The design and implementation of the Durham university seismic processing system

Poulter, Michael John

How to cite:

Poulter, Michael John (1982) The design and implementation of the Durham university seismic processing system, Durham theses, Durham University. Available at Durham E-Theses Online: http://etheses.dur.ac.uk/7778/

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full Durham E-Theses policy for further details.
The Design and Implementation
of the Durham University
Seismic Processing System

By

Michael John Poulter

A thesis submitted for the Degree of
Doctor of Philosophy at the University of Durham

Graduate Society March 1982

The copyright of this thesis rests with the author.
No quotation from it should be published without
his prior written consent and information derived
from it should be acknowledged.
A NERC Research Grant, in late 1978 permitted the Department of Geological Sciences, at the University of Durham, to purchase a pdp 11/34 minicomputer system. Together with a pdp 8/e, already possessed by the Department, this system was intended to fulfill two roles; provide a computer tool for research work into seismic reflection methods and provide a system for production processing of seismic reflection data acquired by the Department, mainly as a result of marine geophysical investigations.

This thesis describes the design of Systems level software, and its implementation, to allow the computer systems to be easily used as a general research tool, and the design and implementation of a suite of programs, to provide the basic facilities of a seismic reflection processing system. At the end of this work it was possible to reach a number of conclusions on how both the hardware and software could be developed to provide a more powerful system for the future.
Acknowledgments

I would like to thank the NERC for funding this project and Professor Bott for providing me with the opportunity of carrying out this work in the Department of Geological Sciences at the University of Durham. I would also like to thank the members of staff of the Department, especially Neil Goulty and my supervisor Harry Peacock for all their help. In a technical project, such as this, one is always at the mercy of the equipment and so I would like to extend many thanks to the technical staff of the Department, in particular George Ruth and Dave Asbery, for helping me to keep it all working.

In difficult times, one is always lucky to be surrounded by friends and so I extend my thanks to the many fellow students who provided me with help and advice, with a special mention for Alan Nunns and Tom Armstrong, whose example helped me to complete this work.

A special vote of thanks is due to my employers, British Petroleum Ltd, for allowing me to use their facilities in producing the text and diagrams in the thesis.

Finally the most important thank you must go to my Wife, Anne, without whose patience and loyal support none of this would have been possible.
Contents

Chapter 1: Introduction 1

Introduction To Hardware 1

Project Aims 2

Scope of the Work 4

Chapter 2: Seismic Reflection Principles 6

Data Acquisition 7

Recording 9

Processing 11

Summary 18

Chapter 3: Hardware and Systems Software 20

Hardware 20

Systems Software 23

Linking the Two Minicomputers 26

RT-11 to OS/8 Transfer 27

Floating Point Transfer 28
Chapter 4: Seismic Processing Software

Overview

Overall Design Considerations

Data Format

Data Input and Output

Seismic Processing System

Synthetics

Demultiplex
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>63</td>
</tr>
<tr>
<td>Pre-Stack Data Analysis</td>
<td>65</td>
</tr>
<tr>
<td>Pre-Stack Processing</td>
<td>67</td>
</tr>
<tr>
<td>Edit</td>
<td>68</td>
</tr>
<tr>
<td>Polarity Reversal</td>
<td>69</td>
</tr>
<tr>
<td>Gain Application</td>
<td>69</td>
</tr>
<tr>
<td>Muting</td>
<td>71</td>
</tr>
<tr>
<td>Frequency Filtering</td>
<td>72</td>
</tr>
<tr>
<td>Bandpass Filtering</td>
<td>73</td>
</tr>
<tr>
<td>Bandreject Filtering</td>
<td>73</td>
</tr>
<tr>
<td>Deconvolution</td>
<td>74</td>
</tr>
<tr>
<td>Wiener Spiking Deconvolution</td>
<td>75</td>
</tr>
<tr>
<td>Prediction Error Deconvolution</td>
<td>80</td>
</tr>
<tr>
<td>Trace Normalisation</td>
<td>82</td>
</tr>
<tr>
<td>Pre-Stack Processing Summary</td>
<td>83</td>
</tr>
<tr>
<td>Velocity Analysis and Stacking</td>
<td>84</td>
</tr>
<tr>
<td>Coherence Velocity Analysis</td>
<td>87</td>
</tr>
<tr>
<td>Velocity Analysis Display</td>
<td>89</td>
</tr>
<tr>
<td>NMO corrected gathers</td>
<td>90</td>
</tr>
</tbody>
</table>
Chapter 5: Using The System

