UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

Shipboard Playback for Modification II of Ocean Bottom Instrument Package

by

J. J. Fredericks, G. I. Evenden, and J. E. Dodd

Open-File Report 85-111

Woods Hole, Massachusetts

1985

TABLE OF CONTENTS

ALPHABETICAL INDEX OF SOFTWARE

INTRODUCTION

The U.S. Geological Survey's (USGS) Ocean Bottom Instrument Package (OBIP) is a recoverable ocean bottom data-capture device for seismic and other data. The software described in this report was written for use with data that have been captured using the software listed in the USGS Open-File Report 84-842. The format is described in Appendix A of this report. The "C" programming language was used for all routines, excluding the cartridge and 9-track tape utilities which were written in assembler using ALLOY IDXS-100 interface protocol. The plotting utilities were written using the DM/PL instruction set for the Houston Instrument Incremental Plotter. The TELETEK microprocessor interfaced with an S-100 bus was used for creation of the software and playback of the data.

The program TRANSCRB is used to copy any cartridge to a 9-track tape. The program was modified to simultaneously create a directory of the contents of the tape as it is copied. With the modification, the cartridge tape must be in the format which is described in the Appendix A of this report.

The program OPLOT allows shipboard plotting of recovered OBIP data that have been transcribed to a 9-track tape. The operator may request, by time, a series of events which will be retrieved from the tape and plotted according to specified plotting parameters.

The program SHOTD is used to enter the shot point number, shot instant and shot range data for each shot point of the project. Shot point specifications are retrieved from the data file by the program OPLOT.

Transcrb

Description

The program was written by G.I. Evenden to copy the contents from the recovered OBIP cartridge tape written in TIP format to a TIP formatted 9-track tape. The utilities are specific to the TIP format and Alloy Interface (IDXS-100) protocol and are listed in Appendix B. The routines will ignore an end-of-file (EOF) which is detected at the beginning of a track and will expect a double EOF as the end of data flag. The program was modified by J.J. Fredericks so as to create a directory while copying the contents.

Direction for use

When the user wants to save the tape directory, the run-line must contain an output filename. To run the utility type:

TRANSCRB [-rn] [-tn] [-en])dev:filename.type

The "-rn" option is used to set the maximum number of records to be transcribed. The "-t" option allows the user to begin copying from a track other than the default track 0. The "-en" option is used to specify the number of errors allowed before termination of the copy.

Information about the tape from the GPHEADER record is dumped to "stdout". The default "stdout" is the terminal, but the specification of ")dev:filename.type" diverts the output to the file named. A list of series parameters from the GPHEADER record is listed to "stdout" and is followed by a list of parameters for each experiment that has been generated on the tape. The following parameters are displayed for each

experiment:

REC	-	the record number on the cartridge tape
TREC	-	the record number on the cartridge tape on a given track
S	-	series number
EX	-	experiment number
MO	-	start time of window (month)
DA	-	(day)
HR	-	(hour)
MN	-	(minute)
SE.FF	-	(seconds to the hundredths)

The record numbers listed just before the track changes do not reflect the number of physical records on a track, as the parameters are only printed when a last block flag is present in the TIP header.

```
/*                          SOFTWARE

   Transcription routine: TRANSCRB

**
**    to copy data from OBIPS cartidges to 9-track, 1600bpi
**   magnetic tape.
*/
int  numbport = 2;       /* number of config. ports */
int  cartport = 0xff;    /* port no. for cartridge drive */
int  tapeport = 0xff;    /* port no. for 9-track drive   */

#include "a:libc.h"
#include "a:tapeio.h"

char  buff[9000];   /* I/O buffer */

/*  data control blocks for 'tape' routine */
DCB  cartdcb = {0, 0, 0x60, 0, 0, 0, 0, 0, &buff};
DCB  tapedcb = {0, 0, 0, 0, 0, 0, 0, 0, &buff};
PDCB  pcart = &cartdcb, ptape = &tapedcb;

int maxrec = 32767, maxerr = 10;

main(argc, argv)
int argc;
char *argv[];
{
     int i, n;
     char *p;

          /* get and check arguments */
     if (argc > 1) { /* arguments given */
          if (*argv[1] == '?') {
               fprintf(stderr,"TRANSCRB [-tre]");
               fprintf(stderr," --- cart to tape utility\n");
               exit(0);
          }
          while (argc-- > 1) {
               p = argv[argc];
               if (*p++ == '-') {
                    i = tolower(*p++);
                    n = atoi(p);
                    switch (i) {
                    case 't':
                         pcart->drive |= n & 3;
                         break;
                    case 'r':
                         maxrec = n;
                         break;
                    case 'e':
                         maxerr = n;
                         break;
                    default:
                         fprintf(stderr,"invalid option\n");
```

```
                    exit(1);
                }
            }
        }
    }

        /* check if program configured */
    if (cartoort == 0xff || tapeoort == 0xff) {
        fprintf(stderr,"TRANSCRB not configured\n");
        exit(1);
    }
        /* configure DCB's */
    pcart->port = cartoort;
    ptape->port = tapeoort;
        /* check cartridge */
    iscart(pcart);
        /* check tape */
    istape(ptape);
        /* transcribe */
    n = ctcopy(ptape, pcart, maxrec, maxerr);
        /* done */
    printf("approx. %d records transcribed\n", n);

    exit(0);

}
```

```
/* ctcopy - transcribe cartridge to 9-track

*/
#include "libc.h"
#include "tapeio.h"
extern char buff[];
ctcopy(tdcb, cdcb, maxrec, maxerr)
PDCB tdcb, cdcb;
int maxrec, maxerr;
{
     int ce, te, t, rec;
     int trec; /* trec is the track record counter */
     int eoflag; /* flag to signal EOF on first rec of track */
     FILE *fl,*fopen();

        /* temporary patch
        fl = fopen("lst:","w");  */

          /* setup operations */
     cdcb->cmd = READ;
     te = 0;
     trec = 0;
          /* do initial read exec */
     if (ce = tape(cdcb))
          terror("cart exec", 0);

          /* transcription loop */
     for (rec = 1; !keybrk() && !ce && !te &&
                    rec (= maxrec; rec++) {
               /* check status of last cart READ */
          if (cdcb->status)
             if (ce = check(cdcb,"cart",rec)) {
               if (ce & EOTRACK) {
                         /* bump track number */
                    t = cdcb->drive & 3;
                    if (++t ( 4) {
                         cdcb->drive &= 0xfc;
                         cdcb->drive += t;
                         ce = 0;
                         trec = 0;
                    }
               }
               if (ce == EOFT && !trec) { /* EOFT is at beginning of track */
                    ce = 0;
                    eoflag = 1;
                    }

               /* check 'maxerr' situation */
               if (ce && ce != EOFT && maxerr-- ) 0) {
                    ce = 0;
                    printf("will write %d bytes of rec. %d\n",
                         cdcb->size, rec);
               }
          }
          else  /* no cart read error bump trec */
```

```
            trec++;

            /* exec next read and write data on 9-track */
        if (!ce) {
            if (eoflag) { /* last cart read found EOF at first track */
                rec--;
                eoflag = 0;
                continue;
                }
            if (buff[13])
                /* a trailer and/or header is present */
                dirct(rec,trec);
             /* temporary patch
             decode(&buff[16],f1);
             fclose(f1);     */
            tdcb->size = cdcb->size;
                /* issue next READ exec */
            if (ce = tape(cdcb))
                terror("cart exec(B)", rec);
                /* check status of last WRITE */
            if (tdcb->status)
                te = check(tdcb,"tape",rec);
             /* co tape WRITE */
            if (!te && tdcb->size ) 0) {
                tdcb->cmd = LENGTH;
                if (!(te=tape(tdcb)) && !(te=tape(tdcb))) {
                    tdcb->cmd = WRITE;
                    te = tape(tdcb);
                }
                if (te)
                    terror("tape exec",rec);
            }
        }
        /* check for  end of data flag */

    }
        /* check last tape exec and do an EOF on tape */
    if (!te) { /* write EOF if tape not bad */
        if (tdcb->status) /* do check, if necessary */
            te = check(tdcb,"tape",rec);
        tdcb->cmd = WEOF;
        if (!te && !(te = tape(tdcb)))
            check(tdcb,"tape",rec);
        else
            terror("tape exec for EOF",rec);
    }

    return rec - 1;

}
```

```
/* dirct - to create a directory for transcrb during tape copy

*/
#include "libc.h"
#define MAXHEADRLEN 360
extern char buff[];
dirct(rec,trec)
int rec,trec;   /* tape block counter & track block counter */
{
        int n,i,offset;
        static int nseries;
        float timr;

        if (buff[1] == 'G' && buff[2] == 'P')  /* GPHEADR RECORD -
                                                  DUMP TO STDOUT) */
                {
                i = 0;
               while (i (= MAXHEADRLEN) {
                if (buff[i] == NULL)
                   {
                   if (buff[++i] == NULL)
                      break;
                   }
                else if ((buff[i] ) 31 && buff[i] ( 127) :: buff[i] == '\n')
                   putchar(buff[i++]);
                else
                   i++;
                }
                putchar ('\n');
                /* now deal with series parameters in trailer */

                for (offset = 7952,nseries=0; (buff[offset+3]);
                    offset += 24, nseries++)
                  {
                  printf ("Series: %d\n",nseries + 1);
                  printf ("no chans: %d, series type: %c, no exps: %x\n",
                    (int)buff[offset+1]/2, buff[offset+2],buff[offset+3]);
                  printf ("Start time for series: ");
                  for (n=4;n(=8;n++)
                        printf ("%x ",buff[offset+n]);
                  printf ("\nStop time for series: ");
                  for (n=9;n(=13;n++)
                        printf ("%x ",buff[offset+n]);
                  printf ("\n");
                  printf("maxsams: %d, offset(secs): %x, period(mins): %x\n",
                        (int)buff[offset+18]*256+(int)buff[offset+19],
                        buff[offset+20], buff[offset+21]);
                  printf ("Number of recs per OBIP window: %x\n\n",
                        buff[offset+14]);
                }
                printf ("nseries: %d\n\n",nseries);
                printf (" REC TREC S EX MD  DA HR MN SE.=    TIMER\n");
                }
        else  if (buff[1] == ' ' && buff[2] == ' ') {
                printf ("Test record read.\n");
```

```
            }

        /* now deal with the trailer that should be in the last 256 bytes */

        else {
           /* for now just list out experiment parameters for each exp. */
           /* first print series, expno, mo, day, hour, min */
           printf (" %3d %4d %x %2x %x%x  %x%x %x%x %x%x",
                rec,trec,buff[8171],
                buff[8172],buff[8184],buff[8183], buff[8181],buff[8180],
                buff[8179],buff[8178],buff[8177],buff[8176]);
           timr = 1.0 -(float)((int)buff[8186] + 256*(int)buff[8187])/0x3FFF;
           if (timr > 0.1 && timr < 0.9)
                printf ("**");
           printf (" %x%x.%x  %7.3f\n",
                buff[8175],buff[8174],buff[8173],timr);
            /*printf (" %x%x.%x [%x%x] %d\n",
                buff[8175],buff[8174],buff[8173],buff[8186],buff[8187],
                (int)buff[8186] + 256*(int)buff[8187]);*/
        }
return;
}
```

