UPDATE NOTICE NO. 1

Introduction to RT-11

AD-5281B-T1

March 1981

NEW AND CHANGED INFORMATION

This update contains changes and additions to the *Introduction to RT-11* (AA-5281B-TC).

Additional copies of this update to the *Introduction to RT-11* may be ordered from the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754. Order Number: AD-5281B-T1. The order number of the base manual is AA-5281B-TC.

Copyright © 1981 by Digital Equipment Corporation
Maynard, Massachusetts

INSTRUCTIONS

The enclosed pages are to be placed in the *Introduction To RT-11* as replacements for, or additions to, current pages. The changes made on replacement pages are indicated in the outside margin by change bars (▮) for additions and bullets (●) for deletions.

KEEP THIS NOTICE IN YOUR MANUAL TO MAINTAIN AN UP-TO-DATE RECORD OF CHANGED PAGES

Old Page	New Page
Title page/Copyright	Title page/Copyright
5-9/5-10	5-9/5-10
5-19/5-20	5-19/5-20
—	9-2.1
9-5/9-6	9-5/9-6
14-3/14-4	14-3/14-4
14-5/14-6	14-5/14-6
14-11/14-12	14-11/14-12
15-7/15-8	15-7/15-8
A-3/A-4	A-3/A-4
Reader's Comments	Reader's Comments

March 1981

This document is an introductory manual for the RT-11 operating system. Its purpose is to acquaint new users with the RT-11 commands that perform common system operations. This manual presents the background material necessary to understand the system operations. It also contains a series of command examples and demonstration exercises that complement the text.

Introduction to RT-11

Order No. AA-5281B-TC

SUPERSESSION/UPDATE INFORMATION: This manual supersedes Order No. DEC-11-ORITA-A-D, DN1. This manual includes Update Notice No. 1, (AD-5281B-T1).

OPERATING SYSTEM AND VERSION: RT-11 V4.0

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, March 1980
Updated, March 1981

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

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1980, 1981 by Digital Equipment Corporation
All Rights Reserved.

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EduSystem	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

M12700

10/81-15

The K (Kill) command erases the entire line following the pointer and positions the pointer at the beginning of the next line in the text. Type:

```
*K(ESC)L(ESC)(ESC)
AR ENDOWED BY THEIR CREATOR, THAT AMONG
*
```

The pointer is now at the beginning of the next line in the text. As you can see, this line also contains an error, the word ARE is incorrectly spelled as AR. Use the J command to jump over two characters, and insert the E. Then verify the line:

```
*2J(ESC)IE(ESC)V(ESC)(ESC)
ARE ENDOWED BY THEIR CREATOR, THAT AMONG
*  ↑
```

GET

The arrow shows where the pointer is now positioned. This line still contains an error — it is missing the words WITH CERTAIN INALIENABLE RIGHTS, which should follow the word CREATOR. You can count the number of characters from the pointer to the second R in CREATOR and then jump the pointer by this number, or you can use the G (Get) command. The G command searches, from the pointer, for the first occurrence of a specified character string and leaves the pointer at the end of that string. Use the G command to search for the string OR (in CREATOR); then insert the missing words and list the lines that have changed. Notice how you use the carriage return to break the line into two parts (the (SP) symbol is used to show where you should insert spaces):

```
*GOR(ESC)I(SP)WITH(SP)CERTAIN(RET)
INALIENABLE(SP)RIGHTS(ESC)-A(ESC)2L(ESC)(ESC)
ARE ENDOWED BY THEIR CREATOR WITH CERTAIN
INALIENABLE RIGHTS, THAT AMONG
*
```

To list both lines, it was necessary to move the pointer back to the beginning of the first line you changed; this was done by the -A command. The 2L command then listed both lines. Notice where the pointer is; it was moved by the -A command and was not repositioned by the L command.

You must be careful when you use the Get command, because the character string you specify must be unique if you want the pointer to move to the correct spot. For example, if the characters OR had occurred anywhere after the pointer and before the word CREATOR, the pointer would have stopped there instead, and you would have inserted text in the wrong place.

The final errors in this text occur in the last line. The words THE PURSUIT OF are missing, and the word HAPLENESS is a misspelling. Use the Get command to move the pointer to the word

AND and insert the missing text. Move the pointer again with the Get command to the PLE of HAPLENESS; erase the LE, and insert PI. Then verify the line:

```
*GAND(ESC)I(SP) THE(SP) PURSUIT(SP) OF(ESC)(ESC)
*GPLE(ESC)-2D(ESC)IPI(ESC)V(ESC)(ESC)
THESE ARE LIFE, LIBERTY AND THE PURSUIT OF HAPPINESS.
*
```

CTRL/L

Large text files of 50 lines or more should be delimited into pages. To do this, insert a form feed into the text at the place where you want the page to end. A form feed is typed as a CTRL/L (hold the CTRL key down and type the L key), which the editor recognizes as a page break.

