indicator is set on if the condition specified is satisfied. If the specified condition is not satisfied, the resulting indicator is set off. See Chapter 16 for information on how resulting indicators are used with each operation code.
- If you use the same indicator to test the results of more than one operation, the last operation determines the indicator setting.
- You cannot use resulting indicators when the result field contains a nonindexed array.

Additional Information

After a resulting indicator is on, it remains on until one of the following occurs:

- The operation is repeated and the result resets the indicator
- The conditions the indicator specifies are not met
- The indicator is set off by another method (such as the SETOF operation)

Using a control-level indicator as a resulting indicator does not automatically set on lower-level indicators.

Using an external indicator as a resulting indicator allows you to set the indicator, then to test the indicator value after the program exits.

15.7.12 Comments

Use columns 60 through 74 for comments.

Column Number	Allowable Values	Explanation
60-74	Any character	Documents the program line

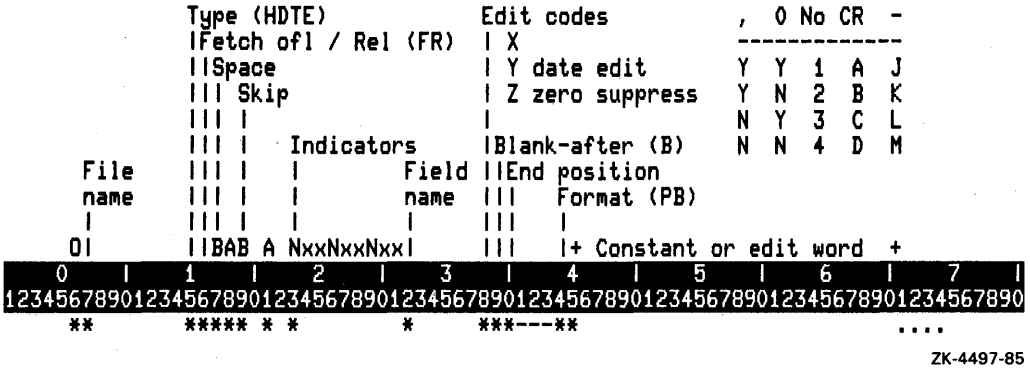
Output Specification (O)

15.8 Description

The Output specification describes the records and fields in an output, update, or input (with the ADD option) file. Columns 7 through 37 describe the record and columns 23 through 70 describe the position and format of each field in the record.

15.8.1 Output Specification Format

The format of the Output specification is as follows:



ZK-4497-85

15.8.2 Specification Type

Use column 6 to identify the type of specification for every program line.

Column Number	Allowable Values	Explanation
6	O	Indicates that this program line is an Output specification

15.8.3 File Name

Use columns 7 through 14 to name the output, update, or input (with the ADD option) file.

Column Number	Allowable Values	Explanation
7-14	File name	Identifies the name of the output file

Rules

- Use the same file name you specified in the File Description specification. An output file can be a file you specified as an output file, an update file, or an input file with A in column 66 of the File Description specification.
- Left justify this entry.
- If columns 7 through 14 are blank, VAX RPG II assumes that the information in this program line describes a field or record from the file last named.

All the records for a single file need not be described together.

15.8.4 AND and OR Lines

If you need more than three indicators to condition record output, or if you want to output a record under a number of conditions, use columns 14 through 16 to enter AND or columns 14 through 15 to enter OR.

Column Number	Allowable Values	Explanation
14,15	OR	Performs the output operation when the conditions for all indicators in columns 23 through 31 in either program line are met
14-16	AND	Performs the output operation when the conditions for all indicators in columns 23 through 31 in both program lines are met

Rules

- You must use at least one indicator on a program line in an OR or AND relationship.
- If you use AND, columns 17 through 22 must be blank.
- If you use AND or OR, columns 7 through 13 must be blank.
- You can use AND and OR lines only with record description entries, not with field description entries.
- You can specify an unlimited number of AND or OR lines.

In the following example, if these conditions are satisfied, VAX RPG II writes the specified fields and constants:

— Indicator 01 is off

— OR, indicator 01 is on

— AND, indicator 23 is off

```
          Type (HDTE)            Edit codes          , 0 No CR -
          |Fetch ofl / Rel (FR)  | X                 ------------
          ||Space                | Y date edit       Y Y 1 A J
          ||| Skip               | Z zero suppress   Y N 2 B K
          ||| |                  |                   N Y 3 C L
          ||| |  Indicators      |Blank-after (B)    N N 4 D M
 File     ||| |  |        Field  ||End position
 name     ||| |  |        name   |||  Format (PB)
  |       ||| |  |         |     |||  |
 O|       ||BAB A NxxNxxNxx|     |||  |+ Constant or edit word +
   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890
 **       ***** * *       *     ***---**                        ....
 O       OR          N01
 O       OR           01
 O       AND         N23
 O                        PN        3
 O                     01           28 '01'
 O                        PNAME     10
 O                        WHOUSE    12
 O                        COLOR     17
 O                        WEIGHT    20
 O                        QTY       24
```

ZK-4498-85

15.8.5 Record Type

Use column 15 to specify the point in the VAX RPG II program cycle at which a record is output. Heading records are normally used to describe the heading information in the output report, such as column names, page numbers, and the date. Detail records contain the data from input and calculation operations at detail time. Total records usually contain the data from the result of calculations on several detail records at total time. Exception records are written as a result of using the EXCPT operation in a Calculation specification.

Column Number	Allowable Values	Explanation
15	Blank	Indicates that this program line describes a field or constant
	H	Indicates that this program line describes a heading record
	D	Indicates that this program line describes a detail record
	T	Indicates that this program line describes a total record
	E	Indicates that this program line describes an exception record

Rules

- You must specify a record type for every output record.
- Records of the same type are tested for output and written in the order in which you specify them in the Output specifications.

Additional Information

There is no difference between a heading record and a detail record. The different entries are for documentation purposes only.

The following example defines heading, detail, total, and exception records.

```
                Type (HDTE)              Edit codes          , 0 No CR  -
                |Fetch ofl / Rel (FR)   | X                  ------------
                ||Space                 | Y date edit        Y  Y  1  A  J
                ||| Skip                | Z zero suppress    Y  N  2  B  K
                ||| |                   |                    N  Y  3  C  L
                ||| |   Indicators      |Blank-after (B)     N  N  4  D  M
     File       ||| |   |        Field  ||End position
     name       ||| |   |        name   |||   Format (PB)
     |          ||| |   |        |      |||   |
     O|         ||BAB A NxxNxxNxx|      |||   |+ Constant or edit word  +
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
     12345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *     ***---**                          ....
     OOUT44A  E            12
     O        OR              16
     O        OR              LR
     O                          N          3
     O                          PNAME     10
     O                          WHOUSE    12
     O                          COLOR     17
     O                          WEIGHT    20
     O                          QTY       24
     O                                    26 'E'
     O                 N12 16             28 '16'
     O                  12 01             28 '12'
     O                  LR                28 'LR'
     O         H
     O                          N          3
     O                          PNAME     10
     O                          WHOUSE    12
     O                          COLOR     17
     O                          WEIGHT    20
     O                          QTY       24
     O                                    26 'H'
     O                        01          28 '01'
     O         D         17
     O        OR           N01
     O        OR            01 01
     O        AND          N23
     O                          N          3
     O                        01          28 '01'
     O                          PNAME     10
     O                          WHOUSE    12
     O                          COLOR     17
     O                          WEIGHT    20
     O                          QTY       24
     O                                    26 'D'
     O         T
     O                          N          3
     O                          PNAME     10
     O                          WHOUSE    12
     O                          COLOR     17
     O                        01          28 '01'
     O                          WEIGHT    20
     O                          QTY       24
     O                                    26 'T'
```

ZK-4499-85

15.8.6 ADD and DEL Options

Use columns 16 through 18 to add and delete records. See Chapter 8 for information on adding and deleting records.

Column Number	Allowable Values	Explanation
16-18	ADD	Adds a record to an input, output, or update file with an indexed, direct, or sequential file organization
	DEL	Deletes the last record read in the update file with an indexed or direct file organization

Rules

- You can add records to input, output, and update files that reside on disk. Therefore, the File Description specification must contain DISK in columns 40 through 46 and A in column 66.
- You can delete records only from update files that reside on disk.
- ADD or DEL must appear on the same line that defines the record type for the record you want to add or delete.
- If a line in an OR relationship follows an ADD or DEL entry, the ADD or DEL entry applies to both lines.

The following example adds records to the file:

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     OOUT43A  DADD    N1P
     O                        N          3
     O                        PNAME     10
     O                        WHOUSE    12
     O                        COLOR     17
     O                        WEIGHT    20
     O                        QTY       24
```

ZK-4500-85

The following example deletes the last record read from the update file:

```
                Type (HDTE)                Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)      | X                 -------------
                ||Space                    | Y date edit       Y Y 1 A J
                ||| Skip                   | Z zero suppress   Y N 2 B K
                ||| |                      |                   N Y 3 C L
                ||| |   Indicators         |Blank-after (B)    N N 4 D M
     File       ||| |   |        Field     ||End position
     name       ||| |   |        name      |||  Format (PB)
     |          ||| |   |        |         |||  |
    O|          ||BAB A NxxNxxNxx|         |||  |+ Constant or edit word +
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *      ***---**                          ....
     OOUT45B    EDEL    N25
     OOUT45C    EDEL    N25
```

ZK-4501-85

15.8.7 Fetch Overflow or Release

Use column 16 to specify fetch overflow for a printer file, or release for a WORKSTN file. Fetch overflow causes VAX RPG II to check whether the overflow indicator assigned to the printer output file is on before printing total, detail, or exception records. See Chapter 9 for information on overflow.

Release terminates processing of the WORKSTN file. See Chapter 6 for information on release.

Column Number	Allowable Values	Explanation
16	F	Executes the overflow routine if overflow has occurred
	R	Release is performed on the WORKSTN file

Rules

- An entry in this column is valid only for printer output files with overflow lines (F), or a WORKSTN file (R).
- Do not specify an overflow indicator on the same line as fetch overflow.

- If you specify an OR relationship between two lines, you must specify fetch overflow for each record type that requires it in both lines of the OR relationship.

Additional Information

VAX RPG II fetches an overflow routine when overflow occurs and all conditions specified by the indicators in columns 23 through 31 are met. When you specify fetch overflow, only overflow output associated with the file containing the executed fetch routine is output. The overflow routine does not automatically advance to the next page.

The following example specifies fetch overflow:

```
                Type (HDTE)             Edit codes          ,  0 No CR  -
                |Fetch ofl / Rel (FR)   | X                  ------------
                ||Space                 | Y date edit        Y  Y  1  A  J
                ||| Skip                | Z zero suppress    Y  N  2  B  K
                ||| |                   |                    N  Y  3  C  L
                ||| |   Indicators      |Blank-after (B)     N  N  4  D  M
     File       ||| |   |         Field ||End position
     name       ||| |   |         name  |||   Format (PB)
      |         ||| |   |           |   |||   |
     O|         ||BAB A NxxNxxNxx|     |||   |+ Constant or edit word  +
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **        ***** * *         *      ***---**                          ....
     OOUT66A   EF 1          01
     O         AND           02 03
```

ZK-4502-85

15.8.8 Space Before and Space After

Use columns 17 and 18 to define the format of a printer output file. Use column 17 to specify the number of lines to advance before printing the next line of output. Use column 18 to specify the number of lines to advance after printing a line of output.

Column Number	Allowable Values	Explanation
17	Blank	Does not advance before printing a line of output.
	0-3	Specifies the number of lines the printer will advance before printing a line of output. A value of zero allows overprinting.
18	Blank	Does not advance after printing a line of output.
	0-3	Specifies the number of lines the printer will advance after printing a line of output. A value of zero allows overprinting.

Rules

- If you leave columns 17 through 20 blank for a record specification line, VAX RPG II automatically spaces one line after printing the output line.
- If there are no entries in columns 17 through 20 of an OR line, VAX RPG II uses the entries in a preceding line.
- You cannot define the spacing and skipping for an AND line.

Additional Information

Because you can space up to only three lines before and after a line of output, you cannot specify more than five blank lines between output lines using entries in columns 17 and 18. Spacing to or past the overflow line causes VAX RPG II to set on the overflow indicator.

15.8.9 Skip Before and Skip After

Like the space before and space after columns, columns 19 through 22 define the format of a printer output file. Unlike the entries in columns 17 and 18, the entries in columns 19 and 20 can be used to specify more than five lines between lines and to specify a move to the next page.

Use column 19 to specify the line number the printer must move to before printing a line of output. Use column 20 to specify the line number the printer must move to after printing a line of output.

Output Specification (O)

Column Number	Allowable Values	Explanation
19,20	Blank	Specifies no skipping before printing a line of output
	01-99	Causes the printer to move to the line number you specify before printing a line of output
	A0-A9	Causes the printer to move to the line number you specify 100 (A0) to 109 (A9) before printing a line of output
	B0-B2	Causes the printer to move to the line number you specify 110 (B0) to 112 (B2) before printing a line of output
21,22	Blank	Specifies no skipping after printing a line of output
	01-99	Causes the printer to move to the line number you specify after printing a line of output
	A0-A9	Causes the printer to move to the line number you specify 100 (A0) to 109 (A9) after printing a line of output
	B0-B2	Causes the printer to move to the line number you specify 110 (B0) to 112 (B2) after printing a line of output

Rules

- Follow the same rules in Section 15.6.10 for AND and OR lines.
- You can specify entries in all space and skip columns for a single program line. When you do, VAX RPG II executes the entries in the following order: skip before, space before, print the output line, skip after, and space after.
- Specifying a skip entry past the overflow line causes VAX RPG II to set on the overflow indicator. See Chapter 9 for more information.
- If you specify a skip entry to the same line number that the printer is currently on, no skipping takes place.
- If you specify a skip entry to a line number less than the current line number, the printer advances to that line number on the next page.
- The skip entry cannot exceed the entry for forms length (columns 18 and 19 of the Line Counter specification). If there is no Line Counter specification, the skip entry cannot exceed the default, line 66.

15.8.10 Example

The following example causes VAX RPG II to perform the following operations:

- Skip to line 27 and space two lines before printing the output line
- Skip to line 30 and space three lines after printing the output line