OPLOT

Description

OPLOT is a shipboard playback program for plotting OBIP traces on the Houston Instrument Incremental Plotter using the DM/PL plotting instruction set. The program was written in the C programming language by J.J. Fredericks for the TELETEK-S100 microprocessor. The tape10 is configured for a 9T Cipher drive on ioport 30(hex) using ALLOY IDXS-100 Interface protocol.

Direction for use

The user must have on-line the 9T OBIP tape, which has been transcribed from the cartridge tape using the program TRANSCRB.

The plotter should be on-line with the T-bar switch selecting the plotter. Unless a plot-file is specified on the run-line, all plot instructions will be sent to "list:".

If the OBIP tape does not contain the GPHEADER block or the processing is not to begin at the beginning of the tape, the GPHEADER block must be on the disc from which the user is operating.

A file containing a list of shot numbers, t0 (yr mo day hr min se.ff), and ranges must exist before OPLOT can be run. This file may be created using the utility SHOTD. Traces will be selected from this list of t0's by the program OPLOT. The file

must contain all parameters for each shot number, in the order given, separated by a space. See Appendix C for a listing of SHOTD.

To run the program enter:

OPLOT drv:name.type [plot-deffered-file]

(where drv:name.type is the shot descriptor file from which the t0 times are retrieved)

(plot-deffered-file is optional and is only used in test modes to defer the plot from actually plotting)

A dump of the text portion of the GPHEADER record is placed on the screen. Then the user is prompted as follows:

Specify series no: The program parses out the series parameters from the GPHEADER which is either on the OBIP tape or read from the GPHEADER disc file. The program does not check for a change in the series as it processes the tape, so the parameters which have been retrieved (number of channels, sampling rate and sampling frequency or interval) will remain the same throughout the run.

Enter kv & wbv: Specify two floating point values, separated by a space, which represent the normalizing velocity

and the water break velocity. If the range for a given shot is zero, the traveling time correction and water break cut-off is not computed and the trace continues until "tmax" or the end of the OBIP window. For a series of shots with zero range values, the traces will be plotted at one kilometer intervals (for spacing of traces).

Enter shot1,shotl,shoti: Enter integer values, separated by spaces, representing the first shot, last shot and the shot interval to be plotted. The subroutine TLOAD will open the shotc.dat file and load the request parameters from the first shot (shot1) to the last shot (shotl) at shot intervals (shoti). There are no defaults for these values. Up to 100 shots can be processed in a run.

Enter pre-amp gain: Enter a floating point value which is used to correct the decoded amplitudes from the OBIP for the pre-amp setting.

Enter channel no: Enter the number of the channel that is to be plotted (1,2,3, or 4). The user is not prompted for the channel number if only one channel is present in the OBIP data.

Tmax & tlen: Enter the maximum number of seconds to be plotted and the length in inches of tmax, separated by a space. This determines the scaling factor for each trace according to the sampling rate. The minimum time is always zero.

Rmax, rmin & rlen: Enter the maximum range(km), minimum range(km) and the length of the range axis(inches), separated by spaces. The scale factor for the positioning of the traces is computed using these values, where rlen is the length of the axes from rmin to rmax. When range values in the SHOTD file are zero, traces can be spaced at one inch intervals by entering rmax equal to the number of traces plus one; rmin equal to zero; and, rlen equal to the number of traces plus one.

The program will then plot the axes. They have not been labelled because the task is time consuming, so the user must make a note of the parameters chosen. The progam will request the user to enter a return which should only be answered when the axes have been drawn and you are ready to process the data.

Scale factor for trace: Enter a scaling factor which will be applied to the trace. If 1.0 is specified, the plot will use the scale of 15 counts per volt. This allows a trace that has a maximum amplitude of +-5.0V to be contained in 2 cm. The plotter scale for the HI Incremental plotter is 78.7 counts per centimeters or 200 counts per inch.

Amp.factor: R0 & Alpha: Enter a range base (r0) and an alpha factor (separated by a space) to amplify the amplitude signal as the range increases. If the range is 0.0, no range amplitude factoring is done on the signal. The algorithm for amplitude scale factoring for non-zero range traces is:

plotcounts = (signal volts)*(plotcounts/volt)*(scalefactor)*

((range/r0)**alpha)

The program then commences the plotting. If the system halts, secure all interface connectors and try again. If the plotter starts acting up, it may be that the data are coming through the buffer too fast for the plotter to handle. A patch to wait after each trace could be implemented -- but that would require that someone be available to continuously monitor the program and is obviously an undesirable solution. An electrostatic plotter may alleviate the problem, but it is more costly.

Misc

The sampling interval is computed as follows:

```
secint = 4096.*nblocks/(nsams*nchans);
```

This value is added to the time read from each record on the OBIP tape and is compared to the t0 to determine whether or not the t0 falls within the current OBIP window read.

When a t0 is found, the next t0 requested may be within the same OBIP window. But, it may not be before the current OBIP window. **All times (t0) requested must be in increasing order.** The tape is not searched backwards.

For further software development, the code which is dependant on the header and trailer formats were flagged by

three asterisks (***). The amplitudes were decoded in the routine DECODE based on each amplitude's occupying 16-bits, with the least significant 8-bits in the first 8 bits(15-8), the most significant 4-bits in the last four bits (0-3), and the gain code in bits 4-7. (See DECODE for complete description.)

An amplitude greater than 90.0 volts flags the end of an OBIP window. If this becomes a problem due to modification of the routine DECODE, an end-of-window flag should be implemented.