Since this text file is only five lines long, there is really no need to delimit it as a page. However, for the sake of practice, insert a form feed at the end of this file. Then move the pointer to the beginning of the text buffer and list the entire text. Compare your text with the following example. If errors remain in your file, fix them by using the commands described so far.

```
*G.(ESC)I(RET)
(CTRL/L)
```

(CTRL/L echoes as eight line feeds.)

```
(ESC)B(ESC)/L(ESC)(ESC)
WE HOLD THESE TRUTHS TO BE SELF-EVIDENT,
THAT ALL MEN ARE CREATED EQUAL, THAT THEY
ARE ENDOWED BY THEIR CREATOR WITH CERTAIN
INALIENABLE RIGHTS, THAT AMONG
THESE ARE LIFE, LIBERTY AND THE PURSUIT OF HAPPINESS.

*
```

This text is correct in spelling and content, but the last two lines should be justified to make them easier to read. The pointer is currently at the beginning of the text. Use the G command to search for the character string AMONG; then insert and delete text to justify the lines. Finally, list the text again:

```
*GAMONG(ESC)I(SP) THESE(SP) ARE(ESC)A(ESC)10D(ESC)B(ESC)/L(ESC)(ESC)
WE HOLD THESE TRUTHS TO BE SELF-EVIDENT,
THAT ALL MEN ARE CREATED EQUAL, THAT THEY
ARE ENDOWED BY THEIR CREATOR WITH CERTAIN
INALIENABLE RIGHTS, THAT AMONG THESE ARE
LIFE, LIBERTY AND THE PURSUIT OF HAPPINESS.

*
```

CREATING THE DEMONSTRATION PROGRAMS

Following are two demonstration programs. One is written in the FORTRAN IV programming language and one is written in the MACRO–11 assembly language. Both programs are used in later chapters of this manual, and both contain intentional misspellings and errors.

Use the editor to create these programs. Type them exactly as they are shown, including errors. Use tabs and spaces to format each line as shown (remember that tab stops are positioned every eight spaces across the terminal page). Make sure that the FORTRAN program is formatted properly so that a source comparison described in the next chapter will operate properly. Except for the comment lines (those beginning with a C), begin all lines with a tab. Use any of the editing commands described in this chapter. Activate the display editor and immediate mode if you wish.

When you have finished, check each file carefully. The two files should match those shown here exactly, including tabs and spaces. Correct any errors that you find that are not intentional. Obtain a listing of each file by using B ESC/L ESC ESC before closing the file.

Create the FORTRAN file first. Call it GRAPH.FOR and use the system volume for storage. Then create the MACRO program. Call it SUM.MAC and again use the system volume for storage.

NOTE

Knowledge of the FORTRAN IV and MACRO–11 languages is not necessary to create these demonstration programs.

The following program, GRAPH.FOR, is the FORTRAN demonstration program.

```
C GRAPH.FOR     VERSION 1
C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
C OF AN EXTERNAL FUNCTION, FUN(X,Y)
C THE LIMITS OF THE PLOT ARE DETERMINED BY THE DATA STATEMENTS
C 'STAB' IS FILLED WITH A TABLE OF WEIGHT FLAGS
C 'STRING' IS USED TO BUILD A LINE OF GRAPH FOR PRINTING

        SCAL(ZMIN,ZMAX,MAXZ,K)=ZMIN+FLOAT(K-1)*(ZMAX-ZMIN)/FLOAT(MAXZ-1)
        LOGICAL*1 STRING(13,3),STAB(100)
        DATA XMIN,XMAX,MAXX/-5,5,45/
        DATA YMIN,YMAX,MAXY/-5,5,72/
        DATA FMIN,FMAX/0.0,1.0/
        CALL SCOPY('- 1 2 3 4 5 6 7 8 9 +',STAB)
        MAXFLEN(STAB)
        DO 20 IX=1,MAXX
            X=SCAL(XMIN,XMAX,MAXX,IX)
            CALL REPEAT('*',STRING,MAXY)
            IF(IX.EQ.1 .OR. IX.EQ.MAXX) GOTO 20
                DO 10 IY=2,MAXY-1
                    Y=SCAL(YMIN,YMAX,MAXY,IY)
                    IFUN=2+INT(FLOAT(MAXF-3)*(FUN(X,Y)-FMIN)/(FMAX-FMIN))
10                  STRING(IY)=STAB(MIN0(MAXF,MAX0(1,IFUN)))
30          CALL PUTSTRING(7,STRING,' ')
        CALL EXIT
        END

        FUNCTION FUN(X,Y)
        R=SQRT(X**2+Y**2)
        FUN=X*Y*R*EXP(-R))**2
        RETURN
        END
```