```
                Type (HDTE)            Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)  | X                  ------------
                ||Space                | Y date edit        Y Y 1 A J
                ||| Skip               | Z zero suppress    Y N 2 B K
                ||| |                  |                    N Y 3 C L
                ||| |   Indicators     |Blank-after (B)     N N 4 D M
     File       ||| |   |        Field  ||End position
     name       ||| |   |        name   |||   Format (PB)
      |         ||| |   |         |     |||   |
     O|         ||BAB A NxxNxxNxx|     |||   |+ Constant or edit word  +
      0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *      ***---**                          ....
     O          D 232730N1P 18
     O                                   N          3
     O                                   PNAME     10
     O                                   WHOUSE    12
     O                                   COLOR     17
     O                                   WEIGHT    20
     O                                   QTY       24
```

ZK-4503-85

15.8.11 Indicators

Use columns 24 and 25, 27 and 28, and 30 and 31 to enter previously assigned indicators to condition a line of output.

Output Specification (O)

Column Number	Allowable Values	Explanation
24-25 27-28 30-31	Blank	Outputs the record or field.
	Indicator	Outputs the record or field when the indicator you specify is on.
25	*	Repeat line. If the preceding record was output, this record will be output. If the preceding field was output, this field will be output.
23,26,29	N	Outputs the record or field when the indicator is off. N in column 23 conditions the indicator in columns 24 and 25. N in column 26 conditions the indicator in columns 27 and 28. N in column 29 conditions the indicator in columns 30 and 31.

Rules

- When you want an indicator to condition an entire record, enter the indicator on the line that specifies the type of record. When you want an indicator to condition a field, enter the indicator on the same line as the field name (columns 32 through 37).
- If you specify more than one indicator on a line, the indicators form an AND relationship.
- You can use overflow indicators on AND or OR lines; however, you can associate only one overflow indicator with a group of output indicators. If a line is to be considered an overflow line, the overflow indicator must appear on the main specification line or on an OR line.
- If you use an overflow indicator, it must be the same one assigned to the file on the File Description specification.
- You cannot use overflow indicators to condition exception output lines, but you can use them to condition fields in an exception record.

Additional Information

You can use one of the following indicators in columns 24 through 25, 27 through 28, and 30 through 31:

- Record-identifying (01-99)
- Control-level (L1-L9)
- Last-record (LR)

- Matching-record (MR)
- Halt (H1-H9)
- External (U1-U8)
- Overflow (OA-OG and OV)
- K (KA-KZ and K0-K9)
- First-page (1P)
- Asterisk indicator (*)

Note that VAX RPG II outputs those detail and heading lines conditioned by the first-page (1P) indicator, no indicator, or all negative indicators (N in columns 23, 26, or 29) before reading the first record from a file. Therefore, use the 1P indicator to condition only heading and detail output lines that do not depend on data from an input record. For a line with no indicators or all negative indicators that requires data from an input record, use a negative first-page indicator (N1P in columns 23 through 31) to prevent the line from being output before reading the first record.

Because the 1P indicator is set off after the first detail-time output, it can be used only to condition heading and detail lines.

If you use a control-level indicator with a total record and no overflow indicator, VAX RPG II writes the record when a control break occurs and after VAX RPG II processes the last record of a control group. If you use a control-level indicator with a detail record and no overflow indicator, VAX RPG II writes the record when a control break occurs and after it processes the first record of a new control group. If you use a control-level indicator with an overflow indicator, VAX RPG II writes the record when a control break occurs and passes the overflow line.

If you have two or more output records that are to be output when the same conditions are set, you can specify the conditions on one Output specification record line and then use * in column 25 on following Output specification record lines of the same type. If you have two or more output fields that are to be output when the same conditions are set, you can specify the conditions on one Output specification field line and then use * in column 25 on following Output specification field lines.

Output Specification (O)

The following example causes VAX RPG II to print the specified fields in the detail record if the 1P indicator is off:

```
                Type (HDTE)              Edit codes          ,  0 No CR  -
                |Fetch ofl / Rel (FR)    | X                 -------------
                ||Space                  | Y date edit       Y  Y  1  A  J
                ||| Skip                 | Z zero suppress   Y  N  2  B  K
                ||| |                    |                   N  Y  3  C  L
                ||| |   Indicators       |Blank-after (B)    N  N  4  D  M
 File           ||| |   |        Field    ||End position
 name           ||| |   |        name     |||   Format (PB)
 |              ||| |   |        |        |||   |
O|              ||BAB A NxxNxxNxx|        |||   |+ Constant or edit word  +
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *      ***---**                          ....
     OOUT50A    D       N1P
     O                            N          3
     O                            PNAME     10
     O                            WHOUSE    12
     O                            COLOR     17
     O                            WEIGHT    20
     O                            QTY       24
     O                            PAGE      30
```

ZK-4504-85

15.8.12 Field Name

Use columns 32 through 37 to specify the field name that identifies the item to be written to the output file.

Column Number	Allowable Values	Explanation
32-37	Blank	Indicates the presence of a constant in columns 45 through 70.
	Name	Specifies the name of the item to print. The item can be a field name, table or array name, array element, or one of the following special words: PAGE, PAGE1-7, UDAY, UMONTH, UYEAR, UDATE, *IN, *INxx, and *PLACE. See Chapter 9 for information on special words.

Rules

- All field names must have been previously defined in an Input, Calculation, or Extension specification.
- Left justify this entry.
- You cannot enter a field name if you enter a constant in columns 45 through 70.
- If you enter a field name in columns 32 through 37, columns 7 through 22 must be blank.
- If you specify a nonindexed array name, the entire array is output.

The following example specifies fields in the detail record:

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)   | X                  ----------
                ||Space                 | Y date edit        Y Y 1 A J
                ||| Skip                | Z zero suppress    Y N 2 B K
                ||| |                   |                    N Y 3 C L
                ||| |   Indicators      |Blank-after (B)     N N 4 D M
  File          ||| |   |       Field   ||End position
  name          ||| |   |       name    |||  Format (PB)
  |             ||| |   |       |       |||  |
 O|             ||BAB A NxxNxxNxx|       |||  |+ Constant or edit word +
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     **         ***** * *         *      ***---**                          ....
     OOUT50A    D         N1P
     O                              N          3
     O                              PNAME     10
     O                              WHOUSE    12
     O                              COLOR     17
     O                              WEIGHT    20
     O                              QTY       24
     O                              PAGE      30
```

ZK-4505-85

15.8.13 EXCPT Name

When the record type is an exception record (indicated by an E in column 15), a name can be placed in columns 32 through 37 of the record line. The EXCPT operation can specify the name assigned to a group of records to be written. The name is called an EXCPT name.

Column Number	Allowable Values	Explanation
32-37	Blank	Identifies exception output records to be written when an EXCPT opcode with a blank factor 2 is executed.
	Name	Identifies exception output records to be written when an EXCPT opcode with the same name in factor 2 is executed.

Rules

- An EXCPT name must follow the rules for field names.
- An EXCPT name cannot be the same as a file name, field name, data structure name, array name, table name, label, or subroutine name.
- A group of any number of output records can use the same EXCPT name, and the records do not have to be consecutive records.

15.8.14 Edit Codes

Use column 38 to specify an edit code. Edit codes allow you to perform a variety of editing functions on the data in a numeric output field.

Column Number	Allowable Values	Explanation
38	1	Prints a number with commas before every third digit to the left of the decimal point, prints a zero balance, and suppresses signs and leading zeros.
	2	Prints a number with commas before every third digit to the left of the decimal point and suppresses a zero balance, signs, and leading zeros.
	3	Prints a number without commas, prints a zero balance, and suppresses signs and leading zeros.
	4	Prints a number without commas and suppresses a zero balance, signs, and leading zeros.

Column Number	Allowable Values	Explanation
	A	Prints a number with commas before every third digit to the left of the decimal point, prints a zero balance, uses CR to represent a negative sign, and suppresses leading zeros.
	B	Prints a number with commas before every third digit to the left of the decimal point, suppresses a zero balance, uses CR to represent a negative sign, and suppresses leading zeros.
	C	Prints a number without commas, prints a zero balance, uses CR to represent a negative sign, and suppresses leading zeros.
	D	Prints a number without commas, suppresses a zero balance, uses CR to represent a negative sign, and suppresses leading zeros.
	J	Prints a number with commas before every third digit to the left of the decimal point, prints a zero balance and prints a negative sign, and suppresses leading zeros.
	K	Prints a number with commas before every third digit to the left of the decimal point, suppresses a zero balance, prints a negative sign, and suppresses leading zeros.
	L	Prints a number without commas, prints a zero balance and a negative sign, and suppresses leading zeros.
	M	Prints a number without commas, suppresses a zero balance, prints a negative sign, and suppresses leading zeros.
	X	Performs no editing.
	Y	Edits a date field using the format mm/dd/yyyy or the format dd/mm/yyyy, if you specify inverted print. If the first digit of a date field is zero, it is suppressed.
	Z	Suppresses signs and leading zeros.

Rules

- If you use an edit code in column 38, columns 45 through 70 must be blank unless you specify an edit code modifier.
- If you use an edit code to edit an array, VAX RPG II leaves two spaces to the left between the elements of the array.
- You cannot use edit codes on numeric data in packed or binary format.

Additional Information

To prevent overlapping of the output fields, leave enough space for the characters so that the edit code will insert into the output field.

Unedited numeric output fields with negative values are output with the overpunched representation of the sign. For example, -1 will be output as J, -2 as K, and so on. See Chapter 14 for information on overpunched data format. Therefore, use an edit code or edit word to prevent the output of an overpunched representation of a sign.

Table 15–6 shows the results of several edit code examples.

Table 15–6: Edit Codes and Examples

Edit Code	+12345.67	+1234567	-1234.567	-1234567	Print Zero Balance
none	1234567	1234567	123456P	123456P	yes
1	12,345.67	1,234,567	1,234.567	1,234,567	yes
2	12,345.67	1,234,567	1,234.567	1,234,567	no
3	12345.67	1234567	1234.567	1234567	yes
4	12345.67	1234567	1234.567	1234567	no
A	12,345.67	1,234,567	1,234.567CR	1,234,567CR	yes
B	12,345.67	1,234,567	1,234.567CR	1,234,567CR	no
C	12345.67	1234567	1234.567CR	1234567CR	yes
D	12345.67	1234567	1234.567CR	1234567CR	no
J	12,345.67	1,234,567	1,234.567-	1,234,567-	yes

Table 15–6 (Cont.): Edit Codes and Examples

Edit Code	+12345.67	+1234567	-1234.567	-1234567	Print Zero Balance
K	12,345.67	1,234,567	1,234.567-	1,234,567-	no
L	12345.67	1234567	1234.567-	1234567-	yes
M	12345.67	1234567	1234.567-	1234567-	no

15.8.15 Blank After

Use column 39 to specify blank after, which causes VAX RPG II to reset the contents of the output field after writing it. VAX RPG II resets alphanumeric data with blanks and numeric data with zeros. Specifying blank after is especially useful when accumulating totals for each control group.

Column Number	Allowable Values	Explanation
39	B	Causes VAX RPG II to reset the field after writing it

Rules

- This column must be blank for look-ahead fields, fields defined by an EXTRN operation, constants, and the following special words: UDATE, UDAY, UMONTH, UYEAR, and *PLACE.
- If indicators condition the field you want to reset, the same indicators condition blank after.
- If you specify blank after for a field that you want to write more than once, enter B in this column on the last line specifying output for that field. Otherwise, the field will be reset before being output again.

Output Specification (O)

The following example specifies blank after for PAGE3:

```
                Type (HDTE)              Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)    | X                 ------------
                ||Space                  | Y date edit       Y  Y  1  A  J
                ||| Skip                 | Z zero suppress   Y  N  2  B  K
                |||  |                   |                   N  Y  3  C  L
                |||  |   Indicators      |Blank-after (B)    N  N  4  D  M
     File       |||  |   |        Field  ||End position
     name       |||  |   |        name   |||  Format (PB)
     |          |||  |   |        |      |||  |
    O|          ||BAB A NxxNxxNxx|      |||  |+ Constant or edit word  +
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         ***** * *          *      ***---**                        ....
     OOUT50A    D       N1P
                .
                .
                .
     O                          PAGE3 ZB  50
```

ZK-4506-85

15.8.16 End Position and Form

Use columns 40 through 43 to indicate the location of an output field or constant. Enter the number of the position for the rightmost character. For example, if a field contains 20 characters and you specify 40 as the end position, the output might appear as follows:

```
   0    |    1    |    2    |    3    |    4    |
12345678901234567890123456789012345678901234567890
                    South LyndeboroughNH
```

The numbers above this example are for reference only; they do not appear in the output.

Use K in column 42 if you are specifying a WORKSTN form name.

Column Number	Allowable Values	Explanation
40-43	1-9999	Indicates the position of the rightmost character in an output field or constant
42	K	Indicates WORKSTN form name begins in column 45.

Rules

- If fields overlap, the last field you specify on the Output specification is the only field that is completely written.
- When specifying the end position for an array, use the rightmost position of the last element in the array.
- The end position must be less than or equal to the record length (columns 24 through 27 of the File Description specification) of the file to which the record belongs.
- Right justify this entry.
- Leading zeros can be omitted.