The plotter was timed in a search and plot mode, with traces scattered throughout the first series. The average time for finding and plotting a trace was approximately 3.6 minutes per 5 second trace.

```
                        /* SOFTWARE */


/* Dolot.h  - include file for oolot*/

#define MAXHEADRLEN 500
#define MAXSHOTNOCHARS 11
#define MAXSHOTS 100
struct shotd {
        char shotno[MAXSHOTNOCHARS];
        double t0;      /* t0 as computed in tload */
        int water;      /* (water break time - t0)* nsams/sec */
        int series,
            expno; /* series & experiment number as determined in tload */
        float range;    /*  range of shot for use in labeling trace */
        };
```

```
/* OPLOT: to plot a group of OBIP shots on shipboard plotter */
/*       OBIP format:  See U.S.G.S. Open-file Report 84-842 */
#include "libc.h"
#include "tapeio.h"
#include "oplot.h"
   char tbuff[9000];
   DCB tapedcb = {0,0,0,0,0,0,0,0,&tbuff};
   PDCB ptape = &tapedcb;
   int tapeport = 0xE0;         /* assigns E0(hex) as tape port */

struct shotd shot[MAXSHOTS],*pshot;
int series,     /* series number to pars nchans & nsams */
   maxsams,     /* max samples computed from max secs in params */
    nshots,     /* number of shots to be plotted as determined in tload */
                /*    the shotd array addressing is from 0 to nshots-1  */
    nblks,      /* no. blocks per record */
    nsrec,      /* number of secs per physical record (varies with nochans) */
    chano,      /* channel number to be plotted */
    nchans,     /* number of channels recorded for each sample */
    nsams;      /* number of samples per sec (128 or 256) */

float pagain,   /* pre-amp gain */
      rf,r0,rmin, rmax, /* rf(alpha),r0, & max range in km */
      rscale,tscale, /* counts/km , counts/sec or counts/sample */
      ascale,       /*counts/mv */
      secint;   /* secs between windows - will be f(period) */


char string[80],*strptr;

main(argc,argv)
int argc;
char **argv;
{
       FILE *fi,*fopen();
       int ns, /* current shot number (0 to nshots-1) */
           byteno;  /* byteno of first sample to load for plotting */

       if (argc < 2) {
              puts ("\n Usage: OPLOT shotdatfile  plotoutfile\n");
              exit(1);
              }

       /* initialize tape drive */
       /*** tinit contains OBIP format dependant code */
       tinit (ptape);
       nsrec = 4096/nchans;

       /* initialize plotter output - if not specified as a file
                                      output is directed to :lst */
       init( (argc > 2)? argv[2] : 0);

       /* load variables from shot data file (first argument on run line) */
       if ( (fi = fopen(argv[1], "r") ) == NULL) {
              printf ("Can't open %s\n",argv[1]);
```

```
        exit(1);
        }
tload(fi);
fclose (fi);

/* select plotting parameters & plot axes */
params();

/* for each shot: find shot window on OBIP 9T-tape,
                  load from first specified sample &
                  plot data */
/*** tfind,plotln  & decode contain OBIP format dependant code */
for (ns = 0; ns < nshots; ns++) {
        if ((byteno = tfind(ns)) < 0)
                continue;
        plotln( (byteno+(chano-1)*2) ,ns);
        }

closep();       /* close plotting */
exit(0);
}
```

```
/* PARAMS.C */
#include "plot.h"
params()
{
        extern int nchans,chano,maxsams, nsams;
        extern float pagain,r0,rf,rmin,rmax,tscale,rscale,ascale;
        float sfctr,tmax, tlen, rlen;

        printf ("Enter pre-amp gain: ");
        scanf ("%f",&pagain);

        chano = 1;
        if (nchans != 1) {
           printf ("Enter channel no: ");
           scanf("%d",&chano);
           }

        printf ("Tmax & tlen :");
        scanf ("%f %f",&tmax,&tlen);
        maxsams = tmax*nsams;
        if (tlen > 20.0) {
               printf ("Time axis too long.\n");
               exit(1);
               }

        printf ("Rmax, rmin & rlen: ");
        scanf ("%f %f %f",&rmax,&rmin,&rlen);
        rscale = (rlen/(rmax-rmin))*200;        /* counts per km */
        tscale = (tlen/tmax)*200;
        scale (rscale,tscale);
        axis(rmax,rmin,1.0,tmax,0.0,1.0,"KM","SECS");

        /* now set px to counts per millivolt & py to counts per sample */
        printf ("Scale factor for trace: ");
        scanf ("%f",&sfctr);

        printf ("Amp factor: r0 & alpha: ");
        scanf ("%f %f",&r0,&rf);

        ascale = 15.0*sfctr;
        tscale = tscale/nsams;
        scale (ascale,tscale);
        return;
}
```

```
/*  plot.h - include file for uplot routines */

#define SIZE1 0.56
#define SIZE2 0.28
int fl;  /* file identifier returned from open in init() */
float px = 78;  /* pxcount is in plotter counts/data unit */
float py = 78;
```

```
                    /* UPLOT.C */
                /* plotting subroutines */
             /* using DM/PL instruction set */


#include "LIBC.H"
#include "plot.h"
#define HOME "0,0"
#define ERR -1
#define MAXBYTES 255
extern int errno;
char buff[MAXBYTES];
int nbytes;     /* number of bytes to dump to fi or byte no to start sprintf */
int rofset;     /* x offset in board counts for range adjustment */
/* ********** INIT - to initialize plotter ********** */
init(pname)
char *pname;
   {
   if (pname)
      fi = open(pname,O_CREAT+O_WRONLY);
   else
      fi = open("lst:",O_WRONLY);
   if (fi == ERR) {
        printf("Error opening file, errno = %d\n",errno);
        exit(1);
        }
   sprintf (buff,";;0\n");
   write(f1,buff,4);
   nbytes = 0;
   return (f1);
   }
/* ********** AXIS - to draw axes for plot ********** */
axis(xmax,xmin,xtint,ymax,ymin,ytint,xlabel,ylabel)
double xmax,xmin,xtint,ymax,ymin,ytint;
char *xlabel,*ylabel;
{
   char label[6];
   int isize,ix,iy,ixmin,iymin,kpen;
   float x,y,tlen;

   rofset = 0;
   kpen = 1;
   movep(0.0,ymax,&kpen);
   movep(0.0,ymin,&kpen);
   movep(xmax-xmin,ymin,&kpen);
   movep(xmax-xmin,ymax,&kpen);
   movep(0.0,ymax,&kpen);
   oflush();
   wait();
   wait();
  /* draw ticks along axes */
   tlen = 40/dx;
   while (ymax >= ymin) {
        kpen = 1;
        movep(0.0-tlen,ymax,&kpen);
```

```
        movep(0.0,ymax,&kpen);
        ymax -= ytint;
        };
    wait();

    tlen = 40/oy;
    x = 0.0;
    while (x (= xmax - xmin) {
        kpen = 1;
        movep(x,ymin-tlen,&kpen);
        movep(x,ymin,&kpen);
        x += xtint;
        };
    pflush();
    wait();
    return;
    }
/* ********** CLOSEP - to home pen & close plot device or file ********** */
closep ()
{
    pflush();
    printf ("Closing plotting routine.\n");
    sprintf (buff,":;AU0 0 0\n");
    if (write(fl,buff,10) != 10)
        printf ("Error closing plot.\n");
    close(fl);
    return;
}
/* ********** WAIT - to hold up plotter until user response ********** */
wait()
{
    int i;
    char label;
    printf ("RETURN to continue:");
    scanf("%c",&label);
    return;
}
/* ********** MOVEP - to move pen to x,y ********** */
movep(x,y,kpen)
double x, y;    /* coordinates in units */
int *kpen;      /* to indicate pen up & pen down
                   pen is always left down after call to movep */
{
    if (*kpen) {
        if(nbytes) pflush();
        sprintf (buff,":;AU%5d %5d ;:D",(int) (x*ox)+ rofset,(int)(y*oy) );
        *kpen = 0;
        nbytes = 19;
        }
    else {
        if (nbytes +12 ) MAXBYTES)  /* dump buffer & wait for it */
            pflush();
        sprintf(&buff[nbytes],"%5d %5d ", (int)(x*ox) + rofset,(int) (y*oy) );
        nbytes += 12;
        }
```

```
    return;
    }
/* ********** PFLUSH - to flush plot buffer ********** */
pflush()
{
        if ( (nbytes -= 1) ) 0) {
            if (write(fl,buff,nbytes) != nbytes)
                printf ("Error writing plot buffer.\n");
            nbytes = 0;
            }
        return;
}
/* ********** SCALE - to reset plot_counts/unit ********** */
scale(ix,iy)
double ix,iy;   /* the no board counts per plot unit */
{
        px = ix;
        py = iy;
        return;
        }
/* ********** RSHIFT - to offset each trace ********** */
rshift(x)
double x;
        /* to set rofset which is the x offset due to range adjustment */
        {
        extern float rscale,rmin;
        rofset = (int) (rscale*(x-rmin));
        return;
        }
/* ********** LABEL - to label at x, y (in user units) ********** */
label(x,y,title)
int x,y;
char *title;
{
        int ipen;

        ipen = 1;
        movep( (double)x,(double)y,&ipen);
        sprintf(&buff[nbytes],";:S41 %s_",title);
        nbytes += (strlen(title) + 8);
        pflush();
        return;
        }
```

```
/* TLOAD.C - to load shot parameters from SHOTD file */

#include "oplot.h"
#include "jjfinc.h"
extern struct shotd shot[],*pshot;
extern int nsams,nshots;
extern char string[],*strptr;
tload(f1)
FILE *f1;
{
        int shot1,shot1,shoti;          /*first,last shot & shot interval */

        int year,mon,day,hour,min,isec; /* time read from shot dat file */
        double sec,tw;

        int ns, nskip;
        char *fgets();
        double fltime(),getfloat(),t;
        float kv,wbv;

        /* get normalizing velocity & water break velocity from shot.dat */
        kv = 6.0;
        wbv = 1.2;
        printf("Enter kv & wbv: ");
        scanf ("%f%f",&kv,&wbv);
        printf ("kv: %f, wbv:%f\n",kv,wbv);

        printf ("Enter shot1 shot1 shoti: ");
        scanf ("%d %d %d",&shot1,&shot1,&shoti);

        nshots = 0;
        nskip = 1;

        for (ns = 1; ns (= shot1; ns++) {
                if (fgets(string,80,f1) == NULL) {
                   printf ("EOF after shot %d ",ns-1);
                   break;
                   }
                if (ns ( shot1)
                   continue;
                if ( !(--nskip) ) {     /* load shotdata & set nskip */
                   pshot = &shot[nshots];
                   strptr = string;
                   getwdo(pshot->shotno);
                   year = getint();
                   mon  = getint();
                   day  = getint();
                   hour = getint();
                   min  = getint();
                   sec  = (float) getfloat();
                   shot[nshots].range = (float) getfloat();
                   t = fltime(year,mon,day,hour,min,sec);
                   shot[nshots].t0 = t + (double) (shot[nshots].range/kv);
                   tw = t + (double) (shot[nshots].range/wbv);
                   tw -= shot[nshots].t0;
```

```
                shot[nshots].water = (int) (tw*(float)nsams);
                if (++nshots >= MAXSHOTS) /* too many samples requested */
                     {
                     printf ("Too many shots requested ");
                     break;
                     }
                nskip = shoti;
