Systematic Redesign
Modifying Object Code

Phil Curry
Alvin Community College

Many different types of files may be maintained on a computer system, right? Well in actuality this statement is false. There are only two types of files, data files and program files. The data structures and contents of the files are basically immaterial to the type of file it is. The way a file is being used determines the type of file it is. Let us prove this concept by looking at several files that would be typically found on a computer system.

A :LISTF command is done to examine file information on a particular file.

```
:LISTF DATA,2
ACCOUNT= ISIS        GROUP= MISC

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

DATA               80B  FA           4          4   3         3  1  1
```

```
The file contents are as follows:

R.P. Gee              1401 Printer Street        White Plains    NY
Anne C. Cobol         P.O. Box X3J4              Boston          MA
N.C. Seay             4 Tran Street              Los Angeles     CA
Flop E. Disc          256 Sector                 Half Track      IL
```

What does the file contain? The answer is fairly easy, the file contains data.

A :LISTF command is done to examine file information for another file.

```
:LISTF FSOURCE,2
ACCOUNT= ISIS        GROUP= MISC

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FSOURCE            80B  FA          14         14   3         6  1  1
```

```
The file contents are as follows:

$CONTROL USLINIT,SOURCE,WARN
$TITLE "SUMMATION PROGRAM"
      PROGRAM SUMM
```

```
      IMPLICIT INTEGER (A-Z)
      SUM=0
C
   10 DISPLAY "ENTER NUMBER: "
      ACCEPT NUMBER
      IF (NUMBER.EQ.-1) GO TO 20
      SUM=SUM+NUMBER
      GO TO 10
   20 DISPLAY "The sum of the numbers is: ", SUM
      STOP
      END
```

What does the file contain? Again, the file contains data. The file does not contain a program as some would answer. This file is used as input data to a Fortran compiler which produces a USL file as output. When a :PREP command is used at the MPE level or the -PREPARE command is used in the Segmenter, the USL file is used as data to the Segmenter and a file containing object code is produced.

Looking at one other file on the system, a :LISTF command shows the following:

```
:LISTF FOBJECT,2
ACCOUNT=  ISIS        GROUP=  MISC

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FOBJECT   PROG    128W  FB           5          5   1        6  1  1
```

The file contents (as viewed with FCOPY) are as follows:

```
:RUN FCOPY.PUB.SYS

HP32212A.3.17 FILE COPIER (C) HEWLETT-PACKARD CO. 1982

>FROM=FOBJECT;TO=;OCTAL;CHAR

FOBJECT RECORD 0 (%0, #0)

00000: 004600 000001 000003 000001 ........
00004: 000002 001440 000000 177777 ... ....
00010: 000004 000000 000000 177777 ........
00014: 000000 000003 000001 177777 ........
00020: 000000 000000 000000 000000 ........
00024: SAME: TO 000034-1
00034: 140400 000124 000000 000000 ...T....
00040: SAME: TO 000050-1
00050: 000400 000600 001000 000171 .......y
00054: 000371 000000 000000 177310 ........
00060: 177000 177000 177000 000002 ........
00064: 001362 000576 001374 000000 ...~....
00070: 002003 117103 002601 001400 ...C....
00074: 152703 043117 041112 141075 ..FOBJ.=
00100: 052040 002001 000767 000400 T ......
00104: 000200 001120 000601 046511 ...P..MI
00110: 051503 020040 020040 044523 SC    IS
00114: 044523 020040 020040 000000 IS    ..
00120: 000000 000000 177000 177000 ........
00124: 177004 177006 000007 000006 ........
00130: 177511 000017 177310 000001 .I......
00134: 000042 000001 000000 000000 ."......
00140: 000642 000362 177310 177310 ........
00144: 177310 000000 000000 000000 ........
00150: 177310 000001 000112 000000 .....J..
```

```
00154: 000000 000003 001400 152703 ........
00160: 177730 000000 000001 001544 .......d
00164: 001560 001600 000001 000000 .p......
00170: 000000 000713 000400 130564 .......t
00174: 001400 064445 000000 000000 ..i%....

FOBJECT RECORD 1 (%1, #1)

00000: 003000 002400 177777 000000 ........
00004: SAME: TO 000200-1

< CONTROL Y >

2 RECORDS PROCESSED *** 0 ERRORS
```

What kind of file is this? The answer to this question is not as easy as the others. The file could be a data file or a program. We must run :LISTDIR2.PUB.SYS to answer this question.

```
:RUN LISTDIR2.PUB.SYS

LISTDIR2 C.01.00 (C) HEWLETT-PACKARD CO., 1977
TYPE 'HELP' FOR AID

>LISTF FOBJECT;PASS
**********
FILE: FOBJECT.MISC.ISIS

FCODE: PROG              FOPTIONS: STD,BINARY,FIXED
BLK FACTOR: 1            CREATOR: MANAGER
REC SIZE: 256(B)         LOCKWORD:
BLK SIZE: 128(W)         SECURITY--READ:    ANY
EXT SIZE: 6(S)                   WRITE:   ANY
# REC: 5                        APPEND:  ANY
# SEC: 6                        LOCK:    ANY
# EXT: 1                        EXECUTE: ANY
MAX REC: 5                     **SECURITY IS ON
MAX EXT: 1               COLD LOAD ID: %15106
# LABELS: 0              CREATED: THU, 23 JUN 1983
MAX LABELS: 0            MODIFIED: THU, 23 JUN 1983
DISC DEV #: 3            ACCESSED: THU, 23 JUN 1983
DISC TYPE: 0             LABEL ADR: %146665
DISC SUBTYPE: 9          SEC OFFSET: %1
CLASS: DISC              FLAGS: NO ACCESSORS
FCB VECTOR: %0

# SEG: 1                 TOTAL DB: %3
STACK: %1440             DL: %0
MAXDATA: DEFAULT         CAP: IA,BA
>EXIT
```

Since the display does not indicate the file is loaded, this is a data file. The loader uses this file as data when a program is to be run that is not currently loaded on the system. Once the program is loaded, we would be correct in saying that this file contains a program. When the program is loaded, the file cannot be modified since the operating system has control over it. However, if the file is not being used for program execution, it may be accessed just as any other data file on the system.

The perpetual data concept should be quite obvious. The output of a process is data to another process. A person uses a terminal and a text editor to produce a file containing source code. This source code is data to a compiler which produces a USL file, and so on. Any of the files throughout the process may be accessed and modified before con- tinuing to the next logical step. What will be demonstrated is the taking of the output from the Segmenter, prepared object code, and modifying it before it is used as data to the loader.

DEBUG and DECOMP are used quite extensively in this paper. The reader is assumed to be familiar with DEBUG and the architecture of the HP 3000 including the structure of the data stack. You will learn quickly there are several tools which are invaluable. Several manuals which will be necessary are the Machine Instruction Set, System Reference, System Intrinsics and the SPL Reference manual along with the Instruction Decoding Pocket Guide. A copy of the System Tables Manual could also prove helpful.

Many people wonder why in the world would anyone want to modify object code. The only practical answer is "Any time a program needs to be modified and the source code is not available". If the source code is available, it would be much simpler to make the necessary changes to the code and create a new object file. All of us, however, have programs on our systems which we have no source code for. An example would be some of the programs found in the Contributed Software Library. Many times people will contribute only object code to make sure all changes to the source code are controlled at their site.

Many times though, we need to make changes to a program for which we have no source code. One example of this would be the run-time information placed in the the object code by the Segmenter. Looking at the listing produced by LISTDIR2 we can see there is one code segment, the total global-DB area is three words, the stack parameter used at run time is 1440 (octal), the DL size in words is zero, the maxdata parameter used at run time is the default (-1), and the capabilities of the program are IA and BA. The values mentioned are stored in record zero of the object code. The data structure for prepared object code can be found in the System Tables Manual in Chapter 10. There is also an excellent article in the proceedings for the 1978 HP User's Group North America meeting held in Denver which explains in a more narrative fashion the structure of object code and USL files.

Record zero of all object code contains the following:

Word (zero based)	Contents
0	Flags (Capabilities) Bit 7: BA 8: IA 9: PM 12: MR 14: DS 15: PH
1	Number of segments in the program
2	The number of words in the global-DB area of the program's run-time stack
5	The initial stack size
6	The initial DL size (zero if DL= not specified in PREP)
7	The maxdata specification (-1 if maxdata= not specified in PREP)

By knowing the above, any of the parameters which are set by the Segmenter when the object code is created may be changed by updating the appropriate word in the object code. One would not want to modify the number of segments unless you are going to add a segment to the object code (no easy task). The total global-DB value one would rarely want to change. By altering the total global-DB up, however, one can set aside some global area that the original program knows nothing about to be used by the patch being implemented or a patched in procedure.

