RSX

MULTI-TASKER

JULY 1985

RSX Multitasker

Table Of Contents

The Bag Of Tricks: MACRO-11

Bruce R. Mitchell
Machine Intelligence and Industrial Magic
PO Box 601
Hudson, WI 54016

This column covers MACRO-11 bag-of-tricks routines, as stated in previous issues of the Multi-Tasker. All MACRO programmers are encouraged to submit their favorite routines to the Multi-Tasker so that these useful, interesting, or just plain bizarre tricks can be put out before the SIG in general for the admiration and edification of all.

In this month's column, we have a bit of code which is at once both quite trivial and very useful.

When the Taskbuilder builds a task, it leaves information in the storage locations for R3 and R4 (general registers) in the task header. The information left in those locations is the task's identification, as specified either through the .IDENT directive or the Taskbuild IDENT= directive. When the task is loaded, of course, those stored locations are loaded into the CPU's general registers.

If the task captures those registers as soon as it initializes, it is able to do a number of useful things with that information. It can print it on the system console as part of a startup message, to log the version number of a task which must be up to current revision level. It can send it to cooperating tasks, to inform them of what level of interchange protocol it can understand. It can gleefully squirrel it away and hoard it where nobody will ever find it (admittedly not very useful). Other uses are possible.

An example of the entry point of a task which captures this information follows. Note that the information supplied in the registers is Radix-50 and must be converted to ASCII to be useful.

```
IDENT:  .BLKB   6               ; Storage for ASCII version name


        .PAGE
        .SBTTL  NTRYPT Entry Point


;       Before doing anything else, load and save the
;       task version number

NTRYPT: MOV     #IDENT,R0       ; Load version field address in R0
        MOV     R3,R1           ; Get first 3 Rad50 characters
        CALL    $C5TA           ; Convert to ASCII
        MOV     R4,R1           ; Get second 3 Rad50 characters
        CALL    $C5TA           ; Convert to ASCII

        ...                     ; task continues ...

        .END    NTRYPT
```

Building RPT For Faster Error Reports

B. Z. Lederman
Bankers Trust Co.
P.O Box 318, Church Street Station
New York, N.Y. 10015

Recently, I received a complaint that it was taking too long to obtain the error log summary reports. (The reports are automatically printed out as part of our SHUTUP procedure, as a warning in case there were disk errors while making backup copies of system disks, or other errors.) When I looked into the matter, I discovered that the RPT program was very heavily overlayed, and that it might run faster if built with fewer or no overlays. I built a copy of the task with no overlays at all (using the supervisor mode FCS resident library on M-Plus), and tested it on a lightly loaded 11/70 using some of the typical report commands we use here and a log file which was 391. blocks long and contained data from 07-Nov-84 through 16-Jan-85. The results were quite impressive. A scan for errors which occured during the past week with the command

```
RPT LOG1.RPT=[1,6]LOG.ERR;11/W:N/SU:A/DA:P:7
```

was reduced in time from 3 minutes 40 seconds to only 40 seconds. A report listing all error log entries in short format in the file:

```
RPT LOG3.RPT=[1,6]LOG.ERR;11/W:N
```

was reduced in time from 15 minutes 24 seconds to 3 minutes 56 seconds (the resulting report file is 510. blocks long). I compared the output of the two versions of RPT with the CMP utility, and found them to be identical. Measuring the two tasks with System Accounting and SPM-11 shows a great reduction in the system resources consumed by the task: not only are all of the disk overlays gone, but the number of CPU cycles consumed and disk QIOs issued are also reduced. Both versions extend themselves to create internal work space, and end up about the same size when doing large reports (they will both approach the 32K word task size limit). I also did a little testing with an identical non-overlayed version, but which also is built for I-and-D space, so it would have even more internal work space available. It expanded to 46K words, but had only a slight improvement in resources used (somewhat fewer disk I/Os, probably from not having to re-read from the ERRLOG.ULB library) over the non-overlayed task without I-and-D space. We are using this version of RPT on our development system now, and plan to use it on all of our systems in the future.

If you do not have a system which supports I-and-D space or Supervisor mode this approach does not work, as there won't be enough internal work space left after all of the program and FCS code is built into the task image. The program will give you summaries, but cannot process any of the device packets as it apparently cannot get the appropriate modules out of the ERRLOG library and into memory. Though I have not done much testing in this area (as I have a version which suits my needs) I did test a version which has fewer overlays than that distributed by DEC, but which had enough work space to process an error log file. This is an area where someone could do a little testing to find out how "flat" the program can be made, and still have enough space left to work. There might also be a balance between reading overlays and having to read the ERRLOG library.

The following is the command file to build the non-overlayed version. Please be aware of the usual DECUS disclaimer, that all information is supplied "AS IS", with no warranty whatsoever.

```
; TKB BUILD FILE FOR RPT Non-overlayed
; ON AN RSX-11M-PLUS SYSTEM
; LINKING TO FCSFSL
; B. Z. Lederman
;
```

```
OU:[3,54]RPTFSL/-FP/CP/MM/ID, MP:[1,34]RPTID/-SP/MA =
;
[1,24]RPT/LB:CTLRPT:GLODAT:FILEIO:INTUTL:VALUTL:INTERP
[1,24]RPT/LB:OPERAT:ARITH:RSXASC:CNVFUN:COMFUN:CNDFUN
[1,24]RPT/LB:RPTINI:USRFUN:CTLFUN:RPTFUN:FILEGE:FILECM
[1,24]RPT/LB:FILEOP:MEMORY:LIMITS:INTSTA:PKTFUN:FILEIN:OPRSUB
[1,24]RPT/LB:MODHAN:FILECO:INTEXT:INTPUT
[1,24]RPT/LB:RADVAL:FILERE:PAGBRK:LOKFUN:FORMAT:INTWRI
[1,24]RPT/LB:CODFUN:TIMCNV:STRFUN:VALRAD:TIMFUN:ARITHE
[1,24]RPT/LB:PDP11:MSGRPT:MSGCOM:DISPLA:DECLAR:SIGNAL
;
[1,24]NEISLB/LB
;
/
TASK=...RPT
IDENT=02.01
;
PAR=GEN:0:0
PRI=50
UNITS=10
;
ASG=TI:1            ; COMMAND LUN
ASG=TI:2            ; ERROR LUN
ASG=SY:3            ; INPUT LUN
ASG=SY:4            ; OUTPUT LUN
ASG=SY:5            ; REPORT LUN
ASG=TI:6            ; USER LUN
ASG=SY:7            ; CONTROL LUN
ASG=SY:10           ; MODULE LUN
;
SUPLIB=FCSFSL:SV
;
; Parameters Options
;
GBLDEF=LPLINE:74
;
; USERCM is non-zero to have RPT prompt for the control file to
use.
;
GBLDEF=USERCM:0
;
; Module RWLONG is not used
;
GBLDEF=..RWLG:177777
GBLDEF=FDAMOD:177777
GBLDEF=FDAOUT:177777
;
; To increase the size of the execution stack
; increase the extension by multiples of
; 10 (octal) bytes.
;
EXTSCT=XCSTK0:300
;
```

```
; To increase the size of the value stack,
; increase the size of PSECT VLSTK0 by multiples
; of 2.
;
EXTSCT=VLSTK0:50
;
; To increase the size of the value heap,
; increase the size of PSECT VHEAP0
;
EXTSCT=VHEAP0:732
;
; To increase the size of the declaration stack
; increase the extension by multiples of
; 10 (octal) bytes.
;
EXTSCT=DCSTK0:10
/
```

I havn't tried to change any of the values in the build file to see if they would have an effect on the speed of the report.