Be sure to allow enough room for the number of characters in each output field and for the editing characters you specified using an edit code or edit word.

The following example shows the rightmost character in each of these positions:

- First field is in character position 22
- Second field is in character position 40
- Third field is in character position 57
- Fourth field is in character position 75

Output Specification (O)

```
                Type (HDTE)              Edit codes          , 0 No CR  -
                |Fetch ofl / Rel (FR)    | X                 ------------
                ||Space                  | Y date edit       Y  Y  1  A  J
                ||| Skip                 | Z zero suppress   Y  N  2  B  K
                ||| |                    |                   N  Y  3  C  L
                ||| |   Indicators       |Blank-after (B)    N  N  4  D  M
     File       ||| |   |        Field    ||End position
     name       ||| |   |        name     |||   Format (PB)
      |         ||| |   |         |       |||   |
     O|         ||BAB A NxxNxxNxx|       |||   |+ Constant or edit word  +
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     **         ***** * *         *      ***---**                        ....
     O                                        22 'EMPLOYEE NUMBER'
     O                                        40 'EMPLOYEE NAME'
     O                                        57 'REG EARNINGS'
     O                                        75 'OVER EARNINGS'
```

ZK-4507-85

The output appears as follows:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      EMPLOYEE NUMBER     EMPLOYEE NAME     REG EARNINGS     OVER EARNINGS
```

The numbers above this example are for reference only; they do not appear in the output.

15.8.17 Format

If an output field contains numeric data, use column 44 to specify overpunched decimal, packed decimal, or binary data format. The packed decimal and binary formats conserve disk space.

Column Number	Allowable Values	Explanation
44	Blank	Indicates that the field contains either alphanumeric characters or numeric data and is in overpunched decimal format
	P	Indicates that numeric data is in packed format
	B	Indicates that numeric data is in binary format

Leave this entry blank for the output field if you specify an edit code, edit word, or the special word *PLACE.

The following example specifies packed decimal format for the field QTYP and binary format for the field QTYB.

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)    | X                 ------------
                ||Space                  | Y date edit       Y Y 1 A J
                ||| Skip                 | Z zero suppress   Y N 2 B K
                ||| |                    |                   N Y 3 C L
                ||| |   Indicators       |Blank-after (B)    N N 4 D M
  File          ||| |   |        Field    ||End position
  name          ||| |   |        name     |||  Format (PB)
   |            ||| |   |         |       |||  |
  O|            ||BAB A NxxNxxNxx|       |||  |+ Constant or edit word +
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    **          ***** * *          *      ***---**                          ....
    O                              QTYP       32P
    O                              QTYB       38B
```

ZK-4508-85

15.8.18 Edit Code Modifiers, Constants or Form Names, and Edit Words

This section describes the options you can use to specify edit code modifiers, constants or form names, and edit words.

15.8.18.1 Edit Code Modifiers

Use columns 45 through 47 to specify edit code modifiers. Edit code modifiers can replace suppressed zeros to the left of the decimal point with asterisks (asterisk fill) or put a dollar sign ($) before the leftmost character (floating currency symbol). To specify these modifiers, enter the appropriate value described as follows.

Column Number	Allowable Values	Explanation
45-47	'*'	Replaces suppressed zeros to the left of the decimal point with asterisks (*).
	'Symbol'	Places the currency symbol before the first significant digit in a numeric field. The currency symbol is the same symbol you define in column 18 of the Control specification.

Rules

- Enclose edit code modifiers in apostrophes.
- The floating currency symbol will not be printed for a zero balance when you use an edit code that suppresses a zero balance.
- You cannot use the floating currency symbol or asterisk fill with simple (X, Y, and Z) edit codes.
- You can specify a currency symbol before an asterisk fill by making the following entries:
  - Column 38 (edit code)—specify one of the combined edit codes.
  - Place a currency symbol constant one space before the beginning of the edited field on the Output specification.
  - Place an asterisk enclosed in apostrophes ('*') in columns 45 through 47 on the same line as the edit code.

In the following example, if the field OTEARN, which is four digits long with two decimal positions, contains a zero balance, VAX RPG II prints a dollar sign before the asterisk fill.

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)    | X                 ----------
                ||Space                  | Y date edit       Y Y 1 A J
                ||| Skip                 | Z zero suppress   Y N 2 B K
                |||  |                   |                   N Y 3 C L
                |||  |  Indicators       |Blank-after (B)    N N 4 D M
  File          |||  |  |        Field   ||End position
  name          |||  |  |        name    |||  Format (PB)
   |            |||  |  |         |      |||  |
  O|            ||BAB A NxxNxxNxx|      |||  |+ Constant or edit word +
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
    **       ***** * *         *      ***---**                          ....
    O                                      56 '$'
    O                          OTEARN1     61 '*'
```

ZK-4509-85

The output might appear as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
                                                      $**.**
```

VAX RPG II uses a dollar sign ($) as the currency symbol unless you specify another symbol in column 18 (currency symbol) of the Control specification.

15.8.18.2 Constants or Form Names

Use columns 45 through 70 to specify constants or WORKSTN form names. Place a double quotation mark (") in column 45 to specify a long character literal as a constant or form name. Constants are used to describe constant data in an output file. A WORKSTN form name requires a K in column 42.

Column Number	Allowable Values	Explanation
45	Double quotation (")	Causes VAX RPG II to print the characters within single quotation marks on the line(s) that follow. All the rules for long character literals on Calculation specifications apply when used on Output specifications. See Section 15.7.5 for information on long character literals.
45-70	Any character	Causes VAX RPG II to print the characters in columns 45 through 70 for non-WORKSTN files. For WORKSTN files, the constant is used as a form name for a form to be displayed when column 42 contains K.

Rules

- Constants can contain up to 24 characters.
- You must enclose constants within single quotation marks ('). Use the keyboard apostrophe mark as the single quotation mark (for example, 'Subroutine'). The single quotation marks are not printed.
- When using constants, leave columns 32 through 39 and column 44 blank.
- To include an apostrophe in a constant, you must use two consecutive apostrophes to represent one apostrophe (for example, 'Subroutine''s calculations').

15.8.18.3 Edit Words

Use columns 45 through 70 to specify edit words. Edit words can be used to edit a numeric field. Edit words consist of three parts: the body, sign status, and expansion. The body is the portion of the edit word that provides space for the digits from the field to be edited. The body begins at the leftmost character position of the edit word and ends at the rightmost character position that is to contain a digit from the field to be edited.

The sign status is the portion of the edit word that is used to specify whether the field is positive or negative and to specify a constant, if needed. The sign status begins at the first character position to the right of the body and ends with CR or a negative sign (-). If you specify one of these symbols and the field is positive, blank spaces will be substituted in

the edited field. If you use CR or a negative sign and the field is negative, that symbol will be substituted in the edited field.

If an edit word contains no CR or a negative sign to the right of the rightmost character that is to contain a digit, the edit word does not contain a sign status portion.

The expansion consists of characters that will be printed regardless of the field's sign status. The expansion begins immediately after the sign status (or body, if no sign status is used) and continues to the end of the edit word.

The following table describes those characters you can use in the body of the edit word.

Column Number	Allowable Values	Explanation
45-70	Blank	Indicates that the position in the edited field is to contain the digit from the same position in the numeric field.
	0	Indicates that the field is to be zero-suppressed. Place the zero in the rightmost position where zero-suppression is to stop. Each leading zero that appears to the left of and including the stop position of the numeric field is replaced with a blank space in the edited field. The first zero VAX RPG II encounters is the zero-suppression character. Any zero appearing after the first zero is treated like any other character. Zero-suppression begins at the leftmost position in the data and continues up through the stop position unless a nonzero digit is encountered to the left of the stop position. If VAX RPG II encounters a nonzero digit to the left of the stop position, zero-suppression stops at the position where the nonzero digit is encountered; that digit and all following digits to the right of the nonzero digit are printed.

Column Number	Allowable Values	Explanation
	*	Indicates that the field is to be edited using asterisk protection. Place the asterisk in the rightmost position where asterisk protection is to stop. Each leading zero that appears in the data to the left of and including the stop position is replaced with an asterisk. The first asterisk VAX RPG II encounters is the asterisk protection character. Any asterisk appearing after the first asterisk is treated like any other character.
	&	Indicates that the position in the edited field is to be a blank space.
	Symbol	Prints the currency symbol. If the currency symbol appears in the body of the edit word immediately to the left of the zero suppression, it is printed immediately to the left of the first significant digit in the edited field. This type of currency symbol is called a floating currency symbol. A floating currency symbol cannot be used with asterisk protection. The currency symbol in the leftmost position of the edit word indicates that the dollar sign is to be printed in that exact position in the edited field. This type of currency symbol is called a fixed currency symbol.
45-70	Decimal point or comma	Indicates the exact position in the edited field where it is to be printed. If a decimal point or comma appears to the left of the most significant digit, VAX RPG II will replace it with a blank space or, if asterisk protection is specified, with an asterisk.
45-70	Any other character	Prints the characters in the edited fields, if the position is to the right of the most significant digit in the edited field. Any character to the left of the most significant digit in the edited field is replaced with a blank space or an asterisk if asterisk protection is specified.

The following example shows both the floating and fixed currency symbol types:

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)  | X                   ----------
                ||Space                | Y date edit         Y Y 1 A J
                ||| Skip               | Z zero suppress     Y N 2 B K
                ||| |                  |                     N Y 3 C L
                ||| |   Indicators     |Blank-after (B)      N N 4 D M
 File           ||| |   |        Field ||End position
 name           ||| |   |        name  |||  Format (PB)
  |             ||| |   |        |     |||  |
 O|             ||BAB A NxxNxxNxx|     |||  |+ Constant or edit word +
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
 **             ***** * *         *     ***---**                          ....
 O                               FLOAT     45 '$0    '
 O                               FIXED     45 '$      '
```

ZK-4510-85

In the example, if FLOAT and FIXED contain the characters 1234, the output appears as follows:

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
                                                  $1234
                                               $  1234
```

In the following example, VAX RPG II prints a comma before the fifth digit from the right and a decimal point before the rightmost two digits:

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)  | X                   ----------
                ||Space                | Y date edit         Y Y 1 A J
                ||| Skip               | Z zero suppress     Y N 2 B K
                ||| |                  |                     N Y 3 C L
                ||| |   Indicators     |Blank-after (B)      N N 4 D M
 File           ||| |   |        Field ||End position
 name           ||| |   |        name  |||  Format (PB)
  |             ||| |   |        |     |||  |
 O|             ||BAB A NxxNxxNxx|     |||  |+ Constant or edit word +
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
 **             ***** * *         *     ***---**                          ....
 O                               FLD       45 '$ ,   . '
```

ZK-4511-85

In the preceding example, if FLD contains the data 123456, the output appears as follows:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
                                    $1,234.56
```

In the preceding example, if FLD contains the data 56, the output appears as follows:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
                                    $      56
```

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)   | X                  -------------
                ||Space                 | Y date edit        Y  Y  1  A  J
                ||| Skip                | Z zero suppress    Y  N  2  B  K
                ||| |                   |                    N  Y  3  C  L
                ||| |   Indicators      |Blank-after (B)     N  N  4  D  M
 File           ||| |   |       Field   ||End position
 name           ||| |   |       name    |||   Format (PB)
  |             ||| |   |       |       |||   |
 O|             ||BAB A NxxNxxNxx|       |||   |+ Constant or edit word  +
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
 **       ***** * *          *      ***---**                          ....
 O                           FLD         45 '$ ,    .  &BALANCE'
```

ZK-4512-85

In the preceding example, if FLD contains the data 123456, the output appears as follows:

```
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
12345678901234567890123456789012345678901234567890123456789012345678901234567890
                                $1,234.56 BALANCE
```

The following table describes those characters you can use in the status portion of the edit word.

Column Number	Allowable Values	Explanation
45-70	CR or -	Indicates that the specified symbol (CR or a negative sign (-)) is to be printed in the edited field if the data is negative. If the edited field is positive, the specified symbol is replaced by blanks.
	Any character	Prints the specified character(s) in the edited field if the data is negative; otherwise the character is replaced by a blank. If an ampersand (&) is specified, it will be replaced by a blank space.

```
                Type (HDTE)             Edit codes          , 0 No CR -
                |Fetch ofl / Rel (FR)   | X                 ------------
                ||Space                 | Y date edit       Y Y 1 A J
                ||| Skip                | Z zero suppress   Y N 2 B K
                ||| |                   |                   N Y 3 C L
                ||| |  Indicators       |Blank-after (B)    N N 4 D M
  File          ||| |  |        Field    ||End position
  name          ||| |  |        name     |||  Format (PB)
   |            ||| |  |        |        |||  |
  O|            ||BAB A NxxNxxNxx|       |||  |+ Constant or edit word +
    0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
    **        ***** * *         *       ***---**                      ....
    O                           FLD         45 '$ ,   .  &CR&BALANCE'
```

ZK-4513-85

In the preceding example, if FLD contains the data -123456, the output appears as follows:

```
    0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
                              $1,234.56 CR BALANCE
```

Output Specification (O)

Rules

- Leave column 38 (edit code) blank.
- You must complete columns 32 through 37 (field name) and columns 40 through 43 (end position).
- Edit words can be used only with overpunched numeric data. (Column 44 (data format) must be blank.)
- Edit words can be up to 24 characters long.
- Enclose edit words in apostrophes.
- The number of replaceable characters in an edit word must be greater than or equal to the number of digits in the numeric field.
- If you want all leading zeros to be printed, increase the edit word by one position to the left of the leftmost digit and place a zero in that position.
- When the floating currency symbol is used, the sum of the number of blanks and the zero-suppression in the edit word must be equal to or greater than the number of digits in the edited field. A floating currency symbol is not counted as a digit position.
- Any zeros or asterisks following the leftmost zero suppression code or asterisk protection are treated as constants and are not replaced with digits.