The following program, SUM.MAC, is the MACRO demonstration program.

```
        .TITLE SUM.MAC  VERSION 1

        .MCALL .TTYOUT, .EXIT, .PRINT


        N = 70.          ;NO. OF DIGITS OF 'E' TO CALCULATE

;       'E' = THE SUM OF THE RECIPROCALS OF THE FACTORIALS
;       1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...

EXP:    .PRINT  #MESSAG          ;PRINT INTRODUCTORY TEXT
        MOV     #N,R5            ;NO. OF CHARS OF 'E' TO PRINT
FIRST:  MOV     #N+1,R0          ;NO. OF DIGITS OF ACCURACY
        MOV     #A,R1            ;ADDRESS OF DIGIT VECTOR
SECOND: ASL     @R1              ;DO MULTIPLY BY 10 (DECIMAL)
        MOV     @R1,-(SP)        ;SAVE *2
        ASL     @R1              ;*4
        ASL     @R1              ;*8
        ADD     (SP)+,(R1)+      ;NOW *10, POINT TO NEXT DIGIT
        DEC     R0               ;AT END OF DIGITS?
        BNE     2ND              ;BRANCH IF NOT
        MOV     #N,R0            ;GO THRU ALL PLACES, DIVIDING
THIRD:  MOV     -(R1),R3         ;BY THE PLACES INDEX
        MOV     #-1,R2           ;INIT QUOTIENT REGISTER
FOURTH: INC     R2               ;BUMP QUOTIENT
        SUB     R0,R3            ;SUBTRACT LOOP ISN'T BAD
        BCC     FOURTH           ;NUMERATOR IS ALWAYS < 10*N
        ADD     R0,R3            ;FIX REMAINDER
        MOV     R3,@R1           ;SAVE REMAINDER AS BASIS
                                 ;FOR NEXT DIGIT
        ADD     R2-2(R1)         ;GREATEST INTEGER CARRIES
                                 ;TO GIVE DIGIT
        DEC     R0               ;AT END OF DIGIT VECTOR?
        BNE     THIRD            ;BRANCH IF NOT
        MOV     -(R1),R0         ;GET DIGIT TO OUTPUT
FIFTH:  SUB     #10.,R0          ;FIX THE 2.7 TO .7 SO
                                 ;THAT IT IS ONLY 1 DIGIT
        BCC     FIFTH            ;(REALLY DIVIDE BY 10)
        ADD     #10+'0,R0        ;MAKE DIGIT ASC II
        .TTYON                   ;OUTPUT THE DIGIT
        CLR     @R1              ;CLEAR NEXT DIGIT LOCATION
        DEC     R5               ;MORE DIGITS TO PRINT?
        BNE     FIRST            ;BRANCH IF YES
        .EXIT                    ;WE ARE DONE

EXP:    .REPT   N+1
        .WORD   1                ;INIT VECTOR TO ALL ONES
        .ENDR

MESSAG: .ASCII  /THE VALUE OF E IS:/ <15><12> /2./ <200>
        .EVEN

        .ENDEXP
```

To link the example FORTRAN program, you must include FORLIB.OBJ in SYSLIB. For instructions on how to include FORLIB in SYSLIB, refer to Section 3.4 in the *RT-11 FORTRAN IV Installation Guide*.

point to look at the listing produced by the compiler, because more information is shown there. Print the listing on either the line printer or terminal, using one of the following commands:

Long Command Format

(Line printer)

```
.PRINT (RET)
Files? GRAPH.LST (RET)
```

(Terminal)

```
.TYPE (RET)
Files? GRAPH.LST (RET)
```

Short Command Format

(Line printer)

```
.PRINT GRAPH.LST (RET)
```

(Terminal)

```
.TYPE GRAPH.LST (RET)
```

Your listing should look like the following example.