The following are the build command file and ODL file to build a version with fewer overlays than the distributed version of RPT.

```
; TKB BUILD FILE FOR RPT with fewer overlays.
; ON AN RSX-11M-PLUS SYSTEM (or RSX-11M)
; LINKING TO FCSRES
; B. Z. Lederman
;
OU:[3,54]RPTRES/-FP/CP/MM, MP:[1,34]RPTRES/-SP/MA = RPTRES/MP
;
TASK=...RPT
IDENT=02.01
;
PAR=GEN:0:0
PRI=50
UNITS=10
;
ASG=TI:1            ; COMMAND LUN
ASG=TI:2            ; ERROR LUN
ASG=SY:3            ; INPUT LUN
ASG=SY:4            ; OUTPUT LUN
ASG=SY:5            ; REPORT LUN
ASG=TI:6            ; USER LUN
ASG=SY:7            ; CONTROL LUN
ASG=SY:10           ; MODULE LUN
;
LIBR=FCSRES:RO
;
; Parameters Options
;
GBLDEF=LPLINE:74
;
```

```
; USERCM is non-zero to have RPT prompt for the control file to
use.
;
GBLDEF=USERCM:0
;
; Module RWLONG is not used
;
GBLDEF=..RWLG:177777
GBLDEF=FDAMOD:177777
GBLDEF=FDAOUT:177777
;
; To increase the size of the execution stack
; increase the extension by multiples of
; 10 (octal) bytes.
;
EXTSCT=XCSTK0:300
;
; To increase the size of the value stack,
; increase the size of PSECT VLSTK0 by multiples
; of 2.
;
EXTSCT=VLSTK0:50
;
; To increase the size of the value heap,
; increase the size of PSECT VHEAP0
;
EXTSCT=VHEAP0:732
;
; To increase the size of the declaration stack
; increase the extension by multiples of
; 10 (octal) bytes.
;
EXTSCT=DCSTK0:10
/
```

```
;
; TKB ODL FILE FOR RPT
; ON AN RSX-11M-PLUS SYSTEM (or RSX-11M)
; LINKING TO FCSRES
; CREATED BY SYSGEN3.CMD VERSION 1.82
;
;  B. Z. Lederman  Reduce number of overlays.
;
   .NAME   RPT

   .ROOT   BASE-*(EXECUT, MESSAG), DECSIG-*(DECLAR, SIGNAL)

BASE:      .FCTR   [1,24]RPT/LB:CTLRPT:GLODAT:FILEIO-BASE1
BASE1:     .FCTR   [1,24]RPT/LB:INTUTL:VALUTL-BASE2
BASE2:     .FCTR   LB:[1,1]SYSLIB/LB:FCSFSR:SAVR1-LIB
```

```
EXECUT:     .FCTR   MAIN-*(OPEN, other)
other:      .fctr   INPUT-OUTPUT-CODFUN
MAIN:       .FCTR   [1,24]RPT/LB:INTERP-MAIN1
MAIN1:      .FCTR   [1,24]RPT/LB:OPERAT:ARITH:RSXASC-MAIN2
MAIN2:      .FCTR   [1,24]RPT/LB:CNVFUN:COMFUN:CNDFUN-LIB
;
; File handling overlays
;
OPEN1:      .FCTR   [1,24]RPT/LB:RPTINI:USRFUN:CTLFUN
OPEN2:      .FCTR   [1,24]RPT/LB:RPTFUN-[1,24]NEISLB/LB
FILEGE:     .FCTR   [1,24]RPT/LB:FILEGE:FILECM-GCML
OPEN:       .FCTR   OPEN1-OPEN2-FILEGE-FILEOP
FILEOP:     .FCTR   [1,24]RPT/LB:FILEOP
GCML:       .FCTR   LB:[1,1]SYSLIB/LB:.GCML
;
; Execution overlays
;
INPUT:      .FCTR
   [1,24]RPT/LB:MEMORY:LIMITS:INTSTA-LIB-INPUT1-INPUT2
INPUT1:     .FCTR   [1,24]RPT/LB:PKTFUN:FILEIN:OPRSUB-LIB
INPUT2:     .FCTR   [1,24]RPT/LB:MODHAN:FILECO:INTEXT:INTPUT-LIB

OUTPUT:     .FCTR
   [1,24]RPT/LB:RADVAL:FILERE:PAGBRK:LOKFUN:FORMAT:INTWRI-LIB

CODFUN:     .FCTR
   [1,24]RPT/LB:CODFUN:TIMCNV:STRFUN:VALRAD:TIMFUN:ARITHE-LIB

MESSAG:     .FCTR   FILERE-PDP11-DISPLA
FILERE:     .FCTR   [1,24]RPT/LB:FILERE:PAGBRK-LIB
PDP11:      .FCTR   [1,24]RPT/LB:PDP11-MSGRPT-MSGCOM
MSGRPT:     .FCTR   [1,24]RPT/LB:MSGRPT
MSGCOM:     .FCTR   [1,24]RPT/LB:MSGCOM
DISPLA:     .FCTR   [1,24]RPT/LB:DISPLA
;
; DECSIG co-tree
;
   .NAME    DECSIG
DECLAR:     .FCTR   [1,24]RPT/LB:DECLAR
SIGNAL:     .FCTR   [1,24]RPT/LB:SIGNAL

LIB:        .FCTR   [1,24]NEISLB/LB-LB:[1,1]SYSLIB/LB

            .END
```

Password Encryption For RSX-11M

Wayne E. Kesling, Sr. Engr.
Monsanto Research Corp.
Mound Rd., M-225
Miamisburg, Ohio 45342

The need for password encryption for RSX-11M users is partially due to the fact that a priviledged user can TYPE the file that contains accounting information ([0,0]RSX11.SYS) and see the password of all users on the system. During boot-up, anyone can CONTROL-C out of the start-up command file before the BYE command and TYPE the above file. To prevent the person from obtaining the passwords in this manner, they can be encrypted before they are entered into this file by ACNT. The encryption implementation that is described below is a one-way process. That is, there is encryption but no decryption. Keep in mind that the encryption routine described is strictly an example. Any other algorithm could be used, including the DES algorithm. However, the one described would be very difficult to reverse and, therefore, would be very difficult to break without a decryption routine. The use of a KEY to EXCLUSIVE-OR with the three-word password is unnecessary with a one-way process but is included to give an example of the use of a KEY. Also, if a KEY is to be used, it should not be a part of the source code as in this example. If it is a part of the source code, it should be disguised in some way that would make it less obvious to an intruder. Another way of obtaining a key would be to read a hardware register that has been preconditioned to a known value. If, for example, you used the contents of the date code register in a hardware clock, the contents of the RSX11.SYS file would change every day for every password. This would require a task to run sometime after midnight each day to update the contents of RSX11.SYS. The fact that the encrypted passwords changed every day would be transparent to the users, but would be very confusing to an intruder.

It should be noted that, when encryption is in use, ACNT will not display the passwords. However, ACNT should be removed from the system to prevent anyone from creating their own account after gaining access by hitting CONTROL-C during boot-up.

There are two major task to perform to implement password encryption under RSX-11M. They are:

Modification of build files.
Writing the encryption routine.