Chapter 16

Operation Codes

VAX RPG II operation codes perform calculations on the operands you specify in Calculation specifications. In the following sections, operation codes are grouped by function and discussed individually in detail. A summary of all operation codes is provided at the end of this chapter, in Table 16–1.

16.1 Arithmetic Operation Codes

This section describes arithmetic operation codes which perform a variety of functions, ranging from adding two operands to taking the square root of an operand.

When you use arithmetic operation codes, you must consider the following restrictions and default characteristics:

- You can use arithmetic operation codes only with numeric fields and numeric literals.
- VAX RPG II aligns the operands according to their decimal points before performing any arithmetic operation. VAX RPG II aligns the result on the decimal point in the result field, which could cause truncation.
- The contents of factor 1 and factor 2 do not change during an arithmetic operation unless the same field is used as the result field.
- Any existing data in the result field is replaced with the result of the current operation.
- Make sure the field length of the result field is large enough to hold the result of the operation. Otherwise, the result of the operation is truncated before being placed in the result field.

- You can specify half adjust (column 53 of the Calculation specification) for any arithmetic operation except an MVR operation and the DIV operation immediately preceding it.
- You can specify the same field for factor 1 and factor 2 and/or the result field, if desired.
- You can leave factor 1 blank. If you do, the statement is treated as if the result field were specified in factor 1.
- You can specify an entire array as an operand of the ADD, SUB, Z-ADD, Z-SUB, MULT, DIV, and SQRT operation codes. See Chapter 11 for information on using arrays in calculations.
- No field in an arithmetic operation can be longer than 15 digits.
- VAX RPG II performs all arithmetic operations algebraically.
- The result of all arithmetic operations is signed. The sign of the result of an arithmetic operation depends on the operation.

  Addition:

  — If factor 1 and factor 2 have like signs, the result field has the same sign.

  — If factor 1 and factor 2 have unlike signs, the result field uses the sign of the factor with the largest absolute value.

  Subtraction:

  — Change the sign of factor 2 (positive to negative or negative to positive) and use the same rules as for addition.

  Multiplication:

  — If factor 1 and factor 2 have like signs, the sign of the result field is positive.

  — If factor 1 and factor 2 have unlike signs, the sign of the result field is negative.

  Division:

  — If factor 1 and factor 2 have like signs, the sign of the result field is positive.

  — If factor 1 and factor 2 have unlike signs, the sign of the result field is negative.

  — The sign of the remainder is the same as the sign of factor 1.

16.1.1 ADD Operation

The ADD operation adds the contents of factor 1 to factor 2 and puts the sum in the result field. If you leave factor 1 blank, the statement is treated as if the result field were specified in factor 1.

16.1.2 Z-ADD Operation

The Z-ADD operation assigns the value of factor 2 to the result field.

16.1.3 SUB Operation

The SUB operation subtracts the contents of factor 2 from the contents of factor 1 and puts the difference in the result field. If you leave factor 1 blank, the statement is treated as if the result field were specified in factor 1.

16.1.4 Z-SUB Operation

The Z-SUB operation multiplies the contents of factor 2 by -1 and puts the result in the result field.

16.1.5 MULT Operation

The MULT operation multiplies factor 1 by factor 2 and puts the product in the result field. If you leave factor 1 blank, the statement is treated as if the result field were specified in factor 1.

The field length of the result field for a MULT operation should equal the sum of the field lengths of factor 1 and factor 2. This procedure ensures that the result field can contain the maximum value.

16.1.6 DIV Operation

The DIV operation divides factor 1 by factor 2 and puts the quotient in the result field. If you leave factor 1 blank, the statement is treated as if the result field were specified in factor 1.

Factor 2 cannot be zero. If it is, a run-time error occurs. The remainder is lost unless you use the MVR operation immediately following the DIV operation.

16.1.7 MVR Operation

MVR moves the remainder from the division operation on the preceding line to the result field. The decimal position of the remainder is the greater of either of the following:

1. The number of decimal positions specified for factor 1
2. The sum of the number of decimal positions specified for factor 2 and the result field of the preceding DIV operation

The sign of the remainder is the same as the sign of factor 1 in the DIV operation.

Because DIV and MVR operation codes work together, use the same indicators to condition both operations.

You cannot specify half adjust (column 53 of the Calculation specification) for a DIV operation immediately followed by an MVR operation.

You cannot use the MVR operation if, in the immediately preceding DIV operation, you specified an entire array (nonindexed) in the result field.

16.1.8 SQRT Operation

The SQRT operation calculates the square root of factor 2, half adjusts the value, and puts the result into the result field. The result of a SQRT operation is always half adjusted. Factor 2 cannot be a negative number. If the field contains a negative number, a run-time error occurs. If you use a negative numeric literal, a compile-time error occurs.

16.1.9 XFOOT Operation

The XFOOT operation puts the sum of all the array elements into the result field. Factor 2 contains the name of the array. If the result field contains an array element of the array you specify in factor 2, the original value of the element is used during the operation.

You can half adjust the contents of the result field.

16.1.10 Example

The following example demonstrates the use of arithmetic operation codes:

```
                                                    Field length
      Control level                                 |  Decimal positions
      |                                             |  |Half adjust (H)
      | Indicators          Operation               |  ||
      | | |                 |                       |  ||Resulting
      | | |        Factor   |    Factor    Result|  |||indicators
      | | |        1        |    2         field |  ||+ - 0
    C| NxxNxxNxx|           |    |         |     |  ||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *         *    *         *     *--*** * * *
  12 C           PURCH     SUB  DWNPAY    AMTFIN  62H
  13 C           AMTFIN    MULT .18       FINCHG  62H
  14 C           FINCHG    ADD  AMTFIN    AMTDUE  62H
  15 C           AMTDUE    DIV  13        MTHPAY  52
  16 C                     MVR            REMAIN  42
  17 C                     Z-ADD5.00      TOTDUE  62
  18 C                     ADD  AMTDUE    TOTDUE
       .
       .
       .
  30 C                     Z-SUB10.00     EARPAY  42H
  31 C                     Z-ADD6.99      LATCHG  32H
       .
       .
       .
  46 C                     SQRT FINCHG    TAX     42H
```

ZK-4514-85

For the preceding example, the following table lists the data in each operand before and after the operation.

Program line	Factor 1	Operation	Factor 2	Result field
12	122.99	-	100.00	22.99
13	1200.00	*	.18	216.00
14	216.00	+	1200.00	1416.00
15	1416.00	/	13	108.92
16		MVR		0.04
18		ADD	1416.00	1421.00
30		Z-SUB	10.00	-10.00
31		Z-ADD	6.99	6.99
46		SQRT	216.00	14.70

16.2 Move Operation Codes

MOVE operation codes transfer data from a field in factor 2 to the result field. Although the contents of factor 2 remain unchanged, you can move all or part of the field in factor 2 and either retain or change the format of the data as you move it.

In move operations, VAX RPG II ignores the decimal positions of numeric fields. You cannot use resulting indicators with any move operation.

16.2.1 MOVE Operation

The MOVE operation transfers the contents of factor 2 to the result field. The transfer begins with the rightmost character of factor 2 to the rightmost character of the result field. If the result field is not large enough to accommodate factor 2, VAX RPG II moves only enough characters (beginning with the rightmost character) to fill the result field. If the field length of the result field is longer than factor 2, the leftmost characters of the result field are not changed. If VAX RPG II transfers numeric data, the sign of the result field is the same as the sign of factor 2.

When you move an alphanumeric field to a numeric field, VAX RPG II converts the digit portion of each character to its corresponding numeric character and then moves the numeric character to the result field. VAX RPG II converts the zone portion of the rightmost character to its corresponding sign and then moves the sign to the rightmost character position of the numeric result field, where it becomes the sign of that field.

16.2.2 MOVEA Operation

The MOVEA operation transfers data from factor 2 to the result field. Either factor 2 or the result field must contain an array or array element. If you specify an array element, it specifies the beginning position of the transfer. Both factor 2 and the result field must be character fields or arrays.

You can move several contiguous array elements to a single field or move a single field to several contiguous array elements.

Movement of data from factor 2 to the result field begins with one of the following:

1. The leftmost character of the first element in the array, if you specify an entire array (nonindexed)
2. The leftmost character of the element you specify, if you specify an array element (indexed)
3. The leftmost character of the field, if you specify a field

The length of factor 2 and the result field is determined by the length of one of the following:

1. An entire array, if you specify an entire array (nonindexed)
2. An array from the specified array element to the end of the array, if you specify an array element (indexed)
3. A field, if you specify a field

If the field length of factor 2 is greater than the field length of the result field, VAX RPG II does not move the excess rightmost characters. If the field length of the result field is greater than the field length of factor 2, the rightmost characters in the result field remain unchanged.

Array element boundaries are ignored in a MOVEA operation. Therefore, movement of data into the result field can end in the middle of an array element.

16.2.3 MOVEL Operation

The MOVEL operation transfers the contents of factor 2 to the result field. The transfer begins with the leftmost character of factor 2 to the leftmost character of the result field.

When the field length of factor 2 is equal to the field length of the result field, the following rules apply:

- If factor 2 contains alphanumeric data and the result field is alphanumeric, VAX RPG II moves characters without changing them.
- If factor 2 contains numeric data and the result field is numeric, the sign of factor 2 becomes the sign of the result field.
- If factor 2 contains numeric data and the result field is alphanumeric, VAX RPG II moves the sign in the rightmost character position.
- If factor 2 contains alphanumeric data and the result field is numeric, each character is converted to its corresponding numeric digit and moved to the result field. The zone portion of the rightmost character in factor 2 is used to determine the sign of the result field.

When the field length of factor 2 is longer than the field length of the result field, the following rules apply:

- If factor 2 contains alphanumeric data and the result field is alphanumeric, VAX RPG II moves only the number of characters needed to fill the result field.
- If factor 2 contains numeric data and the result field is numeric, the sign of factor 2 becomes the sign of the result field.
- If factor 2 contains numeric data and the result field is alphanumeric, the result field contains only numeric data; that is, the sign of factor 2 is not used.
- If factor 2 contains alphanumeric data and the result field is numeric, the leftmost characters of factor 2 are converted to the corresponding numeric digits and moved to the result field. The zone portion of the rightmost character in factor 2 is used to determine the sign of the result field.

When the field length of factor 2 is shorter than the field length of the result field, the following rules apply:

- If factor 2 contains either numeric or alphanumeric data and the result field is numeric, VAX RPG II moves the digits of numeric fields or the corresponding numeric digits of factor 2, if alphanumeric, into the

leftmost character positions of the result field. The sign of the result field remains unchanged.

- If factor 2 contains either numeric or alphanumeric data and the result field is alphanumeric, VAX RPG II moves the data into the result field beginning with the leftmost character position. The rightmost character positions in the result field remain unchanged.

16.2.4 Example

In the following example, the preexecution-time array ARR1 is read from the input file ARRFILE and is copied to the execution-time array DUPARR. The array is modified by moving the input field INPFLD to the second and third elements of the array. In addition, the field MYREC consists of the first element in ARR1 and the last three characters of the third element in ARR1.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
      .
      .
      .
E    ARRFILE         ARR1    6   6  5
E                    DUPARR      6  5
IARRFILE AA  01
I                                        1  10 INPFLD
C                     MOVEAARR1      DUPARR
C                     MOVEAINPFLD    ARR1,2
C                     MOVE ARR1,3    MYREC   8
C                     MOVELARR1,1    MYREC
```

ZK-4515-85

For the preceding example, the input file (ARR1) contains the following data:

Field	Data
ARR1,1	12345
ARR1,2	67890
ARR1,3	ZZZZZ
ARR1,4	ZZZZZ

Field	Data
ARR1,5	ZZZZZ
ARR1,6	ZZZZZ
INPFLD	ABCDEFGHIJ

After executing the program, the fields would contain the following data:

```
MYREC = 12345HIJ
ARR1,1 =  12345
ARR1,2 =  ABCDE
ARR1,3 =  FGHIJ
ARR1,4 =  ZZZZZ
ARR1,5 =  ZZZZZ
ARR1,6 =  ZZZZZ
DUPARR =  1234567890ZZZZZZZZZZZZZZZZZZZZ
```

16.3 SET Operation Codes

SET operation codes (SETON and SETOF) set indicators on and off. They affect only those indicators in columns 54 through 59.

If you use SET operation codes to set control-level indicators on and off, they do not affect lower-level control-level indicators.

16.3.1 SETON Operation

The SETON operation sets on the indicators you specify in columns 54 and 55, 56 and 57, and 58 and 59. You cannot set the first-page (1P) or matching-record (MR) indicators on.

If you use SETON to set the LR indicator on at total time, processing stops after VAX RPG II finishes total-time output operations.

If you use SETON to set the LR indicator on at detail time, processing stops after VAX RPG II finishes the next total-time output operation.

If you use SETON to set halt indicators on and they are not set off before VAX RPG II finishes detail-time output operations, processing stops.

16.3.2 SETOF Operation

The SETOF operation sets off the indicators you specify in columns 54 and 55, 56 and 57, and 58 and 59. You cannot set the first-page (1P) or matching-record (MR) indicators off.

In the following example:

- The SETON operation sets indicators 11 and 22 on.
- The SETOF operation sets indicator 33 off.