Now that we know how to change the parameters set by the Segmenter, we need to know how to modify the instructions generated by the compiler. One way to do this is to use

the program DECOMP which is in the Contributed Software Library. DECOMP allows the dumping of object code, showing the op codes for the program instructions, and allows the modification of these instructions. Please notice that I stated modification of instructions. Other means must be taken for inserting instructions into object code. Deleting instructions is another matter. Any instruction can be quickly "deleted" by changing it to a NOP, which is the mnemonic for a "No operation" or do nothing instruction.

Let us look at a typical problem and a solution. We have a program which opens an Image data base with exclusive access to generate a report. There is a problem in that this program must be run when no one is accessing the Image data base. We do not have the source code for the program but we need to change the call to DBOPEN to use mode 5 rather than mode 3 to allow concurrent access to the data base.

First off, we will need to verify the fact that the DBOPEN is really using mode 3. The program calls DBEXPLAIN when a DBOPEN fails to show what happened. In the display we are shown that mode 3 was in fact used.

```
:RUN XTEST;LIB=P

OPEN ERROR OF DATA BASE

IMAGE ERROR AT %000010: CONDITION WORD = -1
DBOPEN, MODE 3, ON ISISDB
DATA BASE IN USE


END OF PROGRAM
:
```

If the program did not do this, we could use DBUTIL's SHOW command to verify that the program did indeed open the data base with a mode of 3.

Next we need to get a load map for the program to see where DBOPEN is called from in the program. The format for LMAPs can be found in the MPE Commands manual in appendix D.

```
:RUN XTEST;LMAP;MAXDATA=10000;LIB=P

PROGRAM FILE XTEST.MISC.ISIS

CANYDATE1          PROG 2  13    3 PSL  2  13   7
SEMESTER           PROG 2  11    3 PSL  0   2   4
ISISPREP           PROG 2   2    3 PSL  0   7   4
CVRI               PROG 2  14    2 PSL  2  20   7
CCOMPRESS          PROG 2  12    3 PSL  2  15   7
                           13    2
CFLAGS             PROG 2  10    2 PSL  2   7  14
                            7    1
SORTEND            PROG 0  23    3 SSL  0   1 206
C'PERFORM          PROG 0  22    3 SSL  0  11 232
SORTINITIAL        PROG 0  21    3 SSL  0  11 206
C'SORTERR          PROG 0  20    3 SSL  0  16 232
C'CLOSE            PROG 0  17    3 SSL  0   2 232
C'SORTINITIAL      PROG 0  16    3 SSL  0  11 231
C'ACCEPT           PROG 0  14    3 SSL  0  50 231
DBOPEN             PROG 2   3    3 SSL  2   2 276
DBOPEN             PSL  2  11    4 SSL  2   2 276
DBCLOSE            PROG 2  16    2 SSL  2  13 267
SORTINPUT          PROG 0  15    2 SSL  0   3 206
C'TST'             PROG 0  11    2 SSL  0  12 231
TERMINATE'         PROG 0  10    3 SSL  0   2  41
                            7    2
DBEXPLAIN          PROG 2   7    3 SSL  2   1 300
                            3    2
DBGET              PROG 2   2    2 SSL  2   2 272
C'ENDPAR           PROG 0  17    2 SSL  0   3 232
                           14    1
C'DISPLAY'FIN      PROG 0   6    3 SSL  0  36 231
                            6    2
                           13    1
C'DISPLAY'FC       PROG 0  12    1 SSL  0  37 231
C'DISPLAY'INIT     PROG 0   4    3 SSL  0  34 231
                            4    2
                           11    1
C'DISPLAY'L        PROG 0   5    3 SSL  0  33 231
                            5    2
                           10    1
C'EDITMOVEAN       PROG 0   6    1 SSL  0   2 231
C'WRITE'           PROG 0   5    1 SSL  0  20 232
SORTOUTPUT         PROG 0   4    1 SSL  0   2 206
C'EDITMOVEN        PROG 0   3    1 SSL  0   1 231
C'OPEN'            PROG 0  15    3 SSL  0  23 232
                            2    1
QUIT               PROG 0  15    1 SSL  0  21  16
                           10    0
QUIT               PSL  0  36   14 SSL  0  21  16
QUIT               PSL  0   5   10 SSL  0  21  16
QUIT               PSL  0  32    7 SSL  0  21  16
C'GOTO             PROG 0  12    2 SSL  0   4 232
                            7    0
COBOLTRAP          PROG 0   3    0 SSL  2  44 231

DEBUG              PROG 0   2    0 SSL  0  10  52
PRINT              PSL  0  40   14 SSL  0  35  45
PRINT              PSL  0   4   10 SSL  0  35  45
PRINT              PSL  0  12    4 SSL  0  35  45
WHO                PSL  0  37   14 SSL  0  30  45
DLSIZE             PSL  0  11   10 SSL  0  25 100
GENMESSAGE         PSL  0  10   10 SSL  0  11  65
PRINTFILEINFO      PSL  0   7   10 SSL  0   2  46
PRINTFILEINFO      PSL  0  23    4 SSL  0   2  46
FOPEN              PSL  0   6   10 SSL  0   1   4
FOPEN              PSL  0  16    4 SSL  0   1   4
```

```
RTOI'           PSL  0  40   7 SSL  0 102 203
COMMAND         PSL  0  37   7 SSL  0   1  25
FMTDATE         PSL  0  36   7 SSL  0  14  65
CLOCK           PSL  0  35   7 SSL  0  16  35
CALENDAR        PSL  0  34   7 SSL  0  17  35
FMTCALENDAR     PSL  0  33   7 SSL  0  15  65
CONVERTDATE     PSL  0  31   7 SSL  0  23  65
MYCOMMAND       PSL  0  30   7 SSL  0  27  45
FCONTROL        PSL  0  27   7 SSL  0   6 104
FCONTROL        PSL  0  20   4 SSL  0   6 104
FSETMODE        PSL  0  26   7 SSL  0   4 104
FREAD           PSL  0  25   7 SSL  0   4  15
FREAD           PSL  0  17   4 SSL  0   4  15
FWRITE          PSL  0  24   7 SSL  0   3  15
FWRITE          PSL  0  13   4 SSL  0   3  15
FREADDIR        PSL  0  25   4 SSL  0   1  15
TERMINATE       PSL  0  24   4 SSL  0   1  41
DFIX            PSL  0  21   4 SSL  0   3 205
DFLOAT'         PSL  0  15   4 SSL  0 151 203
EXTIN'          PSL  0  14   4 SSL  0  41 205

301 302 303 304

OPEN ERROR OF DATA BASE

IMAGE ERROR AT %000010: CONDITION WORD = -1
DBOPEN, MODE 3, ON ISISDB
DATA BASE IN USE


END OF PROGRAM
:
```

DBOPEN is called in the program code from segment 3 and is at external segment transfer table entry 3. By looking at the LMAP, we can tell that mode 5 can be used since there are no calls to DBLOCK or any of the Image intrinsics which modify the data base.

Next, DECOMP is run to find where in segment 3 the PCALs to DBOPEN occur. The first command entered is to determine where the segment transfer table begins in the code segment.

```
:RUN DECOMP.LIB.SYS

HP3000 DECOMPILER 6.1

FILE NAME? XTEST
TYPE 'HELP' FOR ASSISTANCE.

-F 3.
  3.500   100601  SEGMENT TRANSFER TABLE (PL-%23)  SORTEND
  3.501   104600  SEGMENT TRANSFER TABLE (PL-%22)  C'PERFORM
  3.502   104601  SEGMENT TRANSFER TABLE (PL-%21)  SORTINITIAL
  3.503   107200  SEGMENT TRANSFER TABLE (PL-%20)  C'SORTERR
  3.504   101200  SEGMENT TRANSFER TABLE (PL-%17)  C'CLOSE
  3.505   104602  SEGMENT TRANSFER TABLE (PL-%16)  C'SORTINITIAL
  3.506   111600  SEGMENT TRANSFER TABLE (PL-%15)  C'OPEN'
  3.507   124202  SEGMENT TRANSFER TABLE (PL-%14)  C'ACCEPT
  3.510   105576  SEGMENT TRANSFER TABLE (PL-%13)  CANYDATE1
  3.511   106576  SEGMENT TRANSFER TABLE (PL-%12)  CCOMPRESS
  3.512   101177  SEGMENT TRANSFER TABLE (PL-%11)  SEMESTER
  3.513   101042  SEGMENT TRANSFER TABLE (PL-%10)  TERMINATE'
  3.514   100534  SEGMENT TRANSFER TABLE (PL-%7)  DBEXPLAIN
  3.515   117202  SEGMENT TRANSFER TABLE (PL-%6)  C'DISPLAY'FIN
  3.516   115602  SEGMENT TRANSFER TABLE (PL-%5)  C'DISPLAY'L
  3.517   116202  SEGMENT TRANSFER TABLE (PL-%4)  C'DISPLAY'INIT
```

```
  3.520   101132  SEGMENT TRANSFER TABLE (PL-%3)  DBOPEN
  3.521   103577  SEGMENT TRANSFER TABLE (PL-%2)  ISISPREP
  3.522   000000  SEGMENT TRANSFER TABLE (PL-%1)
  3.523   040023  STT LENGTH = %23
```