NOTE

You do not need to understand the FORTRAN IV language or the way this program works to successfully complete the exercises in this chapter.

```
FORTRAN IV      V02.1-10   Thu 11-Dec-80 01:18:50              PAGE 001

      C GRAPH.FOR       (VERSION PROVIDED)
      C THIS PROGRAM PRODUCES A PLOT ON THE TERMINAL
      C OF AN EXTERNAL FUNCTION, FUN(X,Y)
      C THE LIMITS OF THE PLOT ARE DETERMINED BY THE DATA STATEMENTS
      C 'STAB' IS FILLED WITH A TABLE OF HEIGHT FLAGS
      C 'STRING' IS USED TO BUILD A LINE OF GRAPH FOR PRINTING
0001        SCAL(ZMIN,ZMAX,MAXZ,K)=ZMIN+FLOAT(K-1)*(ZMAX-ZMIN)/FLOAT(MAXZ-1)
0002        LOGICAL*1 STRING(13,3),STAB(100)
0003        DATA XMIN,XMAX,MAXX/-5,5,45/
0004        DATA YMIN,YMAX,MAXY/-5,5,72/
0005        DATA FMIN,FMAX/0.0,1.0/
0006        CALL SCOPY('- 1 2 3 4 5 6 7 8 9 +',STAB)
0007        MAXF=LEN(STAB)
0008        DO 20 IX=1,MAXX
0009           X=SCAL(XMIN,XMAX,MAXX,IX)
0010           CALL REPEAT('*',STRING,MAXY)
0011           IF(IX.EQ.1 .OR. IX.EQ.MAXX) GOTO 20
0013            DO 10 IY=2,MAXY-1
0014                Y=SCAL(YMIN,YMAX,MAXY,IY)
0015                IFUN=2+INT(FLOAT(MAXF-3)*(FUN(X,Y)-FMIN)/(FMAX-FMIN))
0016  10            STRING(IY)=STAB(MIN0(MAXF,MAX0(1,IFUN)))
0017  30        CALL PUTSTR(7,STRING,' ')
0018        CALL EXIT
0019        END
FORTRAN IV      Diagnostics for Program Unit .MAIN.

In line 0003,  Error:   Modes of variable 'XMIN' and data item differ
In line 0004,  Error:   Modes of variable 'YMIN' and data item differ
In line 0008,  Error:   Reference to undefined statement label
In line 0012,  Error:   Reference to undefined statement label
In line 0016,  Error:   Wrong number of subscripts for array 'STRING'

FORTRAN IV      Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000334 (  110. words)

Name   Type  Offset     Name   Type  Offset     Name   Type  Offset
FMAX   R*4   000230     FMIN   R*4   000224     IFUN   I*2   000312
IX     I*2   000274     IY     I*2   000300     K      I*2   000256
MAXF   I*2   000260     MAXX   I*2   000272     MAXY   I*2   000276
MAXZ   I*2   000254     MAX0   I*2   000316     MIN0   I*2   000314
```

```
X       R*4   000262      XMAX    R*4   000266      XMIN    R*4   000214
Y       R*4   000302      YMAX    R*4   000306      YMIN    R*4   000220
ZMAX    R*4   000250      ZMIN    R*4   000244

Local and COMMON Arrays:

Name      Type    Section Offset  ------Size-----  Dimensions
STAB    L*1       $DATA   000047  000144 (   50.)  (100)
STRING  L*1 Vec   $DATA   000000  000047 (   20.)  (13,3)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name    Type    Name    Type    Name    Type    Name    Type    Name    Type
EXIT     R*4    FLOAT    R*4    FUN      R*4    INT      I*2    LEN      I*2
PUTSTR   R*4    REPEAT   R*4    SCAL     R*4    SCOPY    R*4

FORTRAN IV      V02.1-10    Thu 11-Dec-80 01:18:55              PAGE 001

0001          FUNCTION FUN(X,Y)
0002          R=SQRT(X**2+Y**2)
0003          FUN=X*Y*R*EXP(-R))**2
***** P
0004          RETURN
0005          END

FORTRAN IV      Diagnostics for Program Unit FUN

In line 0003,  Error:   [See source listing]

FORTRAN IV      Storage Map for Program Unit FUN

Local Variables, .PSECT $DATA, Size = 000020 (     8. words)

Name    Type  Offset      Name    Type  Offset      Name    Type  Offset
FUN     R*4   000004 Eqv R        R*4   000010      X       R*4 @ 000000
Y       R*4 @ 000002

Subroutines, Functions, Statement and Processor-Defined Functions:

Name    Type    Name    Type    Name    Type    Name    Type    Name    Type
SQRT     R*4
```

The first part of the listing shows the main program unit and consists of the language statements up to, but not including, the function. This is followed by a diagnostics list, then by a storage map. Next the language statements comprising the function program unit are listed, again followed by a diagnostics list and a storage map.

Before considering the individual sections of the program listing, first examine the program logic to determine what this program should do. The first few lines of this program are user comment lines that briefly describe the program. Essentially, this program produces on the terminal a graph of a "three-dimensional" function, FUN(X, Y). The graph is plotted using 45 lines down and 72 characters across the terminal page. The limits of the X and Y axes are +5 and -5. The third dimension, height, is a real number within the range 0 to 1 and is represented in the listing as a number within a scale of 1 to 9. These dimensions are illustrated in Figure 9-3.

The SCAL function determines the value of the next coordinate on the graph. The statements within the DO loops calculate the coordinates using the SCAL function and determine the height value. This is done for an entire line of coordinates across the terminal page. The entire line is then printed on the terminal, using the CALL PUTSTR statement; the number 7 in this statement is the FORTRAN method of naming the terminal as the output device. This procedure is repeated until all 45 lines of the graph have been printed.

rately with artificial data. After you test all parts individually, you can test routine and module linkage — system testing — to see that all related code fits together properly.

Check the program with test data. A standard method for checking out modules is to write a test program that calls the program with possible options. The test should cause the program to execute all steps in all algorithms. Check programs first with representative data, then with improper data (data that is not in the correct range or size). You should also do volume testing to see that the program works successfully with a representative amount of data.

Each programming language has special debugging aids for examining immediate states. For example, BASIC has a STOP statement that you can insert at strategic points in the program. When the program arrives at a STOP statement, it pauses so that you can use BASIC's immediate mode to examine variables, values, and so on. Use an immediate mode GO TO statement pointing to the appropriate line number to continue execution.