If you are running a stand-alone system, there are four files to modify.

```
ACNBLD.ODL - TKB Overlay descriptor file for ACNT
ACNBLD.CMD - TKB Build command file for ACNT

HELBLD.ODL - TKB Overlay descriptor file for HELLO
HELBLD.CMD - TKB Build command file for HELLO
```

If you are running DECnet, there are two additional files that must be modified. These are responsible for building the Network Verification Program (NVP).

```
NVPBLD.ODL - TKB Overlay descriptor file for NVP
NVPBLD.CMD - TKB Build command file for NVP
```

An addition is made to the .ODL files to indicate that the encryption routine is to be included and which overlay segment it is to be placed in. These modifications are as follows:

[1,24]ACNBLD.ODL

Add overlay module F:

```
;
; TKB ODL FILE FOR ACNT
; ON A MAPPED RSX-11M SYSTEM
; USING SYSLIB
; CREATED BY SYSGEN3.CMD VERSION 2.01
;
        .ROOT    A-B-C-D-E-F
A:      .FCTR    DL3:[1,24]MLTUSR/LB:ACNT-DL3:[1,24]MCR/LB:ACTFIL:GETNUM
B:      .FCTR    DL3:[1,24]MCR/LB-LB:[1,1]VMLIB/LB:VMDAT:MRKPG:ALVRT:CVRL
C:      .FCTR    LB:[1,1]VMLIB/LB:FNDPG:ALBLK:RDPAG:RQVCB:GTCOR:EXTSK
D:      .FCTR    LB:[1,1]VMLIB/LB:INIVM-LB:[1,1]EXELIB/LB
E:      .FCTR    LB:[1,1]SYSLIB/LB:CATB:C5TA:CBTA-SY:[1,54]RSX11M.STB/SS
F:      .FCTR    DR0:[5,4]ENCODE
;
        .END
```

****

[1,24]HELBLD.ODL

Add -DR0:[5,4]ENCODE to segment HELO:

ENCODE.MAC was in the user's UIC [5,4], but could be in a Library.

```
;
; TKB ODL FILE FOR HELLO/HELP
; ON A MAPPED RSX-11M SYSTEM
; USING SYSLIB
; CREATED BY SYSGEN3.CMD VERSION 2.03
;
        .ROOT   A-*(HELO,HELP),FCS
A:      .FCTR   DL3:[1,24]MLTUSR/LB:HELROT-DL3:[1,24]MCR/LB:ACTFIL-A2
A2:     .FCTR   LB:[1,1]SYSLIB/LB:CAT5:CATB:CBTA:CDDMG:SAVR1-LIB1
HELO:   .FCTR   DL3:[1,24]MLTUSR/LB:HELLO-LIB2-LIB1-DR0:[5,4]ENCODE
HELP:   .FCTR   DL3:[1,24]MLTUSR/LB:HLP-*(STTAB,SUB)
LIB1:   .FCTR   SY:[1,54]RSX11M.STB/SS-LB:[1,1]EXELIB/LB/SS
LIB2:   .FCTR   DL3:[1,24]MCR/LB:COLOG:FMTDV:GNBLK:GETNUM
STTAB:  .FCTR   DL3:[1,24]MLTUSR/LB:HSTTAB-LB:[1,1]SYSLIB/LB:.TPARS-LIB1
SUB:    .FCTR   DL3:[1,24]MLTUSR/LB:HLPSUB
;
;

***** Note: This is only a partial listing. *****

;
        .END

*******************************************************************
```

[6,24]NVPBLD.ODL

Add -DR0:[5,4]ENCODE-OVR to segment R00:

ENCODE.MAC was in the user's UIC [5,4], but could be in a Library.

```
;
; NVP (network verification) task overlay description file
;
        .ROOT   R00
;
R00:    .FCTR   IN:[132,24]NVP/LB:NVPMAI:NVPDAT:ACTFIL-DR0:[5,4]ENCODE-OVR
OVR:    .FCTR   (OVR1,OVR2,OVR3)
OVR1:   .FCTR   *IN:[132,24]NVP/LB:NVPINI
OVR2:   .FCTR   *IN:[132,24]NVP/LB:NVPOPN-FCS
OVR3:   .FCTR   *IN:[132,24]NVP/LB:NVPCTL:NVPUTI:NVPVFY:NVPSAI
;

***** Note: This is only a partial listing. *****

;
        .END

*******************************************************************
```

The modification to the .CMD files is simply to comment out the global definition of ENCRPT. This is necessary to prevent multiple definitions of ENCRPT since the encryption routine must declare this global variable.

In each file the only change is the addition of the semicolon in the statement

```
;GBLDEF=ENCRPT:0
```

****

[1,24]ACNBLD.CMD

```
;
; TKB BUILD FILE FOR ACNT
; ON A MAPPED RSX-11M SYSTEM
; USING SYSLIB
; CREATED BY SYSGEN3.CMD VERSION 2.03
;
SY:[1,54]ACNT/AL/-FP/PR/-IP/MM,[1,34]ACNT/SP=
DL3:[1,24]ACNBLD/MP
TASK=ACNT
UNITS=5
ASG=TI:1,SY:2:3
PAR=GEN:0:60000            ;IF ALL OTHERS 12K      ;WMG028
IDENT=05.00                                        ;**-1
GBLDEF=N$MLIN:64    ; OCTAL NUMBER OF LINES PER PRINTED PAGE
GBLDEF=W$KLUN:4     ; WORKFILE LUN
GBLDEF=N$MPAG:20    ; FAST PAGE SEARCH PAGE COUNT
GBLDEF=W$KEXT:24    ; WORK FILE EXTENSION SIZE (BLOCKS)
;GBLDEF=ENCRPT:0    ; ADDRESS OF PASSWORD ENCRYPTION SUBROUTINE (0=NOT USED)
GBLDEF=T$KINC:256   ; TASK INCREMENT SIZE
GBLDEF=T$KMAX:0     ; TASK MAX SIZE
/
```

****

[1,24]HELBLD.CMD

```
;
; TKB BUILD FILE FOR HEL
; ON A MAPPED RSX-11M SYSTEM
; USING SYSLIB
; CREATED BY SYSGEN3.CMD VERSION 2.03
;
SY:[1,54]HEL/-FP/AL/PR/-IP/MM,[1,34]HEL/SP=
DL3:[1,24]HELBLD/MP
```

```
;
TASK=...HEL
IDENT=02.01
STACK=64
UNITS=7
ASG=TI:1:7
ASG=SY:2,CO:3
PAR=GEN:0:0
GBLDEF=HELP$P:101   ; OCTAL PRIORITY OF "HELP"
GBLDEF=$DNTSB:0             ; ADDRESS OF DECNET SUBROUTINE (0=NOT USED)
GBLDEF=$USESB:0             ; ADDRESS OF USER SUBROUTINE (0=NOT USED)
;GBLDEF=ENCRPT:0    ; ADDRESS OF PASSWORD ENCRYPTION ROUTINE (0=NOT USED)
GBLDEF=TIMOUT:20    ; SECONDS BEFORE TIMEOUT ON PROMPTS
GBLREF=$DIV
GBLREF=$MUL
/
```

```
******************************************************************************
```

```
[6,24]NVPBLD.CMD
```

```
;
; NVP (network verification) task build command file
;
OU:[6,54]NVP.TSK/MM/PR/CP/-FP,MP:[6,34]NVP/-SP=
OU:[6,24]NVPBLD/MP
PAR=GEN:0:0
TASK=NVP...
STACK=60
PRI=150
UIC=[1,1]
UNITS=3
;GBLDEF=ENCRPT:0             ;PASSWORD ENCRYPTION ROUTINE (0 = NOT USED)
//
```

The second, and most demanding, task is to write the encryption routine. This routine determines how secure your passwords are. It may be a simple mathematic- al manipulation of the ASCII characters or it may incorporate the DES encryption algorithm.