The last instruction will be 0.477 since the segment transfer table starts at 3.500, so we look for a PCAL to DBOPEN in 0.0 through 0.477.

```
-F "PCAL  DBOPEN",0/477

  3.10    031003  2.    PCAL  DBOPEN

-EXIT

END OF PROGRAM
:
```

DBOPEN is called from one place in the program, segment 3 offset 10. Now we need to run the program and invoke Debug to determine a memory location to use which contains the value 5 to reference in the DBOPEN for the mode parameter.

```
:RUN XTEST;LIB=P;DEBUG


*DEBUG* 0.0
```

A breakpoint is set at the call to DBOPEN.

```
?B 3.10
?R


*BREAK* 3.10
```

Next we can look at the program segment to verify we broke at the correct point in the program.

```
?DP-5,6,C
P-5      031002  PCAL STT 2
P-4      040022  LOAD P+22
P-3      040022  LOAD P+22
P-2      040022  LOAD P+22
P-1      040022  LOAD P+22
P+0      031003  PCAL STT 3
```

Next we look at the run time stack to see what was placed on the stack in preparation for the call to DBOPEN. DBOPEN has 4 parameters so we look at the top 4 items on the stack. Notice also in the dump of the program segment above that we can see where the 4 loads to top of stack occur.

```
?D S-3,4
S-3      000513 000535 000540 000471
```

Since the intrinsic DBOPEN uses call by reference for all it's parameters, we know that these values on the stack are memory addresses and not actual values.

The parameters are located in the stack as follows:

Location	Parameter for DBOPEN
S-3	BASE
S-2	PASSWORD
S-1	MODE

```
    S-0                 STATUS
```

We can verify this by displaying the contents of memory at these locations.

```
    ?D 513,10,A
    DB+513      ISISDB
    ?D 535,10,A
    DB+535      ; ..............
    ?D 540,I
    DB+540      +00003
```

We know that the DBOPEN is for the data base ISISDB using the password of the creator (;) with an open mode of exclusive access, mode 3.

Next we look at the memory locations around the mode parameter to see if there is a location containing a 5 which can be substituted. We look for an alternate memory reference rather than changing the value of DB+540 since this location could be referenced elsewhere in the program and changing the contents could produce unpredictable results.

```
    ?D 530,20,I
    DB+530      +08224 +08224 +08224 +08224 +08224 +15136 +00001 +00002
    DB+540      +00003 +00004 +00005 +00006 +00007 +00000 +00000 +00000
```

A pattern of values can be found by looking at DB+536 through DB+544. The values contained in these locations are 1 through 7. Since this pattern is found, we will assume that these are being used as constants in the program for the Image calls. This could be verified by setting breakpoints at calls to DBFIND, DBGET and DBCLOSE and examining the locations passed to these calls, however since we are highly certain these values are alright to use as constants, we will continue.

Next we will modify the contents of S-1, the parameter for the mode, from DB+540 which contains the value 3, to DB+542 which contains the the value 5.

```
    ?MS-1
    S-1       000540 :=542
    ?R

    Do you wish to extract only on-campus sections?
```

As we can see the data base opened and the program continued executing.

We have now found out what needs to be changed in the call to DBOPEN. Now we need to patch the object code to call DBOPEN using the address DB+542 for the mode parameter rather than DB+540.

```
    :RUN DECOMP.LIB.SYS

    HP3000 DECOMPILER 6.1

    FILE NAME? XTEST
    TYPE 'HELP' FOR ASSISTANCE.

    -3.0
    FILE  XTEST.MISC.ISIS
    SEGMENT 3          LENGTH 524


      3.0      170400  ..    LRA   P- 000      <<=0>><-- Procedure Entry
      3.1      051604  S.    STOR  Q- 004
      3.2      031400  3.    EXIT  %0

      3.3      031002  2.    PCAL  ISISPREP
      3.4      040022  @.    LOAD  P+ 022      <<=26>>
      3.5      040022  @.    LOAD  P+ 022      <<=27>>
      3.6      040022  @.    LOAD  P+ 022      <<=30>>
      3.7      040022  @.    LOAD  P+ 022      <<=31>>
```

```
3.10    031003  2.    PCAL  DBOPEN
3.11    040020  @.    LOAD  P+ 020      <<=31>>
3.12    043700  G.    LOAD  S- 000,I
3.13    000106  .F    DELB, ZERO
3.14    001706  ..    CMP   , ZERO
3.15    141202  ..    BE    P+ 002      <<=17>>
3.16    006400  ..    NOT   , NOP
3.17    017714  ..    BRE   P+ 014,I    <<=33->62>>
3.20    040012  @.    LOAD  P+ 012      <<=32>>
3.21    021002  ".    LDI   2
3.22    031004  2.    PCAL  C'DISPLAY'INIT
3.23    021006  ".    LDI   6
3.24    170010  ..    LRA   P+ 010      <<=34>>
3.25    140024  ..    BR    P+ 024      <<=51>>
3.26    000513  .K    DECX, MPYL
3.27    000535  .]    DECX, XAX
3.30    000540  .`    DECX, DEL
3.31    000471  .9    INCX, FIXT
3.32    001655  ..    DXCH, FNEG
3.33    000027  ..    NOP   , DTST
3.34    000005  ..    NOP   , DECX
3.35    013517  .O    TCBC  BIT 15
3.36    050105  PE    TBA   P+ 105      <<=143>>
3.37    047040  N     LOAD  DB+040,I,X
3.40    042522  ER    LOAD  P- 122,I    <<=177716>>
3.41    051117  RO    STOR  DB+117
3.42    051040  R     STOR  DB+040
3.43    047506  OF    LOAD  Q+ 106,I,X
3.44    020104   D    ----  << DOUBLE WORD >>
3.45    040524  AT    ????
3.46    040440  A     LOAD  P- 040      <<=6>>
```

Again, notice the 4 load instructions before the PCAL to DBOPEN at 3.10. The third load instruction at 3.6 shows a load of P+22. To the reside of the instruction <<=30>> is shown, indicating the location ferenced. Looking at the contents of 3.30 we notice the value 540. This needs to be changed to 542 using the modify command.

```
-M 3.30
  3.30    000540  .`    DECX, DEL                          :=542
  3.30    000542  .b    DECX, LDXB
-E

END OF PROGRAM
```

Finally we can run the program to verify that the patch is correct.

```
:RUN XTEST;LIB=P;DEBUG


*DEBUG* 0.0
?B 3.10
?R


*BREAK* 3.10
?D S-3,4
S-3      000513 000535 000542 000471
?D 542,I
DB+542    +00005
?R

Do you wish to extract only on-campus sections?
```

By looking at the contents of the stack we can observe the address for the mode parameter passed is indeed DB+542 and the contents of the address is 5.

Let's look at a second example. We have a program that uses a tape as input data and creates a report. We know the tape file is opened with the file name "CBM004". We would like to produce the same report using a disk file as input so we enter the appropriate file equation and run the program.

```
:FILE CBM004;DEV=DISC
:RUN XREPORT
```

After waiting a while we press the break key and type RECALL. We find that the program is still expecting to use a tape as input.

```
:RECALL
THE FOLLOWING REPLIES ARE PENDING:
?11:06/#S76/109/LDEV# FOR "CBM004" ON TAPE (NUM)?
```

We suspect the program has the bit set in the foptions for FOPEN which indicates to disallow file equations. To verify this we need to run the program again with the LMAP option to find what segments call FOPEN in the program.

```
:RUN XREPORT;LMAP

PROGRAM FILE XREPORT.MISC.ISIS

IIO'            PROG 0  16   0 SSL  0  34 205
DIO'            PROG 0  15   0 SSL  0  33 205
DATELINE        PROG 3  14   0 SSL  0  35 205
FSET            PROG 3  13   0 SSL  0  16 205
FOPEN           PROG 0  12   0 SSL  0   1   4
PRINTFILEINFO   PROG 3  11   0 SSL  0   2  46
SIO'            PROG 0  10   0 SSL  0  30 205
RIO'            PROG 0   7   0 SSL  0  32 205
TERMINATE'      PROG 0   6   0 SSL  0   2  41
FMTINIT'        PROG 0   5   0 SSL  0  21 205
TFORM'          PROG 0   4   0 SSL  0  22 205
OVFL'           PROG 0   3   0 SSL  0 143 203

301
```