                }
           }
      printf ("%d shots will be plotted.\n",nshots);
      return;
}
```

```
/* TFIND.C - to position tape on a requested OBIP window */

/*  *** this routine is highly dependant on OBIP tape format - search
       for occurrences of 'tbuff' to find dependant code */

#include "oplot.h"
extern int nblks,nsrec,nsams,nchans; /*no sams per rec, no sams per sec,
                                no chans per sample*/
extern struct shotd shot[];
extern char tbuff[];
tfind(ns)
int ns; /*ns = address of shotdat buffer */
{
        double timr,ttape,fltime(); /* time read from trailer on OBIP tape */
        int year,mon,day,hour,min;
        int blk,samskp,modskp;
        double sec;
        extern float secint;

        printf("Finding shotno:%s\n",shot[ns].shotno);
        do {
        tread(1);
        if (!(tbuff[13]))
          continue;
        year = ((tbuff[8185]  & 0360))>4)*10 + (tbuff[8185] & 017);
        mon = (int)tbuff[8184]*10 + (int)tbuff[8183];
        day = (int)tbuff[8181]*10 + (int)tbuff[8180];
        hour = (int)tbuff[8179]*10 + (int)tbuff[8178];
        min = (int)tbuff[8177]*10 + (int)tbuff[8176];
        sec = (int)tbuff[8175]*10+(int)tbuff[8174]+((int)tbuff[8173]/10.0);
        ttape = fltime(year,mon,day,hour,min,sec);
        timr = (float)(tbuff[8186] + 256.0*tbuff[8187])/0x3FFF;
        if (timr (= 0.10)  /* clock ahead of timer */
          ttape -= timr;
        else {
          if (timr ( 0.90)
               printf ("Timer/clock out of sync.\n");
          ttape += (1.0 - timr);
          }
        }  while (ttape + secint ( shot[ns].t0);

        /* find t0 within OBIP window */
        samskp = (shot[ns].t0-ttape)*nsams;     /*samples to skip */
        modsko = samsko/nsrec;
        blk = nblks;
        tread(-1);      /* back-up one rec */
        /* printf("samskp: %d, nsrec: %c, modsko: %d\n".samsko,nsrec,modsko);*/
        switch (modsko) {
        case '\0':      if(--blk ) 0) tread(-1); /* backspace tape 1,2 or 3 recs */
        case '\1':      if(--blk ) 0) tread(-1);
        case '\2':      if(--blk ) 0) tread(-1);
                        tread(1);       /* read a record unless t0 is in  */
        case '\3':      break;          /* the last rec of sample         */

        default:
```

```
        printf ("T0 not in OBIP window.\n");
                return(-1);
}
/* determine the beginning byte from which to plot the data */
samsko = ((samsko - mocsko*nsrec)*nchans*2) + 16; /* 16 byte header */
/* printf ("Sams to skip: %d\n",samsko); */
return(samskp);
}
```

```
/* FLTIME.C - to compute the seconds of a date */

double fltime(year,month,day,hour,min,sec)
int year,month,day,hour,min;
double sec;
        {
        double rtime;
        rtime = sec + 60.0*(min + 60.0*(hour + jday(year,month,day)*24) );
        /* rtime is jday(secs) + hour(sec) + min(sec) + sec */
        return rtime;
        }


/* JDAY.C - to compute the julian day */

jday(yr,mo,day)
int yr,mo,day;
{
        static int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
        /* adjust for leap years */
        days[1] = (mod(4,yr)?28:29);
        --mo;  /* don't add in current month */

        while (--mo >= 0 )
                day = day + days[mo];
        return (day);
        }
```

```
/* MOD.C */
mod(num,val)
int num,val;
{
   if (val >= 0) {
        while (val >= num)
           val -= num;
        return(val);
        }
   else if (val < 0) {
        while (val <= -num)
           val += num;
        return (val);
        }
}
```

```
/* DECODE.C - to decode an OBIP signal from a word */

/* first byte is the least significant byte
   first four bits of next byte is the gain code
   last four bits of byte is the most significant byte of the signal */

#include "oplot.h"
double
decode(byteno,lastbyte)
int *byteno;    /*byte no to decode */
int *lastbyte;  /*last byte in current record*/
        {
        extern char tbuff[];
        extern float pagain;
        extern int chano;

        static float rval[17] = {1.,2.,4.,8.,16.,32.,64.,128.,256.,512.,1024.,
                                 2048.,4096.,8192.,16384.,32768.,65536.};
        static float adsens = 10.0/4096.0;

        unsigned int lsb,isiglv,gain;
        double siglev;

        /* check for end of data in physical record & OBIP window */
        if (*byteno >= *lastbyte) {     /*end of data in record */
            if (tbuff[13]) {            /***end of data in OBIP window */
                printf ("End of data in OBIP window before maxsams.\n");
                return(99);
                }
            tread(1);   /* get next rec */
            *byteno = 16 + (chano-1)*2;
            *lastbyte = ((int) tbuff[15])*128 + 16;
        /***   last block flag subtract last 256 bytes */
            }

        lsb = (int) tbuff[*byteno];
        gain = (int) tbuff[(*byteno) + 1];
        isiglv = (gain & 000017)*0400 + lsb;
        gain = (gain & 000360) >>4;

        siglev = adsens*(float)((int)isiglv-2048) / (pagain*rval[gain+1]) ;
        if (siglev > 32767)  siglev = 32767;
        /* for debugging
        printf("gain: %d, isiglev: %d, siglen: %f\n",gain,isiglv,siglev); */
        return (siglev);
        }
```

```
/* PLOTLN.C - to move to new range & plot an event */

#include "oplot.h"
plotln(byteno,ns)
int byteno,ns;
{
   int sams,ival,kpen,samax,lastbyte;
   extern int maxsams,chano,nchans;
   extern struct shotd shot[],*pshot;
   extern char tbuff[];
   extern float r0,rf,tscale,ascale;
   double rval, pow(),decode();
   static int erange = 1;       /*range = increments of 1 if 0.0 in shot.dat file */
   /* starting at byteno (which is the location of the first channel of the
        first sample to be plotted) the data must be parsed for chano &
        decoded & scaled for plotting */


   /* maxsams = max of tw or tmax */
   if (shot[ns].range == 0.0)  samax = maxsams;
   else
     samax = (maxsams > shot[ns].water? shot[ns].water:maxsams);
  /* samax = max no of samples to be plotted */
   kpen = 1, ival =0;
   lastbyte = 8207 - tbuff[13]*256;     /* maximum bytes/block */
                  /*** last block of rec subtract 256 byte trailer */
   /*     printf     ("firstbyte:%d,      samax:     %d,lastbyte: %d\n",byteno,samax,lastbyte);*/
   rval = shot[ns].range;
   if (rval > 0.0)      /* apply range-amplitude scale factoring */
        scale( ascale*pow(rval/r0,rf),tscale);
   else
         rval = erange++;
   rshift(rval);
   for (sams =1;sams (= samax; sams++) {
        rval = decode (&byteno,&lastbyte);
        if (rval > 99.0)
           break;
        /* printf ("amp: %7.3f\n",rval); */
        moveo(rval,(double)ival,&kpen);
        ival++;
        byteno += nchans*2;
        }
   kpen = 1;
   /* mark max amp +- 5.0 on axis
   moveo(5.0,10.0,&kpen);
   movep(5.0,0.0,&kpen);
   movep(-5.0,0.0,&kpen);
   movep(-5.0,10.0,&kpen); */
   pshot = &shot[ns];
   rval = -1.0*(125.0/tscale);
   label (0,(int)rval,pshot->shotno); /* posn @ 0(mv),-5/8(in) */
   pflush();    /* flush plotting buffer before next sample */
   return;
}
```

```
/* TREAD.C - to read or backspace a record on Alloy tape */

#include "tapeio.h"
tread(nrec)
int nrec;
     {
     extern PDCB ptape;
     extern char tbuff[];
     int te,im;
     static char *imode[] = {"backspacing ","reading rec "};
     static int recno = 0;

     /* printf ("%s tape at recno: %d\n", imode[(nrec(0?0:1)],recno); */

     if (nrec )= 0)     /* read a record */
        ptape->cmd = READ;
     else {     /* backspace a record on the tape */
        ptape->ba = 0;
        ptape->cmd = BR;
        }
      if ( (te = tape(ptape)) )
        {
        im = (nrec )=0?1:0);
        printf ("While %s :",imode[im]);
        terror ("Err on  tape rec ", recno);
        return(te);
        }
     if (ptape->status)
        check(ptape,"Status error", recno);
     if (te)
        return (te);
     (nrec )= 0? recno++:recno--);
     return(0);
}
```

```
/* TINIT.C - to initialize tape, read test record and rpheacer
             loads parameters from GPHEADR file when not present
             on the tape).  Also, prompts for series number.  */

#include "oplot.h"
#include "libc.h"
#include "tapeio.h"
tinit(otape)
PDCB otape;
{
   int srsno,te, rec,i;
   char c;
   extern char tbuff[];
   extern int tapeport,nblks,nsams,nchans;
   extern float secint;
   FILE *f1,*fopen();

        otape->port = tapeport;
        otape->status = 0;
        otape->size = 8208;

        otape->cmd = LENGTH;
        if ( (te=tape(otape)) || (te=tape(otape)) )
           {
           if (otape->status)
                te = check(otape, "tape LENGTH call", rec);
           if (te)
                terror ("tape LENGTH");
           }
        otape->cmd = READ;
        if ((te = tape(otape)) || (te = tape(otape)) )
                check(otape,"status err in header",1);


        /* read header & now list it */
                if  (tbuff[1]  == ' ' && tbuff[2] == ' ') /*  get next rec */
                     {
                      te = tape(otape);
                      if ( (te = check(otape,"tape ", 0)) )
                        terror("Tape init ",0);
                     }
                if (tbuff[1] == 'G' && tbuff[2] == 'P')  /* GPHEADR RECORD -
                                                  DUMP TO STDOUT) */
                     printf ("Retreiving series data from tape header.\n");
                else {
                   printf ("Opening GPHEADER file for tape info.\n");
                   f1 = fopen("GPHEADER","r");
                   if (f1 == NULL) {
                        printf ("Error opening GPHEADER. OPLOT aborted.\n");
                        exit(1);
                        }
                   else {       /* read the record into tbuff */
                        if (fread(tbuff[16],8192,8192,f1) != 8192)
                           {
                            printf("Fread failed.\n");
```

```
                exit(1);
                }
            }
        }
i = 0;
while (i (= MAXHEADRLEN) {
        if (tbuff[i] == NULL)
           {
           if (tbuff[++i] == NULL)
              break;
           }
        else if ((tbuff[i] ) 31 && tbuff[i] ( 127) || tbuff[i] == '\n')
           putchar(tbuff[i++]);
        else
           i++;
        }
putchar ('\n');
/* parse out nsams, nchans */
printf ("Specify series no: ");
scanf("%d",&srsno);

nsams = 256;
srsno = (srsno-1)*24;   /* srsno = nbytes offset from first series */
if ( (int) tbuff[7974 + srsno] -1)
   nsams = 128;
nchans = (int) tbuff[7953+srsno]/2;
printf ("Nsams/sec: %d   Nchans: %d\n",nsams,nchans);

nblks = tbuff[7966+srsno];
secint = (float) nblks*4096.0/(float) (nsams*nchans);
printf ("secint: %f\n",secint);
return;
}
```