The address of the password to be encrypted is passed to the routine in R0 and must be saved locally to permit replacement of the original password with the encrypted version. As stated above, the routine must contain the definition

```
    .GLOBL ENCRPT
```

or the label

```
     ENCRPT::
```

and must start with a call to the system routine $SAVAL to save and restore the processor registers. The source code for HELLO and ACNT says all system registers are preserved and can be used. - - - NOT SO. - - -

```
     JSR   PC, $SAVAL
```

The encryption algorithm used in the example is as follows:

1. Save the address that the password is in.

```
     MOV R0,PASWRD
```

2. Convert the six ASCII password characters to RAD50 format. This changes the 3-word information to 2-word.

```
     JSR PC,$CAT5B ;Convert ASCII 3-character string with
                   ;  blanks to RADIX 50 (RAD50).
```

   Set up for this call is as follows;

   1. R0 - Address of first of three characters to be converted.

   2. R1 - Period disposition flag.

```
            0 = Period is terminator.
            1 = Period is valid character.
```

3. $CAT5B saves and restores R3 through R5 and returns the following results:

   1. R0 - Address of the next character in the input buffer.

   2. R1 - The converted RAD50 value.

   3. R2 - The terminating character (last character in the string that was converted, or invalid character that caused termination).

```
      Condition Code:  C (Carry) = 1 Incomplete operation.
                                 = 0 Complete operation.
```

4. Convert the RAD50 data to a random number in floating point format. The RAD50 bit pattern is treated as an integer and is passed to the FORTRAN routine RAN as a seed. Register R5 must point to a data base containing tha data to be converted. The following statements are an example, if DATA is the address of

the data base containing the RAD50 data:

```
          MOV  DATA,R5
          JSR PC,RAN
```

The data base could look like this:

```
NUMBER: .BLKW    3         ; 3-word buffer holding data
DATA:   .WORD    1         ; Beginning of data base
        .WORD    NUMBER    ; Pointer to data buffer
```

The floating point output of RAN is returned in R0 and R1. The fact that the original ASCII password is now represented as a floating point number is irrelevent since all we are interested in is a unique bit pattern.

5. Permute the floating point bit pattern. This rearranges the bit pattern according to a permutation table that is included in code at PERM: and could be the one used by the DES.

   The table must contain one, and only one, value for each bit position in a 16-bit word.

```
i.e.  1,2,4
      10,20,40
      100,200,400
      1000,2000,4000
      10000,20000,40000
      100000
```

6. Add the two permuted floating point words to obtain a third word thus ending up with a 3-word representation of the original 3-word password.

7. these three words with a KEY expression. If the DES algorithm is used, this section of the routine would be different. As noted above, the use of a KEY in a one-way system is not necessary.

8. Replace the original password with the above three words.

9. Clear the CARRY bit to indicate a successful operation and return to the calling program.

```
CLC
RTS PC
```

   In this case, the calling program is actually $SAVAL, which takes care of restoring the processor registers.

The following is a listing of the example ENCODE.MAC routine:

```
   .TITLE ENCRPT
;
;  THIS PROCEDURE ENCRPTS THE PASSWORD BEFORE IT IS
;  USED BY THE SYSTEM. THIS MUST BE ADDED TO THE ODL
;  FILES OF ACNT, HELLO, AND NVP PRIOR TO TASK
;  BUILDING. THE ENCRPT GLOBAL IN THE TASK BUILD CMD
;  FILE MUST BE COMMENTED OUT.
;
;  AUTHOR M. CRAREN
;
   .PSECT ENCRPT
   .GLOBL ENCRPT
;
ENCRPT:    JSR     PC, $SAVAL                       ;STORE ALL REGS
;
;CONVERT FROM ASCII TO RAD50. THIS WILL TAKE A THREE WORD PASSWORD AND
;CONVERT IT TO A TWO WORK NUMBER.
;
   MOV     R0, PASWRD                   ;STORE ADDR OF PASWRD.
   MOV     #NUMBER, R3                  ;PUT ADDR OF NUMBER INTO R3.
   MOV     #1, R1                       ;CONVERT ALL CHAR TO RAD50.
   JSR     PC, $CAT5B                   ;DO THE CONVERSION.
   MOV     R1, (R3)+                    ;PUT RESULT INTO NUMBER.
   MOV     #1, R1                       ;SET IT UP AGAIN.
   JSR     PC, $CAT5B                   ;CONVERT NEXT 3 CHARS.
   MOV     R1, (R3)                     ;STORE THE RESULT.
;
;CONVERT THIS TO A RANDOM NUMBER.
;THIS IS A FORTRAN CALL.
;THE ACTUAL OUTPUT IS IN R0 AND R1 AS A FLOATING POINT NUMBER.
;THIS IS IRREVELANT BECAUSE THE OUTPUT IS TREATED AS A BIT PATTERN.
;
   MOV     #DATA, R5                    ;SET UP ADDRESS FOR RAN.
   JSR     PC, RAN                      ;CONVERT TO RANDOM NUMBER.
                                        ;RESULT IS IN R0 AND R1.
;PERMUTE THE WORD.
;
   MOV     #NUMBER, R5                  ;USE NUMBER TO STORE RESULT.
   JSR     PC, PERMUT                   ;PERMUTE BITS IN FIRST WORD.
   MOV     R3, (R5)+                    ;STORE THE RESULT.
   MOV     R1, R0                       ;GET NEXT WORD OF RESULT.
   JSR     PC, PERMUT                   ;PERMUTE BITS IN THIS WORD.
   MOV     R3, (R5)                     ;STORE THIS RESULT.
   MOV     R3, R4                       ;USE R4 TO FORM SUM.
;
;FILL IN REMAINING WORD OF ENCODED PASSWORD.
;
   ADD     -(R5), R4                    ;ADD TWO PERMUTED WORDS.
   TST     (R5)+                        ;SET UP ADDR FOR THIRD WORD.
   TST     (R5)+
   MOV     R4, (R5)                     ;PUT THE SUM INTO ADDR.
```

```
           MOV     #NUMBER, R1                     ;PUT ADDR OF NUMBER INTO R1.
        ;
        ;XOR KEY VALUE.
        ;
           MOV     KEY, R0                         ;GET KEY VALUE.
           XOR     R0, (R1)+                       ;EXCLUSIVE-OR IT WITH THE
           XOR     R0, (R1)+                       ;THREE WORDS OF THE
           XOR     R0, (R1)                        ;ENCODED PASWRD.
        ;
        ;GET LOCATION OF PASSWORD AND FILL IT IN WITH THE ENCRYPTED PASSWORD.
        ;
           MOV     PASWRD, R0
           MOV     #NUMBER, R1
           MOV     (R1)+, (R0)+
           MOV     (R1)+, (R0)+
           MOV     (R1), (R0)
        ;
        ;CLEARING THE CARRY BIT SIGNALS ENCRPTION WORKED.
        ;
           CLC
           RTS     PC
        ;
        ;SUBROUTINE TO PERMUTE THE PASSWORD.
        ;VALUE TO PERMUTE IS IN R0.
        ;RESULT IS IN R3.
        ;NO REGISTERS ARE PRESERVED.
        ;
        PERMUT: CLR     R3                     ;R3 IS USED FOR OUTPUT.
                MOV     #30., R4               ;16 WORDS WORTH OF PERMUTES.
        MOVE:   ASR     R0                     ;PUT LSB INTO CARRY.
                BCS     ONE                    ;IF A ONE, GO PERMUTE IT.
                BIC     PERM(R4),R3            ;CLEAR PERMUTED POSITION IF 0.
        ;
        ; The above line is not actually needed. All bits were cleared by CLR R3.
        ;
                BR      END                    ;GO GET NEXT BIT TO PERMUTE.
        ONE:    BIS     PERM(R4),R3            ;SET PERMUTED POSITION.
        END:    DEC     R4                     ;DECREMENT FOR NEXT PATTERN
                DEC     R4                     ;IN PERMUTATION TABLE.
                BPL     MOVE                   ;GO CHECK NEXT BIT.
                RTS     PC                     ;RETURN WHEN DONE WITH WORD.
        ;
        ;PERMUTATION TABLE FOR EACH WORD.
        ;
                .PSECT PERM,RO,D,LCL
        ;
        PERM:   .WORD   1000,200,2,20,4
                .WORD   20000,1,40,100,10,10000,400,4000
                .WORD   2000,100000,40000
        ;
        ;VARIABLES FOR DOING CONVERSIONS.
        ;
        .PSECT LOCAL, RW, D, LCL
;
NUMBER: .BLKW   3                       ;3-WORD BUFFER.
DATA:   .WORD   1
        .WORD   NUMBER                  ;ADDRESS OF BUFFER.
PASWRD: .BLKW   1
KEY:    .WORD   54317
        .END
```

In operation, the following occurs:

1. ACNT encrypts the entered password using the ENCRPT subroutine and stores the encrypted version in [0,0]RSX11.SYS.

2. HELLO encrypts the entered password using the ENCRPT subroutine and compares the results to the contents of [0,0]RSX11.SYS.

