U. S. DEPARTMENT OF THE INTERIOR

U.S. GEOLOGICAL SURVEY

Description of an Algorithm and Example Programs for Compression of Digital Files Containing Channelized Data

by

Robert E. Bracken[1]

Open-File Report 00-111

http://greenwood.cr.usgs.gov/pub/open-file-reports/ofr-00-0111

[1]USGS, MS 964, Federal Center, Denver, Colorado 80225

Table of Contents

Introduction

For the purpose of releasing data on a CD-ROM it became necessary to write programs for compressing and uncompressing large ASCII files containing channelized data. To fit the parameters of the data release, nearly 2 Gb (gigabytes) of data had to be stored in about 160 Mb (megabytes) of CD-ROM disk space. This required a compression factor of about 12 (ratio 12:1). An examination of general compression algorithms indicated that they would produce a compression factor of about 4 for these files, which was insufficient. Consequently, the author chose to write a compression algorithm (with its inverse), code the algorithm into standard Fortran 77, and port the resulting compression and uncompression programs to a variety of computer systems.

This report describes the algorithm, discusses how to modify the example programs for application to other data sets, and discusses possibilities for a generalized program derived from the algorithm. Included is the Fortran-77 source code derived for use in the data release described above. The compression and uncompression programs are named wis_cmpr.f and wis_uncm.f respectively. Digital copies of both the source code and the executable code are included on the CD-ROM (Bracken and Nicholson, 2000).

Overview of the Compression Algorithm

In general, the compression algorithm follows three basic steps. The first step is to read a record of the formatted data into a binary integer array. The second step utilizes the (typically) slowly varying nature of each data channel to reduce the magnitude of each value, preserving only the point-to-point variations. By reducing the magnitude in this way, the space required for each value is reduced significantly. The third step removes or reduces the number of leading zeros in each value by encoding it with a compression algorithm and packing it into an output array. The inverse simply undoes each step in reverse order.

Description of the Compression Algorithm

This description is given in reference to the basic algorithm from which the programs were written. Therefore, certain aspects may contain a greater level of generality than is found in the example programs (wis_cmpr.f and wis_uncm.f).

Step 1

Data File Description: The input data must be structured with uniformly-formatted records and channels. The algorithm itself does not impose any restrictions on numbers of records or channels or the definitions of the channels. A typical input file may have 10,000 records, each containing 5 channels. This file would then have 5 data functions, each with 10,000 discrete points. The first function might be a line identifier, the second a line direction, the third longitude, and so forth.

The data file may be formatted (ASCII) or unformatted (binary), and it may contain any variable types (ASCII, integer, floating point, hex, etc). But, all the values must be translatable to integers. Furthermore, once translated, the resulting integer values should be slowly varying within each channel. This allows the differencing in step 2 to reduce the magnitudes of most function values. Consequently, the best results can be obtained when the file is formatted (ASCII) and contains integer or fixed-point (f-floating) formats.

Operation: Each record is read and translated into an appropriate array of integers. For example, following is an input record and a reasonable translation:

```
 L723  N -93.77512 41.31234 51273.68 7821 (input record)
76723 78  -9377512  4131234  5127368 7821 (integer array)
```

The decimal values have been multiplied by the necessary amount to remove the decimal point. The ASCII variables have been translated to their decimal equivalent values. Many other scenarios are possible as long as they are invertible. The magnitudes of the values in the integer array should be kept below 536,936,725 (about 1/4 the maximum integer*4 value). The reason for this restriction will become apparent later, in the step-3 description.

After a record is translated, its integer array is passed to step 2 of the algorithm.

Step 2:

Step 2 is simple but results in a large contribution to the compression. When a new integer array (translated record) is passed to step 2, the previous array is subtracted and the result passed to step 3. Zero is subtracted if the new array is from the first record.

Step 3:

The Forward Process: The subtracted result from step 2 is passed into step 3. Then each subtracted value from the integer array is encoded and packed into an output array. The encoding works at the bit level and is designed to produce a small number of bits for small magnitude values with increasing numbers of bits necessary for larger magnitude values. The encoding for each value includes sign, magnitude, and length information. The maximum length is 4 bytes (32 bits). The left-most bit is the entry point into the encoded value. The following table outlines the encoding:

Table 1: Encoding parameters for compressing an integer value.

VALUE TO ENCODE	ENCODING	ENCODING LEN (BITS)
±(0)	00	2
±(1)	010.	4
±(2-9)	0110....	8
±(10-11)	011100..	8
±(12-43)	011101......	12
±(44-555)	011110..........	16
±(556-131627)	011111..................	24
±(131628-1073873451)	1...............................	32

In the "ENCODING" column, the zero and one bits are a code indicating the location and length of a subsequent binary number. The dots are placeholders for the subsequent binary number that is a translation of the “VALUE TO ENCODE”. For example, 0110 indicates that a value in the ±(2-9) range is to be translated into the 4 bits that follow, indicated in the table by 4 dots (….). The translated binary number is right justified within the 4 dots such that the least-significant bit occupies the position of the right-most dot. In the ±(2-9) range, the negative values of the range -9 to -2 are translated to the binary number by adding 9. The positive values of the range +2 to +9 are translated to the binary number by adding 6, as follows:

```
-9 to -2  ->  0 to  7  ( 0000 to 0111 )
 2 to  9  ->  8 to 15  ( 1000 to 1111 )
```

This procedure is used for all ranges that have placeholder dots, including ±(1). It is apparent that the largest magnitude number is 1,073,873,451. Therefore, in step 1 the magnitude should never be allowed to exceed 536,936,725 because:

$$| \pm 536{,}936{,}725 - \pm 536{,}936{,}725 | < 1{,}073{,}873{,}451.$$

The Inverse Process: To see why things are done in the forward process (encoding), it is helpful to understand the inverse process (decoding). Decoding is outlined in the following table. To use the table, start with the "`CHK BIT`" column and find the necessary bit number (bit 1 is entry point for decoding and is the left-most in the encoded number). Then select the "`BIT STATUS`", 0 or 1. Move to the "`GOTO BIT`" column and if a number is there, go back to the "`CHK BIT`" column and repeat the process with that bit number. If a number is not in the "`GOTO BIT`" column, move on to the right and get the encoding information (for decoding). Finally, use the encoding information to read the binary value and to translate it back to its original value.

Table 2 Decoding procedure for uncompressing an integer value.

CHK	BIT	GOTO	ENCODED-VALUE --------------		ENCODING
BIT	STATUS	BIT	RANGE	BIT LOC	LENGTH
1	0	2			
	1	->	±(131628-1073873451)	2-32	32
2	0		(0)	-	2
	1	3			
3	0	->	±(1)	4- 4	4
	1	4			
4	0	->	±(2-9)	5- 8	8
	1	5,6			
5,6	0,0	->	±(10-11)	7- 8	8
	0,1	->	±(12-43)	7-12	12
	1,0	->	±(44-555)	7-16	16
	1,1	->	±(556-131627)	7-24	24

Miscellaneous Bookkeeping: After a value has been encoded, its length is known and it can be easily packed into an output array with no extraneous bits between encoded values. In the example programs, units of 16 integer values are encoded, packed, and padded on the right to the nearest byte boundary. This is a necessary bookkeeping procedure, but the algorithm does not necessarily require 16-value processing units. The encoded 16-integer units are then written into a 64-byte output array. When the output array is full, it is written to the output file. Again, there is no particular requirement for a 64-byte array.

How to Modify Example Programs for Application to Other Data Sets

To modify the example programs for use with a different data set, first copy wis_cmpr.f and wis_uncm.f from the CD-ROM (Bracken and Nicholson, 2000). (Select among the provided operating systems to find the most appropriate source and "make" files.) In most cases, the changes will entirely involve the input format and the conversion of input values into the i4a array as described in the algorithm, step 1 (above). These changes should be made for the compression program and their inverse for the uncompression program.

If there are not enough channels to fill the 16 i4a elements, simply pad with zeros. (Zeros will compress to 2 bits, more compression than any other value.) If there are more than 16 channels, attempt to combine channels in order to reduce the total number down to 16. In this case, the best channels to combine are those that tend to remain constant for many records at a time, such as line numbers, dates, etc. If this is successful, the conversion is done.

In the compression program, these conversions will all take place in the following three sections:

```
C READ ONE LINE FROM THE INPUT FILE
C CONVERT 19 ASCII NBRS (17 CHANNELS) TO 16 INTEGER*4 VARIABLES
C PUT THE 16 VARIABLES INTO THE INTEGER*4 DIFFERENCING ARRAY
```

In the uncompression program, the conversions will take place in the following sections:

```
C PUT THE 1ST COLUMN OF THE DIFFERENCING ARRAY INTO 16 VARIABLES
C CONVERT 16 INTEGER*4 VARIABLES TO THE 17 VARIOUS FIELDS
C WRITE ONE RECORD TO THE OUTPUT FILE
```

If the input channels cannot be made to fit in 16 channels, the i4a array must be expanded to accommodate. Unfortunately, the sample programs were not written with this possibility in mind, and therefore the task is not a simple matter of changing a parameter. It is recommended that the size of the following arrays be multiplied by an appropriate integer: i4a, mash, iobuff, i4buff, and jbuff. They are found throughout the main programs, subroutines "mash" and "unmash", and subroutine "writeb" and "readb". When these arrays are changed, a number of miscellaneous bookkeeping indices must also be changed.