```
                                                    Field length
     Control level                                  | Decimal positions
     |                                              | |Half adjust (H)
     | Indicators          Operation                | ||
     | |                   |                        | ||Resulting
     | |        Factor     |    Factor    Result|   | |||indicators
     | |        1          |    2         field |   | |||+ - 0
   C| NxxNxxNxx|           |    |         |     |   | |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *        *          *    *         *      *--*** * * *
     C                      SETON                      1122
     C                      SETOF                      33
```

ZK-4516-85

16.4 Subroutine Operation Codes

Subroutine operation codes are used to identify and execute subroutines. A subroutine is a group of Calculation specifications that you can execute more than once in a single program cycle using the ESXR operation code.

You can specify SR in columns 7 and 8 to indicate that the specification is part of a subroutine, although this is optional. You cannot use control-level indicators in columns 7 and 8 of a subroutine. However, you can use any indicator in columns 9 through 17. Also, you can use AN and OR in columns 7 and 8 to specify a relationship between two program lines.

You can use a maximum of 254 subroutines in a program. Subroutines must be placed after all other calculations. Subroutines cannot be nested or recursive. However, you can use the EXSR operation to call one subroutine from another.

16.4.1 BEGSR Operation

The BEGSR operation indicates the beginning of a subroutine and must be the first specification in a subroutine. Factor 1 contains the name of the subroutine. All other columns in the same specification must be left blank except for an optional SR in columns 7 and 8.

16.4.2 ENDSR Operation

The ENDSR operation indicates the end of a subroutine and must be the last specification in a subroutine. Factor 1 can contain a label for a GOTO operation within the subroutine. All other columns in this specification must be blank, except an optional SR in columns 7 and 8.

After the program reaches the ENDSR operation, it returns program control to the specification immediately following the EXSR operation code that invoked the subroutine.

16.4.3 EXSR Operation

The EXSR operation executes a subroutine. Factor 2 contains the name of the subroutine. It must be the same name you used in factor 1 of the BEGSR operation. You can use control-level and conditioning indicators to condition EXSR.

After the program performs the operations in the subroutine, control branches to the specification immediately following EXSR.

16.4.4 Example

In the following example, line 11 causes VAX RPG II to execute the subroutine SUB1. The subroutine consists of lines 22 through 24.

```
                                                    Field length
      Control level                                 |  Decimal positions
      |                                             |  |Half adjust (H)
      | Indicators          Operation               |  ||
      | |                   |                       |  ||Resulting
      | |        Factor     |    Factor    Result|  ||indicators
      | |        1          |    2         field |  ||+ - 0
     C| NxxNxxNxx|          |    |         |     |  ||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
     ** *         *         *    *         *     *--*** * * *
 11 C   02                  EXSR SUB1
        .
        .
        .
 22 CSR             SUB1    BEGSR
 23 CSR 02          HOURS   MULT RATE       DAYPAY  52H
 24 CSR                     ENDSR
```

ZK-4517-85

16.5 BIT Operation Codes

The BIT operation codes set and test bits. You must use one-character alphanumeric fields in factor 2 and the result field.

16.5.1 BITON Operation

The BITON operation sets on the bits you specify in factor 2 in the result field, replacing the value in the result field. Factor 2 contains the source of bits in bit numbers or a field name.

You can set on bit numbers 0 through 7. Zero is the leftmost bit. You must enclose the bit number in apostrophes. For example, to set bits 1, 2, and 3 on, enter '123' in factor 2. You cannot specify a bit number more than once.

The field name is a one-character alphanumeric field, table, or array element. The bits that are on in the field name are set on in the result field. If you specify an array element, each array element must be a one-character field.

You can use indicators in columns 7 through 17, but the following columns must be left blank:

- Columns 18 through 27 (factor 1)
- Column 52 (decimal positions)
- Column 53 (half adjust)
- Columns 54 through 59 (resulting indicators)

16.5.2 BITOF Operation

The BITOF operation sets off the bits you specify in factor 2 in the result field, replacing the value in the result field. To specify operands for BITOF, follow the same guidelines as for BITON.

16.5.3 TESTB Operation

The TESTB operation compares the bits in factor 2 with the corresponding bits in the result field. Factor 2 can contain bit numbers or a one-character alphanumeric field. Bit numbers and one-character alphanumeric fields follow the same rules as those for BITON and BITOF.

Indicators in columns 54 through 59 reflect the status of the bits in the result field; therefore, you must assign at least one resulting indicator. You can set up to three resulting indicators, but no more than two resulting indicators can be identical.

If factor 2 is a field in which all bits are off, no resulting indicator is set on; otherwise, indicators in columns 54 through 59 indicate the result of the comparison as follows:

- VAX RPG II sets the indicator in columns 54 and 55 on, if all bits specified in factor 2 in the result field are off.
- VAX RPG II sets the indicator in columns 56 and 57 on, if some bits specified in factor 2 in the result field are on and some are off.
- VAX RPG II sets the indicator in columns 58 and 59 on, if all bits specified in factor 2 in the result field are on.

You can use indicators in columns 7 through 17, but the following columns must be left blank:

- Columns 18 through 27 (factor 1)
- Column 52 (decimal positions)
- Column 53 (half adjust)

16.5.4 Example

In the following example:

- Line 34 sets on the bits 1, 2, and 3 in the result field FLD1.
- Line 35 tests the bits 1, 2, and 3 in the result field FLD1. If all the bits are on, indicator 11 is set on. If one or more of the bits are off, indicator 11 is set off.
- Line 36 sets off the bits 4, 5, and 6 in the result field FLD2.
- Line 37 tests the bits 4, 5, and 6 in the result field FLD2. If all the bits are off, indicator 22 is set on. If one or more of the bits are on, indicator 22 is set off.

```
                                                    Field length
      Control level                                 | Decimal positions
      |                                             | |Half adjust (H)
      | Indicators          Operation               | ||
      | |                   |                       | ||Resulting
      | |        Factor     |    Factor    Result   | ||indicators
      | |        1          |    2         field    | ||+ - 0
     C| NxxNxxNxx|          |    |         |        | ||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *          *      *--*** * * *
  34 C                      BITON'123'      FLD1
  35 C                      TESTB'123'      FLD1           11
  36 C                      BITOF'456'      FLD2
  37 C                      TESTB'456'      FLD2        22
```

ZK-4518-85

16.6 COMP Operation Code

The COMP operation tests fields for certain conditions. Based on the result of the comparison, you can assign resulting indicators to condition calculation and output operations.

The COMP operation compares the contents of factor 1 to the contents of factor 2. An indicator in columns 54 through 59 indicates the result of the comparison as follows:

- If factor 1 is greater than factor 2, VAX RPG II sets on the indicator in columns 54 and 55.
- If factor 1 is less than factor 2, VAX RPG II sets on the indicator in columns 56 and 57.
- If factor 1 is equal to factor 2, VAX RPG II sets on the indicator in columns 58 and 59.

You must specify at least one resulting indicator. The result field must be left blank.

When using the COMP operation, consider the following restrictions and default characteristics:

- If you compare numeric fields, the fields are aligned at their implied decimal point. Fields are filled with zeros to the left and right of the decimal point until both fields are equal in length. For example, if you compare 1234.56 to 1.2, VAX RPG II fills the second field (1.2) with zeros (0001.20) until both fields are equal in length.
- If you compare alphanumeric fields of unequal lengths, the fields are aligned at the leftmost character. Shorter fields are filled with blanks until the two fields are equal in length.
- VAX RPG II compares numeric fields algebraically.
- Positive numeric fields are greater than negative numeric fields.
- If you have specified an alternate collating sequence, VAX RPG II translates character fields to the alternate collating sequence before comparing them.
- You cannot compare an alphanumeric field to a numeric field.
- You cannot compare entire arrays (nonindexed).

In the following example, if the contents of the field CODE are greater than 1, VAX RPG II sets on indicator 11 and sets off indicators 22 and 33. If the contents of CODE are less than 1, VAX RPG II sets on indicator 22 and sets off indicators 11 and 33. If CODE is equal to 1, VAX RPG II sets on indicator 33 and sets off indicators 11 and 22.

```
                                                          Field length
     Control level                                        | Decimal positions
     |                                                    | |Half adjust (H)
     | Indicators          Operation                      | ||
     | |                   |                            | ||Resulting
     | |         Factor    |    Factor    Result|       | |||indicators
     | |         1         |    2         field |       | |||+ - 0
    C| NxxNxxNxx|         |    |         |     |       | |||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *         *    *         *    *--*** * * *
     C             CODE      COMP '1'              112233
```

ZK-4519-85

16.7 Input and Output Operation Codes

You can use the following operation codes to alter the normal input and output sequence, enabling the program to read and write records during calculations.

16.7.1 CHAIN Operation

The CHAIN operation reads a record from a file during calculations and places the contents of the record into the fields you specify on the Input specification. You can read records randomly from a sequential, direct, or indexed file.

Rules

- If you want to read a record from a sequential or direct file, factor 1 must contain the relative record number of that record. If you want to read a record from an indexed file, factor 1 must contain a field name or a literal that is the key of that record. The field length of the field or literal specified in factor 1 must be the same as the field length of the key.

- Factor 2 contains the name of the file from which the record is read. This file must be the same file you describe with a C or an F in column 16 (type) in the File Description specification.
- You can use any indicator in columns 7 through 17, but columns 43 through 53 and 56 and 57 must be left blank. If you condition the chained or full-procedural file with an external indicator, use the same indicator to condition the CHAIN operation.
- You can specify an indicator in columns 54 and 55 to verify the CHAIN operation. If VAX RPG II cannot locate the record, it sets on the indicator in these columns. If you do not use an indicator in columns 54 and 55, and VAX RPG II cannot locate the record, a run-time error occurs. If VAX RPG II cannot locate the record, you can add a record to the chained file (if you use a resulting indicator to indicate that a record has not been found), but you cannot update the record.
- You can specify on a CHAIN operation that if a record is locked, to set on an indicator. Enter the indicator for a locked record in columns 58 and 59 of a Calculation specification. If you specify an indicator in columns 58 and 59, the program will not wait for the record to become unlocked before proceeding and will set on the indicator to show that the requested record is locked. If you do not specify the indicator, the program will wait until any record lock is released before proceeding. This indicator is allowed only on CHAIN operations to files that are marked as SHARE (S or R in column 68 on File specifications). The file cannot be an output file. Note that if another program has locked a record for update, but uses the file sharing option R, then a CHAIN operation that accesses that record will be successful; no lock will be seen and an indicator in columns 58 and 59 will be set off.
- If you chain to a file with packed keys, the field in factor 1 of the CHAIN operation must be numeric and have the same number of digits as the key in the chained or full-procedural file. Packed key fields can be up to eight bytes long.
- If you use one or more chained or full-procedural files during the same program cycle and the previous CHAIN operation was successful, any record-identifying indicators you use remain on throughout the cycle. If you use a chained file more than once during the same program cycle, only the last record processed can be updated during output, unless you specify exception output for each CHAIN operation.
- The CHAIN operation is also used to load a direct file (a chained output file). Use the CHAIN operation to position the file to the record you want to add to the file.

See Chapter 8 for information on processing files.

In the following example:

- Line 33 retrieves the record from the input file FILE1 with the relative record number specified in the field RECNO. If the record is not found, VAX RPG II sets on indicator 11. If the record is found, VAX RPG II sets off indicator 11.
- Line 34 branches to a TAG operation to terminate the program if the previous CHAIN operation causes a record-not-found error.
- Line 55 retrieves the record with the key 761 from the indexed file FILE2. If a record with a key of 761 does not exist, VAX RPG II sets on indicator 22. If a record with a key of 761 does exist, VAX RPG II sets off indicator 22.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FFILE1   IC  F      80                DISK
     FFILE2   IC  F      80  5AI    10 DISK
          .
          .
          .
 33 C             RECNO     CHAINFILE1              11
 34 C   11                   GOTO END
          .
          .
          .
 55 C             761       CHAINFILE2              22
```

ZK-4520-85

16.7.2 DSPLY Operation

The DSPLY operation allows you to display on line up to 511 characters at run time. VAX RPG II can do the following:

- Display up to 511 characters from a field without suspending program execution. Factor 1 names the field to display.
- Display the number of characters up to one less than your screen width from the result field. The program suspends execution after displaying the result field. The cursor is positioned at the next line where you can enter a new value for the result field from the terminal.

When entering a new value for the result field, terminate the input by pressing either the RETURN key or the TAB key. If you press only a RETURN key or a TAB key for the new value of the result field, the data in the result field remains the same.

When using the DSPLY operation, observe the following restrictions and default characteristics:

- You cannot change the contents of a literal; therefore, do not specify a literal in the result field.
- The maximum length of the result field is one character less than the screen width.

When entering data for the result field, consider the following characteristics:

- You do not need to fill numeric data with leading zeros.
- Numeric data is aligned on the decimal point when entered into the result field.
- Alphanumeric data is left-justified when entered into the result field.
- If you enter no characters and press either the RETURN key or the TAB key, the value in the result field remains unchanged.

16.7.3 Example

In the following example, if the data in NUMBER is greater than 100.0, the number will be displayed on the screen and will be followed on the next line by the current value of RESNO. Then, you can enter a new value for RESNO and press the RETURN key.