We know that there is only 1 segment in the program and that FOPEN is at STT entry 12. Now we need to find where in the program FOPEN is called.

```
:RUN DECOMP.LIB.SYS

HP3000 DECOMPILER 6.1

FILE NAME? XREPORT
TYPE 'HELP' FOR ASSISTANCE.
```

Display the segment transfer table to find the last instruction in the segment.

```
-F 0.
  0.3015  116122  SEGMENT TRANSFER TABLE (PL-%16)  IIO'
  0.3016  115522  SEGMENT TRANSFER TABLE (PL-%15)  DIO'
  0.3017  116522  SEGMENT TRANSFER TABLE (PL-%14)  DATELINE
  0.3020  107122  SEGMENT TRANSFER TABLE (PL-%13)  FSET
  0.3021  100406  SEGMENT TRANSFER TABLE (PL-%12)  FOPEN
  0.3022  101047  SEGMENT TRANSFER TABLE (PL-%11)  PRINTFILEINFO
  0.3023  114122  SEGMENT TRANSFER TABLE (PL-%10)  SIO'
  0.3024  115122  SEGMENT TRANSFER TABLE (PL-%7)  RIO'
  0.3025  101042  SEGMENT TRANSFER TABLE (PL-%6)  TERMINATE'
  0.3026  110522  SEGMENT TRANSFER TABLE (PL-%5)  FMTINIT'
  0.3027  111122  SEGMENT TRANSFER TABLE (PL-%4)  TFORM'
  0.3030  161572  SEGMENT TRANSFER TABLE (PL-%3)  OVFL'
```

```
      0.3031  001403  SEGMENT TRANSFER TABLE (PL-%2)  SRS05S
      0.3032  000000  SEGMENT TRANSFER TABLE (PL-%1)
      0.3033  040016  STT LENGTH = %16
```

The last instruction will be 0.3014, so we look for a PCAL to FOPEN in 0.0 through 0.3014.

```
    -F "PCAL  FOPEN",0/3014

      0.1574  031012  2.    PCAL  FOPEN
      0.1703  031012  2.    PCAL  FOPEN
    -E

    END OF PROGRAM
```

Now we run the program and use DEBUG to set breakpoints at the two calls to FOPEN.

```
    :RUN XREPORT;DEBUG


    *DEBUG* 0.1403
    ?B 1574,1703
    ?R


    *BREAK* 0.1574
```

We know the parameters used in FOPEN by looking in the System Intrinsics manual. The layout of the stack before a call to FOPEN is as follows:

Location	Parameter for FOPEN
S-16	FORMALDESIGNATOR
S-15	FOPTIONS
S-14	AOPTIONS
S-13	RECORD SIZE
S-12	DEVICE
S-11	FORMS MESSAGE
S-10	USER LABELS
S-7	BLOCKING FACTOR
S-6	NUMBER OF BUFFERS
S-5	FILE SIZE (DOUBLE WORD)
S-3	NUMBER OF EXTENTS
S-2	INITIAL ALLOCATION
S-1	FILE CODE
S-0	OPTION VARIABLE MASK

We look now at the top 17 locations on the stack.

```
?D S-16,17
S-16     001416 002001 000000 177660 001426 000000 000000 000014
S-6      020040 020040 020040 020040 020040 020040 017440
```

Next we look at the file name used in the FOPEN. Since there are two calls to FOPEN we need to know if this is the open for the tape file or the report file.

```
?D 1416/2,10,A
DB+607    CBM004 .TAPE ...
```

This is indeed the FOPEN for the file CBM004. Let's look at the parameters for record size and device.

```
?=177660,I
 =-80
?D 1426/2,10,A
DB+613    TAPE .....
```

The file is opened for an 80 byte record and looking at the foptions value of 2001, we know bit 5 was indeed set to disallow file equations. Let's change the value of the foptions at S-15 to 1 to allow file equations then change the data stored at DB+613 and DB+614 to indicate a disk device rather than tape.

```
?M S-15
S-15     002001 :=1
?M 613,2
DB+613    052101 :="DI"
DB+614    050105 :="SC"
?D 613,2,A
DB+613    DISC
?R


*BREAK* 0.1703
```

This is the second FOPEN so resume execution.

```
?R


END OF PROGRAM

:
```

As we can see the program did not wait for a tape request so the disk file was used as input rather than the tape.

We would like to have the program allow file equations and default to disk rather than default to tape as it currently does. We start by running DECOMP to find where value 2001, the foptions value, is stored. We do not know the address using DEBUG since the foptions parameter is passed by value.

```
:RUN DECOMP.LIB.SYS

HP3000 DECOMPILER 6.1

FILE NAME? XREPORT
TYPE 'HELP' FOR ASSISTANCE.
```

First we decompile the statements close to the PCAL to FOPEN to see how the parameters are placed on the stack.

```
-1540
  0.1540  163001  ..    STD   DB+001,I
  0.1541  000647  ..    ZERO, FLT
```

```
0.1542  163000  ..    STD   DB+000,I
0.1543  140005  ..    BR    P+ 005      <<=1550>>
0.1544  041502  CB    LOAD  Q+ 102
0.1545  046460  M0    LOAD  P- 060,I,X <<=1465->53131>>
0.1546  030064  04    ----
0.1547  020016   .    MOVE  PB-DB SDEC=2
0.1550  034404  9.    LDPN  %4          <<=1544>>
0.1551  034403  9.    LDPN  %3          <<=1546>>
0.1552  140004  ..    BR    P+ 004      <<=1556>>
0.1553  052101  TA    MTBA  P+ 101      <<=1654>>
0.1554  050105  PE    TBA   P+ 105      <<=1661>>
0.1555  020000   .    MOVE  PB-DB SDEC=0
0.1556  040403  A.    LOAD  P- 003      <<=1553>>
0.1557  034403  9.    LDPN  %3          <<=1554>>
0.1560  000600  ..    ZERO, NOP
0.1561  171707  ..    LRA   S- 007
0.1562  010201  ..    LSL   1 BIT
0.1563  040016  @.    LOAD  P+ 016      <<=1601>>
0.1564  000600  ..    ZERO, NOP
0.1565  025120  *P    LDNI  80
0.1566  171707  ..    LRA   S- 007
0.1567  010201  ..    LSL   1 BIT
0.1570  000700  ..    DZRO, NOP
0.1571  021014  ".    LDI   12
0.1572  035006  :.    ADDS  %6
0.1573  040007  @.    LOAD  P+ 007      <<=1602>>
0.1574  031012  2.    PCAL  FOPEN
0.1575  051707  S.    STOR  S- 007
0.1576  035406  ;.    SUBS  %6
0.1577  051406  S.    STOR  Q+ 006
0.1600  140003  ..    BR    P+ 003      <<=1603>>
0.1601  002001  ..    ADD , DELB
0.1602  017440  .     TSBC  BIT 32,X
0.1603  141502  .B    BNE   P+ 002      <<=1605>>
0.1604  140042  ."    BR    P+ 042      <<=1646>>
0.1605  041406  C.    LOAD  Q+ 006
0.1606  031011  2.    PCAL  PRINTFILEINFO
0.1607  000707  ..    DZRO, DZRO
0.1610  021002  ".    LDI   2
0.1611  172003  ..    LRA   P+ 003,I    <<=1614->1645>>
0.1612  031005  2.    PCAL  FMTINIT'
0.1613  140020  ..    BR    P+ 020      <<=1633>>
```

We can see that the file name CBM004 is P-relative data and is loaded onto the stack by the LDPN instructions at 0.1550 through 0.1551. We can also see that the device type is loaded onto the stack by the LOAD and LDPN instructions at 0.1556 through 0.1557. The address of the file name is pushed onto the stack at 0.1561 then this word address is converted to a byte address by shifting the value left 1 bit at 0.1562. The value for the foptions is pushed onto the stack at the next instruction 0.1563, which is LOAD P+16. The value to the side of the instruction indicates the P-relative address of P+16 is 1601. By looking at the value at 0.1601 we see it is 2001. This value needs to be changed to the value one as we did using DEBUG.

```
-M 1601
  0.1601  002001  ..    ADD , DELB                          :=1
  0.1601  000001  ..    NOP , DELB
```

Finally, we need to change the value at 0.1553 through 0.1554 from "TAPE" to "DISC" so the file will default to the disk device.

```
-M 1553/1554
  0.1553  052101  TA    MTBA  P+ 101      <<=1654>>         :="DI"
  0.1553  042111  DI    LOAD  P+ 111,I    <<=1664->36270>>
  0.1554  050105  PE    TBA   P+ 105      <<=1661>>         :="SC"
  0.1554  051503  SC    STOR  Q+ 103
-D 1553/1554
  0.1553  042111 051503                                     DISC
-E
```

```
    END OF PROGRAM
```