Possibilities for a Generalized Program

A generalized program, one that could compress (and uncompress) a variety of channelized data file types could be derived from the algorithm. The primary task would be to design a generalized front end from the algorithm step 1. From that point, steps 2 and 3 would be implemented with a variable-length integer array to accommodate various record lengths. The only significant restriction is that the data file must be organized as channelized data.

The front end would have to be capable of recognizing a data type and converting it to an appropriate array of integers. This could be as simple as having the operator input a format each time the program is executed. Or, it could become sophisicated to the point of recognizing file types and deriving formats internally. Even header information could be handled as an uncompressed attachment to a compressed file. Any of the systems could be set-up to include conversion of either formatted (ASCII) or unformatted (binary) input.

Finally, it may be possible to improve the encoding algorithm (step 3) to produce a greater compression. Tests have shown that once an ASCII file is compressed using this algorithm, an additional 30% compression can be obtained by using an existing general compression program. This indicates that the current algorithm has a small amount of wasted space.

References Cited

Bracken, R.E. and Nicholson, S.W., 2000, Aeromagnetic Surveying in Wisconsin, 1998-99: Digital Data Files: U.S. Geological Survey open-file report 99-527. CD-ROM.

APPENDIX A

Example Compression Program: wis_cmpr.f

```
C
C_______________________________________________________________
C
C     PROGRAM  W I S _ C M P R
C_______________________________________________________________
C
C PROGRAM WIS_CMPR COMPRESSES ASCII DATA FILES OF A SPECIFIC
C FORMAT, GENERATED BY HIGH-SENSE GEOPHYSICS LIMITED AS THE FINAL
C DATA FILES FOR THE WISCONSIN 1998/99 AEROMAGNETIC SURVEY.  THIS
C PROGRAM IS NOT A GENERAL COMPRESSION ALGORITHM, BUT RATHER
C UTILIZES SPECIFIC FORMATING KNOWLEDGE ABOUT THE FILES.  IT
C PARTICULARLY TAKES ADVANTAGE OF THE FACT THAT THE DATA ARE
C ARRANGED IN A SPECIFIC ASCII FORMAT AND THAT MOST DATA CHANNELS
C DO NOT CHANGE BY LARGE AMOUNTS FROM RECORD TO RECORD.
C
C THE ALGORITHM EMPLOYS THE FOLLOWING 3 STEP PROCESS:
C
C     1) CONVERT THE 17 ASCII-FORMATTED NUMBERS INTO 16 INTEGER*4
C        VARIABLES,
C     2) REDUCE THE ABSOLUTE VALUE OF EACH OF THE 16 INTEGER
C        VALUES BY FINDING THE DIFFERENCE BETWEEN IT AND
C        THE CORRESPONDING VALUE IN THE PREVIOUS RECORD,
C     3) COMPRESS THE DIFFERENCES USING THE FACT THAT SMALLER
C        ABSOLUTE VALUES REQUIRE LESS STORAGE SPACE.
C
C THE COMPRESSION ROUTINE USED IN STEP 3 IS NOT A GENERAL
C ROUTINE.  IT DOES NOT UTILIZE ANY PROPRIETARY FORMAT, NOR IS IT
C AS EFFICIENT AS IS MIGHT BE POSSIBLE.
C
C
C NOTES:
C
C 1) SOME COMPILERS SPECIFY RECORD LENGTHS IN 4-BYTE WORDS AND
C OTHER COMPILERS USE BYTES.  IF COMPILING THIS SOURCE CODE, THE
C OPEN STATEMENTS MAY NEED MODIFICATION.
C
C 2) THE ORIGINAL FILES USED THE CONVENTION THAT, WHERE
C APPLICABLE, A LEADING ZERO BEFORE THE DECIMAL WAS SUPPRESSED (
C FOR EXAMPLE -.73 INSTEAD OF THE CONVENTIONAL -0.73 ).
C THEREFORE, OBTAINING AN EXACT MATCH OF THE UNCOMPRESSED FILES
C TO THE ORIGINAL FILES MAY BE DEPENDENT UPON PROPER SELECTION OF
C COMPILER OPTIONS TO PRODUCE THE SUPPRESSED LEADING ZERO.
C
C 3) THE ORIGINAL FILES HAD 17 TRAILING PAD SPACES IN EACH ASCII
C RECORD.  THIS PROGRAM HAS INCLUDED THOSE PADS FOR COMPARISON
C AND CONSISTANCY PURPOSES.  BEYOND THAT, THEY SERVE NO PURPOSE
C AND CAN BE REMOVED IF DESIRED.
C
C 4) THE COMPRESSED FILES SHOULD ALWAYS BE PORTED IN A "BINARY"
C MODE.  THE UNCOMPRESSED OR ORIGINAL FILES CAN USUALLY BE PORTED
C IN EITHER "BINARY" OR "ASCII" MODE BUT TO AVOID CONFUSION
C SHOULD BE PORTED IN "ASCII" MODE.
C
C 5) IN ASCII FORMATTED FILES, SOME MACHINES USE A SINGLE NEWLINE
C CHARACTER (ASCII DECIMAL 10) TO DESIGNATE THE END OF A RECORD.
C OTHERS USE A CARRIAGE-RETURN (ASCII DECIMAL 13) AND NEWLINE
C CHARACTER.  IF THE ORIGINAL FILES ARE PORTED ACROSS PLATFORMS
```

```
C IN A "BINARY" MODE, THE UNCOMPRESSED FILES MAY BE A DIFFERENT
C LENGTH THAN THE ORIGINAL FILES DUE TO THE EXTRA CARRIAGE RETURN
C CHARACTER.
C
C
C PROGRAM WIS_CMPR WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991220.
C VERSION 1.1, 19991229. (I4CNV MAKES BYTE SWAPPING TRANSPARENT)
C
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
C IMPLIED.
C
C
      program wis_cmpr
C
C
C DECLARATIONS
C
C     I/O
      integer*4 itty,otty,idsk,odsk
      parameter(itty=5,otty=6,idsk=10,odsk=11)
      character*132 ifile,ofile
      integer*4 io0,ioutb,irec,noutb,nrec
      integer*4 nread
C
C     MISC INDEX VARIABLES
      integer*4 i,j
C
C     INPUT DATA
      integer*4 ifl
      character*2 adir
      real*8 dlond,dlatd,dutmxm,dutmym,dfid
      integer*4 iyd,iyd1,iyd2,iut,iut1,iut2
      real*8 drdrm,dbarm,dgpsm,ddiunt,drmgnt,ddmgnt,dimgnt,dlmgnt
C
C     FOUR-BYTE INTEGER CONVERTED VARIABLES
      integer*4 i4fld,i4lon,i4lat,i4utx,i4uty,i4fid,i4yjd,i4utc,
     &          i4rdr,i4bar,i4gps,i4diu,i4rmg,i4dmg,i4img,i4lmg
      integer*4 i4a(16,3)
C
C     COMPRESSION SUBROUTINE
      integer*4 nbytes
      byte mash(64)
C
C     OUTPUT BUFFER
      byte iobuff(128)
      integer*4 i4buff(32)
      equivalence(iobuff,i4buff)
C
C
C PROGRAM DESCRIPTION
C
      write(otty,801)
```

```
  801 format(//,' This program COMPRESSES ascii data files from',
     & ' the Wisconsin 98/99',/,' aeromag survey, flown by',
     & ' High-Sense Geophysics Limited.')
C
C
C GET NAME OF INPUT AND OUTPUT FILES
C
      write(otty,802)
  802 format(/,' Type name of input ascii file',/,' * ',$)
      read(itty,803) ifile
  803 format(a132)
C
      write(otty,804)
  804 format(' Type name of output compressed file',/,' * ',$)
      read(itty,803) ofile
C
C
C OPEN INPUT AND OUTPUT FILES
C
      open(idsk,file=ifile,status='old',form='formatted')
C
C     NOTE:  CERTAIN COMPILERS MAY USE 4-BYTE UNITS FOR RECL=
c      open(odsk,file=ofile,status='new',form='unformatted',
c     & access='direct',recl=16)
C
C     NOTE:  OTHER COMPILERS MAY USE 1-BYTE UNITS FOR RECL=
      open(odsk,file=ofile,status='new',form='unformatted',
     & access='direct',recl=64)
C
C
C INITIALIZATIONS
C
C     NUMBER OF INPUT RECORDS (143 BYTES PER RECORD)
      nread=0
C
C     DIFFERENCING ARRAY
      do j=1,3
        do i=1,16
          i4a(i,j)=0
        enddo
      enddo
C
C     THE OUTPUT BUFFER INDEX
      io0=0
C
C     CURRENT OUTPUT RECORD NUMBER
C     (RECORD 1 IS RESERVED FOR TOTAL NUMBER OF DATA BYTES)
      irec=1
C
C     CURRENT NUMBER OF DATA BYTES OUTPUT TO FILE
      ioutb=0
C
C
C READ ONE LINE FROM THE INPUT FILE
C
  207 read(idsk,805,end=106)
     & ifl,adir,dlond,dlatd,dutmxm,dutmym,dfid,iyd1,iyd2,iut1,
     & iut2,drdrm,dbarm,dgpsm,ddiunt,drmgnt,ddmgnt,dimgnt,dlmgnt
  805 format(i6,a2,2f10.4,2f10.1,f9.1,2i3,2i4,f8.2,2f7.1,
     & 5f10.2)
      nread=nread+1
C
C
C CONVERT 19 ASCII NBRS (17 CHANNELS) TO 16 INTEGER*4 VARIABLES
C
      i4fld=                ifl*10000
     &   +(ichar(adir(1:1))-32)*100
     &   +(ichar(adir(2:2))-32)
      i4lon=nint(dlond *1.d4)
      i4lat=nint(dlatd *1.d4)
      i4utx=nint(dutmxm*1.d1)
      i4uty=nint(dutmym*1.d1)
      i4fid=nint(dfid  *1.d1)
        iyd=iyd1*1000+iyd2
      i4yjd=     iyd
        iut=iut1*10000+iut2
      i4utc=     iut
      i4rdr=nint(drdrm *1.d2)
      i4bar=nint(dbarm *1.d1)
      i4gps=nint(dgpsm *1.d1)
      i4diu=nint(ddiunt*1.d2)
      i4rmg=nint(drmgnt*1.d2)
      i4dmg=nint(ddmgnt*1.d2)
      i4img=nint(dimgnt*1.d2)
      i4lmg=nint(dlmgnt*1.d2)
C
C