SHOTD

Description

The program SHOTD may be used to create or update a disc file containing shot numbers, shot instants, and ranges for each shot.

Direction for use

To run the program enter:

SHOTD dev:filename.type

NOTE: The file named must not exist!!! There is a bug in the Manx Aztec C fopen call. It should append to an existing file, but it does not. All data are completely lost. Check before you enter the file name just to be sure it is an original.

The program then prompts the user as follows:

Enter Shot Number: enter the shot number which can be a string of ten or fewer characters or digits. **A period, ".", will stop the execution of the program.**

Enter T0: For first shot of each run, fields must be given an initial value. Enter:

y[DD]DD mDD dDD hDD MM SE.FS

where DD is a one or two digit integer value to be assigned to a field: where each field is specified by a character tag to identify it as follows: y[ear], m[onth], d[ay] or h[our]. The minute (MM) and second (SE.FS for seconds to the

hundredths) are entered without a field specifier.

For all subsequent shot data entered, the year, month, day, and hour will not change unless exolicitly changed by entering the field specifier and a new value before entering the minute and second of the shot instant.

If the user enters a oeriod (.) as the first character after the oromot, the orogram will restart the current shot and prompt the user for a new shot number and its corresoonding data.

Enter Range: The range in KM should be entered as a floating ooint value. If it is a fraction of a kilometer, **a leading zero must be entered.** Entering a oeriod (.) as the first character in resoonse to the oromot will cause a restart of the current shot entry.

The orogram then orints out on the terminal the values entered by the user, and the user is asked to verify the entry:

OK? A "y" flags the orogram to write the data to the soecified outout file. A soace will only cause the orogram to ask again. Any other resoonse causes the orogram to restart the current shot entry.

/* SOFTWARE */

```
/* ********** SHOTD ********** */
#include "a:jjfinc.h"
struct shot_dat{
        char *shot_label;
        int year,month,day,hour,min;
        float sec, range;
        } shotn = {"ABC",0,0,0,0,0,0.0,0.0};
char shotno[MAXCHARS];
char string[MAXCHARS],*strptr;
main(argc, argv)  /* SHOT_DAT to input shot data to file */
int argc;
char *argv[];
        {
        FILE *fp, *fopen();
        struct shot_dat *pshotn = &shotn;
        int stat;

        shotn.shot_label = &shotno;


        if (argc == 1) {
                printf("No output file specified, SHOT_DAT filename\n");
                exit(0);
                }
        if ((fp = fopen(argv[1],"a")) == NULL) {
                fprintf(stderr,
                        "shot_dat:  can't open %s\n", argv[1]);
                exit(1);
                }


        while((stat=getdat(pshotn)) != EOF)
                {
                if (stat == RESTART)  continue;
                if (verify(pshotn) == CONTINUE) fcopy (fp,pshotn);
                }
        exit(0);
        }

/* ********** VERIFY ********** */
verify(pshotn)
        struct shot_dat *pshotn;
        {
        printf("*******************************************\n");
        printf("Shot Number:  %s\n",pshotn->shot_label);
        printf("year: %d, month: %d, day: %d, hour: %d, min: %d, sec: %5.2f\n",
                 pshotn->year,pshotn->month, pshotn->day,
                pshotn->hour, pshotn->min, pshotn->sec);
        printf("Range:  %6.2f\n",pshotn->range);
        if (isok()) return (CONTINUE);
        return (RESTART);
        }
```

```
/* ********** GETDAT ********** */
getdat(pshotn)
        struct shot_dat *pshotn;
        {
        double getfloat();
        printf("*********************************************\n");
        if (prompt("Enter Shot Number: ") == EOF)
                return(EOF);
        strptr = string;
        getwdq(pshotn->shot_label);
        do {
        if (prompt("Enter T0:  ", string) == EOF)  return(RESTART);
           if (string[0] == '?')
                        printf("[ynnnn][mmn] [dnn] [hnn] min sec.fs\n");
        } while (string[0] == '?');
        strptr = string;
        while (*strptr != EOS)
                {
                switch (*strptr) {
                case '.':
                        return (RESTART);
                case ' ':
                case ',':
                        strptr++;
                        break;
                case 'y':
                        pshotn->year = getint();
                        break;
                case 'm':
                        pshotn->month = getint();
                        break;
                case 'd':
                        pshotn->day = getint();
                        break;
                case 'h':
                        pshotn->hour = getint();
                        break;
                default:
                        pshotn->min = getint();
                        pshotn->sec = getfloat();
                        break;
                }
                }
        strptr = string;
        if (prompt("Enter Range:  ", string) == EOF) return (RESTART);
        pshotn->range = getfloat();
        }
/* ********** FCOPY ********** */
fcopy (fp,pstruct) /* to copy a struct 'rec' to a file */
FILE *fp;
struct shot_dat *pstruct;
        {
        fprintf(fp,"%-8s%-5d%-4d%-4d%-4d%-4d%-8.2f%-8.2f\n",
                pstruct->shot_label,pstruct->year,pstruct->month,
                pstruct->day,pstruct->hour,pstruct->min,
```

```
        pstruct->sec,pstruct->range);
}
```

APPENDIX A

Magnetic Tape Format

for the Ocean Bottom Seismometer

The tape format conforms to the Tape Interchange Package (TIP) format described in the Operator's Guide #OG - 100055 rev 0981 published by the Alloy Engineering Company, Inc. , 12 Mercer Rd. , Natick, MA 01760.

There are some extensions to the TIP format used in the U.S. Geological Survey Ocean Bottom Instrument Package tape format. These include:

1) The stipulation that all records must be 8208 bytes in length.

2) The presence of a test record as the first record on the tape.

3) The presence of a general purpose header as the second record on the tape.

4) Note that in the OBIP program, the word 'file' is used to describe all the records up to and including the record whose 'Last Block' flag is set. End-of-file marks are not used for this purpose.

5) The last 256 bytes of the final record of each file are reserved for the storing of series and experiment parameters.