Now we run the program using DEBUG to verify that our patch is correct.

```
    :RUN XREPORT;DEBUG

    *DEBUG* 0.1403
    ?B 1574
    ?R

    *BREAK* 0.1574
```

Looking at S-15,'we see the foptions is now 1, and looking at DB+613 we see the contents is now "DISC".

```
    ?D S-16,17
    S-16      001416 000001 000000 177660 001426 000000 000000 000014
    S-6       020040 020040 020040 020040 020040 020040 017440

    ?D 1426/2,10,A
    DB+613    DISC ..........
    ?R

    END OF PROGRAM
```

The final example will show multiple solutions to solve a problem. We have a program which allows one to change the terminal type for $STDIN. The program does not need SM capability, yet it checks to see if the person running the program is MANAGER.SYS, and if not, the program terminates.

```
    :RUN XDEV


    USER:  MANAGER
    ACCOUNT:  ISIS
    GROUP:    MISC
    LOGON DEVICE:      23
    You must be MANAGER.SYS to run this program.

    END OF PROGRAM
```

A simple solution would be to write our own WHO subroutine which always returns the user as MANAGER.SYS and place it in an SL. The problem in doing this is that all programs which call the WHO intrinsic and use this SL will always return the user as MANAGER.SYS and we want only this one program to have MANAGER.SYS returned to it.

We start by looking at the program file information using LISTDIR2.PUB.SYS.

```
    :RUN LISTDIR2.PUB.SYS

    LISTDIR2 C.01.00 (C) HEWLETT-PACKARD CO., 1977
    TYPE 'HELP' FOR AID

    >LISTF XDEV
    **********
    FILE: XDEV.MISC.ISIS

    FCODE: PROG           FOPTIONS: STD,BINARY,FIXED
    BLK FACTOR: 1         CREATOR: **
    REC SIZE: 256(B)      LOCKWORD: **
    BLK SIZE: 128(W)      SECURITY--READ:    ANY
    EXT SIZE: 7(S)                WRITE:   ANY
    # REC: 6                      APPEND:  ANY
```

```
# SEC: 7                          LOCK:    ANY
# EXT: 1                          EXECUTE: ANY
MAX REC: 6                      **SECURITY IS ON
MAX EXT: 1              COLD LOAD ID: %15115
# LABELS: 0             CREATED: MON, 18 JUL 1983
MAX LABELS: 0           MODIFIED: MON, 18 JUL 1983
DISC DEV #: 2           ACCESSED: MON, 18 JUL 1983
DISC TYPE: 0            LABEL ADR: **
DISC SUBTYPE: 8         SEC OFFSET: %1
CLASS: DISC             FLAGS: NO ACCESSORS
FCB VECTOR: %0

# SEG: 1                TOTAL DB: %25
STACK: %1440            DL: %0
MAXDATA: DEFAULT        CAP: IA,BA
>EXIT

END OF PROGRAM
```

We note that there is only one segment in the program. Next we run DECOMP to find where the WHO intrinsic is called and where the data for user, account, group and logon terminal is stored.

```
:RUN DECOMP.LIB.SYS

HP3000 DECOMPILER 6.1

FILE NAME? XDEV
TYPE 'HELP' FOR ASSISTANCE.

-F 0.
  0.360   114122  SEGMENT TRANSFER TABLE (PL-%13)  SIO'
  0.361   116122  SEGMENT TRANSFER TABLE (PL-%12)  IIO'
  0.362   101042  SEGMENT TRANSFER TABLE (PL-%11)  TERMINATE'
  0.363   110522  SEGMENT TRANSFER TABLE (PL-%10)  FMTINIT'
  0.364   111122  SEGMENT TRANSFER TABLE (PL-%7)   TFORM'
  0.365   122572  SEGMENT TRANSFER TABLE (PL-%6)   F'CONTRAP
  0.366   100442  SEGMENT TRANSFER TABLE (PL-%5)   TERMINATE
  0.367   100406  SEGMENT TRANSFER TABLE (PL-%4)   FOPEN
  0.370   114046  SEGMENT TRANSFER TABLE (PL-%3)   WHO
  0.371   103101  SEGMENT TRANSFER TABLE (PL-%2)   FCONTROL
  0.372   000021  SEGMENT TRANSFER TABLE (PL-%1)   TYPESET
  0.373   040013  STT LENGTH = %13

-F "PCAL  WHO",0/357


  0.47    031003  2.    PCAL  WHO
```

The WHO intrinsic is called from one place in the program, at 0.47. Now we need to find where the locations are placed on the stack in preparation for the call to WHO so we know where the data is to be stored.

```
  0.21    035007  :.    ADDS  %7       <-- Primary Entry TYPESET
  0.22    171700  ..    LRA   S- 000
  0.23    010201  ..    LSL   1 BIT
  0.24    051404  S.    STOR  Q+ 004
  0.25    035004  :.    ADDS  %4
  0.26    171700  ..    LRA   S- 000
  0.27    010201  ..    LSL   1 BIT
  0.30    051405  S.    STOR  Q+ 005
  0.31    035004  :.    ADDS  %4
  0.32    171700  ..    LRA   S- 000
```

```
0.33    010201  ..    LSL   1 BIT
0.34    051406  S.    STOR  Q+ 006
0.35    000706  ..    DZRO, ZERO
0.36    033405  7.    LLBL  TERMINATE
0.37    031006  2.    PCAL  F'CONTRAP
0.40    000706  ..    DZRO, ZERO
0.41    041406  C.    LOAD  Q+ 006
0.42    041405  C.    LOAD  Q+ 005
0.43    041404  C.    LOAD  Q+ 004
0.44    000600  ..    ZERO, NOP
0.45    171401  ..    LRA   Q+ 001
0.46    021035  ".    LDI   29
0.47    031003  2.    PCAL  WHO
```

We can see the parameters passed to the WHO intrinsic from the System Intrinsics manual. The parameters are placed on the stack as follows:

Location	Parameter for WHO
S-10	MODE
S-7	CAPABILITY
S-6	LOCAL ATTRIBUTES
S-5	USER NAME
S-4	GROUP NAME
S-3	ACCOUNT NAME
S-2	HOME GROUP
S-1	LOGON TERMINAL
S-0	OPTION VARIABLE MASK

Looking at 0.40 through 0.45 we see the loads to top of stack of the addresses for the parameters and at 0.46 the placing on top of stack of the option variable mask. The option variable mask is 35 octal or 29 decimal. The binary representation of 35 octal is 00/011/101. The bits which are set indicate which parameters, from right to left, that are passed to the intrinsic. The parameters for the WHO intrinsic are as follows:

WHO (mode,capability,lattr,usern, groupn,acctn,homen,term); O-V

Alligning the bits with the parameters of the intrinsic, we find the parameters passed are the fourth, fifth, sixth and eighth, which are USERN, GROUPN, ACCTN and TERM.

Looking at the loads to top of stack at 0.40 through 0.45, we see at 0.40 three words with the contents of zero loaded, which are the first three unreferenced parameters. At 0.41, we see a load of Q+6, which contains the byte address for the fourth parameter, USERN. At 0.42, we see a load of Q+5, which contains the byte address for the fifth parameter, GROUPN. At 0.43, we see a load of Q+4, which contains the byte address for the sixth parameter, ACCTN. At 0.44, a load of one word containing zero for the unreferenced seventh parameter. Finally, at 0.45, we see a load of the address of Q+1, which is the eighth parameter, TERMN.

We can run the program using DEBUG to verify this.

```
:RUN XDEV;DEBUG


*DEBUG* 0.21
?B 47
?R


*BREAK* 0.47
?DS-10,11
S-10      000000 000000 000000 000124 000114 000104 000000 000034
S+0       000035
?B 50
?R


*BREAK* 0.50
?D 124/2,4,A
DB+52     MANAGER
?D 114/2,4,A
DB+46     MISC
?D 104/2,4,A
DB+42     ISIS
?D 34,I
DB+34     +00023
?R


USER:  MANAGER
ACCOUNT:  ISIS
GROUP:    MISC
LOGON DEVICE:      23
You must be MANAGER.SYS to run this program.

END OF PROGRAM
```

Now that we know where the intrinsic WHO is being called and where the information is being stored, we can decide on a solution.

