RSX
MULTI-TASKER

JULY 1985
RSX MULTITASKER

Recovering Files From a Damaged Files-11 Disk

Judah Levine
Joint Institute 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 overwrote 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 FNDNIL.

* 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.
RSX MULTITASKER

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.PTN, 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.PTN;N was prepared using EDT, for example, its header block may be ABC.TMP;I while its directory entry will be ABC.PTN;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
C INTEGER*2 IBUF(256) !BUFFER TO STORE DISK BLOCK
BYTE IUIC(2) !UIC IN HEADER BEGINS IN WORD 5
EQUIVALENCE (IUIC(1),IBUF(5)) !RETRIEVAL PNTRS BEGIN AT 51
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
BYTE BBLRTV(4)
EQUIVALENCE (IBLRTV,BBLRTV(1)) !USED TO STORE FIRST BLOCK OF THE

22
INTEGER*2 ISB(2), IPRL(6)  ! USED IN QIO-CALL

INTEGER*4 ILow, IHIGH
INTEGER*2 IBL(2)
EQUHANCE (ILow, IBL(1))  ! USED TO CONVERT I*4 FOR QIO CALL

DATA IOATT '/1400'O/
DATA IORLB '/1000'O/
DATA IODET '/2000'O/

! ATTACH DEVICE
! READ LOGICAL BLOCK
! DETACH DEVICE

THE FOLLOWING ASSIGN STATEMENT SETS LOGICAL UNIT 1 TO THE DISK TO BE SEARCHED. THE STRING 'DRO:' SHOULD CHANGE IF NECESSARY. THE THIRD PARAMETER IS THE NUMBER OF CHARACTERS IN THE DEVICE NAME.

NOTE THAT SINCE THE DISK HAS A CORRUPTED FILE STRUCTURE AND THEREFORE CANNOT BE MOUNTED, THE ASSIGN WILL PRODUCE FORTRAN ERROR 43. THIS ERROR IS NOT FATAL AND MAY BE IGNORED.

CALL ASSIGN (1,'DRO:',4)

OPEN COMMAND FILE AND LISTING FILE

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')

SET SEARCH LIMITS. THESE VALUES MAY HAVE TO BE CHANGED DEPENDING ON WHAT TYPE OF DISK IS BEING SEARCHED. ILow IS ALWAYS ZERO TO START. IHIGH SHOULD BE SET TO THE SIZE OF THE DISK (AS DETERMINED FROM PIP /FR, FOR EXAMPLE). NOTE THAT THE FIRST BLOCK ON THE DISK IS NUMBER 0.

ILow=0
IHIGH=131679

ATTACH DEVICE -- STOP ON EXECUTIVE REJECT OR ON ATTACH FAILURE.

CALL WTQIO(IOATT,1,1,ISB,IPRL,IDS)
IF(IDS .NE. 1) THEN
WRITE(4,1)IDS
1 FORMAT( 'ATTACH REJECTED, IDS='06)
STOP
ENDIF
IF(ISB(1) .NE. 1) THEN
WRITE(4,2)ISB
2 FORMAT( 'ATTACH FAILED, ISB='206)
STOP
ENDIF
CALL GETADR(IPRL(1),IBUF(1))  ! GET ADDRESS OF INPUT BUFFER
RSX MULTITASKER

IPRL(2)=512
IPRL(3)=0
888 READ(3,23,END=999) IL,(NAME(I),I=1,IL) !READ FILE NAME, END ON EOP
23 FORMAT(Q,9A1)

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 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 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 CONVERT BLOCK NUMBER TO FORMAT REQUIRED BY QIO CALL
C 9 IPRL(4)=IBL(2)
IPRL(5)=IBL(1)
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='06,' AT BLOCK='110)
STOP
ENDIF
IF(ISB(1) .NE. 1) THEN
WRITE(4,4)ISB,ILOW
4 FORMAT(' READ FAILED, ISB='206,' AT BLOCK='110)

24
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
IF(IBUF(I+JJ+2) .NE. INAME(JJ)) GO TO 5
!SEARCH FOR 3 OR 4 WORDS
35 CONTINUE
WRITE(4,6)ILOW
6 FORMAT(' DIRECTORY ENTRY IN BLOCK='I10)
CALL R50ASC(12,IBUF(I+3),RNAME)
!DIRECTOIR ENTRY FOUND
!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(IX,06,1X,06,1X,06,1X,9A1,1X,3A1,1X,06)
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)
8 FORMAT(' HEADER BLOCK FOUND, BLOCK='I10,
+ ' NAME='9A1,1X,3A1';'05)
WRITE(4,100)IBUF(2),IBUF(3)
100 FORMAT(' NUM,SEQ='06,1X,06)
WRITE(4,101)IUIIC(2),IUIIC(1)
101 FORMAT(' OWNER=['03,,'03,'])
WRITE(4,102)IBUF(13),IBUF(14)
!LAST BLOCK AND LAST BYTE
102 FORMAT(' EOF IN BLOCK'13' AT BYTE'14)
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)
RSX MULTITASKER

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
   ILOW=ILOW+1
   MOVE ON TO NEXT BLOCK
   IF(ILOW.LT.IHIGH) GO TO 9
   GO BACK AND READ