FORTRAN IV has a special DEBUG statement indicator, a D in the first column of a statement line. Operations in statements marked with a D can perform useful debugging functions, such as printing intermediate results. You can treat such statements as source text (and thus execute them) or as comments (and thus ignore them), depending on whether you use a special compiler command option. In addition, FORTRAN IV has a traceback feature that locates the actual program unit and line number of a run-time error. If the program unit is a subroutine or function subprogram, the error handler traces back to the calling program unit and displays the name of that program unit and the line number where the call occurred. This process continues until the calling sequence has been traced back to a specific line number in the main program unit. Finally, FORTRAN IV has an optional interactive debugger called FDT (FORTRAN DEBUGGING TECHNIQUE) that can be linked with a user program.

For MACRO-11 users, RT-11 provides a special on-line debugging tool called ODT (On-line Debugging Technique). This is provided as part of the RT-11 operating system and is an object program on your system volume. It is used exclusively for debugging assembled MACRO-11 programs.

The use of ODT is described next for MACRO-11 users and for those FORTRAN IV users who will be combining MACRO and FORTRAN program code. Other users can continue to Chapter 15, or go back and perform one of the other language demonstrations. Refer to the reading path outlined in the Preface.

USING THE ON-LINE DEBUGGING TECHNIQUE

ODT is an interactive debugging tool that allows you to monitor program execution from the console terminal. ODT is provided as the object module ODT.OBJ on your system volume. To use it, you link ODT.OBJ with the assembled MACRO program that needs debugging. You then start execution of the resulting load module, not at the transfer address of your program, but at the entry point of the ODT module (shown on the linker load map as the global symbol O.ODT). Once ODT is started, you can use its special debugging commands to control the execution of your assembled machine language program from the console terminal, to examine memory locations, to change their contents, and to stop and continue program execution.

The MACRO demonstration program in Chapter 11 still contains one error, which you can locate and correct using ODT. Several ODT debugging commands are demonstrated in the process.

Throughout the examples in this chapter you need to refer to the program assembly listing that you produced in Chapter 11 (SUM) and stored on the storage volume. Print it now on either the terminal or line printer:

Long Command Format

(Line printer)

```
.PRINT(RET)
Files? VOL:SUM.LST(RET)
```

(Terminal)

```
.TYPE(RET)
Files? VOL:SUM.LST(RET)
```

Short Command Format

(Line printer)

```
.PRINT VOL:SUM.LST(RET)
```

(Terminal)

```
.TYPE VOL:SUM.LST(RET)
```

```
SUM.MAC VERSION 1       MACRO V04.00  12-DEC-80 00:02:09 PAGE 1

     1                                  .TITLE SUM.MAC  VERSION 1
     2
     3                                  .MCALL .TTYOUT, .EXIT, .PRINT
     4
     5
     6
     7        000106                    N = 70.          ;NO. OF DIGITS OF 'E' TO CALCULATE
     8
     9                                  ;       'E' = THE SUM OF THE RECIPROCALS OF THE FACTORIALS
    10                                  ;       1/0! + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...
    11
    12 000000                           EXP:    .PRINT  #MESSAG          ;PRINT INTRODUCTORY TEXT
    13 000006  012705  000106           FIRST:  MOV     #N,R5            ;NO. OF CHARS OF 'E' TO PRINT
    14 000012  012700  000107                   MOV     #N+1,R0          ;NO. OF DIGITS OF ACCURACY
    15 000016  012701  000124'                  MOV     #A,R1            ;ADDRESS OF DIGIT VECTOR
    16 000022  006311                   SECOND: ASL     @R1              ;DO MULTIPLY BY 10 (DECIMAL)
    17 000024  011146                           MOV     @R1,-(SP)        ;SAVE *2
    18 000026  006311                           ASL     @R1              ;*4
    19 000030  006311                           ASL     @R1              ;*8
    20 000032  062621                           ADD     (SP)+,(R1)+      ;NOW *10, POINT TO NEXT DIGIT
    21 000034  005300                           DEC     R0               ;AT END OF DIGITS?
    22 000036  001371                           BNE     SECOND           ;BRANCH IF NOT
    23 000040  012700  000106                   MOV     #N,R0            ;GO THRU ALL PLACES, DIVIDING
    24 000044  014103                   THIRD:  MOV     -(R1),R3         ;BY THE PLACES INDEX
    25 000046  012702  177777                   MOV     #-1,R2           ;INIT QUOTIENT REGISTER
    26 000052  005202                   FOURTH: INC     R2               ;BUMP QUOTIENT
    27 000054  160003                           SUB     R0,R3            ;SUBTRACT LOOP ISN'T BAD
    28 000056  103375                           BCC     FOURTH           ;NUMERATOR IS ALWAYS   10*N
    29 000060  060003                           ADD     R0,R3            ;FIX REMAINDER
    30 000062  010311                           MOV     R3,@R1           ;SAVE REMAINDER AS BASIS
    31                                                                   ;FOR NEXT DIGIT
    32 000064  060261  177776                   ADD     R2,-2(R1)        ;GREATEST INTEGER CARRIES
    33                                                                   ;TO GIVE DIGIT
    34 000070  005300                           DEC     R0               ;AT END OF DIGIT VECTOR?
    35 000072  001364                           BNE     THIRD            ;BRANCH IF NOT
    36 000074  014100                           MOV     -(R1),R0         ;GET DIGIT TO OUTPUT
```

```
37 000076  162700  000012          FIFTH:  SUB     #10.,R0         ;FIX THE 2.7 TO .7 SO
38                                                                  ;THAT IT IS ONLY 1 DIGIT
39 000102  103375                          BCC     FIFTH           ;(REALLY DIVIDE BY 10)
40 000104  062700  000070                  ADD     #10+'0,R0       ;MAKE DIGIT ASCII
41 000110                                  .TTYOUT                 ;OUTPUT THE DIGIT
42 000114  005011                          CLR     @R1             ;CLEAR NEXT DIGIT LOCATION
43 000116  005305                          DEC     R5              ;MORE DIGITS TO PRINT?
44 000120  001334                          BNE     FIRST           ;BRANCH IF YES
45 000122                                  .EXIT                   ;WE ARE DONE
46
47 000124  000107                  A:      .REPT   N+1
48                                         .WORD   1               ;INIT VECTOR TO ALL ONES
49                                         .ENDR
50
51 000342     124     110     105  MESSAG: .ASCII  /THE VALUE OF E IS:/ <15><12> /2./ <200>
   000345     040     126     101
   000350     114     125     105
   000353     040     117     106
   000356     040     105     040
   000361     111     123     072
   000364     015     012     062


SUM.MAC VERSION 1       MACRO V04.00  12-DEC-80 00:02:09 PAGE 1-1

      000367     056     200
     52                                    .EVEN
     53
     54          000000'                   .END    EXP




SUM.MAC VERSION 1       MACRO V04.00  12-DEC-80 00:02:09 PAGE 1-2
SYMBOL TABLE

A       000124R         FIFTH   000076R         FOURTH  000052R         N       = 000106        THIRD   000044R
EXP     000000R         FIRST   000012R         MESSAG  000342R         SECOND  000022R

. ABS.  000000      000
        000372      001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  8448 WORDS  ( 33 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  35 PAGES
DK:SUM,DK:SUM/C=DK:SUM




SUM.MAC VERSION 1       MACRO V04.00  12-DEC-80 00:02:09 PAGE S-1
CROSS REFERENCE TABLE (CREF V04.00 )

A       1-15    1-47#
EXP     1-12#   1-54
FIFTH   1-37#   1-39
FIRST   1-14#   1-44
FOURTH  1-26#   1-28
MESSAG  1-12    1-51#
N       1-7#    1-13    1-14    1-23    1-47
SECOND  1-16#   1-22
THIRD   1-24#   1-35




SUM.MAC VERSION 1       MACRO V04.00  12-DEC-80 00:02:09 PAGE M-1
CROSS REFERENCE TABLE (CREF V04.00 )

.EXIT   1-3#    1-45
.PRINT  1-3#    1-12
.TTYOU  1-3#    1-41
```

Now link the MACRO program object module (SUM.OBJ) stored on the storage volume (VOL:) with ODT.OBJ by using the /DEBUG option, and print a load map directly on the terminal or line printer, choosing one of the following commands:

LINK/DEBUG

Long Command Format

(Line printer)

```
.LINK/MAP/DEBUG RET
Files? VOL:SUM RET
```

(Terminal)

```
.LINK/MAP:TT:/DEBUG RET
Files? VOL:SUM RET
```

Short Command Format

(Line printer) (Terminal)

```
.LINK/MAP/DEBUG VOL:SUM(RET)          .LINK/MAP:TT:/DEBUG VOL:SUM(RET)
RT-11 LINK  V06.01          Load Map          Fri 11-Jan-80 13:11:26
SUM   .SAV         Title:   ODT       Ident:   V04.00

Section  Addr   Size    Global  Value   Global  Value   Global  Value

 . ABS.  000000 001000   (RW,I,GBL,ABS,OVR)
         001000 000372   (RW,I,LCL,REL,CON)
$ODT$    001372 006152   (RW,I,LCL,REL,CON)
                             0.ODT   001624

Transfer address = 001624, High limit = 007542 = 1969. words
```

Look at the load map, and note that ODT starts at address 1372. The two modules together, ODT and SUM, reside in memory up to location 7542, the high limit. Look at the symbol table listing for the MACRO program. This shows that the program is 372 (octal) bytes long and starts at location 1000.

To load and start execution of the load module, use the monitor RUN command. The RUN command brings the entire load module, called SUM.SAV, into the absolute (physical) memory locations shown in the load map and begins execution automatically at the starting, or transfer, address of the first module in memory, which is ODT. Type:

Long and Short Command Format

```
.RUN SUM(RET)
 ODT  V04.00
*
```

ODT prints an identifying message on the terminal and an asterisk indicating that you are in ODT command mode and can enter an ODT command. You are now using ODT to control the execution of your program.[1] ODT commands let you execute the entire program or just portions of it, examine individual locations, examine the contents of the PDP-11 general registers, and change the contents of any locations in your program you wish. If you make a mistake while you are typing any commands, type the DEL key; ODT responds with a ? and an asterisk, allowing you to enter another command.