6) Note that in the OBIP program, the word 'block' is used for what is normally referred to as a 'record' by the rest of the world .

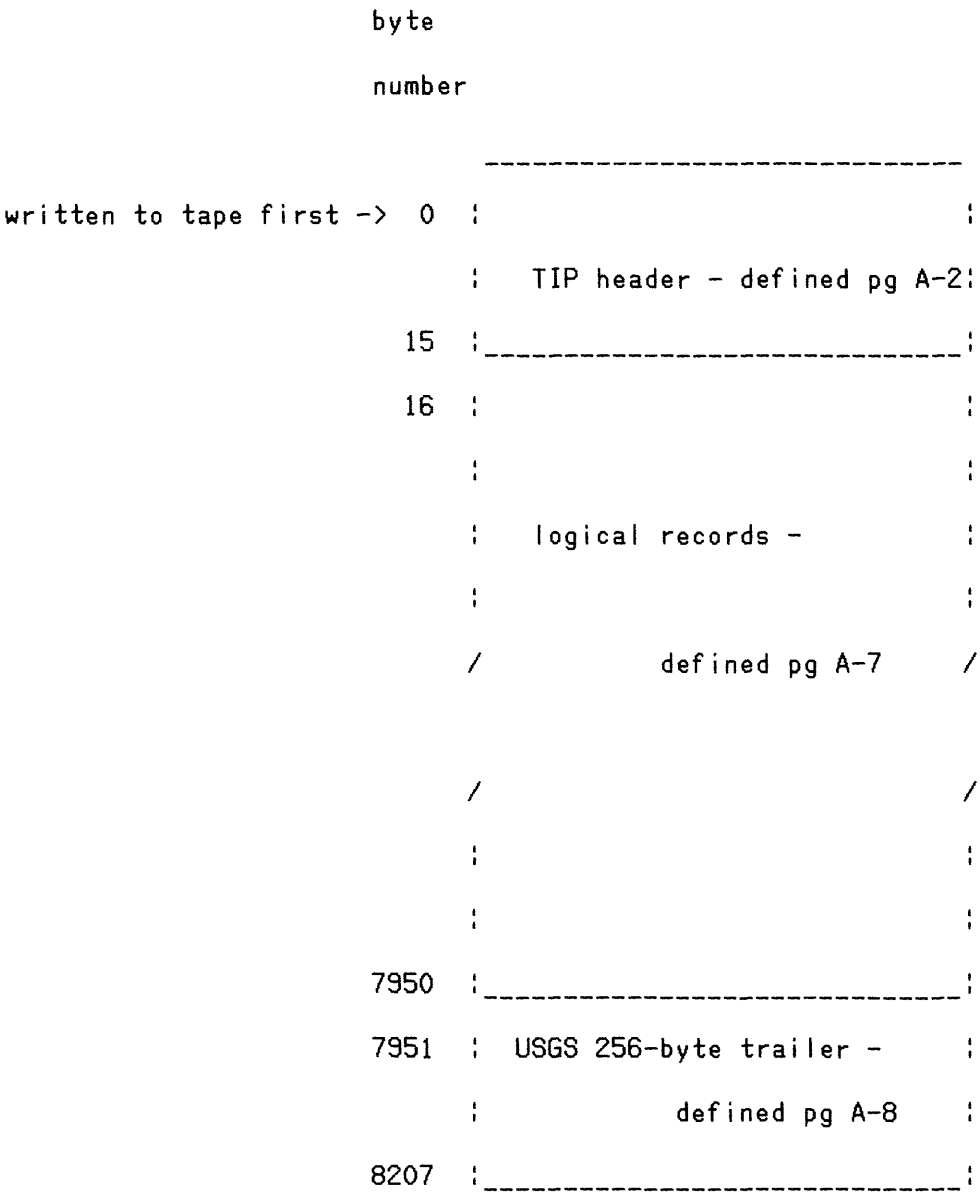
7) An end-of-file (EOF) mark is present at the beginning of each of the last three tracks on the tape. The first track is started with a TIP beginning-of-tape mark. The use of two end-of-file marks indicates the end of data on the tape.

The 16-byte TIP record header (file control block) is formatted as follows:

Byte #	Description
0	always equals zero
1 - 8	ASCII file name - can be anything provided it uniquely defines each file (exceptions: record #1 (test record)- must be all blanks. record #2 (GP header)- must be 'GPHEADER'
9 - 11	ASCII file type (or extension) - this can be anything
12	always equals zero
13	Last Block Flag. Signals that a particular block is the last one of a file. Value of 1 = last block, 0 = more follows. Must equal 1 in test record and General Purpose Header.
14	always equals zero
15	equals the number of 128-byte sectors containing active data in this record. The maximum value is 64. (40H)

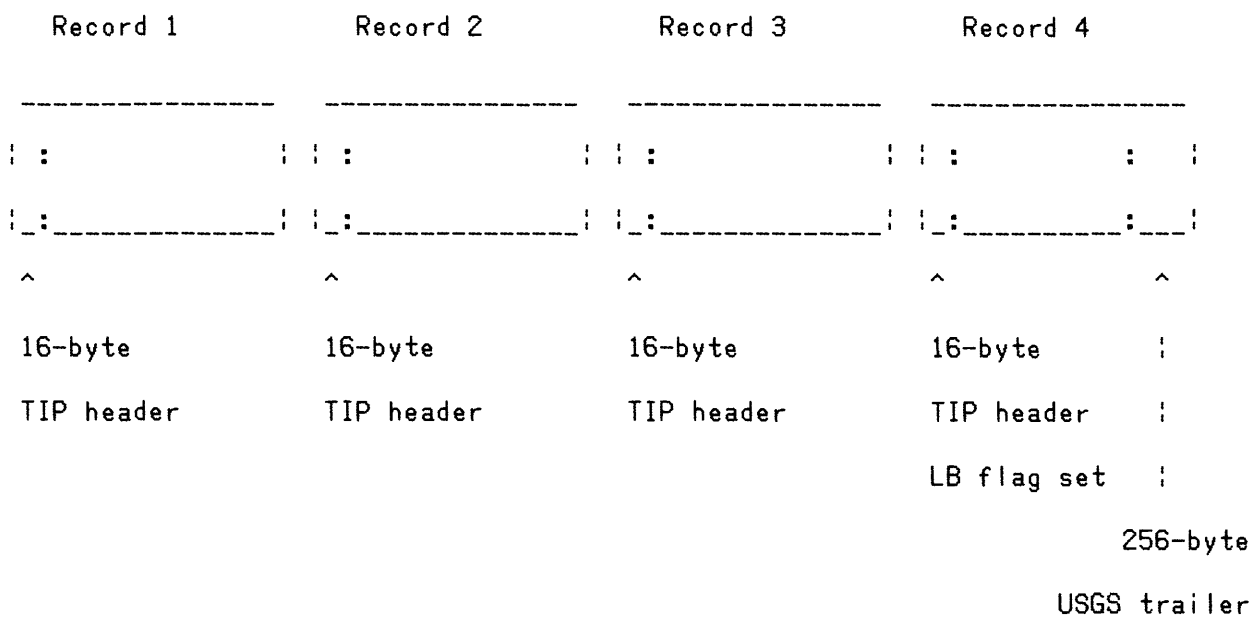
Discussion of Physical Records:

Every physical record on the tape will be 8208 bytes long. End-of-file (EOF) marks are used only to mark the start of the last three tracks on the tape. And, a double end-of-file indicates the end of data on the tape. Files are ended in the TIP format by setting the 'Last Block' flag in the TIP header. The TIP records are laid out like this:



The USGS OBIP program allows the investigator to choose to write 1, 2 or 4 physical records per file.

An OBIP example: If it is decided to write four TIP records per file, it must be noted in the General Purpose Header record 'Buffer Size' field. Then, four TIP physical records will be recorded in main memory. Note that the 'Last Block' flag must be set in the fourth record header. Finally, the file will be written to tape, looking like:



Format of the General Purpose Header Record:

The following items are written in the General Purpose Header record .

1) Cruise information, Chief Scientist, Ship Name, etc.
   These data start at address 16 and can continue to address 7950.
   They are stored in ASCII and include all carriage returns and line feeds. A double NULL (0) must be used to signal the end of this section.

Starting at address 7952, series parameters for each series are written every 24 bytes for up to 9 series. The address "base + n" specifies an address relative to the "base" address which is 7952,7976,7800,etc.

2) The number of channels times two is stored at address <base + 1>

3) The series type is stored at address <base + 2> and may be either E (for event mode) or T (for timed window mode).

4) The number of experiments to be recorded in the given series is stored at address <base + 3>. It may be any integer between 1 and 99.

5) The start time and the stop time for the given series is stored in bytes <base + 4> through <base + 13> as year, month, day, hour and minute. The start time is written first, then the stop time. All components are written in packed BCD(two chars/byte).

6) The number of blocks per file (physical records per experiment) is stored in byte <base + 14>.

7) The most significant byte of the maximum samples per experiment is stored in <base+18> and the least significant byte is stored in <base + 19>.

8) For a series which has been recorded in the window mode, byte <base + 20> will contain the offset in seconds.