3. When doing a DECnet file transfer, NFT uses the network verification program NVP which performs the same operation as HELLO. The password is encrypted and compared to the contents of [0,0]RSX11.SYS.

With a little patience and a lot of luck, this can be implemented without too much difficulty. It can be performed during SYSGEN phase II when asked if you want to PAUSE to EDIT files. For a running system, do SYSGEN phase III to rebuild priviledged tasks and resave the system with the BOOT block. Also, don't forget to rebuild DECnet if it is in use.

HAPPY ENCRYPTING!!!!

Recovering Files From a Damaged Files-11 Disk

Judah Levine
Joint Insitute for Laboratory Astrophysics
National Bureau of Standards and University of Colorado
Boulder, Colorado 80309

We operate a pair of RM03 disks using a single controller on a PDP 11/70. When the controller failed recently, it over-wrote the home block and many of the low-numbered logical blocks on both volumes. When the controller was repaired, we were left with two volumes which could not be mounted, and which were therefore unusable as file-structured devices.

As is so often the case in these situations, many of our most important programs had not been backed up to tape. We decided to attempt to recover as many of the program text files as possible. Since most of our programs are written in FORTRAN, a typical file name to recover might be ABC.FTN. We first attempted to search for the directory entry for this file. If the directory entry could be found, we tried to find the header block. If we could find the header block, we used the retrieval pointers stored there to recover the full file text and to copy the text to another volume. This process, which we describe in more detail below, was able to recover about 90% of the FORTRAN text files and averted what might have been a very serious setback to our work. It worked because most of the disk blocks were intact, although many of the system pointers had been over-written. Since we were writing an emergency recovery task and not a full file-management system, we decided to recover only FORTRAN or MACRO text files that were not so large or so fragmented so as to require more than one header block. This turned out not to be a significant limitation for us. We also chose not to recover task images since these could be easily recovered from the FORTRAN text. Recovery of other than text files does not present any unique obstacles, but some care must be taken to be sure that the structure of the file described in the old header block is faithfully copied to the new header block. Error detection is also more difficult in this case.

We now discuss our method in some detail. The root of all of the programs is the ability to read any physical block of an unmounted disk. The code we used for doing this is shown in program FNDFIL,

----

* The material presented here is based on our experience in recovering data following a hardware failure. It is necessarily specific to the Files-11 disk format and thus product names are used. This does not imply an endorsement by the National Bureau of Standards.

lines 25 - 38 and 61 - 73.

The Directory Entry

The directory entry for every file is 8 words long. The first and second words are the file number and file sequence number, respectively. In normal operation, these point to and validate the file header block. The fourth, fifth and sixth words are the filename in radix-50, the seventh word is the file extension in radix-50 and the eight word is the version number in binary. There are 32 eight-word entries in a physical disk block. They occupy words 1-8, 9-16, ..., 249-256. To search for the entry for file ABC.FTN, we convert the name to radix-50 and compare the resulting string to words 4-7, 12-15, ..., 252-255 of every block on the disk. If we find a match, the last word of the 8-word group gives the version number. If the disk were not corrupted, the first two words of the 8-word group would allow us to read the file header block and thereby the retrieval pointers. But these pointers cannot be used directly since we do not know where the index file is located.

The Header Block

The header block for every file is 256 words long (we assume that extension headers are not necessary) and corresponds exactly to a physical block on the volume. The name of the file described by the header is stored in words 24-27, again in radix-50. To find the header block for a given file, we therefore perform a second comparison between the name of the file we are looking for (in radix-50) and these four words of every block on the disk. Some care must be used in this search, since the name of a file in the directory and the name stored in the header block may not be the same. If file ABC.FTN;N was prepared using EDT, for example, its header block may be ABC.TMP;1 while its directory entry will be ABC.FTN;N. We deal with this possibility in practice by entering a file type consisting of blanks. As can be seen from the program, this results in a search for file ABC.*;* both in the directory search and in the header search. The match between a given header and a given directory entry can be confirmed by examining the file number and file sequence number of both entries.

Once the header block has been found, we extract the owner's UIC, which should correspond to the known owner of the file, the length of the file in blocks, the last significant byte of the last block and the retrieval pointers. These pointers are 4-byte quantities beginning in word 51. Each pointer consists of three bytes giving the starting physical block and one byte giving the number of consecutive blocks in the segment. These parameters can then be used to recover the file.

Text Recovery

Program FNDFIL is guaranteed to find both the directory entry and the header block of any file, if these structures are still intact. It may also find extraneous blocks including entries for deleted files and blocks containing binary data which accidentally match the search string. The recovery program is therefore a separate program. It accepts a series of retrieval pointers and lengths and reads these blocks consecutively to recover the file. It writes the recovered text to any convenient file on another volume. Since the files we seek to recover were prepared using EDT or EDI, they use implied carriage control. Each line of text begins with a 16-bit word-aligned quantity giving the length of the line in characters. If the length is odd, a dummy character is added to the end of the line so that the next character count will also be word aligned. Lines may span two physical blocks on the disk without warning -- indeed the physical block boundary may occur even between the character count and the text. Program RECOVR deals with any of these possibilities. If a file does not completely fill the last block, the bytes after the end of the last line are simply ignored.

Program RECOVR will fail if one of the physical blocks in the retrieval pointer chain has been over-written as a result of the original hardware failure or by subsequent valid writes after the failure has occurred and before the disk can be stopped. Since the start of a block almost never coincides with the start of a line, the program begins reading most blocks in mid-line, and an error is often not immediately apparent. The error is usually detected when the program interprets the next word following what it thinks of as the end of the current line as the byte count for the next line. If this word contains two ASCII characters (as is likely to happen if the block is over-written by a fragment of another text file), the resultant line length will be extremely large. Even if the block is a binary fragment, the probability that the word yields a line length that is reasonable is vanishingly small. Although the probability of detecting an over-written block is therefore quite high, there is no guaranteed way to recover the text of subsequent blocks. We have sometimes been able to recover subsequent pieces of the file by skipping the block in question, starting with the next block in the retrieval-pointer chain and looking for a byte of zero. Since all lines are less than 256 bytes long and since zero is not an ASCII code, a zero byte is an almost certain indication that the upper byte of a count word has been detected and that the start of a new line has been found (unless the block has been over-written by a binary block in which case all is probably lost). This permits the program to be re-synchronized and to continue. The subsequent text may or may not be meaningful, and must be carefully examined. Program RECOVR does not deal with these situations and simply stops if an unreasonably large line length is detected. The techniques discussed here were applied by hand, but could easily be added to RECOVR if desired.

Conclusions

Using these methods, we recovered about 90% of the text files immediately with no errors. Another 5% of the files were recovered with some gaps. In the remaining 5% of the cases, the recovered text was too fragmentary to be useful.

This hardware failure points out a weakness in our standard disk to disk backup procedure -- especially when both disks are operated by the same controller. A single hardware malfunction can destroy both copies of a file system with almost no warning.