C PUT THE 16 VARIABLES INTO THE INTEGER*4 DIFFERENCING ARRAY
C
      i4a( 1,3)=i4fld
      i4a( 2,3)=i4lon
      i4a( 3,3)=i4lat
      i4a( 4,3)=i4utx
      i4a( 5,3)=i4uty
      i4a( 6,3)=i4fid
      i4a( 7,3)=i4yjd
      i4a( 8,3)=i4utc
      i4a( 9,3)=i4rdr
      i4a(10,3)=i4bar
      i4a(11,3)=i4gps
      i4a(12,3)=i4diu
      i4a(13,3)=i4rmg
      i4a(14,3)=i4dmg
      i4a(15,3)=i4img
      i4a(16,3)=i4lmg
C
C
C SHIFT ARRAY VARIABLES AND DIFFERENCE THEM
C
      do i=1,16
        i4a(i,1)=i4a(i,2)
        i4a(i,2)=i4a(i,3)
        i4a(i,3)=i4a(i,2)-i4a(i,1)
      enddo
```

```
C
C
C COMPRESS THE ARRAY INTO THE SHORTEST POSSIBLE BIT GROUP
C
      call masher(i4a(1,3),mash,nbytes,*913)
C
C
C LOAD MASH() INTO IO BUFFER AND WRITE WHEN NECESSARY
C
      do i=1,nbytes
        iobuff(io0+i)=mash(i)
      enddo
      io0=io0+(i-1)
C
C     CHECK WHETHER TO PERFORM A WRITE
      if(io0.ge.64) call writeb(odsk,iobuff,io0,irec,ioutb,*912)
      goto 207
C
C
C END OF INPUT FILE REACHED
C
C     FINISH WRITING TO OUTPUT FILE
  106 if(io0.gt.0) then
        call writeb(odsk,iobuff,io0,irec,ioutb,*912)
        goto 106
      endif
C
C     PUT NUMBER OF RECS AND DATA BYTES IN 1ST REC OF OUTPUT FILE
      nrec=irec
c      i4buff(1)=nrec
      call i4cnv(0, nrec,iobuff(1))
      noutb=ioutb
c      i4buff(2)=noutb
      call i4cnv(0,noutb,iobuff(5))
      io0=8
      irec=0
      call writeb(odsk,iobuff,io0,irec,ioutb,*912)
      goto 999
C
C
C EXIT PROCEDURE
C
C     io0 INDEX ERROR
  912 write(otty,811) io0
  811 format(//,' (MAIN)  ERROR.  io0 =',i12,' > 128',//)
      goto 999
C
C     NUMBER TOO LARGE ERROR
  913 write(otty,814)
  814 format(//,' (MAIN)  ERROR.  Number > 1,073,873,451',//)
      goto 999
C
  999 write(otty,808) nread,nread*143
  808 format(/,' Input: ',i12,' records',i12,' bytes')
      write(otty,809) nrec,nrec*64
  809 format(  ' Output:',i12,' records',i12,
     & ' bytes (w/overhead)')
```

```
      write(otty,810) dble(nrec)*6.4d3/1.43d2/nread,noutb
  810 format(  ' Ratio: ',f12.2,'%       ',i12,
     & ' bytes (compressed)',/)
C
      close(unit=idsk)
      close(unit=odsk)
      end
```

```
C
C_______________________________________________________________
C
C     SUBROUTINE  W R I T E B
C_______________________________________________________________
C
C SUBROUTINE WRITEB WRITES A BINARY RECORD TO A DIRECT ACCESS
C OUTPUT FILE.  THE FILE MUST BE OPENED BEFORE CALLING WRITEB.
C
C INPUT ARGUMENT:
C   ODSK   - INTEGER*4.  UNIT NUMBER OF A DIRECT ACCESS OUTPUT
C            FILE.  THE FILE MUST HAVE BEEN OPENED WITH A RECORD
C            LENGTH OF 64 BYTES BEFORE CALLING THIS SUBROUTINE.
C
C INPUT/OUTPUT ARGUMENTS:
C   IOBUFF - BYTE.  ARRAY OF LENGTH 128 BYTES CONTAINING THE DATA
C            TO BE OUTPUTTED.  THE FIRST 64 BYTES OF THE ARRAY
C            CONTENTS WILL BE OUTPUTTED.  ANY REMAINING ELEMENTS
C            WILL BE SHIFTED TO THE LEFT (TOWARD LOWER ELEMENT
C            NUMBERS) BY 64 BYTES.  IF THERE ARE LESS THAN 64
C            BYTES OF VALID DATA IN THE ARRAY, NOTHING WILL BE
C            SHIFTED.  HOWEVER THE VALID-DATA INDEX, IO0, WILL BE
C            ADJUSTED TO REFLECT THE OUTPUT.  (IN THE OUTPUT FILE
C            SHORT RECORDS WILL BE PADDED WITH ZEROS.)  SEE ALSO
C            ARGUMENT IO0.
C   IO0    - INTEGER*4.  THE VALID-DATA INDEX OF ARRAY IOBUFF.
C            VALID DATA ARE CONTAINED IN ALL ELEMENTS OF IOBUFF
C            TO THE LEFT OF IO0 INCLUSIVE.  ALL ELEMENTS TO THE
C            RIGHT OF IO0 EXCLUSIVE ARE NON-DATA AND MAY BE
C            MODIFIED BY THIS SUBROUTINE.  IF DURING THE CALL
C            IO0 IS GREATER THAN ZERO, A WRITE WILL BE PERFORMED,
C            IOBUFF WILL BE SHIFTED LEFT 64 BYTES, AND IO0 WILL
C            BE DECREASED 64.  IF AFTER THE SHIFT, IO0 BECOMES
C            LESS THAN ZERO, IT WILL BE INCREASED BACK TO ZERO.
C            IF DURING THE CALL, IO0 IS LESS THAN 1, A WRITE WILL
C            NOT BE PERFORMED AND NO OTHER ACTION WILL BE TAKEN.
C   IREC   - INTEGER*4.  THE CURRENT RECORD NUMBER OF THE OUTPUT
C            FILE.  IF DURING THE CALL, IO0 IS GREATER THAN ZERO,
C            IREC WILL BE INCREMENTED BEFORE WRITING.  IF IO0 IS
C            LESS THAN 1, IREC WILL NOT BE CHANGED AND A WRITE
C            WILL NOT BE PERFORMED.
C   IOUTB  - INTEGER*4.  A COUNTER TO TRACK THE NUMBER OF
C            VALID-DATA BYTES WRITTEN TO THE OUTPUT FILE.  IOUTB
C            WILL BE INCREASED BY AN AMOUNT EQUALLING THE NUMBER
C            OF VALID-DATA BYTES WRITTEN.  FOR EXAMPLE, IF DURING
C            THE CALL IOUTB=128, IREC=2, AND IO0=5, AFTER THE
C            CALL, IOUTB=133, IREC=3, AND IO0=0.
C
C RETURNS:
C   NORMAL - WHENEVER THE PRESCRIBED OPERATIONS ARE COMPLETED.
C   1ST    - ONLY IF DURING THE CALL, IO0 IS GREATER THAN 128.
C
C
C SUBROUTINE WRITEB WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991220.
C
```

```
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
C IMPLIED.
C
C
      subroutine writeb(odsk,iobuff,io0,irec,ioutb,*)
C
C
C DECLARATIONS
C
C     INPUT ARGUMENT
      integer*4 odsk
C
C     INPUT/OUTPUT ARGUMENTS
      byte iobuff(*)
      integer*4 io0,irec,ioutb
C
C     WRITING ARRAY
      byte jbuff(64)
C
C     MISC INDEX
      integer*4 i
C
C
C CHECK VALUES OF INDEX VARIABLES
C
      if(irec.lt.0) irec=0
      if(io0.lt.1) then
        io0=0
        goto 990
      endif
      if(io0.gt.128) goto 991
C
C
C PUT DATA INTO WRITING ARRAY AND COUNT NUMBER OF BYTES GOING OUT
C
      if(io0.ge.64) then
C
C       FULL RECORD
        do i=1,64
          jbuff(i)=iobuff(i)
        enddo
        ioutb=ioutb+64
      else
C
C       PARTIAL RECORD
        do i=1,io0
          jbuff(i)=iobuff(i)
        enddo
        do i=io0+1,64
          jbuff(i)=0
        enddo
        ioutb=ioutb+io0
      endif
C
```

```
C
C WRITE TO OUTPUT FILE
C
      irec=irec+1
      write(odsk,rec=irec) jbuff
C
C
C SLIDE IOBUFF() TO THE LEFT AND ADJUST io0 ACCORDINGLY
C
      do i=65,io0
        iobuff(i-64)=iobuff(i)
      enddo
      io0=io0-64
      if(io0.lt.0) io0=0
C
C
C EXIT PROCEDURE
C
  990 return
  991 return 1
      end
```

```
C
C_________________________________________________________________
C
C     SUBROUTINE  M A S H E R
C_________________________________________________________________
C
C SUBROUTINE MASHER COMPRESSES AN ARRAY OF 16 FOUR-BYTE INTEGERS.
C THE METHOD USED CODIFIES THE NUMBER OF BITS NECESSARY TO
C DESCRIBE THE VALUE AND THEN STORES BOTH THE CODE AND THE VALUE
C IN AN EQUAL OR SMALLER NUMBER OF BITS.  THE CODE IS DESIGNED SO
C THAT NO INFORMATION IS LOST.  SMALL VALUES ARE COMPRESSED WITH
C MUCH LESS OVERHEAD THAN LARGE NUMBERS.
C
C INPUT ARGUMENT:
C   I4A    - INTEGER*4.  16-ELEMENT ARRAY CONTAINING THE INTEGERS
C            TO BE COMPRESSED.  THE FULL RANGE OF A 4-BYTE
C            INTEGER CANNOT BE COMPRESSED BECAUSE 1 BIT IS
C            NECESSARY AS AN ENTRY POINT INTO THE CODE.
C            THEREFORE, THE MAXIMUM RANGE BECOMES PLUS OR MINUS
C            1,073,873,451 = 2^0+2^3+2^1+2^5+2^9+2^17+2^30.  THIS
C            IS SLIGHTLY MORE THAN 1/2 THE RANGE OF A FULL
C            32-BIT INTEGER.  THE REASON THE FULL 32-BIT RANGE
C            HAS NOT BEEN MADE AVAILABLE ARISES FROM
C            COMPLICATIONS SURROUNDING THE POTENTIAL FOR
C            EXPANSION OF A RECORD RATHER THAN COMPRESSION.
C
C OUTPUT ARGUMENTS:
C   MASH   - BYTE.  64-ELEMENT ARRAY CONTAINING THE COMPRESSED
C            CODE AND VALUES.  UNDER MOST CIRCUMSTANCES MASH WILL
C            NOT BE FULLY UTILIZED, BUT IT MUST BE 64 BYTES TO
C            HANDLE THE POSSIBILITY THAT NO COMPRESSION OCCURRED.
C   NBYTES - INTEGER*4.  THE ACTUAL NUMBER OF BYTES REQUIRED BY
C            THE COMPRESSED CODE AND NUMBERS.  THE CODE AND