The first solution uses brute force. We decide to let the WHO intrinsic be called but will bypass the checking of the data for being the user MANAGER and the account SYS. This will allow any user to run this program. We use DECOMP to determine where the checking is done for the user to be MANAGER.SYS.

```
0.166   042105  DE    LOAD  P+ 105,I   <<=273->44674>>
0.167   053111  VI    STOR  DB+111,I
0.170   041505  CE    LOAD  Q+ 105
0.171   035040  :     ADDS  %40
0.172   040407  A.    LOAD  P- 007      <<=163>>
0.173   034407  9.    LDPN  %7          <<=164>>
0.174   034406  9.    LDPN  %6          <<=166>>
0.175   034405  9.    LDPN  %5          <<=170>>
0.176   021016  ".    LDI   14
0.177   171707  ..    LRA   S- 007
0.200   010201  ..    LSL   1 BIT
0.201   031013  2.    PCAL  SIO'
0.202   035407  ;.    SUBS  %7
0.203   171401  ..    LRA   Q+ 001
0.204   031012  2.    PCAL  IIO'
0.205   031007  2.    PCAL  TFORM'
0.206   041406  C.    LOAD  Q+ 006
```

```
0.207   140005  ..    BR    P+ 005      <<=214>>
0.210   046501  MA    LOAD  P- 101,I,X <<=107->20147>>
0.211   047101  NA    LOAD  DB+101,I,X
0.212   043505  GE    LOAD  Q+ 105,I
0.213   051040  R     STOR  DB+040
0.214   170404  ..    LRA   P- 004      <<=210>>
0.215   010201  ..    LSL   1 BIT
0.216   021010  ".    LDI   8
0.217   020243  .     CMPB  PB-DB      SDEC=3
0.220   145503  .C    BNE   P+ 003,I    <<=223->236>>
0.221   041404  C.    LOAD  Q+ 004
0.222   140006  ..    BR    P+ 006      <<=230>>
0.223   000013  ..    NOP , MPYL
0.224   051531  SY    STOR  Q+ 131
0.225   051440  S     STOR  Q+ 040
0.226   020040        MVB   PB-DB SDEC=0
0.227   020040        MVB   PB-DB SDEC=0
0.230   170404  ..    LRA   P- 004      <<=224>>
0.231   010201  ..    LSL   1 BIT
0.232   021010  ".    LDI   8
0.233   020243  :     CMPB  PB-DB      SDEC=3
0.234   141502  .B    BNE   P+ 002      <<=236>>
0.235   140050  .(    BR    P+ 050      <<=305>>

0.236   000707  ..    DZRO, DZRO
0.237   021002  ".    LDI   2
0.240   172003  ..    LRA   P+ 003,I    <<=243->304>>
0.241   031010  2.    PCAL  FMTINIT'
0.242   140030  ..    BR    P+ 030      <<=272>>
0.243   000041  .!    NOP , ZROB
0.244  054557 Yo  TBX  P- 157   <<=65>>
0.245   072440  u     ADDM  P- 040,I    <<=205->31214>>
0.246   066565  mu    CMPM  P- 165,I,X <<=61->40464>>
0.247   071564  st    ADDM  Q+ 164
0.250   020142   b    MVLB  SDEC=2
0.251   062440  e     CMPM  P- 040,I    <<=211->47312>>
0.252   046501  MA    LOAD  P- 101,I,X  <<=151->21161>>
0.253   047101  NA    LOAD  DB+101,I,X
0.254   043505  GE    LOAD  Q+ 105,I
0.255   051056  R.    STOR  DB+056
0.256   051531  SY    STOR  Q+ 131
0.257   051440  S     STOR  Q+ 040
0.260   072157  to    ADDM  P+ 157,I    <<=437>>
0.261   020162   r    SCU   SDEC=2
0.262   072556  un    ADDM  P- 156,I    <<=104->41623>>
0.263   020164   t    SCU   SDEC=0
0.264   064151  hi    CMPM  P+ 151,X    <<=435>>
0.265   071440  s     ADDM  Q+ 040
0.266   070162  pr    ADDM  P+ 162      <<=450>>
0.267   067547  og    CMPM  Q+ 147,I,X
0.270   071141  ra    ADDM  DB+141
0.271   066456  m.    CMPM  P- 056,I,X  <<=213->51253>>
0.272   025426  +.    LDXN  22
0.273   044401  I.    LOAD  P- 001,X    <<=272>>
0.274   011202  ..    IXBZ  P+ 002      <<=276>>
0.275   140402  ..    BR    P- 002      <<=273>>
0.276   021054  ",    LDI   44
0.277   171726  ..    LRA   S- 026
0.300   010201  ..    LSL   1 BIT
0.301   031013  2.    PCAL  SIO'
0.302   035426  ;.    SUBS  %26
0.303   031007  2.    PCAL  TFORM'
0.304   031011  2.    PCAL  TERMINATE'
0.305   000606  ..    ZERO, ZERO
0.306   021040  "     LDI   32
0.307   035014  :.    ADDS  %14
0.310   040004  @.    LOAD  P+ 004      <<=314>>
```

```
  0.311   031004  2.    PCAL  FOPEN
  0.312   051402  S.    STOR  Q+ 002
  0.313   140002  ..    BR    P+ 002      <<=315>>
  0.314   004000  ..    DEL , NOP
  0.315   141202  ..    BE    P+ 002      <<=317>>
  0.316   140037  ..    BR    P+ 037      <<=355>>
```

In instructions 0.166 through 0.205 we see the output of what the logon device is. At 0.206 the address of the USERN, Q+6, is loaded. Next, a branch to 0.214, to bypass the P-relative data "MANAGER ". Next, at 0.214, the address of the P-relative data is loaded onto the stack. Next, the address is logically shifted left 1 bit to form a byte address from the word address which was loaded. Next, a decimal 8, the number of bytes to compare, is loaded onto the stack. At 0.217, the CMPB instruction will compare 8 decimal bytes. At 0.220, the instruction says branch to 0.236 if the bytes compared are not equal. Therefore, if the contents of the data whose address is stored at Q+6 for a length of 8 bytes does not match "MANAGER " then branch to 0.236, otherwise continue. Next, at 0.221, the address of the ACCTN, Q+4, is loaded. The instructions at 0.222 branches around the P-relative data stored in 0.223 through 0.227. Next, at 0.230, the address of the P-relative data is loaded onto the stack. Next, the address is logically shifted left 1 bit to form a byte address from the word address. At 0.232, a decimal 8, the number of bytes to compare, is loaded onto the stack. At 0.233 is the compare bytes instruction, CMPB. At 0.234, the instruction is to branch to 0.236 if the 8 bytes are not equal, otherwise the next instruction at 0.235 is a branch to instruction 0.305.

Now we know the flow of logic for the comparison. If the USERN is not "MANAGER" then branch to 0.236. Next if ACCTN is "SYS" then branch to 0.305, otherwise branch to 0.236. To implement the "brute force" solution, instruction 0.206 is modified to BR P+77, which is a branch to 0.305, which completely bypasses any checking for MANAGER.SYS, and continues processing.

```
-M 206
  0.206   041406  C.    LOAD  Q+ 006                    :=140077
  0.206   140077  .?    BR    P+ 077      <<=305>>
-E

END OF PROGRAM
:RUN XDEV


USER:  MANAGER
ACCOUNT:  ISIS
GROUP:    MISC
LOGON DEVICE:      23
Change to term type: ?10

END OF PROGRAM
:
```

SOLUTION 2.

A second solution is to modify the P-relative data being checked. Since we are in the account "ISIS" we can change the account name being checked from "SYS" to "ISIS". The P-relative data is at 0.224 through 0.227.

```
-M 224/225
  0.224   051531  SY    STOR  Q+ 131                    :="IS"
  0.224   044523  IS    LOAD  P- 123,X    <<=101>>
  0.225   051440  S     STOR  Q+ 040                    :="IS"
  0.225   044523  IS    LOAD  P- 123,X    <<=102>>
-224
  0.224   044523  IS    LOAD  P- 123,X    <<=101>>
  0.225   044523  IS    LOAD  P- 123,X    <<=102>>
```

```
  0.226   020040         MVB   PB-DB SDEC=0
  0.227   020040         MVB   PB-DB SDEC=0
  0.230   170404   ..    LRA   P- 004      <<=224>>
 -E

END OF PROGRAM
:RUN XDEV


USER:  MANAGER
ACCOUNT:  ISIS
GROUP:    MISC
LOGON DEVICE!       23
Change to term type: ?10

END OF PROGRAM
:
```

There are some disadvantages to this method. The first is the user must be MANAGER.ISIS to run the program and cannot be any other user on the system as the patch in solution 1 allows. This may be what you want. The second disadvantage is the error message generated will need to be modified to state "You must be MANAGER.ISIS to run this program.", but as we now know, this can be easily done.

SOLUTION 3.