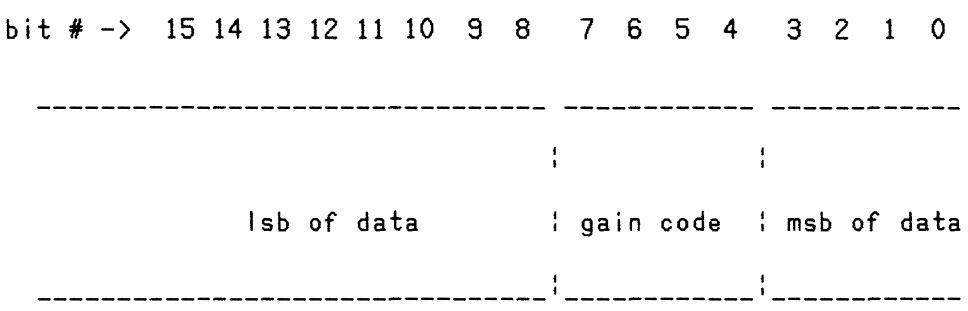
9) Byte <base + 21> will contain the period of the sampling in minutes.

The software for reading and printing the series parameter data can be found in the subroutine DIRCT of the program TRANSCRB.

Specification of Data :

The channel data are interlaced within a given TIP file. Since the number of channels per sample and the sampling rate must remain constant within a given series, it poses no problems in reading the data. The data are decoded using the routine DECODE which is used in the program OPLOT.

Each sample is written as follows:



Specification of the 256-byte USGS trailer:

The following data will be written into the 256-byte trailer and will only be recorded when a file is written. Please note that in the table below, all byte numbers start at 0. Each byte of the time contains one BCD digit in its least significant 4 bits EXCEPT for the year which consists of two packed BCD digits.

byte number relative to start of trailer	absolute byte number in TIP record	description
221	8173	time - tenths of seconds
222	8174	- unit seconds
223	8175	- tens of seconds
224	8176	- unit minutes
225	8177	- tens of minutes
226	8178	- unit hours
227	8179	- tens of hours
228	8180	- unit days
229	8181	- tens of days
230	8182	- day of week (may be blank)
231	8183	- unit months
232	8184	- tens of months
233	8185	- year (two packed BCD)
234 - 235	8186 - 8187	- millisecond timer (LSB followed by MSB - See TFIND for decoding)

APPENDIX B

Tape Utilities (9-track and cartridge)

Basic procedure to communicate with Alloy interface for I/O to either 9-track or cartridge magnetic tape drives.

All I/O operations consist of two calls to 'tape_':

1. Initializes the I/O operation and returns to the calling procedure without making a check as to the successfull completion of the operation.
2. Tests for the completion of the I/O operation and returns status information as well as data in the case of input.

This double execution method allows for interlaced I/O if more than one Alloy interface card is employed.

Execution (in 'C'):

```
struct .... dcb;
int t, tape();
  ....
     t = tape(&dcb);
```

't' will be set to a negative number if a time out condition the 'dcb' status values 'is' and 'ds' must be examined. Remember that these are only set on the second call. Except for a time out value of -268 the calling program can consider the time out return fatal. The -268 return may be caused by rewind or long record or file spacing operations and the calling routine may try again.

```
  See ALLOY.MAC for definition of structure 'dcb'


\
        .z80
        cseg
include ALLOY.MAC               ;MACROS, etc.
SREG    equ     1               ;port status register
LONG    equ     1
```

Basic entry point.

This section saves necessary registers based upon Aztec 'C' register preservation requirements ('bc', 'sp' and 'ix').

The 'ix' register is established as a pointer to the current dcb.

Whether this is a first or second entry is determined by the sign bit in 'dbc->status'.

```
\
;*****************************************************************
tape_::
        ld      hl,2            ;add 2 to
        add     hl,sp           ;  'sp' to get single arg
        push    ix              ;save 'ix'
        push    bc              ;  and 'bc'
        ld      (stksav),sp     ;save current stack position
        ld      sp,hl           ;put computed loc. in 'sp'
        pop     ix              ;get structure pointer
        ld      sp,(stksav)     ;restore stack
        ld      a,(ix+PORT)     ;get port base
        add     a,SREG          ;offset to status port
        call    iosset##        ;tell status checking about it
        ld      a,(ix+STATUS)   ;check if this is continuation
        rlca                    ;  call
        jp      c,nextops       ;yes
        jp      tapeops         ;no, initial command call
;
;*****************************************************************
return:                         ;return point, 'hl' always
                                ;  contains return code
        ld      a,h             ;set z flag
        or      l               ;to value
        ld      sp,(stksav)     ;make sure stack ok
        pop     bc              ;restore 'iy'
        pop     ix              ;restore 'ix'
        ret                     ;go home
```

```
;*****************************************************************
;       First entry operations
;*****************************************************************
\
tapeops:
        OUTCHECK -256
        SETPORT 0
        ld      a,(ix+DRIVE)    ;get tape drive
        out     (c),a           ;send MA
        OUTCHECK -257
        SETPORT 1
        ld      a,(ix+PA)       ;set repeat factor
        out     (c),a
;
        ld      a,(ix+CMD)      ;get command
        and     0fh             ;mask out misc.
        ld      e,a             ;save units in e
        rlca                    ;shift left one
        add     a,e             ;  now X 3
        ld      e,a             ;put into
        ld      d,0             ;  'de' to compute
        ld      hl,swtcha       ;first switch
        add     hl,de
        or      080h            ;set flag and
        ld      (ix+STATUS),a   ;   save value for next call
        jp      (hl)            ;go to operation
swtcha: jp      doca            ;NOP
        jp      doca            ;read
        jp      write           ;write
        jp      doca            ;write eof
        jp      doca            ;forward record
        jp      doca            ;forward file
        jp      doca            ;back record
        jp      doca            ;back file
        jp      doca            ;status
        jp      length          ;set record length
        jp      write           ;write with range check
        jp      search          ;search under mask
        jp      doca            ;forward record
        jp      doca            ;forward file
        jp      doca            ;back record
        jp      doca            ;back file
;
length:
        ld      a,(ix+SIZE+1)   ;flip order of
        ld      (temo),a        ;  bytes
        ld      a,(ix+SIZE)
        ld      (temp+1),a
        ld      hl,temo         ;set source to 'temo'
        ld      de,2            ;set count to 2
        jp      wr..0           ;do remainder as write
search:
        jp      doneok
;
write:                          ;write record
```

```
        ld      d,(ix+SIZE+1)   ;get length of record, mod 256
        ld      e,(ix+SIZE)
        ld      h,(ix+BUFFPTR+1) ;get buffer address
        ld      l,(ix+BUFFPTR)
wr..0:  SETPORT 3
wr..1:  dec     de              ;dec. count
        bit     7,d             ;check if underflow
        jp      nz,doca         ;  yep! do command
        OUTCHECK -266
        outi                    ;outout data
        jr      wr..1           ;still more
;
doca:   OUTCHECK -267
        SETPORT 2
        ld      a,(ix+CMD)      ;get command
        out     (c),a           ;and do it
;
        jp      domeok          ;return for now
```

```
;*****************************************************************
;       return point for second half of i/o operation
;*****************************************************************
\
nextops:
        srl     a               ;shift back right with 0
        ld      e,a             ;set uo
        ld      d,0             ;  for branch
        ld      hl,swtch2       ;    operation
        add     hl,de
        jp      (hl)
swtch2: jp      stat            ;NOP
        jp      read            ;read
        jp      stat            ;write
        jp      stat            ;write eof
        jp      stat            ;forward record
        jp      stat            ;forward file
        jp      stat            ;back record
        jp      stat            ;back file
        jp      stat            ;status
        jp      doneokc         ;set record length
        jp      stat            ;write with range check
        jp      stat            ;search under mask
        jp      stat            ;forward record
        jp      stat            ;forward file
        jp      stat            ;back record
        jp      stat            ;back file
;
stat:
        call    stats
        jp      doneokc
;
read:
        xor     a               ;clear SIZE
        ld      (ix+SIZE),a     ;  to zero
        call    stats           ;get status
        and     040h            ;check status return before
        jp      z,doneokc       ;  with more inout.
        INCHECK -269
        in      e,(c)           ;get length
        ld      (ix+SIZE),e     ;save in SIZE
        INCHECK -269
        in      d,(c)
        ld      (ix+SIZE+1),d
        ld      h,(ix+BUFFPTR+1);set uo inout
        ld      l,(ix+BUFFPTR)  ;buffer
rd..0:  dec     de              ;dec. count
        bit     7,d             ;check if underflow
        jp      nz,doneokc      ;  yep! do command
        INCHECK -270            ;input data
        ini                     ;  loop
        jr      rd..0
;
doneokc:                        ;second call done ok
        xor     a               ;clear status
```

```
        ld      (ix+STATUS),a
comeok: ld      hl,0            ;ok return
        jp      return
;
;
stats:
        INCHECK -268,LONG
        SETPORT 2
        in      a,(c)           ;get DS
        ld      (ix+DS),a       ;put in structure
        INCHECK -269
        in      a,(c)           ;get IS
        ld      (ix+IS),a
        ret
;*****************************************************************
        end
```

```
        MACRO and basic definitions for cartridge and tape
        assembler routines.


  Data structure pointed to by 'ix' register
  This structure is created and maintained by the calling
  routines.

\
PORT    equ     0               ; i/o base port
CMD     equ     PORT+1          ;command
DRIVE   equ     CMD+1           ;drive/track number
PA      equ     DRIVE+1         ;repeat parameter
DS      equ     PA+1            ;drive status
IS      equ     DS+1            ;interface status
STATUS  equ     IS+1            ;status
SIZE    equ     STATUS+1        ;record size
BUFFPTR equ     SIZE+2          ;buffer pointer
;
SETPORT macro   portoff         ;set port in 'c'
        ld      a,portoff
        add     a,(ix+PORT)
        ld      c,a
        endm
;
INCHECK macro   errno,longy     ;check input port status
        local   con
   ifb  (longy)
        call    icheck##        ;call register check routine
   else
        call    lichck##        ;long register check routine
   endif
        jr      c,con           ;ok so skip
        ld      hl,errno        ;not ok, so load error no.
        jp      return          ; and return
con:
        endm
;
OUTCHECK macro  errno   ;check output port status
        local   con
        call    ocheck##        ;call register check routine
        jr      c,con           ;ok so skip
        ld      hl,errno        ;not ok, so load error no.
        jp      return          ; and return
con:
        endm
;
;       work storage area
        common  /ALLOYS/
stksav: ds      2               ;return point stack value
```

```
bcstat: ds      2               ;'bc' reg for status
temp:   ds      2               ;just for grins
        cseg
:
```