The recovery procedure also shows a weakness in the Files-11 protection system. The programs outlined here will work on any unmounted volume and require no special privileges. Any user can use these techniques to copy any file from such a disk, totally bypassing the file protection safeguards of the system. Deleted versions of a file may also be recovered this way. The only way to prevent this is to spin-down (or otherwise take off-line) disks that are not mounted as Files-11 volumes.

```
        PROGRAM FNDFIL
C
C     THIS PROGRAM READS EVERY BLOCK ON A DISK LOOKING
C     FOR EITHER THE DIRECTORY ENTRY OR THE FILE
C     HEADER BLOCK FOR A GIVEN FILE.  WHEN EITHER IS FOUND,
C     THE APPROPRIATE INFORMATION IS PRINTED OUT.
C     THE PROGRAM GETS ITS INPUT FROM A FILE NAMED FIND.LST AND
C     WRITES WHATEVER IT FINDS OUT TO FILE BLOCKS.LST.
C
C     TO FIND A GIVEN FILE, ABC.FTN FOR EXAMPLE, FILE FIND.LST
C     MUST HAVE A 2-LINE ENTRY WITH THE FILENAME ON THE FIRST
C     LINE AND THE EXTENSION ON THE SECOND LINE:
C
C     ABC
C     FTN
C
C     THE PROGRAM WILL FIND ALL VERSIONS OF THE FILE
C
C     IF THE SECOND LINE IS A SERIES OF BLANKS, THE SEARCH IS EQUIVALENT
C     TO LOOKING FOR FILES ABC.*;*
C
      INTEGER*2 IBUF(256)                 !BUFFER TO STORE DISK BLOCK
      BYTE IUIC(2)
      EQUIVALENCE (IUIC(1),IBUF(5))       !UIC IN HEADER BEGINS IN WORD 5
      BYTE IRETRV(410)
      EQUIVALENCE(IRETRV(1),IBUF(51))     !RETRIEVAL PNTRS BEGIN AT 51
      BYTE RNAME(12)
      BYTE NAME(9),EXT(3)
      INTEGER*2 INAME(4)
      INTEGER*4  IBLRTV                   !USED TO STORE FIRST BLOCK OF THE
      BYTE BBLRTV(4)                      !RETRIEVAL POINTER.
      EQUIVALENCE (IBLRTV,BBLRTV(1))
```

```
C
      INTEGER*2 ISB(2),IPRL(6)           !USED IN QIO-CALL
C
      INTEGER*4 ILOW,IHIGH
      INTEGER*2 IBL(2)
      EQUIVALENCE (ILOW,IBL(1))          !USED TO CONVERT I*4 FOR QIO CALL
C
      DATA IOATT /'1400'O/               !ATTACH DEVICE
      DATA IORLB /'1000'O/               !READ LOGICAL BLOCK
      DATA IODET /'2000'O/               !DETACH DEVICE
C
C     THE FOLLOWING ASSIGN STATEMENT SETS LOGICAL UNIT 1 TO THE DISK TO
C     BE SEARCHED.  THE STRING 'DR0:' SHOULD CHANGED IF NECESSARY.  THE
C     THIRD PARAMETER IS THE NUMBER OF CHARACTERS IN THE DEVICE NAME.
C
C     NOTE THAT SINCE THE DISK HAS A CORRUPTED FILE STRUCTURE AND
C     THEREFORE CANNOT BE MOUNTED, THE ASSIGN WILL PRODUCE FORTRAN ERROR
C     43.  THIS ERROR IS NOT FATAL AND MAY BE IGNORED.
C
      CALL ASSIGN (1,'DR0:',4)
C
C     OPEN COMMAND FILE AND LISTING FILE
C
      OPEN(UNIT=3,NAME='FIND.LST',TYPE='OLD',
     + ACCESS='SEQUENTIAL',CARRIAGECONTROL='LIST',
     + READONLY,DISPOSE='SAVE',FORM='FORMATTED')
      OPEN(UNIT=4,NAME='BLOCKS.LST',TYPE='NEW',
     + ACCESS='SEQUENTIAL',CARRIAGECONTROL='LIST',
     + FORM='FORMATTED',DISPOSE='SAVE')
C
C     SET SEARCH LIMITS.  THESE VALUES MAY HAVE TO BE CHANGED DEPENDING
C     ON WHAT TYPE OF DISK IS BEING SEARCHED.  ILOW IS ALWAYS ZERO TO
C     START. IHIGH SHOULD BE SET TO THE SIZE OF THE DISK (AS DETERMINED
C     FROM PIP /FR, FOR EXAMPLE).  NOTE THAT THE FIRST BLOCK ON THE DISK
C     IS NUMBER 0.
C
      ILOW=0
      IHIGH=131679
C
C     ATTACH DEVICE -- STOP ON EXECUTIVE REJECT OR ON ATTACH FAILURE.
C
      CALL WTQIO(IOATT,1,1,,ISB,IPRL,IDS)
      IF(IDS .NE. 1) THEN
      WRITE(4,1)IDS
    1 FORMAT(' ATTACH REJECTED, IDS='O6)
      STOP
      ENDIF
      IF(ISB(1) .NE. 1) THEN
      WRITE(4,2)ISB
    2 FORMAT(' ATTACH,FAILED, ISB='2O6)
      STOP
      ENDIF
      CALL GETADR(IPRL(1),IBUF(1))              !GET ADDRESS OF INPUT BUFFEF
      IPRL(2)=512                             !ALWAYS READ 512 BYTES.
      IPRL(3)=0
  888 READ(3,23,END=999) IL,(NAME(I),I=1,IL)  !READ FILE NAME, END ON EOF
   23 FORMAT(Q,9A1)
C
C     PAD NAME WITH BLANKS IF NECESSARY
C
      IF(IL .LT. 9) THEN
      DO 24 I=IL+1,9
      NAME(I)=' '
   24 CONTINUE
      ENDIF
C
C     READ FILE EXTENSION AND PAD WITH BLANKS IF NECESSARY
C
      READ(3,23,END=999) IL,(EXT(I),I=1,IL)
      IF(IL .LT. 3) THEN
      DO 26 I=IL+1,3
      EXT(I)=' '
   26 CONTINUE
      ENDIF
      WRITE(4,998) NAME,EXT
  998 FORMAT(' BEGIN SEARCH FOR FILE='12A1)
C
C     CONVERT NAME AND EXTENSION TO RADIX-50.  IF EXTENSION IS BLANK,
C     SEARCH IS ON FILENAME ONLY (9 CHARACTERS = 3 WORDS).  IF EXTENSION
C     IS NOT BLANK, SEARCH IF ON FULL NAME (12 CHARACTERS = 4 WORDS).
C
      CALL IRAD50(9,NAME,INAME)
      CALL IRAD50(3,EXT,INAME(4))
      IF(INAME(4) .EQ. 0) THEN
      LIMZ=3
      ELSE
      LIMZ=4
      ENDIF
C
C     CONVERT BLOCK NUMBER TO FORMAT REQUIRED BY QIO CALL
C
    9 IPRL(4)=IBL(2)
      IPRL(5)=IBL(1)
C
C     READ PHYSICAL BLOCK OF DISK.  STOP ON EXECUTIVE REJECT OR
C     READ FAILED.
C
      CALL WTQIO(IORLB,1,1,,ISB,IPRL,IDS)
      IF(IDS .NE. 1) THEN
      WRITE(4,3)IDS,ILOW
    3 FORMAT(' READ REJECTED, IDS='O6,' AT BLOCK='I10)
      STOP
      ENDIF
      IF(ISB(1) .NE. 1) THEN
      WRITE(4,4)ISB,ILOW