Standalone Interpolation 93
CMP Stacking 94
Post-Stack Processing 96
Edit 96
Gain Ramps 97
Mute 97
Deconvolution 97
Frequency Filtering 98
Normalisation 99
Post-Stack Program:- Summary 99
Post-Stack Mix 100
Stacked Section Plotting 101
Header Interrogation 102
Migration 103
Kirchhoff Migration 104
Finite Difference Migration 112
Summary 119

Chapter 5: Using The System

Chapter 5: Using The System

Chapter 5: Using The System
Data Flow
Jan Mayen Data Processing
Jan Mayen Processing
Caribbean Arc 1980
Caribbean Processing Parameters
Summary

Chapter 6: Conclusions

Future Software Development
Demultiplex
Data Exchange Format
Land Data
Other Possible Software Additions
Vibroseis
Improved Filtering
Amplitudes
Summary
Hardware Evolution
Future Evolution Path
Summary

References

Appendix 1

Demultiplex: MPDMXA

Sort: MPSORT

Pre-Stack Processing: MPPRST

Interactive Spectral Analysis Program: MPFANL

Velocity Analysis: MPVEL

NMO corrected gathers: MPCDP

CMP Stacking: MPSTAK

Post-Stack Processing: MPPOST

Post-Stack Trace Mix: MPTMIX

Trace Sequential - Time Slice: MPSLIC

Time Slice to Trace Sequential: MPUSLC

Finite Difference Migration: MPFD15 and MPFD45

Kirchhoff Migration Operator Design: MPOGEN

Kirchhoff Migration: MPKMIG
Stand Alone Interpolation :- ANINT

Internal Header Interrogation :- MPHIST

Synthetic CMP Gathers :- ANSEI

Synthetic Sections :- MPSYNS

Velocity Analysis Display :- MPVCON

Section Plotting :- MPSPLT

Section Plot Background :- MPPBLK

Section Plotting Merge and Post Process :- MPMERG

Gather Plotting(Small Trace separation) :- MPSPLI

Gather Plotting(Large Separation) :- MPGPLI

Quick Raster Plot Processor :- MPPROC

General Purpose Raster Merge and Output :- MPMERP

Appendix 2:

Tape Subroutine Descriptions

Floating Point Transfers

Extended Memory Input/Output Routines

Microcode Routines

Plotting Subroutines
Chapter 1

Introduction

Introduction to Hardware

The Multichannel Seismic Reflection technique is probably the most important single geophysical exploration tool in full time use. However, in exploiting the method fully vast quantities of digitally recorded data are produced. The basic aim of the method is to produce a display from this mass of data which will enable a geological interpretation. In order to produce a final section from which geological information may be derived, several quite sophisticated processing techniques are applied to the data, using powerful computer systems.

The Department of Geological Sciences of the University of Durham has interests in two aspects of the multichannel seismic reflection technique. First, with an established program of marine geophysical investigation, which in the past had utilised single channel seismic reflection, it was only natural that the department would want to employ the multichannel seismic reflection method as a tool in its geophysical investigations. Secondly, research into seismic acquisition and processing methods had been undertaken in an attempt to improve the techniques employed within the seismic reflection method.
Prior to October 1978 research projects investigating multichannel seismic methods have been undertaken using limited amounts of synthetic data on the NUMAC IBM 370. Also small amounts of marine data had been processed using the facilities of the departmental seismic refraction laboratory. However these computer systems were not capable of handling large quantities of seismic reflection data.