[1] Be sure to read Chapter 16 of the *RT-11 System User's Guide* before you use ODT with any of your own programs. You must observe certain precautions when you write your program and when you load it with ODT. For example, you should make sure that ODT is not loaded into memory locations used by your program. There are steps you can take to prevent this from occurring.

(.TTYOUT), but that it occurs somewhere before location 110. So the next step in debugging this program is to set a breakpoint at some earlier point in the program logic and to rerun the program. You must restart ODT to do this. Return to monitor mode by typing CTRL/C. The remainder of the program message prints on the terminal; then the monitor period appears, indicating that you are in monitor mode:

```
*CTRL/C
#VALUE OF E IS:
2.
.
```

Restart ODT and reset relocation register 0:

```
.RUN SUM RET

ODT   V04.00
*1000;0R
```

Set a breakpoint at location 76 (line 37 in the assembly listing), and start program execution at its beginning:

```
*0,76;0B
*0,0;G
TB0;0,000076
```

Again, examine register 0 to verify its contents:

```
*$0/000033 RET
```

By following the program logic in the assembly listing, you know that the value in register 0 should at this point be 33(octal) (2.7, previously multiplied by 10, = 27[decimal] = 33[octal]). So the value in register 0 is correct. From this, you can deduce that the error must occur somewhere between locations 76 and 110. The proper step now is to check the assembly listing, where you find the error at line 40. The decimal point that should follow the 10, identifying it as a decimal 10, is missing. Therefore the program treats the 10 as an octal 10, or 8(decimal), making each digit in the result off by an additive factor of 2. The data in location 106, then, should be 72, not 70. Since this data has not yet been used, you can change it now with ODT and continue program execution; if it had been used, you would need to restart ODT and then change the data. To change the contents of a location, simply open the location, type in the new contents, and close the location, using a carriage return.

```
*0,106/000070 72 RET
```

Now eliminate all breakpoints.

```
;B
```

Continue program execution; the correct results should print:

```
*;P
THE VALUE OF E IS:
2.71828182845904523536028747135266249775724709369995957496696762772407663
.
```

SUMMARY: COMMANDS FOR DEBUGGING PROGRAMS

To Start ODT

LINK/DEBUG
Link the assembled program (the program to be debugged) with the ODT object module.

To Use ODT[1]

(LF)
Close the currently open location and open the next sequential location for examination and possible modification.

(RET)
Close the currently open location.

addr/
Open the location indicated (addr) for examination and possible modification.

addr;G
Begin program execution at the indicated address (addr).

;P
Continue program execution from a previous breakpoint.

addr;nB
Set one of the eight available breakpoints (n) at the indicated address (addr).

;nB
Cancel the indicated breakpoint (n).

;B
Cancel all breakpoints.

addr;nR
Set one of the eight available relocation registers (n) to the relocation constant value indicated by addr.

[1] Only a very few of the available debugging commands have been demonstrated in this chapter. Consult Chapter 21 of the *RT-11 System User's Guide* for all ODT commands.

Long Command Format

```
.LINK/FOREGROUND (RET)
Files? DEMOFG (RET)
```

Short Command Format

```
.LINK/FOREGROUND DEMOFG (RET)
```

Executing The Foreground and Background Jobs

Now you are ready to operate the two-job environment. Many times, you have to consider the devices that are used for output in a foreground/background environment. For example, if your program assumes that the output device is a line printer, and you do not have a line printer or you want to output to another device, use the ASSIGN command. Type this command in the following way, substituting the two-character code from Table 4–2 for the storage volume in place of xx.

Long Command Format

```
.ASSIGN (RET)
Physical device name? xx: (RET)
Logical device name: LP: (RET)
```

Short Command Format

```
.ASSIGN xx: LP: (RET)
```

You do not have to consider the above information for the demonstration programs that are provided, since the foreground job communicates with the background job, and both jobs send their output to the terminal.

LOAD

When you use the FB monitor, you must always load into memory the peripheral device handlers needed by the foreground job. You use the monitor LOAD command to make a device handler permanently resident in memory. For example, if your foreground job requires the use of the line printer, you must load the LP device handler. You must specify the jobtype with the command. For a foreground job, the jobtype is F; for a background job, the jobtype is B. If you have assigned the code LP: to another device, the system automatically loads the assigned handler and you need not enter a LOAD command. If you are using the line printer, type:

Long Command Format

```
.LOAD (RET)
Device? LP:=F (RET)
```

Short Command Format

```
.LOAD LP:=F (RET)
```

FRUN

The command to load and start execution of the foreground job is FRUN, which is similar to the RUN command except that the system automatically loads and starts the execution of the foreground .REL program. You must use the /BUFFER:n option with the FRUN command to execute a FORTRAN foreground job. The argument n represents, in octal, the number of words of memory to allocate. Use this command to start the execution of DEMOFG.REL.