The third solution is a bit exotic, however this technique for patching may be necessary if the data for USERN, ACCTN, GROUPN and TERMN is used elsewhere and you wish them not to reflect what the WHO intrinsic would normally return, but rather other values. In this solution we will patch the program to show the USERN as "MANAGER", the ACCTN as "SYS" the GROUPN as "PUB" and the TERMN as 20 no matter who runs the program.

What we will do is increase the global-DB area by %14 words to allocate memory the program does not reference and place initial values in these locations. Then we will change the array pointers from pointing at there intended memory locations to instead point to our global memory locations we are allocating.

We know by looking at the output from LISTDIR2 that the total global-DB area is %25 words. Global-DB is initialized at run time from data within the object code. Word 2 of record 0 denotes the beginning record number of the image of the global-DB area of the stack. We need to allocate 4 words (8 bytes) of storage for USERN, 4 words for ACCTN and 4 words for GROUPN. What we will do is add 14 octal (12 decimal words) to the value in word 2 of record 0 so the loader will allocate not 25 octal words of global-DB storage but instead 41 octal words.

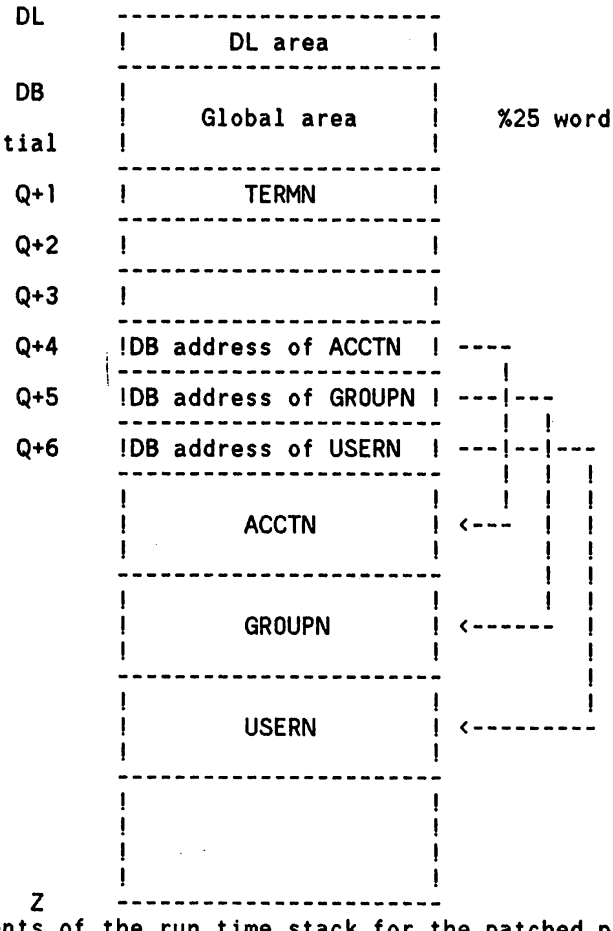
```
:RUN DISKED2.PUB.SYS

DISKED2 C.01.00 (C) HEWLETT-PACKARD CO., 1976
TYPE 'HELP' FOR INFO
>HELP

DISKED2 allows to dump and/or modify : file contents or
any disc sector (sys. mgr capability is required).

 B[ASE] [<ABS SEC #>]
 DEBUG
 DISC <LOG DEV #>
 D[UMP] [ [<REL SEC #>] [, <# OF SEC>] ] OR [<'ALL'>] [, A=ASCII ]
     (AT LEAST ONE PARAMETER MUST BE PRESENT.)
 F[ILE] <FILENAME>
 L[IST] [<DEVICE CLASS>] OR [<LOG DEV #>]
 M[ODIFY] <SEC NUM, REL WORD ADDR [,NUM OF WORDS]>
     (NEW VALUE STARTS WITH : # - DECIMAL, ' - ASCII)
```

```
 W[IDTH]
 E[XIT]

>FILE XDEV
>DUMP 1

LOGICAL SECTOR 1     *** BEGINNING OF DATA ***
SECTOR %00000220402        LDEV = %000004
000:  004600  000001  000025  000001  000002  001440  000000  177777
010:  000005  000000  000021  177777  000022  000004  000001  000001
020:  000000  000000  000000  000000  000000  000000  000000  000000
030:  000000  000000  000000  000000  000000  000374  000000  177774
040:  177774  177774  177774  177774  177774  177774  177774  177774
050:  177774  177774  177774  177774  177774  177774  177774  177774
060:  177774  177774  000000  066001  177777  000002  023401  177777


>M 1,2
LOGICAL SECTOR 1     *** BEGINNING OF DATA ***
SECTOR %00000220402        LDEV = %000004
002:  %000025,41
WRITTEN
>DUMP 1

LOGICAL SECTOR 1     *** BEGINNING OF DATA ***
SECTOR %00000220402        LDEV = %000004
000:  004600  000001  000041  000001  000002  001440  000000  177777
010:  000005  000000  000021  177777  000022  000004  000001  000001
020:  000000  000000  000000  000000  000000  000000  000000  000000
030:  000000  000000  000000  000000  000000  000374  000000  177774
040:  177774  177774  177774  177774  177774  177774  177774  177774
050:  177774  177774  177774  177774  177774  177774  177774  177774
060:  177774  177774  000000  066001  177777  000002  023401  177777
```

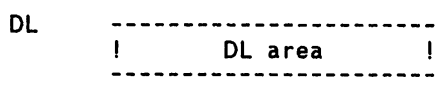
DB-relative data may be changed by modifying the image of the DB area in the object code. Knowing this, we now modify record 1 words %25 through %40 zero based to "MANAGER PUB SYS ". Note that the address for USERN is DB+25, the address for GROUPN is DB+31 and the address for ACCTN is DB+35.

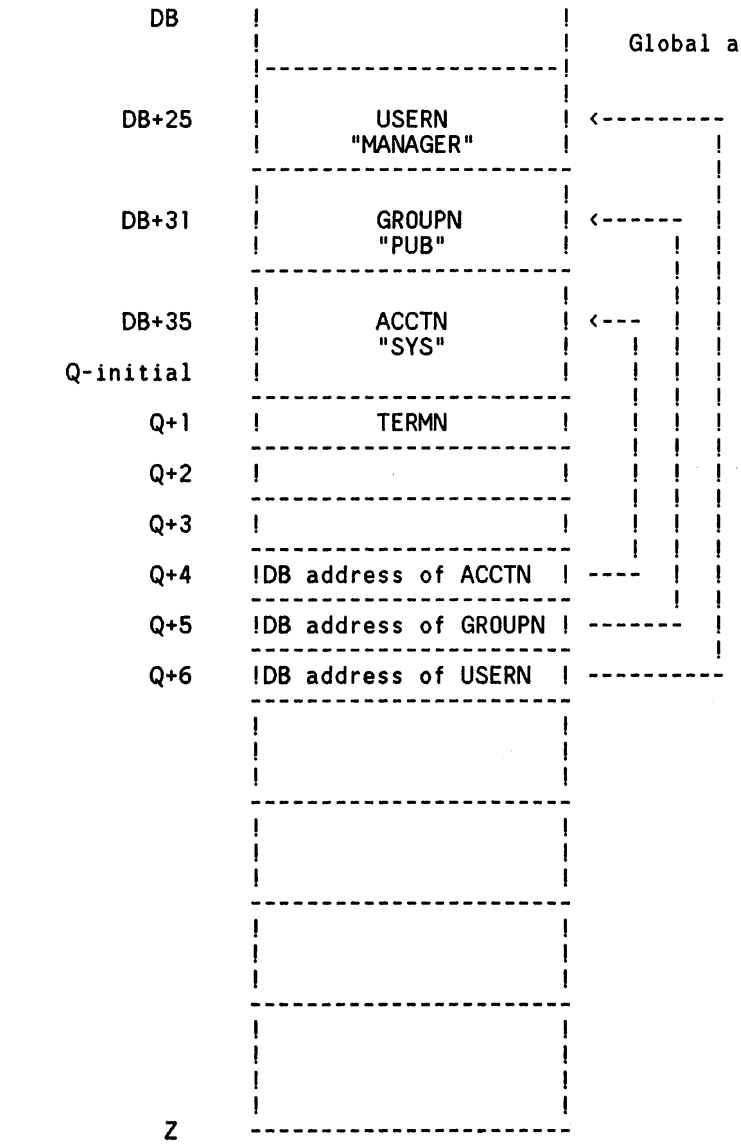
```
>DUMP 2

LOGICAL SECTOR 2
SECTOR %00000220403        LDEV = %000004
000:  000001  000000  000000  000000  000000  000000  000000  000000
010:  000000  000000  000000  000000  000000  000000  000000  000000
020:  000000  000000  003000  002400  177777  000000  000000  000000
030:  000000  000000  000000  000000  000000  000000  000000  000000
040:  000000  000000  000000  000000  000000  000000  000000  000000
050:  000000  000000  000000  000000  000000  000000  000000  000000
060:  000000  000000  000000  000000  000000  000000  000000  000000
070:  000000  000000  000000  000000  000000  000000  000000  000000


>M 2,%25,12
LOGICAL SECTOR 2
SECTOR %00000220403        LDEV = %000004
025:  %000000,'MA'
026:  %000000,'NA'
027:  %000000,'GE'
030:  %000000,'R '
031:  %000000,'PU'
032:  %000000,'B '
033:  %000000,'  '
034:  %000000,'  '
035:  %000000,'SY'
```

```
036:  %000000,'S '
037:  %000000,'  '
040:  %000000,'  '
WRITTEN
>EXIT

END OF PROGRAM

Running DECOMP, we can verify that our modification is correct.

:RUN DECOMP.LIB.SYS

HP3000 DECOMPILER 6.1

FILE NAME? XDEV
TYPE 'HELP' FOR ASSISTANCE.

-D DB+25
 DB+25   046501 047101 043505 051040 050125 041040 020040  MANAGER PUB
 DB+34   020040 051531 051440 020040 020040                   SYS
-EXIT
The contents of the run time stack for the original program is as
follows:
```




```
The contents of the run time stack for the patched program is to
be as follows:
```