C            NUMBERS ARE PADDED ON THE RIGHT TO FILL TO THE NEXT
C            BYTE BOUNDARY.
C
C RETURNS:
C   NORMAL - WHENEVER THE PRESCRIBED OPERATIONS ARE COMPLETED.
C   1ST    - ONLY IF DURING THE CALL, AN ELEMENT OF I4A() IS
C            FOUND TO BE OUTSIDE THE MAXIMUM RANGE
C
C
C SUBROUTINE MASHER WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991220.
C
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
C IMPLIED.
C
C
      subroutine masher(i4a,mash,nbytes,*)
C
C
C DECLARATIONS
```

```
C
C     INPUT ARGUMENT
      integer*4 i4a(16)
C
C     OUTPUT ARGUMENTS
      byte mash(64)
      integer*4 nbytes
C
C     BIT-HOLDING ARRAY
      byte bits(512)
C
C     THE NUMBER TO COMPRESS
      integer*4 n4
C
C     MISC INDICIES
      integer*4 i,j,jb,k,nbitsr
C
C
C CONVERT I4A() INTO COMPRESSED BIT VALUES AND PUT EACH BIT VALUE
C IN EACH CORROSPONDING ELEMENT OF BITS()
C
      jb=0
      do k=1,16
        n4=i4a(k)
C
        if(         n4 .eq.         0) then
          bits(jb+1)=0
          bits(jb+2)=0
          jb=jb+2
        else if(abs(n4).eq.         1) then
          bits(jb+1)=0
          bits(jb+2)=1
          bits(jb+3)=0
          jb=jb+4
          if(n4.lt.0) then
            bits(jb)=0
          else
            bits(jb)=1
          endif
        else if(abs(n4).le.         9) then
          bits(jb+1)=0
          bits(jb+2)=1
          bits(jb+3)=1
          bits(jb+4)=0
          jb=jb+4
          if(n4.lt.0) then
            n4=n4+9
          else
            n4=n4+6
          endif
          do i=3,0,-1
            jb=jb+1
            if(btest(n4,i)) then
              bits(jb)=1
            else
              bits(jb)=0
            endif
```

```
  enddo
else if(abs(n4).le.        11) then
  bits(jb+1)=0
  bits(jb+2)=1
  bits(jb+3)=1
  bits(jb+4)=1
  bits(jb+5)=0
  bits(jb+6)=0
  jb=jb+6
  if(n4.lt.0) then
    n4=n4+11
  else
    n4=n4-8
  endif
  do i=1,0,-1
    jb=jb+1
    if(btest(n4,i)) then
      bits(jb)=1
    else
      bits(jb)=0
    endif
  enddo
else if(abs(n4).le.        43) then
  bits(jb+1)=0
  bits(jb+2)=1
  bits(jb+3)=1
  bits(jb+4)=1
  bits(jb+5)=0
  bits(jb+6)=1
  jb=jb+6
  if(n4.lt.0) then
    n4=n4+43
  else
    n4=n4+20
  endif
  do i=5,0,-1
    jb=jb+1
    if(btest(n4,i)) then
      bits(jb)=1
    else
      bits(jb)=0
    endif
  enddo
else if(abs(n4).le.       555) then
  bits(jb+1)=0
  bits(jb+2)=1
  bits(jb+3)=1
  bits(jb+4)=1
  bits(jb+5)=1
  bits(jb+6)=0
  jb=jb+6
  if(n4.lt.0) then
    n4=n4+555
  else
    n4=n4+468
  endif
  do i=9,0,-1
```

```
            jb=jb+1
            if(btest(n4,i)) then
              bits(jb)=1
            else
              bits(jb)=0
            endif
          enddo
        else if(abs(n4).le.    131627) then
          bits(jb+1)=0
          bits(jb+2)=1
          bits(jb+3)=1
          bits(jb+4)=1
          bits(jb+5)=1
          bits(jb+6)=1
          jb=jb+6
          if(n4.lt.0) then
            n4=n4+131627
          else
            n4=n4+130516
          endif
          do i=17,0,-1
            jb=jb+1
            if(btest(n4,i)) then
              bits(jb)=1
            else
              bits(jb)=0
            endif
          enddo
        else if(abs(n4).le.1073873451) then
          bits(jb+1)=1
          jb=jb+1
          if(n4.lt.0) then
            n4=n4+1073873451
          else
            n4=n4+1073610196
          endif
          do i=30,0,-1
            jb=jb+1
            if(btest(n4,i)) then
              bits(jb)=1
            else
              bits(jb)=0
            endif
          enddo
        else
          goto 991
        endif
      enddo
C
C
C COMPRESS BITS() INTO MASH()
C
      nbytes=(jb-1)/8+1
      nbitsr=8*nbytes-jb
C
      j=0
      do k=1,nbytes-1
```

```
          mash(k)=0
          do i=7,0,-1
            j=j+1
            if(bits(j).ne.0) mash(k)=ior(mash(k),ishft(1,i))
          enddo
        enddo
C
        mash(k)=0
        do i=7,nbitsr,-1
          j=j+1
          if(bits(j).ne.0) mash(k)=ior(mash(k),ishft(1,i))
        enddo
C
C
C EXIT PROCEDURE
C
  990 return
  991 return 1
      end
```

```
C
C_________________________________________________________________
C
C     SUBROUTINE  I 4 C N V
C_________________________________________________________________
C
C SUBROUTINE I4CNV PUTS AN INTEGER*4 VALUE INTO A 4-ELEMENT BYTE
C ARRAY SUCH THAT THE FIRST ELEMENT IS THE MOST-SIGNIFICANT BYTE
C AND THE 4TH ELEMENT IS THE LEAST-SIGNIFICANT BYTE.  IT ALSO
C PERFORMS THE INVERSE OPERATION.  THE RESULT IS THE SAME AS IF
C AN EQUIVALENCE BETWEEN THE NUMBER AND THE ARRAY WAS PERFORMED
C ON A UNIX-LIKE MACHINE.  HOWEVER, IF THE EQUIVALENCE WAS
C PERFORMED ON A VAX-LIKE MACHINE, THE RESULT WOULD BE A REVERSAL
C OF THE BYTES.  THIS SUBROUTINE IS DESIGNED TO CIRCUMVENT THIS
C DISPARITY, SO THAT INTEGER*4 VARIABLES CAN BE WRITTEN TO AND
C READ FROM A FILE WITHOUT HAVING BYTE REVERSAL PROBLEMS AND
C WITHOUT NECESSITATING USE OF COMPILER SWITCHES TO FIX THE
C PROBLEMS.
C
C INPUT ARGUMENT:
C   NVERSE - INTEGER*4.  IF NVERSE IS SET TO 0, THE FORWARD
C            OPERATION WILL BE PERFORMED SUCH THAT THE INTEGER*4
C            VARIABLE WILL BE PUT INTO THE BYTE ARRAY.  IF NVERSE
C            IS SET TO 1, THE INVERSE OPERATION WILL BE PERFORMED
C            SUCH THAT THE BYTE ARRAY WILL BE PUT BACK INTO THE
C            INTEGER*4 VARIABLE.
C
C INPUT/OUTPUT ARGUMENTS:
C   I4A    - INTEGER*4.  THE INTEGER*4 VARIABLE TO BE CONVERTED.
C            IF NVERSE IS 0, I4A IS AN INPUT ARGUMENT.  IF NVERSE
C            IS 1, I4A IS AN OUTPUT ARGUMENT.
C   K1B    - BYTE.  THE 4-ELEMENT BYTE ARRAY TO BE CONVERTED.
C            IF NVERSE IS 0, I4A IS AN OUTPUT ARGUMENT.  IF
C            NVERSE IS 1, I4A IS AN INPUT ARGUMENT.
C
C
C SUBROUTINE I4CNV WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991229.
C
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
C IMPLIED.
C
C
      subroutine i4cnv(nverse, i4a,k1b)
C
C DECLARATIONS
C
C     INPUT ARGUMENT
      integer*4 nverse
C
C     INPUT/OUTPUT ARGUMENTS
      integer*4 i4a
      byte k1b(4)
```

```
C
C     INTERNAL VARIABLES
      integer*4 i0,j0,k0,k
      byte ksgn
      integer*4 jsgn
C
C
C DECIDE WHETHER TO PERFORM FORWARD OR INVERSE OPERATION
C
      if(nverse.ge.1) goto 101
C
C
C PERFORM THE FORWARD OPERATION I4A -> K1B
C (ASSUMES THAT ALL INTEGERS ARE 4-BYTE TWOS COMPLEMENT,
C AND THE HIGH-BIT IS THE SIGN BIT)
C
C     INITIALIZE THE RUNNING VALUE AND THE SIGN BIT VALUE
      j0=i4a
      ksgn=0
C
C     CHECK FOR THE RUNNING VALUE LESS THAN ZERO
      if(j0.lt.0) then
C
C       CLEAR ONLY THE SIGN BIT
        j0=j0+2**30
        j0=j0+2**30
C
C       REMEMBER THE SIGN BIT FOR LATER APPLICATION TO THE MSB
        ksgn=64
      endif
C
C     CALCULATE THE BIT PATTERN FOR EACH OF THE 4 BYTES
      do k=4,2,-1
        i0=j0
        j0=i0/256
        k0=i0-j0*256
        if(k0.gt.127) k0=k0-256
        k1b(k)=k0
      enddo
      k1b(k)=j0
C
C     RESTORE THE SIGN BIT IN THE MOST SIGNIFICANT BYTE
      k1b(k)=k1b(k)-ksgn
      k1b(k)=k1b(k)-ksgn
      goto 990
C
C
C PERFORM THE INVERSE OPERATION K1B -> I4A
C
C     INITIALIZE THE RUNNING VALUE AND THE SIGN BIT VALUE
  101 i4a=k1b(1)
      jsgn=0
C
C     CHECK FOR THE RUNNING VALUE LESS THAN ZERO
      if(i4a.lt.0) then
C
C       CLEAR ONLY THE SIGN BIT
```

```
          i4a=i4a+128
C
C         REMEMBER THE SIGN BIT FOR LATER APPLICATION TO THE MSB
          jsgn=2**30
        endif
C
C       RESTORE THE BIT PATTERN FOR EACH OF THE 4 BYTES
        do k=2,4
          k0=k1b(k)
          if(k0.lt.0) k0=k0+256
          i4a=i4a*256+k0
        enddo
C
C       RESTORE THE SIGN BIT
        i4a=i4a-jsgn
        i4a=i4a-jsgn
C
C
C EXIT PROCEDURE
C
  990 return
        end
```