C
C RESET POINTERS AND LIMITS AND GO TO READ NEXT FILE NAME FOR SEARCH
C
C
ILOW=0
IHIGH=131679
GO TO 888
C
C COME HERE WHEN EOF READ FROM SEARCH FILE. CLOSE UP AND EXIT.
C
C
999 CALL W70QIO(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 PNDFIL
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 PNDFIL. AS IN PNDFIL, THE
C DISK WE ARE SEARCHING IS NAMED DRO IN THIS PROGRAM AND SHOULDBE
C CHANGED AS NECESSARY.
C
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 ARBITRARILY LIMIT TO 75 RETRIEVAL POINTERS.
C
C INTEGER*4 IBLRTV(75)
INTEGER*2 IBLLEN(75)
   !FIRST BLOCK OF FILE SEGMENT
   !NUMBER OF BLOCKS IN SEGMENT
C
C INTEGER*2 ISB(2),IPRL(6)
   !PARAMETERS FOR QIO CALL
C
C INTEGER*4 ILOW
INTEGER*2 IBL(2)
   !ARRAY IBL IS USED TO
RSX MULTITASKER

EQUIVALENCE (ILOW, IBL(1))    !CONVERT I*4 BLOCK NUMBER TO QIO
DATA IOATT /'1400'0/           !ATTACH DEVICE
DATA IORLB /'1000'0/           !READ LOGICAL BLOCK
DATA IODET /'2000'0/           !DETACH DEVICE
CALL ASSIGN (1, 'DRO:', 4)    !SEE COMMENTS IN PROGRAM FNDNIL
ILOW = 0

ATTACH DEVICE, STOP ON EXECUTIVE REJECT OR ATTACH FAILURE
CALL WTTQIO(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='206)
  STOP
ENDIF
CALL GETADR(IPRL(1), IBUF(1))  !ADDRESS OF BUFFER TO STORE BLOCK
IPRL(2) = 512
IPRL(3) = 0
TYPE 22
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

BEGIN REQUESTING RETRIEVAL POINTERS AND LENGTHS AS OUTPUT BY
PROGRAM FNDNIL. COMPUTE TOTAL LENGTH OF FILE AS SUM OF
RETRIEVAL POINTER LENGTHS. TERMINATE INQUIRY ON A ZERO LENGTH

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

GET END OF FILE POINTER (LAST SIGNIFICANT BYTE OF LAST BLOCK) AS
WRITTEN BY FNDNIL.

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
  ! OTHERWISE, ALL 512 BYTES ARE
  ! SIGNIFICANT
JE0F=IEOF
ELSE
JE0F=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 C
CONVERT FIRST BLOCK OF THIS RETRIEVAL POINTER TO QIO FORMAT
C
IL0W=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(ITORB,1,1,ISB,IPRL,IDS)
IF(IDS .NE. 1) THEN
  TYPE 3,IDS,IL0W
3 FORMAT('READ REJECTED, IDS='06,' AT BLOCK='1I0)
  STOP
ENDIF
IF(ISB(1) .NE. 1) THEN
  TYPE 4,ISB,IL0W
4 FORMAT('READ FAILED, ISB='206,' AT BLOCK='1I0)
  STOP
ENDIF
IPOS=1

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
JCAR=0

! POSITION IN OUTPUT BUFFER
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,IL0W,IPOS,JLEN
123 FORMAT(' BLOCK='I10', BYTE='I5', LENGTH='I5')
  TYPE 124,(JCAR(M),M=IPOS-1,IPOS+1)
124 FORMAT(30S)
  GO TO 999
ENDIF
C
28
IF WE ARE AT THE END OF A BLOCK, READ THE NEXT BLOCK IN THIS
RETRIEVAL POINTER SET OR FIRST BLOCK OF THE NEXT RETREIVAL POINTER
SET.

IF(IPOS .GT. JEOF) THEN
  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)
  ENDF

COMPUTE LAST SIGNIFICANT BYTE IN THIS BLOCK, CONVERT TO QIO
FORMAT AND READ THE BLOCK

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

TRANSFER CONSECUTIVE BYTES FROM INPUT BUFFER TO OUTPUT LINE
UNTIL LINE LENGTH IS SATISFIED. IF END OF PHYSICAL BLOCK
IS DETECTED IN THE MIDDLE OF THIS, COMPUTE NEXT BLOCK AS ABOVE
READ THE NEXT BLOCK AND CONTINUE ASSEMBLING THE LINE.

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)
  ENDF

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

WRITE THIS LINE TO OUTPUT FILE
WRITE(2,48) (ILIN(I), I=1,JLEN)
48 FORMAT(150A1)

C IF LINE LENGTH IS ODD, SKIP 1 BYTE TO MAKE SURE LENGTH IS
C ALIGNED ON A WORD BOUNDARY.
C
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
C IF ( IPOS .LT. JEOF ) GO TO 28
C IBLLEN = IBLLEN + 1
C ITOT = ITOT - 1
C IF ( IBLLEN .NE. 0 ) THEN
C ILOW = ILOW + 1
C ELSE
C ICRTV = ICRTV + 1
C IF ( ICRTV .GT. INRTV ) GO TO 999
C ILOW = IBLRTV
C ENDIF
C IF ( ITOT .EQ. 1 ) JEOF = IEOF
C IPOS = 1
C IPRL(4) = IBL(2)
C IPRL(5) = IBL(1)
C CALL WTIQIO(IORLB,1,1,,ISB,IPRL,IDS)
C GO TO 28
C
C COME HERE WHEN ALL BLOCKS HAVE BEEN READ
C
C 999 CLOSE(UNIT=2,DISPOSE='SAVE')
C CALL WTIQIO(IODET,1,1)
C STOP
C END