Some data had been processed by geophysical companies, but it was fairly obvious that a specialist computer system was necessary to allow seismic reflection data to be handled in the department.

The computer hardware forming the building blocks for this system was provided by a NERC grant in October 1978. The equipment purchased with this grant was a Digital Equipment Corporation pdp 11/34, a FPS AP120B array processor, a Pertec 20 Megabyte disc drive and a Versatec printer/plotter. Together with a DEC pdp 8/e and three tape transports already owned by the department, this equipment was assembled into the hardware configuration shown in Fig 1.1.

Project Aims

The software available with the two minicomputers consisted primarily of the operating systems, Assemblers, Fortran compilers and editing and file handling utilities, as would be expected with most minicomputer systems.
Fig 1.1: Hardware configuration of the Durham University seismic processing system.
The main purpose of this project was to provide the basic software for processing seismic reflection data. This software package was to contain most of the basic techniques which would be available on the commercial systems used in industry, so that it would be capable of producing a geologically interpretable, final section from field tapes. Most systems utilised by industrial concerns employ considerably more powerful computers, and those that are based on minicomputers, such as the TIMAP systems, provide a very specialised operating environment, orientated almost solely around seismic data processing. In this respect a different set of design criteria had to be adopted from those which would have been applied in developing a commercial seismic processing system.

It was important that this system should retain much of the flexibility of a general purpose computing system as well as being capable of providing a reasonable data throughput when processing seismic data.

As can be seen from Fig 1.1 an ultrasonic tank for producing synthetic data is attached to the system, via an interface with the pdp 8/e. It was envisaged that this and the other specialised peripherals on the system, such as the Array Processor (AP) and the electrostatic plotter, should be available as easily used tools, to aid research work dealing with seismic reflection methods.

Therefore, once the computer hardware had been installed, software was developed which would allow reasonably rapid processing of seismic reflection data, while at the same time, retaining all the flexibility and power provided by the basic
minicomputer system. A final constraint on the design of the software was that it had to be easily updated. The hardware in existence is envisaged as only the starting point for what should be a continually evolving system. Hence the software design had to take account of the desirability of minimizing software modifications in the event of any hardware upgrades. It was anticipated that the design of the software, on completion would help indicate the areas where hardware upgrades could bring about increase in speed and flexibility, with a minimum of software effort.

Given the constraints mentioned above, it was obvious from the outset that it was necessary to produce software of two quite different types: systems level software and utilities to enable efficient use to be made of all the peripherals attached to the system and allow data transfers between the two minicomputers, and software concerned solely with the processing of seismic data. Although it was necessary for some of the systems level software to be machine specific, the remaining software was designed to be as machine independent as possible.

Scope of the work

This thesis documents the design and implementation of the software which was developed for the minicomputer system as described above. A brief description of the principles of seismic reflection processing is given in chapter 2, in order to provide a basic introduction to the techniques necessary in a seismic processing system. Chapters 3 and 4 contain descriptions of the
systems level and seismic software, respectively. A brief description of how it is envisaged that the system should be used, together with a description of two test runs is given in chapter 5. An objective appraisal of the system, as developed, with suggestions for improvements and a possible evolution path for both the software and the hardware is given in chapter 6. Listings of the software, together with descriptions of their input parameters, are provided in the appendices.
Chapter 2
Seismic Reflection Principles

The seismic reflection method is probably the most widely used geophysical tool, and certainly produces the largest quantities of data. As warrants such an important technique the principles behind it are well explained in standard texts (Waters, 1978; Dobrin, 1977) and review papers (O'Brien, 1977). However, it is well worth a brief consideration of some of the principles of acquiring and processing modern seismic reflection data, in order to introduce some of the considerations involved in designing a basic processing system. Therefore a descriptive account of seismic methods is given in this chapter and the theory of the methods applied is given in chapter 4.