APPENDIX B

Example Uncompression Program: wis_uncm.f

```
C
C_______________________________________________________________
C
C     PROGRAM  W I S _ U N C M
C_______________________________________________________________
C
C PROGRAM WIS_UNCM PERFORMS THE INVERSE OPERATION OF WIS_CMPR.
C THE ORIGINAL FILES ARE ASCII DATA FILES GENERATED BY HIGH-SENSE
C GEOPHYSICS LIMITED AS THE FINAL DATA FILES FOR THE WISCONSIN
C 1998/99 AEROMAGNETIC SURVEY.
C
C FOR A BRIEF DESCRIPTION OF THE COMPRESSION, SEE THE SOURCE CODE
C FOR PROGRAM WIS_CMPR.  THE UNCOMPRESSION (THIS PROGRAM) IS ITS
C EXACT INVERSE AND SHOULD PRODUCE FILES IDENTICAL TO THE
C ORIGINAL INPUT FILES.
C
C NOTES:
C
C 1) SOME COMPILERS SPECIFY RECORD LENGTHS IN 4-BYTE WORDS AND
C OTHER COMPILERS USE BYTES.  IF COMPILING THIS SOURCE CODE, THE
C OPEN STATEMENTS MAY NEED MODIFICATION.
C
C 2) THE ORIGINAL FILES USED THE CONVENTION THAT, WHERE
C APPLICABLE, A LEADING ZERO BEFORE THE DECIMAL WAS SUPPRESSED (
C FOR EXAMPLE -.73 INSTEAD OF THE CONVENTIONAL -0.73 ).
C THEREFORE, OBTAINING AN EXACT MATCH OF THE UNCOMPRESSED FILES
C TO THE ORIGINAL FILES MAY BE DEPENDENT UPON PROPER SELECTION OF
C COMPILER OPTIONS TO PRODUCE THE SUPPRESSED LEADING ZERO.
C
C 3) THE ORIGINAL FILES HAD 17 TRAILING PAD SPACES IN EACH ASCII
C RECORD.  THIS PROGRAM HAS INCLUDED THOSE PADS FOR COMPARISON
C AND CONSISTANCY PURPOSES.  BEYOND THAT, THEY SERVE NO PURPOSE
C AND CAN BE REMOVED IF DESIRED.
C
C 4) THE COMPRESSED FILES SHOULD ALWAYS BE PORTED IN A "BINARY"
C MODE.  THE UNCOMPRESSED OR ORIGINAL FILES CAN USUALLY BE PORTED
C IN EITHER "BINARY" OR "ASCII" MODE BUT TO AVOID CONFUSION
C SHOULD BE PORTED IN "ASCII" MODE.
C
C 5) IN ASCII FORMATTED FILES, SOME MACHINES USE A SINGLE NEWLINE
C CHARACTER (ASCII DECIMAL 10) TO DESIGNATE THE END OF A RECORD.
C OTHERS USE A CARRIAGE-RETURN (ASCII DECIMAL 13) AND NEWLINE
C CHARACTER.  IF THE ORIGINAL FILES ARE PORTED ACROSS PLATFORMS
C IN A "BINARY" MODE, THE UNCOMPRESSED FILES MAY BE A DIFFERENT
C LENGTH THAN THE ORIGINAL FILES DUE TO THE EXTRA CARRIAGE RETURN
C CHARACTER.
C
C
C PROGRAM WIS_UNCM WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991220.
C VERSION 1.1, 19991229. (I4CNV MAKES BYTE SWAPPING TRANSPARENT)
C
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
```

```
C IMPLIED.
C
C
      program wis_uncm
C
C
C DECLARATIONS
C
C     I/O
      integer*4 itty,otty,idsk,odsk
      parameter(itty=5,otty=6,idsk=10,odsk=11)
      character*132 ifile,ofile
C
C     INPUT DATA
      integer*4 ifl,idir1,idir2
      character*2 adir
      real*8 dlond,dlatd,dutmxm,dutmym,dfid
      integer*4 iyd,iyd1,iyd2,iut,iut1,iut2
      real*8 drdrm,dbarm,dgpsm,ddiunt,drmgnt,ddmgnt,dimgnt,dlmgnt
C
C     I*4 CONVERTED VARIABLES
      integer*4 i4fld,i4lon,i4lat,i4utx,i4uty,i4fid,i4yjd,i4utc,
     &          i4rdr,i4bar,i4gps,i4diu,i4rmg,i4dmg,i4img,i4lmg
      integer*4 i4a(16,3)
C
C     COMPRESSION SUBROUTINE
      integer*4 nbytes
C
C     OUTPUT BUFFER
      byte iobuff(128)
      integer*4 i4buff(32)
      equivalence(iobuff,i4buff)
C
C     MISC COUNTERS AND INDICIES
      integer*4 i,iinb,io0,irec,j,ninb,nrec,nwrite
C
C
C PROGRAM DESCRIPTION
C
      write(otty,801)
  801 format(//,' This program UNCOMPRESSES ascii data files',
     & ' from the Wisconsin',/,' 98/99 aeromag survey, flown by',
     & ' High-Sense Geophysics Limited.')
C
C
C GET NAME OF INPUT AND OUTPUT FILES
C
      write(otty,802)
  802 format(/,' Type name of input compressed file',/,' * ',$)
      read(itty,803) ifile
  803 format(a132)
C
      write(otty,804)
  804 format(' Type name of output ascii file',/,' * ',$)
      read(itty,803) ofile
C
C
```

```
C OPEN INPUT AND OUTPUT FILES
C
C
      open(odsk,file=ofile,status='new',form='formatted')
C
C     NOTE:  CERTAIN COMPILERS MAY USE 4-BYTE UNITS FOR RECL=
c      open(idsk,file=ifile,status='old',form='unformatted',
c     & access='direct',recl=16)
C
C     NOTE:  OTHER COMPILERS MAY USE 1-BYTE UNITS FOR RECL=
      open(idsk,file=ifile,status='old',form='unformatted',
     & access='direct',recl=64)
C
C
C INITIALIZATIONS
C
C     TOT NBR OF RECS AND DATA BYTES FROM 1ST REC OF INPUT FILE
      io0=0
      irec=0
      iinb=0
      call readb(idsk,iobuff,io0,irec,iinb,*912)
c      nrec=i4buff(1)
      call i4cnv(1, nrec,iobuff(1))
c      ninb=i4buff(2)
      call i4cnv(1, ninb,iobuff(5))
C
C     INPUT BUFFER INDEX OF BYTES CURRENTLY IN BUFFER
      io0=0
C
C     CURRENT INPUT RECORD NUMBER
C     (RECORD 1 IS RESERVED FOR TOTAL NUMBER OF DATA BYTES)
      irec=1
C
C     TOT NBR OF INPUT DATA BYTES CURRENTLY HAVING BEEN UTILIZED
      iinb=0
C
C     I4 DIFFERENCING ARRAY
      do j=1,3
        do i=1,16
          i4a(i,j)=0
        enddo
      enddo
C
C     NUMBER OF OUTPUT RECORDS (143 BYTES PER RECORD + 17 PADS)
      nwrite=0
C
C
C LOAD IOBUFF() FOR UNCOMPRESSING
C
  207 call readb(idsk,iobuff,io0,irec,iinb,*912)
C
C
C UNCOMPRESS ONE RECORD FROM IOBUFF() INTO I4A()
C
      call unmash(i4a(1,3),iobuff,nbytes,*913)
      iinb=iinb+nbytes
C
```

```
C
C ADD ARRAY VARIABLES AND SHIFT THEM
C
      do i=1,16
        i4a(i,2)=i4a(i,1)+i4a(i,3)
        i4a(i,1)=i4a(i,2)
        i4a(i,2)=i4a(i,3)
      enddo
C
C
C PUT THE 1ST COLUMN OF THE DIFFERENCING ARRAY INTO 16 VARIABLES
C
      i4fld=i4a( 1,1)
      i4lon=i4a( 2,1)
      i4lat=i4a( 3,1)
      i4utx=i4a( 4,1)
      i4uty=i4a( 5,1)
      i4fid=i4a( 6,1)
      i4yjd=i4a( 7,1)
      i4utc=i4a( 8,1)
      i4rdr=i4a( 9,1)
      i4bar=i4a(10,1)
      i4gps=i4a(11,1)
      i4diu=i4a(12,1)
      i4rmg=i4a(13,1)
      i4dmg=i4a(14,1)
      i4img=i4a(15,1)