    4 FORMAT(' READ FAILED, ISB='2O6,' AT BLOCK='I10)
```

```
      STOP
      ENDIF
C
C     TEST TO SEE IF THIS IS A DIRECTORY BLOCK. SEE TEXT
C
      DO 5 I=1,256,8
      DO 35 JJ=1,LIMZ                            !SEARCH FOR 3 OR 4 WORDS
      IF(IBUF(I+JJ+2) .NE. INAME(JJ)) GO TO 5    !EXIT LOOP IF NO MATCH
   35 CONTINUE
      WRITE(4,6)ILOW                             !DIRECTORY ENTRY FOUND
    6 FORMAT(' DIRECTORY ENTRY IN BLOCK='I10)
      CALL R50ASC(12,IBUF(I+3),RNAME)            !CONVERT NAME TO ASCII
C
C     PRINT FILE NUMBER, SEQUENCE NUMBER, NAME AND VERSION
C
      WRITE(4,36)IBUF(I),IBUF(I+1),IBUF(I+2),RNAME,IBUF(I+7)
   36 FORMAT(1X,O6,1X,O6,1X,O6,1X,9A1,1X,3A1,1X,O6)
    5 CONTINUE
C
C     SEE IF THIS MIGHT BE A HEADER BLOCK.  SEARCH FOR NAME AGAIN
C
      DO 10 JJ=1,LIMZ
      IF(IBUF(23+JJ) .NE. INAME(JJ))GO TO 7
   10 CONTINUE
C
C     IF MATCH, PRINT OUT NAME FOUND, OWNER, POINTERS, ETC.
C
      CALL R50ASC(12,IBUF(24),RNAME)                !CONVERT BACK TO ASCII
      WRITE(4,8)ILOW,RNAME,IBUF(28)                 !PRINT NAME AND VERSION
    8 FORMAT('  HEADER BLOCK FOUND, BLOCK='I10,
     + ' NAME=',9A1,1X,3A1';'O5)
      WRITE(4,100) IBUF(2),IBUF(3)
  100 FORMAT(' NUM,SEQ='O6,1X,O6)
      WRITE(4,101)IUIC(2),IUIC(1)
  101 FORMAT(' OWNER=['O3,',',O3,']')
      WRITE(4,102)IBUF(13),IBUF(14)                !LAST BLOCK AND LAST BYTE
  102 FORMAT(' EOF IN BLOCK'I3' AT BYTE'I4)
C
C     PRINT NUMBER OF RETRIEVAL POINTERS.  NOTE THAT THIS NUMBER IS
C     OFTEN TOO LARGE.  RETRIEVAL POINTERS WHICH MAP BLOCKS AFTER THE
C     NUMBER OF BLOCKS PRINTED ABOVE SHOULD BE IGNORED.
C
      WRITE(4,103)IRETRV(1)
  103 FORMAT(' NUMBER OF POINTERS='I3)
      JORG=0
      DO  11 I=1,IRETRV(1)
C
C     CONVERT 3-BYTE RETRIEVAL POINTER TO FORTRAN I*4 VARIABLE.
C     NOTE THAT MOST SIGNIFICANT BYTE OF RETRIEVAL POINTER WILL ALWAYS
C     BE ZERO.
C
      BBLRTV(1)=IRETRV(5+JORG)
      BBLRTV(2)=IRETRV(6+JORG)
      BBLRTV(3)=IRETRV(3+JORG)
      BBLRTV(4)=0
      WRITE(4,104)IRETRV(4+JORG)+1,IBLRTV
  104 FORMAT(' LENGTH AND FIRST BLOCK='I3,1X,I10)
      JORG=JORG + 4                                 !ADVANCE 4 BYTES FOR NEXT
   11 CONTINUE
    7 ILOW=ILOW +1                                  !MOVE ON TO NEXT BLOCK
      IF(ILOW .LE. IHIGH) GO TO 9                   !GO BACK AND READ
C
C     RESET POINTERS AND LIMITS AND GO TO READ NEXT FILE NAME FOR SEARCH
C
      ILOW=0
      IHIGH=131679
      GO TO 888
C
C     COME HERE WHEN EOF READ FROM SEARCH FILE.  CLOSE UP AND EXIT.
C
  999 CALL WTQIO(IODET,1,1)
      CLOSE(UNIT=3,DISPOSE='SAVE')
      CLOSE(UNIT=4,DISPOSE='SAVE')
      STOP
      END



      PROGRAM RECOVR
C
C      THIS PROGRAM RECOVERS A TEXT FILE USING THE
C      POINTERS FOUND BY A PREVIOUS RUN OF FNDFIL
C     THE FILE MUST BE IN STANDARD TEXT FORMAT WITH
C     IMPLICIT CARRIAGE CONTROL.
C
C     THE SYSTEM CALLS IN HERE TO ATTACH A DISK AND READ A PHYSICAL BLOCK
C     ARE THE SAME AS THOSE USED IN PROGRAM FNDFIL.  AS IN FNDFIL, THE
C     DISK WE ARE SEARCHING IS NAMED DR0 IN THIS PROGRAM AND SHOULD BE
C     CHANGED AS NECESSARY.
C
      INTEGER*2 IBUF(256)
      BYTE JCAR(512)
      EQUIVALENCE (JCAR(1),IBUF(1))
      BYTE BLEN(2)
      EQUIVALENCE(BLEN(1),JLEN)
      BYTE ILIN(150),NAME(28)
C
C     ARBRITRARILY LIMIT TO 75 RETRIEVAL POINTERS.
C
      INTEGER*4  IBLRTV(75)                    !FIRST BLOCK OF FILE SEGMENT
      INTEGER*2 IBLLEN(75)                     !NUMBER OF BLOCKS IN SEGMENT
C
      INTEGER*2 ISB(2),IPRL(6)                 !PARAMETERS FOR QIO CALL
C
      INTEGER*4 ILOW
      INTEGER*2 IBL(2)                         !ARRAY IBL IS USED TO
      EQUIVALENCE (ILOW,IBL(1))           !CONVERT I*4 BLOCK NUMBER TO QIO
C
C
      DATA IOATT /'1400'O/                !ATTACH DEVICE
      DATA IORLB /'1000'O/                !READ LOGICAL BLOCK
      DATA IODET /'2000'O/                !DETACH DEVICE
C
      CALL ASSIGN (1,'DR0:',4)            !SEE COMMENTS IN PROGRAM FNDFIL
C
      ILOW=0
C
C     ATTACH DEVICE, STOP ON EXECUTIVE REJECT OR ATTACH FAILURE
C
      CALL WTQIO(IOATT,1,1,,ISB,IPRL,IDS)
      IF(IDS .NE. 1) THEN
      TYPE 1,IDS
    1 FORMAT(' ATTACH REJECTED, IDS='06)
      STOP
      ENDIF
      IF(ISB(1) .NE. 1) THEN
      TYPE 2,ISB
    2 FORMAT(' ATTACH,FAILED, ISB='2O6)
      STOP
      ENDIF
      CALL GETADR(IPRL(1),IBUF(1))        !ADDRESS OF BUFFER TO STORE BLOCK
      IPRL(2)=512                         !ALWAYS READ 512 BYTES (=1 BLOCK)
      IPRL(3)=0
      TYPE 22                            !FILE NAME TO STORE RECOVERED TEXT
   22 FORMAT('$ENTER OUTPUT FILENAME=')
      READ(5,43) IL,(NAME(I),I=1,IL)
   43 FORMAT(Q,28A1)
      NAME(IL+1)=0                        !PAD WITH ZERO BYTE
C
C     BEGIN REQUESTING RETRIEVAL POINTERS AND LENGTHS AS OUTPUT BY
C     PROGRAM FNDFIL.  COMPUTE TOTAL LENGTH OF FILE AS SUM OF
C     RETRIEVAL POINTER LENGTHS.   TERMINATE INQUIRY ON A ZERO LENGTH
C
      ITOT=0
      DO 23 I=1,75
      TYPE 24
   24 FORMAT('$ENTER STARTING BLOCK AND LENGTH=')