Although more and more data is being acquired using three dimensional techniques, the vast majority of seismic reflection data is still acquired in the form of two dimensional profiles, and this is the type of data which will be considered in this thesis.
Data Acquisition

The advent of quick and flexible digital computers in the mid-1960's was probably the main driving force behind the almost universal acceptance of the Common Mid Point (CMP) method (less accurately often referred to as Common Depth Point, CDP method) of seismic data acquisition.

The basic principle of the method is shown in Fig 2.1. In a horizontally stratified medium, shots and receivers can be arranged on the surface such that the seismic ray paths from the shots to the receivers, have impinged on the same subsurface point, below a common midpoint on the surface. After suitable adjustment to take into account the different travel times of primary reflections for different shot-receiver offsets (NMO correction), the seismic traces can be added together (stacked) to produce a trace with improved signal-to-noise ratio, the primary events having been reinforced relative to the noise. This stacked trace can be displayed as a single trace at the Common Mid-Point.

The random noise on the traces, when summed over N traces gives a reduction in amplitude of $N^{0.5}$, while the primary events linearly reinforce. Therefore there is a resultant increase in the signal to noise ratio of $N/(N^{0.5})$ (Meyerhoff, 1966). An added bonus of the method is that secondary reflections (Multiples) are not aligned by the NMO correction used to align the primaries and so they also tend to be reduced in amplitude by stacking.
Fig 2.1: Common Mid-Point Geometry

Fig 2.2: Travel time relation in CMP geometry
This method had been applied, to a limited extent, with the manual manipulation of analogue recordings during the early 1960's, but with the introduction and acceptance of digital recording and processing techniques, the full potential of the method was realised and it has since gained almost universal acceptance.

The field acquisition layouts for the CMP method are relatively straightforward and really quite ingenious, especially at sea (Figs 2.3, 2.4). On land, a recording truck is linked to a transmission cable which is made up from several shorter segments. Arrays of geophones are attached to this cable at regular intervals along the surface, each array providing one seismic channel for recording. At the beginning of a line the source is located off the end of the line and the recording equipment is connected up to the channels at the beginning of the cable. With subsequent shots the source is moved forward to occupy previous receiver positions, with the receiver nearest the shot being disconnected and a new one being connected at the other end of the line, usually by means of a "Roll Along" switch. This means that the correct geometry for CMP acquisition can be easily maintained and the recording truck only has to be moved when the "Roll Along" switch reaches the end of its range. The segmented transmission cable means that once the channels attached to a certain portion of the cable are no longer required for recording, this segment can be disconnected and taken to the end of the cable, for reconnection, enabling data acquisition to be reasonably continuous.
Fig 2.3: Marine acquisition configuration

Fig 2.4: Land acquisition configuration
In the marine environment the acquisition of data in a CMP geometry is even easier to arrange than on land. The usual marine acquisition configuration is to have both source and receiver being towed behind the boat (fig 2.4). The receiver streamer consists of "live" sections of hydrophones, each of which forms one data channel, separated by inactive sections, hence providing a means of keeping a constant receiver separation. CMP coverage is obtained by steaming at a constant speed and synchronising this with the firing rate, so that the source is activated at the point when the streamer has moved forward to provide a new CMP position, by occupying a previous shot position. That is the speed of the boat for full coverage is given by:

\[ dV = 0.5dX/dS \]

where

\( dV \) is the ship's speed

\( dX \) is the receiver spacing

\( dS \) is the shot repetition time interval

Recording

The signals from the hydrophone or geophone arrays are recorded by a digital acquisition system. The basic principle of such a system is shown in Fig 2.5. The signal from each data channel is amplified and then fed into an analogue multiplexer. The multiplexing of the seismic channels in this way means that only one Analogue to Digital (A-D) converter is needed in the system. The output from the A-D converter is usually a 14 to 16
bit integer value with an associated 4 bits of gain information, although instantaneous floating point values are generated by some systems. This output is written to 9 track digital tape in a multiplexed order. The format used to write the data to tape usually adheres to one of the accepted tape formats specified by the Society of Exploration Geophysicists (Barry et al, 1975; Meiners et al, 1972), SEG-A, SEG-B, SEG-C or SEG-D, although there are several accepted versions of each general format. Hence each record on tape represents the multiplexed data for all the receiver channels for a particular source position. These field tapes, together with the positional and other survey information are the raw material from which a processing system has to produce a final section which can be interpreted geologically.