      i4lmg=i4a(16,1)
C
C
C CONVERT 16 INTEGER*4 VARIABLES TO THE 17 VARIOUS FIELDS
C
      ifl=i4fld/10000
      idir1=(i4fld-ifl*10000)/100
      idir2=i4fld-ifl*10000-idir1*100
C
      adir(1:1)=char(idir1+32)
      adir(2:2)=char(idir2+32)
      dlond =dfloat(i4lon)*1.d-4
      dlatd =dfloat(i4lat)*1.d-4
      dutmxm=dfloat(i4utx)*1.d-1
      dutmym=dfloat(i4uty)*1.d-1
      dfid  =dfloat(i4fid)*1.d-1
      iyd   =       i4yjd
          iyd1=iyd/1000
          iyd2=iyd-iyd1*1000
      iut   =       i4utc
          iut1=iut/10000
          iut2=iut-iut1*10000
      drdrm =dfloat(i4rdr)*1.d-2
      dbarm =dfloat(i4bar)*1.d-1
      dgpsm =dfloat(i4gps)*1.d-1
      ddiunt=dfloat(i4diu)*1.d-2
      drmgnt=dfloat(i4rmg)*1.d-2
      ddmgnt=dfloat(i4dmg)*1.d-2
      dimgnt=dfloat(i4img)*1.d-2
      dlmgnt=dfloat(i4lmg)*1.d-2
```

```
C
C
C WRITE ONE RECORD TO THE OUTPUT FILE
C
      write(odsk,805)
     & ifl,adir,dlond,dlatd,dutmxm,dutmym,dfid,iyd1,iyd2,iut1,
     & iut2,drdrm,dbarm,dgpsm,ddiunt,drmgnt,ddmgnt,dimgnt,dlmgnt
  805 format(i6,a2,2f10.4,2f10.1,f9.1,2i3,i4,i4.3,f8.2,2f7.1,
     & 5f10.2,'                 ')
      nwrite=nwrite+1
      if(iinb.ge.ninb) goto 999
      goto 207
C
C
C EXIT PROCEDURE
C
C     io0 INDEX ERROR
  912 write(otty,811) io0
  811 format(//,' (MAIN)  ERROR.  io0 =',i12,' > 128',//)
      goto 999
C
C     NUMBER TOO LARGE ERROR
  913 write(otty,814)
  814 format(//,' (MAIN)  ERROR.  Number > 1,073,873,451',//)
      goto 999
C
  999 write(otty,808) nwrite,nwrite*143
  808 format(/,' Output:',i12,' records',i12,' bytes')
      if(irec.gt.nrec) irec=nrec
      write(otty,809) irec,irec*64
  809 format(  ' Input: ',i12,' records',i12,
     & ' bytes (w/overhead)')
      write(otty,810) dfloat(nwrite)*1.43d4/6.4d1/irec,ninb
  810 format(  ' Ratio: ',f12.2,'%       ',i12,
     & ' bytes (compressed)',/)
C
      close(unit=idsk)
      close(unit=odsk)
      end
```

```
C
C_______________________________________________________________
C
C     SUBROUTINE  R E A D B
C_______________________________________________________________
C
C SUBROUTINE READB READS A BINARY RECORD FROM A DIRECT ACCESS
C INPUT FILE.  THE FILE MUST BE OPENED BEFORE CALLING READB.
C
C INPUT ARGUMENT:
C   IDSK   - INTEGER*4.  UNIT NUMBER OF A DIRECT ACCESS INPUT
C            FILE.  THE FILE MUST HAVE BEEN OPENED WITH A RECORD
C            LENGTH OF 64 BYTES BEFORE CALLING THIS SUBROUTINE.
C
C INPUT/OUTPUT ARGUMENTS:
C   IOBUFF - BYTE.  ARRAY OF LENGTH 128 BYTES CONTAINING THE DATA
C            WHICH HAS BEEN READ FROM THE INPUT FILE.  BEFORE
C            READING NEW DATA, THE DATA IN IOBUFF WILL BE LEFT-
C            SHIFTED TO DROP THE BYTES THAT HAVE ALREADY BEEN
C            UTILIZED IN THE CALLING PROGRAM.  THE NUMBER OF
C            BYTES TO DROP IS CALCULATED FROM THE INPUT VALUES OF
C            IINB, IREC, AND IO0.  ONCE SHIFTED, A 64-BYTE RECORD
C            IS READ INTO THE ELEMENTS TO THE RIGHT OF THE
C            ADJUSTED IO0.  IF THE 128-BYTE IOBUFF ARRAY DOES NOT
C            HAVE ENOUGH ADDITIONAL SPACE FOR A 64-BYTE RECORD,
C            THE READ IS NOT PERFORMED.  ALSO, IF A READ ERROR IS
C            ENCOUNTED (AS MIGHT OCCUR AT THE END OF THE FILE),
C            THE READ IS NOT PERFORMED.
C   IO0    - INTEGER*4.  THE VALID-DATA INDEX OF ARRAY IOBUFF.
C            VALID DATA ARE CONTAINED IN ALL ELEMENTS OF IOBUFF
C            TO THE LEFT OF IO0 INCLUSIVE.  ALL ELEMENTS TO THE
C            RIGHT OF IO0 EXCLUSIVE ARE NON-DATA AND MAY BE
C            MODIFIED BY THIS SUBROUTINE.  IO0 WILL BE MODIFIED
C            DURING THE CALL TO REFLECT THE AMOUNT OF LEFT SHIFT
C            AS DESCRIBED ABOVE IN ARGUMENT IOBUFF.  THE VALUE OF
C            IO0 DURING THE CALL IS USED IN THE CALCULATION FOR
C            DETERMINING THE AMOUNT OF LEFT SHIFT.  IF, AFTER THE
C            LEFT SHIFT, IO0 IS GREATER THAN 63, A NEW RECORD
C            WILL NOT BE READ IN.  IF HOWEVER, A NEW RECORD IS
C            READ IN, IO0 WILL BE MODIFIED AGAIN TO REFLECT THE
C            RIGHT END OF THE NEW DATA.
C   IREC   - INTEGER*4.  THE CURRENT RECORD NUMBER OF THE INPUT
C            FILE.  IF A READ IS TO BE MADE, IREC WILL BE
C            INCREMENTED BEFORE READING.  IF IO0 IS GREATER THAN
C            63, IREC WILL NOT BE CHANGED AND A READ WILL NOT BE
C            PERFORMED.  THE VALUE OF IREC DURING THE CALL IS
C            USED IN THE CALCULATION FOR DETERMINING THE AMOUNT
C            OF LEFT SHIFT OF ARRAY IOBUFF.
C
C INPUT ARGUMENT:
C   IINB   - INTEGER*4.  THE TOTAL NUMBER OF BYTES UTILIZED BY
C            THE CALLING PROGRAM BEFORE CALLING THIS SUBROUTINE.
C            TYPICALLY, IINB WILL BE SLIGHTLY SMALLER THAN THE
C            TOTAL NUMBER OF BYTES READ IN.  AN APPROPRIATE
C            CALCULATION USING IREC, IO0, AND IINB GIVES THE
C            NUMBER OF BYTES AT THE LEFT END OF IOBUFF THAT HAVE
C            ALREADY BEEN UTILIZED BY THE CALLING PROGRAM.  FOR
```

```
C             EXAMPLE IF THREE 64-BYTE RECORDS HAVE BEEN READ IN,
C             IO0=70, AND IINB=150, THEN THE CONTENTS OF IOBUFF
C             WILL BE SHIFTED LEFT 28 BYTES AND THE VALUE OF IO0
C             DECREASED TO 42.  THIS MAKES ROOM FOR A NEW 64-BYTE
C             RECORD TO BE READ IN AND IO0 WILL THEN BE INCREASED
C             TO 42+64=106.  IINB IS NOT CHANGED IN THIS
C             SUBROUTINE.  HOWEVER, THE NUMBER OF BYTES THAT ARE
C             UTILIZED OFF OF THE LEFT END OF IOBUFF BY THE
C             CALLING PROGRAM AFTER THE CALL TO THIS SUBROUTINE
C             MUST BE ADDED TO IINB BY THE CALLING PROGRAM BEFORE