      READ(5,*) IBLRTV(I),IBLLEN(I)
      IF(IBLLEN(I) .EQ. 0) GO TO 25
      ITOT=ITOT + IBLLEN(I)
   23 CONTINUE
C
C     GET END OF FILE POINTER (LAST SIGNIFICANT BYTE OF LAST BLOCK) AS
C     WRITTEN BY FNDFIL.
C
   25 TYPE 26
   26 FORMAT('$ENTER LAST BYTE=')
      READ(5,*)IEOF
      INRTV=I-1
      ICRTV=1
      IF(ITOT .EQ. 1)  THEN            !IF FILE 1 BLOCK LONG, SET LAST BYTE
      JEOF=IEOF                        !OTHERWISE, ALL 512 BYTES ARE
      ELSE                             !SIGNIFICANT
      JEOF=512
      ENDIF
C
C     OPEN NEW FILE FOR RECOVERED TEXT
C
      OPEN(UNIT=2,NAME=NAME,TYPE='NEW',FORM='FORMATTED',
     + ACCESS='SEQUENTIAL',CARRIAGECONTROL='LIST',DISPOSE='SAVE')
C
C     CONVERT FIRST BLOCK OF THIS RETRIEVAL POINTER TO QIO FORMAT
C
      ILOW=IBLRTV(1)
    9 IPRL(4)=IBL(2)
      IPRL(5)=IBL(1)
C
C     READ THE BLOCK.  STOP ON EXECUTIVE REJECT OR READ ERROR.
C
      CALL WTQIO(IORLB,1,1,,ISB,IPRL,IDS)
      IF(IDS .NE. 1) THEN
      TYPE 3,IDS,ILOW
    3 FORMAT(' READ REJECTED, IDS='O6,' AT BLOCK='I10)
      STOP
      ENDIF
      IF(ISB(1) .NE. 1) THEN
      TYPE 4,ISB,ILOW
    4 FORMAT(' READ FAILED, ISB='2O6,' AT BLOCK='I10)
      STOP
      ENDIF
      IPOS=1                       !IPOS IS POINTER TO CURRENT INPUT BYTE
C
C     CONVERT NEXT TWO BYTES TO LINE LENGTH.  SEE TEXT.
C
   28 BLEN(1)=JCAR(IPOS)
      IPOS=IPOS + 1
      BLEN(2)=JCAR(IPOS)
      IPOS=IPOS + 1
      ICAR=0                           !POSITION IN OUTPUT BUFFER
C
C     IF LINE IS TOO LONG, PROBABLY AN ERROR. PRINT CHARACTERS OF INPUT
C     BUFFER USED FOR LINE LENGTH COMPUTATION AND EXIT.
C     SEE TEXT.
C
      IF(JLEN .GT. 150) THEN
      TYPE 123,ILOW,IPOS,JLEN
  123 FORMAT(' BLOCK='I10', BYTE='I5', LENGTH='I5)
      TYPE 124,(JCAR(M),M=IPOS-1,IPOS+1)
  124 FORMAT(3O5)
      GO TO 999
      ENDIF
C
```

```
C     IF WE ARE AT THE END OF A BLOCK, READ THE NEXT BLOCK IN THIS
C     RETRIEVAL POINTER SET OR FIRST BLOCK OF THE NEXT RETREIVAL POINTER
C     SET.
C
      IF(IPOS .GT. JEOF)  THEN
      IBLLEN(ICRTV)=IBLLEN(ICRTV) -1          !DECREMENT BLOCKS IN THIS SET
      ITOT=ITOT - 1                           !AND NUMBER OF BLOCKS LEFT
      IF(IBLLEN(ICRTV) .NE. 0)  THEN          !IF MORE BLOCKS IN THIS SET,
      ILOW=ILOW +1                            !INCREMENT BLOCK COUNTER
      ELSE
      ICRTV=ICRTV + 1                         !ELSE MOVE TO NEXT RETRIEVAL
      IF(ICRTV .GT. INRTV)  GO TO 999         !SET UNLESS DONE
      ILOW=IBLRTV(ICRTV)
      ENDIF
C
C     COMPUTE LAST SIGNIFICANT BYTE IN THIS BLOCK, CONVERT TO QIO
C     FORMAT AND READ THE BLOCK
C
      IF(ITOT .EQ. 1)   JEOF=IEOF
      IPOS=1
      IPRL(4)=IBL(2)
      IPRL(5)=IBL(1)
      CALL WTQIO(IORLB,1,1,,ISB,IPRL,IDS)
      ENDIF
C
C     TRANSFER CONSECUTIVE BYTES FROM INPUT BUFFER TO OUTPUT LINE
C     UNTIL LINE LENGTH IS SATISFIED.  IF END OF PHYSICAL BLOCK
C     IS DETECTED IN THE MIDDLE OF THIS, COMPUTE NEXT BLOCK AS ABOVE
C     READ THE NEXT BLOCK AND CONTINUE ASSEMBLING THE LINE.
C
      DO 27 K=1,JLEN
      ICAR=ICAR + 1
      ILIN(ICAR)=JCAR(IPOS)
      IPOS=IPOS+1
      IF(IPOS .LE. JEOF)  GO TO 27
      IBLLEN(ICRTV)=IBLLEN(ICRTV) -1
      ITOT=ITOT - 1
      IF(IBLLEN(ICRTV) .NE. 0)  THEN
      ILOW=ILOW +1
      ELSE
      ICRTV=ICRTV + 1
      IF(ICRTV .GT. INRTV)  GO TO 999
      ILOW=IBLRTV(ICRTV)
      ENDIF
      IF(ITOT .EQ. 1)   JEOF=IEOF
      IPOS=1
      IPRL(4)=IBL(2)
      IPRL(5)=IBL(1)
      CALL WTQIO(IORLB,1,1,,ISB,IPRL,IDS)
   27 CONTINUE
C
C     WRITE THIS LINE TO OUTPUT FILE
C
```

```
      WRITE(2,48)(ILIN(L),L=1,JLEN)
   48 FORMAT(150A1)
C
C     IF LINE LENGTH IS ODD, SKIP 1 BYTE TO MAKE SURE LENGTH IS
C     ALIGNED ON A WORD BOUNDARY.
C
      IF( (IPOS .AND. 1) .EQ. 0)  IPOS=IPOS + 1
C
C     IF WE ARE NOW AT THE END OF A BLOCK, COMPUTE THE NEXT BLOCK AND READ
C     IT IN.
C
      IF(IPOS .LT. JEOF)  GO TO 28
      IBLLEN(ICRTV)=IBLLEN(ICRTV)-1
      ITOT=ITOT-1
      IF(IBLLEN(ICRTV) .NE. 0) THEN
      ILOW=ILOW +1
      ELSE
      ICRTV=ICRTV + 1
      IF(ICRTV .GT. INRTV) GO TO 999
      ILOW=IBLRTV(ICRTV)
      ENDIF
      IF(ITOT .EQ. 1)  JEOF=IEOF
      IPOS=1
      IPRL(4)=IBL(2)
      IPRL(5)=IBL(1)
      CALL WTQIO(IORLB,1,1,,ISB,IPRL,IDS)
      GO TO 28
C
C     COME HERE WHEN ALL BLOCKS HAVE BEEN READ
C
  999 CLOSE(UNIT=2,DISPOSE='SAVE')
      CALL WTQIO(IODET,1,1)
      STOP
      END
```

Printed in the U.S.A.

DECUS

DECUS SUBSCRIPTION SERVICE
DIGITAL EQUIPMENT COMPUTER SOCIETY
219 BOSTON POST ROAD, (BP02)
MARLBORO, MA 01752

STATUS CHANGE

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Please Delete My Membership Record
(I Do Not Wish To Remain A Member)

DECUS Membership No: ______________

Name: ______________

Company: ______________

Address: ______________

____

State/Country: ______________

Zip/Postal Code: ______________

Mail to: DECUS - Attn: Subscription Service
219 Boston Post Road, BP02
Marlboro, Massachusetts 01752 USA

Bulk Rate
U.S. Postage
PAID
Permit No. 18
Leominster, MA
01453