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     FTTYFILE D   F      81            TTY
     IINFILE  AA  01
     I                                        1   50NUMBER
     C   01       NUMBER    COMP 100.0              10
     C   10       NUMBER    DSPLYTTYFILE   RESNO    50
```

ZK-4521-85

16.7.4 EXCPT Operation

The EXCPT operation allows you to write a variable number of records during detail-time or total-time calculations or to display a form with a WORKSTN file. To do this you must specify the following entries:

- On the Calculation specification:
  - EXCPT, as the operation code
  - Blanks or an EXCPT name in factor 2
- On the Output specification:
  - E in column 15 (Type), for the record you want to write
  - Blanks or an EXCPT name in columns 32 through 37 (field name)

The EXCPT operation writes those records that have an E in column 15 of the Output specification and that satisfy the conditions specified by the conditioning indicators. In addition, if the EXCPT operation has a blank factor 2, only exception records with blanks in columns 32 through 37 of the Output specification will be written; if the EXCPT operation has an EXCPT name in factor 2, only these exception records with the same name in columns 32 through 37 of the Output specification will be written.

You can use indicators in columns 7 through 17 of the Calculation specification. Factor 2 can contain blanks or an EXCPT name. All other columns must be blank.

An EXCPT name can be used on multiple EXCPT output record lines. Only exception records, not heading, detail, or total records, can contain an EXCPT name.

In the following example, line 22 tells VAX RPG II to write the record described in line 89 during calculations if indicators 01, 02, and 03 are on. Line 34 tells VAX RPG II to write the record described in line 95 if indicator 04 is on.

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
 22 C                    EXCPT
    ...
 34 C                    EXCPTHDG
    ...
 77 OOUTFIL
    ...
 89 O       E        01 02 03
    ...
 95 O       E        04      HDG
```

ZK-4660-85

16.7.5 FORCE Operation

The FORCE operation allows you to select the next file from which a record is read during multifile processing. You can select primary or secondary input and update files. Factor 2 contains the name of the file.

You can use conditioning indicators, but all other columns must be left blank.

When VAX RPG II executes a FORCE operation it reads, at the next program cycle, a record from the file you specify. If you specify more than one FORCE operation during the same program cycle, VAX RPG II ignores all FORCE operations except the last.

Although FORCE operations override normal multifile processing, they cannot override the first record selected by the program. Reading the first record occurs in the first cycle before the first pass through calculations.

If a FORCE operation is issued for a file that is at its end-of-file, the file is not selected for processing. In this case, normal multifile processing logic selects the next record.

In the following example, VAX RPG II reads the next record from the file SPECFIL at the next program cycle if indicators 01, 02, and 03 are on.

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |        Factor     |    Factor    Result|  |||indicators
     | |        1          |    2         field |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |  |||> < = +- Comments --+
   0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789
     ** *         *         *    *         *     *--*** * * *
  33 C   01 02 03           FORCESPECFIL
```

ZK-4523-85

16.7.6 READ Operation

The READ operation causes VAX RPG II to read a record from a demand or full-procedural file. Factor 2 contains the name of the file from which a record is read. You can read a record from the following types of input and update files:

- Sequential and direct disk files processed consecutively
- Indexed disk files processed sequentially by key or by limits

You can use any indicators in columns 7 through 17 and in columns 58 and 59. VAX RPG II sets on the indicator in columns 58 and 59 when an end-of-file condition occurs, or for each READ operation following an end-of-file condition. If there is no indicator in columns 58 and 59, a run-time error occurs. Columns 18 through 27 and 43 through 57 must be left blank, except for WORKSTN files, which can have an indicator in columns 56 and 57 to detect an error. See Chapter 6 for information on WORKSTN files.

If VAX RPG II does not open a file because an external indicator is off, a READ operation on the file causes an end-of-file condition to occur.

In the following example, VAX RPG II reads the next record from the file SPECFIL if indicator 99 is off. If an end-of-file condition occurs, VAX RPG II sets on indicator 99. If an end-of-file condition does not occur, VAX RPG II sets on indicator 99.

```
                                                    Field length
      Control level                                 | Decimal positions
      |                                             | |Half adjust (H)
      | Indicators          Operation               | ||
      | |                   |                       | ||Resulting
      | |         Factor    |    Factor    Result   | |||indicators
      | |         1         |    2         field    | |||+ - 0
    C| NxxNxxNxx|           |    |         |        | |||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *         *          *    *         *      *--*** * * *
  22 C  N99                  READ SPECFIL                 99
```

ZK-4524-85

16.7.7 SETLL Operation

The SETLL operation positions a file at the next record with a key that is greater than or equal to the key you specify in factor 1. Factor 2 contains the name of the file for which the lower limit is set. Factor 2 must be an indexed demand or full-procedural file being processed sequentially within limits.

Factor 1 can be a field name or a literal, and must be the same size as the key specified in the File Description specification. If the keys in the file are in packed decimal format, factor 1 must be numeric.

You cannot use a record-address file and the SETLL operation on the same file.

If the program issues a READ operation before issuing a SETLL operation, processing begins with the first record in the file.

When VAX RPG II reaches the end-of-file, you can use another SETLL operation to reposition the file. If the SETLL operation is unsuccessful, you must reposition the file with a successful SETLL operation.

When a SETLL operation is performed on a record whose key is greater than the highest key in the file, the subsequent READ operation will be as if the current record pointer had not changed from the previous READ operation. If a successful SETLL operation is followed by a SETLL operation for a record whose key is greater than the highest key in the

file, with no intervening READ operation, the next READ operation will occur as if the current record pointer had not changed from the previous READ operation. In the latter case, it appears as if the successful SETLL operation never occurred, because it was followed by an unsuccessful SETLL operation.

In the following example, KEY1 contains the lower key limit for the file FILE1. The READ operation retrieves the first record with a key that is greater than or equal to the field KEY1.

```
                                                    Field length
  Control level                                     |  Decimal positions
  |                                                 |  |Half adjust (H)
  | Indicators          Operation                   |  ||
  | |                   |                         |  ||Resulting
  | |        Factor     |    Factor     Result|    |  ||indicators
  | |        1          |    2          field |    |  ||+ - 0
 C| NxxNxxNxx|          |    |          |    |    |  ||> < = +- Comments --+
  0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
  ** *          *          *    *          *     *--*** * * *
  C             KEY1       SETLLFILE1
  C                        READ FILE1
```

ZK-4525-85

16.8 Branching Operation Codes

VAX RPG II performs operations in the order they appear in your program. However, you can use branching operation codes to skip or repeat operations under certain conditions.

16.8.1 GOTO Operation

The GOTO operation transfers program control to the label you specify in factor 2.

The GOTO operation is especially useful in the following situations:

- Skipping calculations when certain conditions occur
- Performing certain calculations for certain record types
- Repeating calculations

You can transfer control in the following ways:

- To a previous line
- From one detail-time calculation line to another
- From one total-time calculation line to another
- From one subroutine calculation to another inside the same subroutine

You cannot transfer control from the following lines:

- Detail-time calculation line to a total-time calculation line
- Total-time calculation line to a detail-time calculation line
- Line inside a subroutine to a line outside that subroutine
- Line outside a subroutine to a line inside a subroutine

When using the GOTO operation, the following columns must be left blank:

- Columns 18 through 27 (factor 1)
- Columns 43 through 48 (result field)
- Columns 49 through 51 (field length)
- Column 52 (decimal positions)
- Column 53 (half adjust)
- Columns 54 through 59 (resulting indicators)

16.8.2 TAG Operation

The TAG operation identifies the line to which program control from a GOTO operation branches. Factor 1 contains the same label you used in factor 2 of the GOTO operation.

You cannot use conditioning indicators (columns 9 through 17) to condition a TAG operation; however, you can use a control-level indicator if the TAG operation is in total-time calculations.

16.8.3 Example

In the following example, VAX RPG II branches to line 66 if indicators 67, 68, and 69 are on.

```
                                                    Field length
     Control level                                  |  Decimal positions
     |                                              |  |Half adjust (H)
     | Indicators          Operation                |  ||
     | |                   |                        |  ||Resulting
     | |         Factor    |    Factor    Result|   |  |||indicators
     | |         1         |    2         field |   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   |  |||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *          *          *    *         *     *--*** * * *
  56 C   67 68 69             GOTO BRCH1
         .
         .
         .
  66 C               BRCH1    TAG
```

ZK-4526-85

16.9 LOKUP Operation Code

The LOKUP operation searches for an entry in a table or an array. Factor 1 contains the search argument and factor 2 contains the name of the table or array. The search argument can be an alphanumeric or a numeric constant, a field name, an array element, or a table name. Search arguments must be the same length and format as the entries in the table or array. For example, both fields must be numeric with the same number of digits, or both must be alphanumeric with the same number of characters.

You must use at least one, but not more than two, resulting indicators to specify the following:

- The type of search (high, low, or equal)
- The result of the search (successful or unsuccessful)

The following list describes the three types of searches:

- A resulting indicator in columns 54 and 55 causes VAX RPG II to search the table or array for the entry that is nearest to but higher in sequence than the search argument. You can specify this search only for sequenced tables and arrays.
- A resulting indicator in columns 56 and 57 causes VAX RPG II to search the table or array for the entry that is nearest to but lower in sequence than the search argument. You can specify this search only for sequenced tables and arrays.
- A resulting indicator in columns 58 and 59 causes VAX RPG II to search the table or array for the entry that is equal to the search argument.

You can use two indicators to test for HIGH and EQUAL or LOW and EQUAL conditions. VAX RPG II searches for an entry that satisfies either condition, with EQUAL given precedence. You cannot specify both HIGH and LOW conditions at the same time.

If the search is successful, VAX RPG II sets on the resulting indicators. If the search is unsuccessful, VAX RPG II sets off the resulting indicators.

16.9.1 Searching Tables

The LOKUP operation can search one table or two related tables. When searching a single table, you must specify the following:

- Factor 1
- Factor 2
- At least one resulting indicator

You can specify conditioning indicators in columns 7 through 17.

When VAX RPG II finds a table entry that satisfies the type of search, it places a copy of the entry in a special storage area. If you repeat the search, the new entry replaces the previous entry in the storage area.

When searching for an entry in two related tables, VAX RPG II searches only the table specified in factor 2. When the search condition is satisfied, VAX RPG II places the corresponding entries in their respective storage areas.

To program a search for an entry in related tables, you must make the following entries:

- Specify the search argument in columns 18 through 27
- Specify the name of the table to be searched in columns 33 through 42 (factor 2).
- Specify the name of the related table in columns 43 through 48 (result field).
- Specify at least one resulting indicator in columns 54 through 59 (resulting indicators).

You can specify conditioning indicators in columns 9 through 17.

Both tables should have the same number of entries. The related table must have as many entries as or more entries than the table to be searched.

Whenever you use a table name in an operation other than a LOKUP operation, the table name refers to the data placed in storage by the last successful LOKUP operation. Then, you can use the table entry in subsequent calculations. If you specify a table name in an operation other than a LOKUP operation but before a successful LOKUP operation occurs, unpredictable results can occur.

If you specify the table name as factor 1 in a LOKUP operation, the contents of the storage area are used as the search argument.

You can also use a table as the result field in operations other than a LOKUP operation. In this case, the contents of the storage area are replaced by the result of the calculation you specify. The corresponding entry in the table is also changed. In this way, you can use calculations to change the contents of tables.

In the following example, TABLIS was defined previously as a table. VAX RPG II searches for the entry that has the same value as the field PARTNO and, if successful, sets on indicator 33.

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |        Factor     |    Factor    Result  |  |||indicators
     | |        1          |    2         field   |  |||+ - 0
    C| NxxNxxNxx|          |    |         |       |  |||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *           *         *    *         *      *--*** * * *
  56 C              PARTNO    LOKUPTABLIS                 33
```

ZK-4527-85

16.9.2 Searching Arrays

LOKUP operations for arrays are the same as those for tables, except that you cannot use the result field. If the element searched for is found, its contents are not moved to a storage area. Indicators reflect whether the element is present.

To program a search for an element in an array, you must specify the name of the array to be searched in columns 33 through 42 (factor 2). Follow the same rules for specifying indicators for arrays as for tables.

You can specify the element to begin searching by adding an index. The index can be a numeric literal or a field. VAX RPG II begins searching at the specified element and continues the search until it finds the element or it reaches the end of the array. If you use a numeric literal to specify the index, VAX RPG II does not change its value to reflect the result of the search. If you use a field to specify the index and the search is unsuccessful, VAX RPG II places the value of 1 in the field. If you use a field to specify the index and the search is successful, VAX RPG II places the number of that array element that satisfied the search (counting from the first element) in the field. Then, you can use the index field to reference that array element in subsequent operations.

If you use an index that is less than or equal to zero, or greater than the number of elements in the array, and you compile the program with the RPG/CHECK=BOUNDS command, VAX RPG II issues a run-time error. If you use an index that is less than or equal to zero, or greater than the number of elements in the array and you do not compile the program with the RPG/CHECK=BOUNDS command, unpredictable results can occur.

16.9.3 Example

In the following example, MNTH was defined previously as a sequenced array and E as a numeric field. Because 1 is assigned to E, VAX RPG II begins searching with the first element of the array and searches for the first entry with a value that is greater than and equal to 1000. If an entry is found, E will contain the index number of the entry and the indicator 99 will be set on. If an entry is not found, E will contain 1 and the indicator 99 will be set off.

```
                                                    Field length
     Control level                                  | Decimal positions
     |                                              | |Half adjust (H)
     | Indicators          Operation                | ||
     | |                   |                        | ||Resulting
     | |        Factor     |    Factor    Result|   | |||indicators
     | |        1          |    2         field |   | |||+ - 0
    C| NxxNxxNxx|          |    |         |     |   | |||> < = +- Comments --+
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *        *          *    *         *      *--*** * * *
     C                      Z-ADD1         E        30
     C           1000       LOKUPMNTH,E                99  99
```

ZK-4528-85

16.10 Subprogram Operation Codes

VAX RPG II programs can call subprograms written in other languages and can pass and receive parameters between the main program and the subprogram. See Chapter 12 for examples of subprogram operation codes.

16.10.1 CALL Operation

The CALL operation transfers control to a subprogram. Factor 2 contains a character literal or a field defined by the EXTRN operation that names the entry point in the subprogram. Factor 2 cannot be a VAX RPG II program name.

The result field can contain the name of the parameter list associated with the PLIST operation. This enables it to share parameters between the main program and the subprogram. You can also specify the individual parameters immediately following the CALL operation code.

Factor 1, half adjust, and the resulting indicators in columns 54 and 55 and 58 and 59, must be blank. However, you can specify a resulting indicator in columns 56 and 57. VAX RPG II sets this indicator on when the subprogram returns an error status.