C             AGAIN CALLING THIS SUBROUTINE.
C
C RETURNS:
C   NORMAL - WHENEVER THE PRESCRIBED OPERATIONS ARE COMPLETED.
C   1ST    - ONLY IF DURING THE CALL, IO0 IS GREATER THAN 128.
C
C
C SUBROUTINE READB WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991220.
C
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
C IMPLIED.
C
C
      subroutine readb(idsk,iobuff,io0,irec,iinb,*)
C
C
C DECLARATIONS
C
C     INPUT ARGUMENT
      integer*4 idsk
C
C     INPUT/OUTPUT ARGUMENTS
      byte iobuff(*)
      integer*4 io0,irec
C
C     INPUT ARGUMENT
      integer*4 iinb
C
C     READING ARRAY
      byte jbuff(64)
C
C     MISC INDICIES
      integer*4 i,io1
C
C
C CHECK VALUES OF INDEX VARIABLES
C
      if(io0.lt.1) io0=0
      if(io0.gt.128) goto 991
      if(iinb.lt.0) iinb=0
      if(irec.lt.0) irec=0
C
```

```
C
C SLIDE IOBUFF() TO THE LEFT AND ADJUST io0 ACCORDINGLY
C
      io1=io0-((irec-1)*64-iinb)
      do i=io1+1,io0
        iobuff(i-io1)=iobuff(i)
      enddo
      io0=io0-io1
      if(io0.lt.0) io0=0
      if(io0.gt.128) goto 991
C
C
C READ FROM INPUT FILE ONLY IF THERE IS ENOUGH ROOM IN IOBUFF()
C
      if(io0.ge.64) goto 990
      irec=irec+1
      read(idsk,rec=irec,err=901) jbuff
C
C
C PUT DATA INTO READING ARRAY
C
      do i=1,64
        iobuff(io0+i)=jbuff(i)
      enddo
      io0=io0+64
      if(io0.gt.128) goto 991
      goto 990
C
C
C EXIT PROCEDURE
C
  901 irec=irec-1
  990 return
  991 return 1
      end
```

```
C
C________________________________________________________________
C
C     SUBROUTINE  U N M A S H
C________________________________________________________________
C
C SUBROUTINE UNMASH UNCOMPRESSES AN ARRAY OF VARIABLE LENGTH INTO
C 16 FOUR-BYTE INTEGERS.  THE METHOD USED UNCODIFIES THE BITS
C THAT DESCRIBE A VALUE AND THEN PLACES THE VALUE IN A 4-BYTE
C INTEGER.  THE NUMBER OF BITS USED IN THE COMPRESSED INTEGER HAS
C BEEN CODIFIED INTO THE BIT PATTERN SO THAN THE BOUNDARIES ARE
C IMPLICIT.
C
C OUTPUT ARGUMENT:
C   I4A    - INTEGER*4.  16-ELEMENT ARRAY CONTAINING THE INTEGERS
C            THAT HAVE BEEN UNCOMPRESSED.  FOR DETAILS, SEE
C            SUBROUTINE MASHER.
C
C INPUT ARGUMENTS:
C   MASH   - BYTE.  64-ELEMENT ARRAY CONTAINING THE COMPRESSED
C            CODE AND VALUES.  UNDER MOST CIRCUMSTANCES THE
C            ENTIRE LENGTH OF MASH WILL NOT BE READ, BUT IT MUST
C            BE 64 BYTES TO HANDLE THE POSSIBILITY THAT NO
C            COMPRESSION HAD OCCURRED.
C   NBYTES - INTEGER*4.  THE ACTUAL NUMBER OF BYTES THAT WERE
C            READ IN FROM MASH TO OBTAIN THE 16 INTEGERS.  THE
C            COMPRESSED STREAM OF 16 INTEGERS ALWAYS ENDS ON A
C            BYTE BOUNDARY.
C
C RETURNS:
C   NORMAL - WHENEVER THE PRESCRIBED OPERATIONS ARE COMPLETED.
C   1ST    - NEVER USED.  THIS SUBROUTINE SHOULD NOT PRODUCE
C            DATA-DEPENDENT ERRORS.  (BUT, IT IS VERY GOOD AT
C            "G.I.G.O.")
C
C
C SUBROUTINE UNMASH WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991220.
C
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
C IMPLIED.
C
C
      subroutine unmash(i4a,mash,nbytes,*)
C
C
C DECLARATIONS
C
C     INPUT ARGUMENTS
      integer*4 i4a(16)
      byte mash(64)
      integer*4 nbytes
C
```

```
C     THE NUMBER TO UNCOMPRESS
      integer*4 n4
C
C     INSTRUCTION ARRAY
C        COL1: INSTRUCTIONS FOR CLEAR BIT
C        COL2: INSTRUCTIONS FOR SET BIT
C        ROW1: ENTRY POINT, EVALUATE 1ST BIT IN THE SERIES
C        POS ARRAY VALUE:  ROW NUMBER FOR EVALUATING NEXT BIT
C        NEG ARRAY VALUE:  EXIT POINT, GOTO-VALUE FOR BRANCHING
      byte inst(7,2)
      data inst /  2,-1,-2,-3, 6,-4,-6,
     &            -8, 3, 4, 5, 7,-5,-7 /
C
C     MISC INDICIES
      integer*4 ibitsr,ipwr,ix,jrow,k,kcol
C
C
C INITIALIZATIONS
C
C     I4A() INDEX
      k=0
C
C     CURRENT BIT LOCATION IN MASH()
      ibitsr=8
      nbytes=1
C
C
C CHECK POSITION IN I4A AND INITIALIZE ENTRY-ROW NBR OF INST()
C
  209 if(k.ge.16) goto 990
      jrow=1
C
C
C INCREMENT CURRENT BIT LOC IN MASH()
C
  200 ibitsr=ibitsr-1
      if(ibitsr.lt.0) then
        ibitsr=7
        nbytes=nbytes+1
      endif
C
C
C FIND INSTRUCTION COLUMN BASED ON VALUE OF CURRENT BIT IN MASH()
C
      if(btest(mash(nbytes),ibitsr)) then
        kcol=2
      else
        kcol=1
      endif
C
C
C FIND INSTRUCTION FOR CURRENT BIT
C
      ix=inst(jrow,kcol)
      if(ix.gt.0) then
        jrow=ix
      else
```

```
          n4=0
          k=k+1
          goto(101,102,103,104,105,106,107,108),-ix
        endif
        goto 200
C
C
C EVALUATE I4A() FOR VARIOUS INSTRUCTIONS FROM INST()
C
C       INSTRUCTION -1, EVALUATE 0-BIT NUMBER
  101 i4a(k)=n4
        goto 209
C
C       INSTRUCTION -2, EVALUATE 1-BIT NUMBER
  102 ibitsr=ibitsr-1
        if(ibitsr.lt.0) then
          ibitsr=7
          nbytes=nbytes+1
        endif
        if(btest(mash(nbytes),ibitsr)) then
          n4=1
        else
          n4=-1
        endif
        i4a(k)=n4
        goto 209
C
C       INSTRUCTION -3, EVALUATE 4-BIT NUMBER
  103 do ipwr=3,0,-1
          ibitsr=ibitsr-1
          if(ibitsr.lt.0) then
            ibitsr=7
            nbytes=nbytes+1
          endif
          if(btest(mash(nbytes),ibitsr)) n4=n4+2**ipwr
        enddo
        if(n4.le.7) then
          n4=n4-9
        else
          n4=n4-6
        endif
        i4a(k)=n4
        goto 209
C
C       INSTRUCTION -4, EVALUATE 2-BIT NUMBER
  104 do ipwr=1,0,-1
          ibitsr=ibitsr-1
          if(ibitsr.lt.0) then
            ibitsr=7
            nbytes=nbytes+1
          endif
          if(btest(mash(nbytes),ibitsr)) n4=n4+2**ipwr