**Common Processing Techniques and their Aims**

Demultiplex..............Get data in trace-sequential form
Amplitude recovery.......Correct for geometric spreading
Sort......................Order into CMP gathers
Edit......................Keep only good data-correct polarity
Deconvolution............Remove source signature
Filter....................Remove noise frequencies
Statics...................Correct to datum
Velocity analysis
NMO correction
Stack....................Improve S/N
Residual statics
Deconvolution
Time varying filtering
Multiplexed format

Trace Sequential format

Fig 2.6: Data organisations
Migration............Image data to correct location

Processing

When field tapes are received at a processing centre the first process applied to the data is demultiplex; that is the data are taken from the SEG multiplexed format and rearranged into a trace-sequential format. The trace-sequential format can be SEG-Y, although this is mostly used for data exchange, or some internal format designed for use only at the processing centre. At the same time as demultiplexing of the data is taking place the samples are formatted into a floating point format compatible with their subsequent digital processing.

Once the data are in a trace-sequential format subsequent operations are much more straightforward and it is usual at this stage to apply a time-varying scale factor to the data, to correct for the geometric divergence of the source energy and transmission losses in the Earth, which result in a reduction in energy with time, in each trace. Therefore amplitude corrections at this point attempt to bring reflections at later arrival times up to a strength comparable with those near the beginning of the trace. The type of function applied is either one calculated to be approximately correct for the losses the data has experienced, or an empirical function which has has been found to work well in practice.
Fig 2.7: Data amplitude decay and recovery
At this point in a processing stream the data can be plotted to make an examination of data quality and to look out for acquisition errors, like channels with the incorrect polarity or "dead" channels. This allows data editing to be performed, so that bad traces can be zeroed or omitted and all the traces are given the correct polarity.

In land data, differing station heights or the varying thickness of a low velocity weathered layer can introduce time delays which vary from trace to trace, and may pose a major obstacle to successful processing by severely degrading reflector continuity. It is therefore necessary to apply static corrections to the data. These are time shifts calculated from survey information to correct the traces so they appear as though they were recorded on a common datum. There are usually residual static errors, and sometimes these sufficiently degrade the data as to require an automatic residual statics procedure to be run. This package attempts to improve the continuity of an event by applying small time shifts to the data, on the assumption that these small time shifts are the errors left behind in the evaluation of the static corrections.

Although static corrections are of major importance in processing land data, they are of only minimal importance in processing marine data. With the data having been recorded close to the surface of a uniform layer of water, the only static corrections which are usually applied are those necessary to correct for the fact that the source and receivers are at a finite depth and not sea level; although if the target is reasonably deep these effects are negligible.
Fig 2.8: Example of Frequency Filtering
After demultiplex the data is in shot order in Common Shot gathers. However, before stacking is attempted the data have to be reordered into CMP order in CMP gathers. This sort operation is purely a reordering of the data based on the original acquisition geometry, in order to get the data into the correct configuration for CMP processing.

Once the data have been sorted into CMP gathers it is likely that filters will be applied to increase the signal-to-noise ratio, by eliminating noise in unwanted frequencies. From an inspection of the data and its power spectrum, the frequency characteristics of both the signal and the noise can be determined. If the two occur at separate frequencies then Bandpass or Bandreject filtering can be used to remove the unwanted effects of the noise frequencies. The filters used in this type of filtering operation are usually designed to be zero phase filters so as not to introduce time delays to the reflection events. In this country it is quite usual to use this type of filtering to remove the noise introduced by pickup of 50 Hz mains electricity noise, which is usually at a higher frequency than the source wavelet.