16.10.2 EXTRN Operation

The EXTRN operation initializes the value of a numeric unscaled field to a link-time constant. You can use the EXTRN operation to perform the following operations:

- Extend the subprogram name to more than eight characters
- Allow your program to access link-time constants, including status codes

You define link-time constants using external names. Use factor 1 to name the field VAX RPG II initializes, using the value of the link-time constant. Use factor 2 to name the external constant. You can use a maximum of 31 characters to name the constant. You must enclose the constant in apostrophes.

The external name must be defined as a global symbol in an object module available to the program at link time. Otherwise, an error will occur at link time.

Factor 1 of the EXTRN operation is defined as a nine-digit numeric field with zero decimal positions. The field cannot be defined elsewhere in the program. Fields defined by an EXTRN operation cannot be used as a result field in a calculation or have blank after specified when used on an Output specification.

Conditioning indicators must be left blank.

16.10.3 GIVNG Operation

The GIVNG operation allows you to define a parameter that receives the return status from the subprogram. See the *VAX/VMS Run-Time Library Routines Reference Manual* for information on the definition of return status. The GIVNG operation must follow the last PARM, PARMV, and PARMD operation following a CALL operation. The result field contains the name of an unscaled numeric field, table, or array element.

Entries in decimal positions and field length are optional. If you specify a field length, the entry for decimal positions must be zero. The following columns must be left blank:

- Columns 9 through 17 (conditioning indicators)
- Columns 18 through 27 (factor 1)
- Columns 33 through 42 (factor 2)
- Column 53 (half adjust)
- Columns 54 through 59 (resulting indicators)

16.10.4 PARM Operation

The PARM operation passes parameters by reference to a subprogram. The result field identifies the parameter. The parameter can be a field, a table, an array element, or an array. Factor 2 can contain a field, a table, an array element, an array, or a literal. The contents of factor 2 are copied into the result field before the subprogram is called. If the result field is numeric, factor 2 must be numeric. In this case, the value in factor 2 is copied into the result field in the same way that a Z-ADD operation is copied. If the result field is alphanumeric, factor 2 must be alphanumeric. In this case, the value is left-justified in the result field and trailing characters are filled with blanks.

The subprogram can change the contents of the result field but cannot change the contents of factor 2. Using factor 2 allows you to pass the values from factor 2 knowing that the subprogram cannot modify the field.

After a successful CALL operation, VAX RPG II copies the contents of the parameter into factor 1. Factor 1 can contain a field, a table, an array, or an array element. The copying is done in the same way as for factor 2. Entries for factor 1 and factor 2 are optional.

Entries in decimal positions and field length are optional. Conditioning indicators must be left blank. VAX RPG II, by default, passes numeric data by reference in packed decimal format.

You can also use the PARM operation to convert to the numeric data type needed by the subprogram being called. Use columns 54 through 59 (resulting indicators) to specify the notation for the parameter. See Chapter 12 for information on specifying parameters.

You can use one of the following access types:

- Read-only (R)—the parameter is read by the subprograms, but not modified.
- Write-only (W)—the parameter is not read by the subprograms, but a new value is supplied by the subprogram.
- Modify (M)—the parameter is read by the subprograms and a new value is supplied by the subprogram.

You can use one of the following data types:

- Word integer (signed) (W)
- Longword integer (signed) (L)
- Quadword integer (signed) (Q)
- F_floating single-precision (F)
- D_floating double-precision (D)
- Numeric string, right overpunched sign (NRO)

See Chapter 14 for information on data types.

You cannot specify an access type or data type if the result field is an entire array (nonindexed).

16.10.5 PARMD Operation

The PARMD operation passes parameters by descriptor to a subprogram. The result field contains the name of the field, the name of an array element, the name of the array, or a literal that identifies the parameter. Long character literals can be used effectively in the PARMD result field. See Section 15.7.5 for information on long character literals.

Entries in decimal positions and field length are optional. The following columns must be left blank:

- Columns 9 through 17 (conditioning indicators)
- Columns 18 through 27 (factor 1)
- Columns 33 through 42 (factor 2)
- Column 53 (half adjust)
- Columns 54 through 59 (resulting indicators)

See the *VAX/VMS Run-Time Library Routines Reference Manual* for information on argument descriptor format.

16.10.6 PARMV Operation

The PARMV operation passes parameters by value to a subprogram. The result field contains the name of an unscaled numeric field, table, array, or an unscaled numeric literal that identifies the parameter.

Entries in decimal positions and field length are optional. If you specify a field length, the entry for decimal positions must be 0. The following columns must be left blank:

- Columns 9 through 17 (conditioning indicators)
- Columns 18 through 27 (factor 1)
- Columns 33 through 42 (factor 2)
- Column 53 (half adjust)
- Columns 54 through 59 (resulting indicators)

16.10.7 PLIST Operation

The PLIST operation identifies the name of the parameter list for a subprogram. Use this operation with the CALL operation to access parameters in the subprogram. You can pass a maximum of 255 parameters.

Factor 1 contains the name of the parameter list. The following columns must be left blank:

- Columns 9 through 17 (conditioning indicators)
- Columns 33 through 42 (factor 2)
- Columns 43 through 48 (result field)

- Columns 49 through 51 (field length)
- Column 52 (decimal positions)
- Column 53 (half adjust)
- Columns 54 through 59 (resulting indicators)

If you want to pass parameters, you must use one of the parameter operations (PARM, PARMD, PARMV) to specify how you want to pass the parameters. Parameter operations must immediately follow the CALL or PLIST operation. Parameter operations must also be in the order expected by the subprogram.

16.10.8 Example

The following example makes a call to the VAX/VMS Run-Time Library routine STR$UPCASE. The call places REP HEAD in the result field RESULT.

```
                                                  Field length
     Control level                                |  Decimal positions
     |                                            |  |Half adjust (H)
     | Indicators          Operation              |  ||
     | |                   |                      |  ||Resulting
     | |        Factor     |    Factor    Result|  ||indicators
     | |        1          |    2         field |  ||+ - 0
    C| NxxNxxNxx|          |    |         |     |  ||> < = +- Comments --+
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
     ** *        *          *    *         *     *--*** * * *
     C                      MOVE 'rep head'HEAD    8
     C           UPCASE     EXTRN'STR$UPCASE'
     C                      CALL UPCASE
     C                      PARMD          RESULT  8
     C                      PARMD          HEAD
```

ZK-4529-85

16.11 TIME Operation Code

The TIME operation code allows you to access the system time of day and the optional system date. Note that the system date may be different from the date accessed with UDATE, UDAY, UMONTH, or UYEAR if the RPG$UDATE logical is used.

The result field must be numeric with 0 decimal places and contain either 6 or 12 digits. A 6-digit result field displays the time in the format hhmmss. A 12-digit result field displays the time and date in the format hhmmssyymmdd.

Table 16–1: Summary of Operation Codes

Operation Code	Factor 1	Factor 2	Result Field	Indicators: Control Level	Indicators: Conditioning	Indicators: Resulting + >	Indicators: Resulting − <	Indicators: Resulting 0 =
28-32	18-27	33-42	43-48	7-8	9-17	54-55	56-57	58-59
ADD(H)	O	R	R	O	O	O +	O −	O 0
BEGSR	R							
BITOF		R	R	O	O			
BITON		R	R	O	O			
CALL		R	O	O	O		O E	
CHAIN	R	R		O	O	O NR		O RL
COMP	R	R		O	O	O H	O L	O EQ
DIV(H)	O	R	R	O	O	O +	O −	O Z
DSPLY	O	R	O	O	O			
ENDSR	O							
EXCPT		O		O	O			
EXSR		R		O	O			
EXTRN	R	R		O				
FORCE		R		O	O			
GIVNG			R	O				
GOTO		R		O	O			
LOKUP 1A	R	R		O	O	O H	O L	O EQ
LOKUP 1T	R	R	O	O	O	O H	O L	O EQ
MOVE		R	R	O	O			
MOVEA		R	R	O	O			
MOVEL		R	R	O	O			
MULT(H)	O	R	R	O	O	O +	O −	O Z
MVR			R	O	O	O +	O −	O Z
PARM	O	O	R	O				
PARMD			R	O				
PARMV			R	O				
PLIST	R			O				
READ		R		O	O			O EOF
SETLL	R	R		O	O			

SETOF[1]				O	O	O	O	O
SETON[1]				O	O	O	O	O
SQRT[H]		R	R	O	O			
SUB[H]	O	R	R	O	O	O +	O −	O Z
TAG	R			O				
TESTB[1]		R	R	O	O	O	O	O
TIME			R					
XFOOT[H]		R	R	O	O	O +	O −	O Z
Z-ADD[H]		R	R	O	O	O +	O −	O Z
Z-SUB[H]		R	R	O	O	O +	O −	O Z

[H] You can specify Half adjust for this operation.

[1] Specify at least one resulting indicator for this operation.
Conditioning indicators are valid only for executable operation codes.
Fields without entries in this table must be blank.

[A] Factor 2 is an array.

[T] Factor 2 is a table.

Legend

+ = positive	− = negative	EOF = end of file
E = error	EQ = equal	O = optional
NR = no record	R = required	Z = zero
ZB = zero or blank	RL = record locked	

ZK-5537-86

Appendix A
Character Sets

Character	ASCII Decimal	Hexadecimal	EBCDIC Decimal	Hexadecimal
NUL	000	00	000	00
SOH	001	01	001	01
STX	002	02	002	02
ETX	003	03	003	03
EOT	004	04	055	37
ENQ	005	05	045	2D
ACK	006	06	046	2E
BEL	007	07	047	2F
BS	008	08	022	16
HT	009	09	005	05
LF	010	0A	037	25
VT	011	0B	011	0B
FF	012	0C	012	0C
CR	013	0D	013	0D
SO	014	0E	014	0E
SI	015	0F	015	0F
DLE	016	10	016	10
DC1	017	11	017	11

Character	ASCII Decimal	Hexadecimal	EBCDIC Decimal	Hexadecimal
DC2	018	12	018	12
DC3	019	13	019	13
DC4	020	14	060	3C
NAK	021	15	061	3D
SYN	022	16	050	32
ETB	023	17	038	26
CAN	024	18	024	18
EM	025	19	025	19
SUB	026	1A	063	3F
ESC	027	1B	039	27
FS	028	1C	028	1C
GS	029	1D	029	1D
RS	030	1E	030	1E
US	031	1F	031	1F
space	032	20	064	40
!	033	21	079	4F
"	034	22	127	7F
#	035	23	123	7B
$	036	24	091	5B
%	037	25	108	6C
&	038	26	080	50
'	039	27	125	7D
(	040	28	077	4D
)	041	29	093	5D
*	042	2A	092	5C
+	043	2B	078	4E
,	044	2C	107	6B
-	045	2D	096	60
.	046	2E	075	4D

Character	ASCII Decimal	Hexadecimal	EBCDIC Decimal	Hexadecimal
/	047	2F	097	61
0	048	30	240	F0
1	049	31	241	F1
2	050	32	242	F2
3	051	33	243	F3
4	052	34	244	F4
5	053	35	245	F5
6	054	36	246	F6
7	055	37	246	F7
8	056	38	248	F8
9	057	39	249	F9
:	058	3A	122	7A
;	059	3B	094	6E
<	060	3C	076	4C
=	061	3D	126	7E
>	062	3E	110	6E
?	063	3F	111	6F
@	064	40	124	7C
A	065	41	193	C1
B	066	42	194	C2
C	067	43	195	C3
D	068	44	196	C4
E	069	45	197	C5
F	070	46	198	C6
G	071	47	199	C7
H	072	48	200	C8
I	073	49	201	C9
J	074	4A	209	D1
K	075	4B	210	D2

Character	ASCII Decimal	Hexadecimal	EBCDIC Decimal	Hexadecimal
L	076	4C	211	D3
M	077	4D	212	D4
N	078	4E	213	D5
O	079	4F	214	D6
P	080	50	215	D7
Q	081	51	216	D8
R	082	52	217	D9
S	083	53	226	E2
T	084	54	227	E3
U	085	55	228	E4
V	086	56	229	E5
W	087	57	230	E6
X	088	58	231	E7
Y	089	59	232	E8
Z	090	5A	233	E9
[	091	5B	074	4A
\	092	5C	224	E0
]	093	5D	090	5A
^	094	5E	095	5F
_	095	5F	109	6D
`	096	60	121	79
a	097	61	129	81
b	098	62	130	82
c	099	63	131	83
d	100	64	132	84
e	101	65	133	85
f	102	66	134	86
g	103	67	135	87
h	104	68	136	88

Character	ASCII Decimal	Hexadecimal	EBCDIC Decimal	Hexadecimal
i	105	69	137	89
j	106	6A	145	91
k	107	6B	146	92
l	108	6C	147	93
m	109	6D	148	94
n	110	6E	149	95
o	111	6F	150	96
p	112	70	151	97
q	113	71	152	98
r	114	72	153	99
s	115	73	162	A2
t	116	74	163	A3
u	117	75	164	A4
v	118	76	165	A5
w	119	77	166	A6
x	120	78	167	A7
y	121	79	168	A8
z	122	7A	169	A9
{	123	7B	192	C0
|	124	7C	106	6A
}	125	7D	208	D0
~	126	7E	161	A1
DEL	127	7F	007	07

Appendix B

Differences Between VAX RPG II and PDP-11 RPG II

This appendix describes the following:

- PDP-11 RPG II functionality that is not supported by VAX RPG II
- VAX RPG II functionality that is supported differently than PDP-11 RPG II functionality
- Additional functionality that is supported only by VAX RPG II

This appendix does not list new features which have been added to VAX RPG II Versions 2.0 and 2.1. See the online release notes for details of the new features in Version 2.1.