        enddo
        if(n4.le.1) then
          n4=n4-11
        else
          n4=n4+8
```

```
      endif
      i4a(k)=n4
      goto 209
C
C     INSTRUCTION -5, EVALUATE 6-BIT NUMBER
  105 do ipwr=5,0,-1
        ibitsr=ibitsr-1
        if(ibitsr.lt.0) then
          ibitsr=7
          nbytes=nbytes+1
        endif
        if(btest(mash(nbytes),ibitsr)) n4=n4+2**ipwr
      enddo
      if(n4.le.31) then
        n4=n4-43
      else
        n4=n4-20
      endif
      i4a(k)=n4
      goto 209
C
C     INSTRUCTION -6, EVALUATE 10-BIT NUMBER
  106 do ipwr=9,0,-1
        ibitsr=ibitsr-1
        if(ibitsr.lt.0) then
          ibitsr=7
          nbytes=nbytes+1
        endif
        if(btest(mash(nbytes),ibitsr)) n4=n4+2**ipwr
      enddo
      if(n4.le.511) then
        n4=n4-555
      else
        n4=n4-468
      endif
      i4a(k)=n4
      goto 209
C
C     INSTRUCTION -7, EVALUATE 18-BIT NUMBER
  107 do ipwr=17,0,-1
        ibitsr=ibitsr-1
        if(ibitsr.lt.0) then
          ibitsr=7
          nbytes=nbytes+1
        endif
        if(btest(mash(nbytes),ibitsr)) n4=n4+2**ipwr
      enddo
      if(n4.le.131071) then
        n4=n4-131627
      else
        n4=n4-130516
      endif
      i4a(k)=n4
      goto 209
C
C     INSTRUCTION -8, EVALUATE 31-BIT NUMBER
  108 do ipwr=30,0,-1
```

```
        ibitsr=ibitsr-1
        if(ibitsr.lt.0) then
          ibitsr=7
          nbytes=nbytes+1
        endif
        if(btest(mash(nbytes),ibitsr)) n4=n4+2**ipwr
      enddo
      if(n4.le.1073741823) then
        n4=n4-1073873451
      else
        n4=n4-1073610196
      endif
      i4a(k)=n4
      goto 209
C
C
C EXIT PROCEDURE
C
  990 return
  991 return 1
      end
```

```
C
C________________________________________________________________
C
C     SUBROUTINE  I 4 C N V
C________________________________________________________________
C
C SUBROUTINE I4CNV PUTS AN INTEGER*4 VALUE INTO A 4-ELEMENT BYTE
C ARRAY SUCH THAT THE FIRST ELEMENT IS THE MOST-SIGNIFICANT BYTE
C AND THE 4TH ELEMENT IS THE LEAST-SIGNIFICANT BYTE.  IT ALSO
C PERFORMS THE INVERSE OPERATION.  THE RESULT IS THE SAME AS IF
C AN EQUIVALENCE BETWEEN THE NUMBER AND THE ARRAY WAS PERFORMED
C ON A UNIX-LIKE MACHINE.  HOWEVER, IF THE EQUIVALENCE WAS
C PERFORMED ON A VAX-LIKE MACHINE, THE RESULT WOULD BE A REVERSAL
C OF THE BYTES.  THIS SUBROUTINE IS DESIGNED TO CIRCUMVENT THIS
C DISPARITY, SO THAT INTEGER*4 VARIABLES CAN BE WRITTEN TO AND
C READ FROM A FILE WITHOUT HAVING BYTE REVERSAL PROBLEMS AND
C WITHOUT NECESSITATING USE OF COMPILER SWITCHES TO FIX THE
C PROBLEMS.
C
C INPUT ARGUMENT:
C   NVERSE - INTEGER*4.  IF NVERSE IS SET TO 0, THE FORWARD
C            OPERATION WILL BE PERFORMED SUCH THAT THE INTEGER*4
C            VARIABLE WILL BE PUT INTO THE BYTE ARRAY.  IF NVERSE
C            IS SET TO 1, THE INVERSE OPERATION WILL BE PERFORMED
C            SUCH THAT THE BYTE ARRAY WILL BE PUT BACK INTO THE
C            INTEGER*4 VARIABLE.
C
C INPUT/OUTPUT ARGUMENTS:
C   I4A    - INTEGER*4.  THE INTEGER*4 VARIABLE TO BE CONVERTED.
C            IF NVERSE IS 0, I4A IS AN INPUT ARGUMENT.  IF NVERSE
C            IS 1, I4A IS AN OUTPUT ARGUMENT.
C   K1B    - BYTE.  THE 4-ELEMENT BYTE ARRAY TO BE CONVERTED.
C            IF NVERSE IS 0, I4A IS AN OUTPUT ARGUMENT.  IF
C            NVERSE IS 1, I4A IS AN INPUT ARGUMENT.
C
C
C SUBROUTINE I4CNV WRITTEN BY ROB BRACKEN, USGS.
C FORTRAN 77, HP FORTRAN/9000, HP-UX RELEASE 11.0
C VERSION 1.0, 19991229.
C
C NOTE:  THIS ALGORITHM IS AN IN-HOUSE PROGRAM GENERATED BY ROB
C BRACKEN AT THE USGS.  IT HAS NOT YET BEEN OFFICIALLY RELEASED
C AND MAY CONTAIN METHODS OR IDEAS THAT SHOULD BE CREDITED TO THE
C WRITER.  NO GUARANTIES OR WARRANTIES ARE GIVEN, EXPRESSED OR
C IMPLIED.
C
C
      subroutine i4cnv(nverse, i4a,k1b)
C
C DECLARATIONS
C
C     INPUT ARGUMENT
      integer*4 nverse
C
C     INPUT/OUTPUT ARGUMENTS
      integer*4 i4a
      byte k1b(4)
```

```
C
C     INTERNAL VARIABLES
      integer*4 i0,j0,k0,k
      byte ksgn
      integer*4 jsgn
C
C
C DECIDE WHETHER TO PERFORM FORWARD OR INVERSE OPERATION
C
      if(nverse.ge.1) goto 101
C
C
C PERFORM THE FORWARD OPERATION I4A -> K1B
C (ASSUMES THAT ALL INTEGERS ARE 4-BYTE TWOS COMPLEMENT,
C AND THE HIGH-BIT IS THE SIGN BIT)
C
C     INITIALIZE THE RUNNING VALUE AND THE SIGN BIT VALUE
      j0=i4a
      ksgn=0
C
C     CHECK FOR THE RUNNING VALUE LESS THAN ZERO
      if(j0.lt.0) then
C
C       CLEAR ONLY THE SIGN BIT
        j0=j0+2**30
        j0=j0+2**30
C
C       REMEMBER THE SIGN BIT FOR LATER APPLICATION TO THE MSB
        ksgn=64
      endif
C
C     CALCULATE THE BIT PATTERN FOR EACH OF THE 4 BYTES
      do k=4,2,-1
        i0=j0
        j0=i0/256
        k0=i0-j0*256
        if(k0.gt.127) k0=k0-256
        k1b(k)=k0
      enddo
      k1b(k)=j0
C
C     RESTORE THE SIGN BIT IN THE MOST SIGNIFICANT BYTE
      k1b(k)=k1b(k)-ksgn
      k1b(k)=k1b(k)-ksgn
      goto 990
C
C
C PERFORM THE INVERSE OPERATION K1B -> I4A
C
C     INITIALIZE THE RUNNING VALUE AND THE SIGN BIT VALUE
  101 i4a=k1b(1)
      jsgn=0
C
C     CHECK FOR THE RUNNING VALUE LESS THAN ZERO
      if(i4a.lt.0) then
C
C       CLEAR ONLY THE SIGN BIT
```

```
          i4a=i4a+128
C
C         REMEMBER THE SIGN BIT FOR LATER APPLICATION TO THE MSB
          jsgn=2**30
        endif
C
C       RESTORE THE BIT PATTERN FOR EACH OF THE 4 BYTES
        do k=2,4
          k0=k1b(k)
          if(k0.lt.0) k0=k0+256
          i4a=i4a*256+k0
        enddo
C
C       RESTORE THE SIGN BIT
        i4a=i4a-jsgn
        i4a=i4a-jsgn
C
C
C EXIT PROCEDURE
C
  990 return
        end
```