Long and Short Command Format

```
.FRUN/BUFFER:n DEMOFG (RET)

F>
FOREGROUND DEMONSTRATION PROGRAM
SENDS A MESSAGE TO THE BACKGROUND PROGRAM "DEMOBG"
EVERY 2 SECONDS, TELLING IT TO RING THE BELL.

B>
```

The foreground program DEMOFG is now running and queuing the message for the background program every two seconds. You now execute the background program DEMOBG to allow it to receive the messages that were queued and to ring the bell.

```
.RUN DEMOBG (RET)
RT-11 DEMONSTRATION PROGRAM
IF INCORRECTLY EDITED, THIS IS THE LAST LINE.
WELL DONE.
```

The bell rings several times in rapid succession as the monitor dequeues the messages, and then every two seconds as the foreground job sends its message to the background job.

You can run other jobs in the background. You can use the background of an FB environment in the same way as the SJ environment. First, terminate the background job DEMOBG, using the double CTRL/C command.

```
(CTRL/C)
(CTRL/C)

.
```

Execute a DIRECTORY command in the background to get a listing of all the .OBJ files on the system volume by typing

```
.DIRECTORY *.OBJ (RET)
```

To activate other bootstraps, set the numbers into the pushbuttons, using the following method (if you make a mistake, push the button labeled CLR, then reenter the number):

1. Push 1000 (read the number from left to right).
2. Push LAD.
3. Push the appropriate buttons for the first number in the Contents column (read the number from left to right).
4. Push DEP; push CLR.
5. Push the appropriate buttons for the next number in the Contents column (read the number from left to right).
6. Repeat steps 4 and 5 until all numbers in the column have been used.
7. Push 1000.
8. Push LAD.
9. Push the button labeled CNTRL, and, while holding it down, push the button labeled START.
10. Continue to step 11 in Chapter 2.

USING A SWITCH REGISTER CONSOLE TO BOOTSTRAP

If your computer has a switch register console on the front panel similar to those shown in Figure A–2, you can use the switches to manually give the computer the bootstrapping information it needs to start the system.

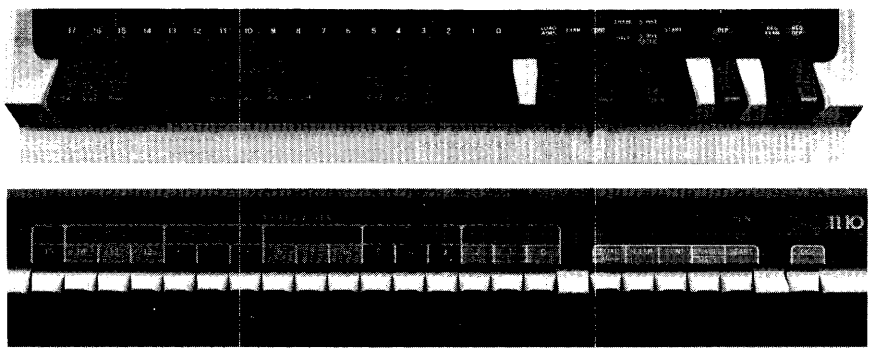

Figure A–2 Switch Register Consoles

Several switches on the console are spring-loaded. This means that the switch moves in only one direction and returns to its initial position after you use it. You must set the remaining switches either up or down as instructed.

The bootstrap for your RT-11 computer system consists of a series of six-digit numbers that you must load into the computer using the switch register console. First, obtain the bootstrap of your system device from the *RT-11 Installation and System Generation Guide*, and copy the numbers into the space provided below. If your system has a hardware bootstrap,[1] the bootstrap consists of only two numbers, which you should copy into the left-hand space; otherwise, the bootstrap consists of two columns of numbers labeled Location and Contents, which you should copy into the right-hand space:

Hardware Bootstrap	Other Bootstraps
Load Address =	
Start Address =	

Next, convert the numbers in the column to binary numbers, using the conversion process shown in Table A-1.

Table A-1: Binary Conversion

Octal		Binary
0	=	000
1	=	001
2	=	010
3	=	011
4	=	100
5	=	101
6	=	110
7	=	111

For example, the number 173100 is converted to 001 111 011 001 000 000. You set this 18-digit binary number into the switch register by placing each individual switch in an up position for a 1 or a down position for a 0. The number 173100 is set into the switch register as follows:

↓↓↑ ↑↑↑ ↓↑↑ ↓↓↑ ↓↓↓ ↓↓↓

The number 012700 is converted to 000 001 010 111 000 000 and is set into the switch register as follows:

↓↓↓ ↓↓↑ ↓↑↓ ↑↑↑ ↓↓↓ ↓↓↓

[1] A hardware bootstrap is bootstrapping information that is already in computer memory but that you must activate by entering a load address and a start address, each a six-digit number.

READER'S COMMENTS

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.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please 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) _____

Name _____ Date _____

Organization _____

Street _____

City _____ 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

CSD SOFTWARE PUBLICATIONS ML 5-5/E45
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

— — Do Not Tear - Fold Here —

Cut Along Dotted Line