VAX RPG II does not support the following PDP-11 RPG II functionality:

- Control specification (H)
  - Column 11 (listing options)—the equivalent functionality is implemented using the /LIST qualifier to the DCL command RPG.
  - Column 15 (debug)—the DEBUG operation code is not supported. Instead, use the VAX/VMS Debugger.
  - Column 25 (source listing)—a single page size for the listing is supported.
- Extension specification (E)
  - Columns 11 through 18 (from file name)—PDP-11 RPG II allows the same table input file to be specified for more than one preexecution-time table or array. VAX RPG II requires a different file for each preexecution-time table or array.

- Calculation specification (C)
  - Columns 28 through 32 (operation)—the DEBUG operation code is not supported.
- Output specification (O)
  - Column 39 (blank after)—specifying blank after for a constant field is not supported.
- General
  - VAX RPG II uses PRN format files for printer output files. This format requires that you use the /NOFEED qualifier with the PRINT command when printing printer output files.
  - PDP-11 RPG II handles both zoned numeric and overpunched decimal data formats for input. VAX RPG II supports only overpunched decimal data format.
  - All user-defined names must be unique for VAX RPG II.
  - VAX RPG II does not support the H0 indicator. Errors detected by VAX RPG II result in run-time errors.
  - VAX RPG II does not recognize L0 as an indicator. Although L0 can be used as a control-level indicator in columns 7 and 8 of the Calculation specification, it cannot be used in an Output specification.

VAX RPG II supports the following functionality differently than PDP-11 RPG II:

- File specification (F)
  - Column 15 (file type)—when O (output) is entered in column 15 and column 66 (file addition) does not contain A, VAX RPG II always creates a new version of the file, even if the file is an indexed or direct file. PDP-11 RPG II accesses an existing indexed or direct file if the specified file is found.
  - Columns 40 through 46 (device code)—VAX RPG II uses the device code to define the functions that can be performed on the associated file. It does not refer to a specific device.
  - Columns 47 through 52 (symbolic device)—VAX RPG II uses the symbolic device in conjunction with the file name specified in columns 7 through 14 (file name) to associate the VAX RPG II file name with the VAX/VMS file specification.

If you specify a symbolic device on the File Description specification and no VAX/VMS logical name translation exists for that symbolic device, then VAX RPG II will have VAX RMS use the symbolic device as the file name.

If a symbolic device consists of all blanks, then VAX RMS will operate as if the symbolic device did not exist (for example, an attempt will be made to translate the file name as a logical name). The symbolic device may consist of any characters and will be passed to VAX RMS when the file is opened.

The file name will be translated as a logical name if a symbolic device is not supplied. The hierarchy by which the file name is constructed by VAX RMS is as follows:

1. File name (symbolic device)
2. Default file name (VAX RPG II file name)
3. Related file name (DAT file type)

The related file name can be overridden by supplying a different file type for the symbolic device.

- Column 68 (share)—VAX RPG II requires you to specify S or R to open a file for shared access. PDP-11 RPG II requires you to use the /MULTIUSER qualifier with RPGASN.

- Input specification (I)
  - Column 43 (data format)—when loading an execution-time array from an Input specification that is in packed decimal or binary format, VAX RPG II requires packed decimal (P) or binary (B) in column 43 of the Input specification. PDP-11 RPG II requires packed decimal (P) or binary (B) in column 43 on the Extension specification.
- Calculation specification (C)
  - A READ operation code on a file that has not been opened because it was conditioned by an external indicator that was off, sets on the end-of-file indicator if one is specified. In this case, PDP-11 RPG II does not set on the end-of-file indicator.
  - VAX RPG II MOVE and MOVEL semantics are not the same as those of PDP-11 RPG II when the result field is numeric. The differences are the following:
    1. VAX RPG II does not perform the "spurious sign" and "sign ignored" cases of PDP-11 RPG II.

2. When the sending field is a character field, VAX RPG II converts the characters in the sending field in the following manner:
   * All valid overpunched decimal characters are converted to the sign and digit they represent.
   * All other characters are converted to a positive zero (+0).
3. If a READ operation is done before a SETLL operation, VAX RPG II reads the record with the lowest key. PDP-11 RPG II reads a record containing blanks in this case.
4. PDP-11 RPG II does not issue an error when the DSPLY operation code is used with a field that is larger than the file record length. VAX RPG II does issue an error message in this case.
5. VAX RPG II will display (DSPLY operation code) the sign of a negative numeric.

- Output specification (O)
  - Columns 17 and 18 (space after)—PDP-11 RPG II assumes a single space after if the entries in columns 17 and 18 are left blank or are zero. VAX RPG II assumes a single space after only if all entries in columns 17 through 22 are left blank. An entry of zero in columns 17 and 18 allows overprinting.
- General
  - ALTSEQ records—VAX RPG II uses hexadecimal representation to specify the entries in ALTSEQ records. PDP-11 RPG II uses octal representation.
  - Tables—For VAX RPG II, a reference to a table before the first LOKUP operation will locate the first element in the table. PDP-11 RPG II returns a blank.
  - VAX RPG II and PDP-11 RPG II support two-word (4-byte) binary data. PDP-11 RPG II places the words in reverse order to what is required by VAX architecture. PDP-11 RPG II data files that include two-word (4-byte) binary data will require conversion to be used by VAX RPG II.
  - VAX RPG II uses the logical name RPG$UDATE to specify UDATE and uses the logical name RPG$EXT_INDS to specify the settings for external indicators.
  - PDP-11 RPG II treats blanks in a numeric field in overpunched decimal format as zeros. VAX RPG II does the same when you use the /CHECK=BLANKS_IN_NUMERICS qualifier with the RPG command.

- VAX architecture reports a reserved operand fault when invalid data is found in a numeric field. For VAX RPG II, this causes a run-time error. PDP-11 RPG II processes invalid numeric data without halting program execution.
- PDP-11 RPG II's run-time system changes the name of the process to the program name while the program is running. VAX RPG II does not do this.
- When compiling with RPG [FOO]TEST.RPG, PDP-11 RPG II places the OBJ, LST, CMD, and ODL files in the [FOO] directory, no matter what the current default directory is.

  VAX RPG II places the OBJ and LIS files into the default directory in a manner similar to other VAX languages. This VAX RPG II operation is similar to the following PDP-11 command lines:

  ```
  $ MCR PDPRPG
  RPG>TEST=[FOO]TEST.RPG
  RPG>CTRL/Z
  ```

  The preceding commands cause PDP-11 RPG II to place the files in the current default directory.

  Use the following command to simulate the PDP-11 RPG II operation:

  ```
  $ RPG [FOO]TEST/LIS/OBJ
  ```

- /NOLIST is the default for invoking the VAX RPG II compiler interactively. PDP-11 RPG II produces a listing file by default.

Note that RPGDMP is not provided with VAX RPG II. A similar functionality is provided with the VAX/VMS Dump and Sort/Merge Utilities (DUMP and SORT/MERGE). See the *VAX/VMS DCL Dictionary* for information on DUMP and SORT/MERGE. See the *VAX/VMS Utility Reference Manual* for information on the three SORT qualifiers, /CONDITION, /FIELD, and /INCLUDE.

VAX RPG II supports the following additional functionality:

- Control specification (H)
  - A Control specification (H) is not required.
  - Column 18 (currency symbol)—you can specify the character to represent the currency symbol.

- Column 21 (inverted print)—an entry of I, D, or J switches the function of decimal point and comma notation in numeric literals and edited formats, and changes the format of UDATE to day, month, and year (ddmmyy).

- File specification (F)
  - Columns 7 through 14 (file name)—file names can be up to eight characters long.
  - Column 16 (file designation)—VAX RPG II does not require a primary file.
  - Column 19 (file format)—you can specify V to indicate that the file contains variable-length records.
  - Columns 24 through 27 (record length)—VAX RPG II allows all files, with the exception of display files, to have a record length of up to 9999 characters.
- Extension specification (E)
  - The definition of compile-time tables and arrays can be mixed with the definition of execution-time and preexecution-time tables and arrays.
- Input specification (I)
  - Columns 15 and 16 (sequence)—you can specify input records with an alphabetic sequence before or after input records with a numeric sequence.
  - Columns 19 and 20 (record-identifying indicator)—you do not have to specify look-ahead fields as the last record in a file.
- Calculation specification (C)
  - Columns 28 through 32 (operation)—the following operations have been added to allow VAX RPG II programs to call subprograms written in other languages and routines in the VAX/VMS Run-Time Library and system services:
    1. CALL—calls a subprogram with optional parameters
    2. PARM—passes a parameter by reference
    3. PARMD—passes a parameter by descriptor
    4. PARMV—passes a parameter by value
    5. GIVNG—returns the status of a subprogram
    6. EXTRN—equates a VAX RPG II name with an external link-time constant
    7. PLIST—defines a parameter list

  - Columns 28 through 32 (operation)—VAX RPG II allows factor 2 and the result field to reference the same array on a MOVEA operation.
  - Columns 49 through 52 (field definition)—you do not have to define fields before they are used, if they are defined within the program.
- Output specification (O)
  - Columns 16 through 18 (ADD/DEL)—you can specify the DEL option to identify the record to be deleted. You can delete records from indexed or direct files.
  - Columns 45 through 70 (edit word)—the number of replaceable characters in the edit word can be greater than or equal to the number of digits in the numeric field.
- General
  - The string containing double slashes (//) and a blank and the string containing a double asterisk (**) and a blank are accepted as delimiters between specifications and any ALTSEQ records, and between compile-time tables and arrays.
  - The special words, *IN and *INxx, can be used to reference indicators as one-position character fields.
  - A user-defined name can contain a pound sign (#), an underscore (_), a dollar sign ($), and an at sign (@).
  - A character field can have a length of up to 9999 characters.

The VAX RPG II editor does not support the following PDP-11 RPG II features:

- Editing of SORT-11 specifications
- VT05 or VT52 terminals
- Hard-copy terminals
- The following qualifiers:
  - /IDENT
  - /PAGE and /NOPAGE
  - /SAVE and /NOSAVE
  - /TERMINAL
- CTRL/D
- SET SKIP command

- Automatically advancing the cursor to the next tab stop if the current field is full
- Displaying a tab stop as data from an input file as the TAB character
- Renaming the input file with a BAK file type

Appendix C

VAX Performance Coverage Analyzer Applied to a VAX RPG II Program

The following command procedure produces execution information by source line. (Note that for purposes of illustration, unimportant information on the accompanying listing is truncated on the right side.)

```
$ rpg/debug/nolist ships
$ link/debug=sys$library:pca$obj.obj/nomap ships
$ run ships
set datafile ships
set counters program_address by line
go
$ pca ships
tabulate/counters/source module ships by line
file pca.lis
exit
$ type pca.lis
                    VAX Performance and Coverage Analyzer                    Page 1
                 Exact Execution Counts (164 data points total)

Percent     Count  Line
SHIPS\
                     1:      H*++
                     2:      H*  FUNCTIONAL DESCRIPTION:
                     3:      H*    This program produces a report of shipments f
                     4:      H*    products broken down by division and departme
                     5:      H*    input file with the shipment data for the pas
                     6:      H*--
                     7:      H
  0.6%           1   8:      FSHIPS   IP  F      41           DISK
  0.6%           1   9:      FSUMREP  O   F      98           LPRINTER
                    10:      E                    QTY         4  2 0
                    11:      LSUMREP   55FL 500L
```

```
12.2%      20   12:   ISHIPS   AA  01
                13:   I                                          1   5 DIV
                14:   I                                          6   7 DEPT
                15:   I                                          8  16 PROD
                16:   I                                         17  24 QTY
                17:   C*

24.4%      40   18:   C   01               XFOOTQTY       PROQTY  30
24.4%      40   19:   C   01      PROQTY   ADD  DEPQTY    DEPQTY  30
                20:   C* (L1 break on DEPT / L2 break on DIV)
 4.3%       7   21:   CL1         DEPQTY   ADD  DIVQTY    DIVQTY  30
 4.3%       7   22:   CL1                  Z-ADDO         DEPQTY
 2.4%       4   23:   CL2         DIVQTY   ADD  FINQTY    FINQTY  40
                24:   C*
 0.6%       1   25:   OSUMREP  H  001   1P
                26:   O                                          48 'PRODUCT SHI
 0.6%       1   27:   O       H 02    1P
                28:   O                          UDATE Y   12
                29:   O                                          48 'PRODUCT SHI
 0.6%       1   30:   O       H  1    1P
                31:   O                                          42 'SHIPMENTS'
 0.6%       1   32:   O       H  2    1P
                33:   O                                          15 'DIVISION  D
                34:   O                                          24 'PRODUCT'
                35:   O                                          48 'Q1  Q2  Q3

12.2%      20   36:   O       D  1     01
                37:   O                L2      DIV        8
                38:   O                L1      DEPT      14
                39:   O                        PROD      25
                40:   O                        QTY   Z   41
                41:   O                        PROQTYZ   48
 4.3%       7   42:   O       T  1     L1
 2.4%       4   43:   O       T  0     L2
                44:   O                        DIV       69
 2.4%       4   45:   O       T  0     L2
                46:   O                        DIV       69
 2.4%       4   47:   O       T 02     L2
                48:   O                        DIVQTYZB  48
                49:   O                                  63 '<== Total f
                50:   O                        DIV       69
 0.6%       1   51:   O       T  0     LR
                52:   O                        FINQTY1   48
                53:   O                                  65 '<== GRAND T
```

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

EADER'S OMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

you find this manual understandable, usable, and well organized? Please make igestions for improvement.

_____

_____

_____

_____

_____

_____

you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

_____

_____

_____

ase indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

ne _____ Date _____

janization _____

eet _____

/ _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here

Cut Along Dotted Line