Next we must find where the values of Q+4, Q+5 and Q+6 are set and patch the code to load the new DB+ values we have allocated instead of the ones generated by the compiler.

```
:RUN DECOMP.LIB.SYS

HP3000 DECOMPILER 6.1

FILE NAME? XDEV
TYPE 'HELP' FOR ASSISTANCE.

-
  0.21    035007  :.    ADDS  %7       <-- Primary Entry TYPESET
  0.22    171700  ..    LRA   S- 000
  0.23    010201  ..    LSL   1 BIT
  0.24    051404  S.    STOR  Q+ 004
  0.25    035004  :.    ADDS  %4
  0.26    171700  ..    LRA   S- 000
  0.27    010201  ..    LSL   1 BIT
  0.30    051405  S.    STOR  Q+ 005
  0.31    035004  :.    ADDS  %4
```

```
0.32    171700  ..    LRA   S- 000
0.33    010201  ..    LSL   1 BIT
0.34    051406  S.    STOR  Q+ 006
0.35    000706  ..    DZRO, ZERO
0.36    033405  7.    LLBL  TERMINATE
0.37    031006  2.    PCAL  F'CONTRAP
0.40    000706  ..    DZRO, ZERO
0.41    041406  C.    LOAD  Q+ 006
0.42    041405  C.    LOAD  Q+ 005
0.43    041404  C.    LOAD  Q+ 004
0.44    000600  ..    ZERO, NOP
0.45    171401  ..    LRA   Q+ 001
0.46    021035  ".    LDI   29
0.47    031003  2.    PCAL  WHO
```

Q+ relative memory is allocated when the program starts. By looking at instruction 0.21, we see 7 words allocated on top of stack. At 0.22 we see the address of S-0 stored at top of stack then logically shifted left 1 bit to form a byte address, then at 0.24 this value is stored at Q+4. Instructions 0.25 through 0.30 allocate 4 words and stores the byte address at Q+5. Instructions 0.31 through 0.34 similarly allocate 4 words and stores the byte address at Q+6. The instruction at 0.35 loads the final 3 words of array allocation onto the stack. Instruction 0.37 arms the CONTROL-Y trap to call TERMINATE if CONTROL-Y is pressed. As stated earlier, instructions 0.40 through 0.46 set up the parameters for the PCAL to the WHO intrinsic at 0.47.

Now that we know where the DB-relative addresses for the arrays are being set, we can patch the instructions for storing the starting addresses of the arrays in Q+4, Q+5 and Q+6 from the original addresses to the DB+ addresses we allocated.

```
-M 22
  0.22    171700  ..    LRA   S- 000          :=021035
  0.22    021035  ".    LDI   29
-M 26
  0.26    171700  ..    LRA   S- 000          :=021031
  0.26    021031  ".    LDI   25
-M 32
  0.32    171700  ..    LRA   S- 000          :=021025
  0.32    021025  ".    LDI   21
```

The instructions show LDI 29, LDI 25 and LDI 21. The numbers in these load immediate instructions are decimal. These numbers in octal are 35, 31 and 25 respectively, which can be seen by looking at the octal representation of the instruction.

We need to leave the adds to top of stack at 0.21, 0.25, 0.31 and 0.35 so the Q+ relative addressing will work. If we eliminated the adds, then some references could possibly be off by the number of words which were added to the stack if direct memory referencing is used.

Next we modify 0.40 to LDI 20 which loads a decimal 20 onto the stack, then instruction 0.41 to store the value in Q+1. Remember, the value of TERMN is stored at Q+1, so now the logon terminal is set to 20.

```
-M 40
  0.40    000706  ..    DZRO, ZERO           :=021024
  0.40    021024  ".    LDI   20
-M 41
  0.41    041406  C.    LOAD  Q+ 006         :=051401
  0.41    051401  S.    STOR  Q+ 001
```

Next we change instructions 0.42 through 0.47, the remainder of the instructions which set up the stack and then calls the WHO intrinsic, to NOP (no operation) instructions.

```
-M 42/47
  0.42    041405  C.    LOAD  Q+ 005                    :=0
  0.42    000000  ..    NOP , NOP
  0.43    041404  C.    LOAD  Q+ 004                    :=0
  0.43    000000  ..    NOP , NOP
  0.44    000600  ..    ZERO, NOP                       :=0
  0.44    000000  ..    NOP , NOP
  0.45    171401  ..    LRA   Q+ 001                    :=0
  0.45    000000  ..    NOP , NOP
  0.46    021035  ".    LDI   29                        :=0
  0.46    000000  ..    NOP , NOP
  0.47    031003  2.    PCAL  WHO                       :=0
  0.47    000000  ..    NOP , NOP
```

Looking at the code as patched we see:

```
-21

  0.21    035007  :.    ADDS  %7       <-- Primary Entry TYPESET
  0.22    021035  ".    LDI   29
  0.23    010201  ..    LSL   1 BIT
  0.24    051404  S.    STOR  Q+ 004
  0.25    035004  :.    ADDS  %4
  0.26    021031  ".    LDI   25
  0.27    010201  ..    LSL   1 BIT
  0.30    051405  S.    STOR  Q+ 005
  0.31    035004  :.    ADDS  %4
  0.32    021025  ".    LDI   21
  0.33    010201  ..    LSL   1 BIT
  0.34    051406  S.    STOR  Q+ 006
  0.35    000706  ..    DZRO, ZERO
  0.36    033405  7.    LLBL  TERMINATE
  0.37    031006  2.    PCAL  F'CONTRAP
  0.40    021024  ".    LDI   20
  0.41    051401  S.    STOR  Q+ 001
  0.42    000000  ..    NOP , NOP
  0.43    000000  ..    NOP , NOP
  0.44    000000  ..    NOP , NOP
  0.45    000000  ..    NOP , NOP
  0.46    000000  ..    NOP , NOP
  0.47    000000  ..    NOP , NOP
-EXIT

END OF PROGRAM
```

Now we run the program.

```
:RUN XDEV


USER:  MANAGER
ACCOUNT:  SYS
GROUP:  PUB
LOGON DEVICE:  20
Change to term type: ?10

END OF PROGRAM
:
```

It should be clear now that object code is not sacred. The instructions in the object code can be decoded and modified as necessary to correct a problem or have the program perform differently than intended. Modification of instructions and P-relative data is quite simple. The insertion of P-relative data and instructions is not shown since it is quite a bit more involved, but is possible.

After patching several programs, one begins to better understand the internals of the

HP 3000 and appreciate the work of a compiler writer. Hopefully, one can now see that object code can be changed just as source code to better accommodate individual needs not considered when the source code was written.

PCAL TERMINATE

Title: Systematic Redesign: Modifying Object Code

Author: Phil Curry Coordinator of Administrative Computing Alvin Community College 3110 Mustang Road Alvin, Texas 77511

Phone: (713) 331-6111

Biographical Sketch:

I have been employed at Alvin Community College since 1976. I am responsible for administrative computing which includes our student records, payroll, personnel, and accounting systems. I have worked with the HP 3000 since 1977 starting with the Series II. I'm a member of the board of directors for the Greater Houston Regional User's Group and am a member of the Contributed Software Library committee. I have made numerous contributions to the CSL including a tape library system, Super Star Trek, and a program which allows one to reset the runonly option in a BASIC program.