Support routine CARTLIB.MAC

Routines to get and check interface status.

The principle purpose of this set of routines is to poll the Alloy interface card port status register for a condition indicating that the card can receive or transmit data. In addition, this polling is limited so that the CPU won't lock-up on indefinite tries if something fails and will eventually return to the calling routine if the I/O card fails to respond.

Entry points:

lichck - does long test on input flag

icheck - tests input flag

ocheck - tests output flag

In all cases the carry flag is set if the device is ready.

The carry flag is zero if not ready after 256 tries (eg. the device 'timed out'). 256*32768 tries are made in 'lichck'.

On input:

'ix' register contains pointer to drive DCB

On output

the 'a' register is lost and meaningless.

```
\
        .z80
include ALLOY.MAC
;
lichck::
        push    bc              ;save 'bc'
        ld      bc,7fffh        ;make big count
lic..0: call    icheck          ;call input check
        jr      c,lic..1        ;OK, go back
        dec     bc              ;decriment long count
        bit     7,b             ;did we underflow?
        jr      z,lic..0        ;no.  try again
lic..1: pop     bc              ;restore 'bc'
        ret                     ;return
;
icheck::
        push    bc              ;save 'bc'
        ld      bc,(bcstat)     ;get stat port
ich..0: in      a,(c)           ;get status
        rrca                    ;put flag
        rrca                    ;  into carry
        jr      c,ich..1        ;jump out if on
        djnz    ich..0          ;try again if off
                                ;fall out means failure
ich..1: pop     bc              ;retore 'bc'
        ret                     ;return
;
ocheck::
        push    bc              ;save 'bc'
        ld      bc,(bcstat)     ;get stat port
och..0: in      a,(c)           ;get status
        rrca                    ;put flag into  carry
        jr      c,och..1        ;jump out if on
        djnz    och..0          ;try again if off
                                ;fall out means failure
och..1: pop     bc              ;retore 'bc'
        ret                     ;return
;
;       entry to set 'bcstat' for later status checks
;       'a' contains status register value
;       all registers restored on return
iosset::
        push    bc              ;save 'bc'
        ld      c,a             ;put port into c
        ld      b,0             ;clear 'b' for loop
        ld      (bcstat),bc     ;save
        pop     bc              ;restore 'bc'
        ret                     ;all done
;
        end
```

```
/*  TAPEIO.H */

/*    Basic control information for cartridge/tape routines
*/
typedef struct {          /* comm. structure to cart I/O proc.   */
     char port;           /* i/o port number */
     char cmd;            /* command
          =  0 : NOP
          =  1 : read
          =  2 : write
          =  3 : write EOF
          =  4 : forward record(s)
          =  5 : forward file(s)
          =  6 : reverse record(s)
          =  7 : reverse file(s)
          =  8 : current status                                  */
     char drive;          /* drive/track                         */
     char pa;             /* repeat factor skip pos              */
     char ds;             /* drive status                        */
     char is;             /* interface status                    */
     char status;         /* 'tape' routine status               */
     int size;            /* block size                          */
     char *buffer;        /* pointer to buffer                   */
}    DCB, *PDCB;

#define READ 1
#define WRITE 2
#define WEOF 3
#define EOTRACK 16
#define LENGTH 9
#define EOFT 14
#define SKIPF 5
#define SKIPR 4
```

```
/* check - check status of dcb */
#include "libc.h"
#include "tapeio.h"
check(dcb, str, rec)
PDCB dcb;
char *str;
int rec;
{
     static char *mess[] = {
          "a FLAG condition exists - do RW",
          "write to protected drive",
          "cmd. to non-reset drive",
          "drive failed to respond",
          "??? - unspecified",
          "??? - unspecified",
          "file mark write verify error",
          "transport abort prior to completion",
          "read fail - hard error",
          "read fail - CRCC check",
          "read fail - short record",
          "read fail - vertical parity",
          "write fail - R-A-W error",
          "write fail - read data not detected",
          "read file - file mark detected",
          "??? - unspecified"
          };
     static char *amess[] = {
          "Abort w/o attempt",
          "Abort with attempt",
          "Syntax rejection or parity"
          };
     int cs, ec;

     if (!dcb->status) {
          printf("CHECK: not in status mode, dev:%s\n", str);
          return 0;  /* not in status mode */
     }
     if (ec = tape(dcb)) {
          terror(str, rec);
          return 32;
     }
     cs = (dcb->is & 0x30) >> 4;
     ec = dcb->is & 0xf;
     if (cs != 0) {
          printf("\nerror:%s", amess[cs-1]);
          if (cs != 3)
               printf(" - %s", mess[ec]);
          printf(", IS-DS-CA:%xh %xh %xh",
               dcb->is,dcb->ds,dcb->cmd);
          if (!ec) /* use unused number */
               ec = 4;
     } else
          ec = 0;
     if (cs != 3 && dcb->ds & 4) { /* at end of track */
          printf(" end of track");
```

```
            ec += 16;
        }
        if (ec)
            printf("\n------)) dev:%s, rec:%d\n",str,rec);
        return ec;
}


terror(str, rec)
char *str;
int rec;
{

        printf("time-out error for %s; record %d\n",
            str, rec);

}
```

```
/* keybrk - check console for abort */
#include "libc.h"
keybrk()
{
      int c;

      c = 0;
      while (bdos(11,0) && (c = bdos(1,0)) != 3) ;
      if (c == 3) { /* ^C */
            orintf("\nconsole abort\n");
            return -1;
      }
      return 0;

}
```

APPENDIX C

Miscellaneous Utilities

```
/* ********** JJFINC.H - include file for jjlib utilities ********** */

#include "a:stdio.h"
#define EOS  '\0'
#define CONTINUE 1
#define RESTART 10
#define MAXCHARS 80
#define YES 1
#define NO 0
```

```
/* JJLIB.C */
#include "jjfunc.h"
extern char string[MAXCHARS],*strptr;
/* ********** GETINT ********** */
getint()
        {
        int ival = 0;
        while(!(isdigit(*strptr)))
                {
                if(*strptr=='\0') return (0);
                strptr++;
                }
        ival = atoi(strptr);
        while((isdigit(*strptr))) strptr++;
        return (ival);
        }
/* ********** GETFLOAT ********** */
double getfloat()
        {
        double value;
        double atof();
        while (!(isdigit(*strptr)))
                {
                if (*strptr == '\0') return (0.0);
                if (*strptr == '.') break;
                strptr++;
                }
        value = atof(strptr);
        while ((isdigit(*strptr)) || *strptr == '.') strptr++;
        return (value);
        }
/* ********** PROMPT ********** */
prompt(prmpt)
        char *prmpt;
        {
        extern char string[];
        char *gets();
        printf("%s", prmpt);
        if (gets(string) == EOF) return(EOF);
        if (string[0] == '.') return (EOF);
        return (CONTINUE);
        }
/* ********** ISOK ********** */
isok() /* to get Y or N from user to validate data */
        {
        char c;
        strptr = string;
        for (prmpt("OK?  ",string);*strptr == ' ';prompt("OK? ",string));
        if (*strptr == 'y' || *strptr == 'Y')  return (YES);
        else return (NO);
        }
```

```
extern char string[];
extern char *strptr;
getwdo(str)
char *str;
{
        /* while non-white alphanumeric copy *strptr to *str */
        while (*strptr != '\0')
        {
                if (*strptr (= 32 || *strptr )= 127)
                {
                        *str = '\0';
                        break;
                }
                *str = *strptr;
                str++;
                strptr++;
        }
        return;
}
```