According to the convolutional model of the reflection trace, (Fig 2.9) the seismic trace is composed of the reflection coefficients of the geological horizons convolved with the source wavelet and contaminated by additive noise. Hence, in order to arrive at a trace which consists of just the reflection coefficients it is necessary to remove the effects of the source wavelet. The application of a filter to the data which compresses the source wavelet into a spike, equivalent to the reflection
Seismic experiment

Earth response  Seismic trace

Fig 2.9: Convolutional model of the seismic trace
coefficient, is known as deconvolution. If the source wavelet has been recorded, or if the wavelet is deduced by averaging over many traces, then an inverse filter can be designed to remove the effect of the waveform from the trace. This ability to reduce the effect of a known wavelet down to a pulse is one of the key principles of the vibroseis method of data acquisition. At sea, it is desirable to measure the far field signatures of airgun arrays, if the water depth permits.

However, in the vast majority of cases the source wavelet is unknown and so an attempt to remove its effect is usually made by attempting to find an estimate of the wavelet from the statistics of the trace (Robinson and Treitel, 1967). These methods are based on the premise that as the primary reflection sequence and the noise are essentially random, the autocorrelation function of the trace is equivalent to the autocorrelation function of the source wavelet. This information and the assumption that the wavelet is minimum phase, which may or may not be true, is used to design a Wiener spiking filter. This filter is the least squares approximation to the filter which would exactly deconvolve the source wavelet into a spike.

The spiking filter is a special case of a range of filters, known as prediction error filters, which can be derived from the statistics of the data. These filters record the error in a prediction of the trace a certain distance ahead from the statistics of the trace. This leads to predictable, events such as multiples and airgun bubble pulse trains, giving small prediction errors, whereas random events such as primary seismic arrivals give high errors. The spiking filter is a prediction
error filter with a prediction distance of 1 (Peacock and Treitel, 1969), but these filters can be used with different prediction distances to remove other unwanted effects. (Fig 2.10)

If the prediction distance is set up to be the same as the period of a long period multiple, then the prediction error filter can be used to dereverberate the trace. Also it can be used to compress an airgun wavelet, to improve resolution, by having a prediction distance just less than one wavelength of the bubble and filter of about the same size. When applied this would tend to leave just the initial pulse and so later events would not be obscured by the bubble pulse train. A compressed pulse so generated could be further compressed using spike deconvolution.

By this stage in a processing sequence there should be an improvement in both resolution and signal to noise ratio and the data would be ready for stacking. However, before the NMO correction can be applied to the data an estimate of the velocity structure has to be made by performing velocity analyses (Taner and Koehler, 1969).

One method of finding the stacking velocities is to produce a range of constant velocity stacks for a portion of the data. It is then possible to find the velocities which produced the best stacks for different events down the trace, and hence derive a stacking velocity function for that region of the data. Another method is to make measures of coherency along hyperbolic scans in a CMP gather, each hyperbola corresponding to a particular velocity for that zero offset travel-time. By repeating this procedure down the traces it is possible to display the coherency
Spiking Deconvolution

Predictive Deconvolution

Fig 2.10: Example of the action of Deconvolution
values as a function of velocity and time. Peaks in this coherency function occur at positions where that particular velocity would result in a good stack at that time, after performing the NMO correction.

These velocity analyses are repeated at regular intervals along the seismic profile so that a set of velocity functions are defined for the entire data set. Using these velocity functions the Normal Moveout correction are applied to the data which are then stacked to produce a CMP stacked section.

The application of the NMO corrections and the stacking procedure are non-linear and produce some undesirable filtering effects, tending to result in a broadening of the primary pulse. Therefore it is usually necessary to apply spiking deconvolution after stack. Also, because of the high levels of broadband noise which are usually present on the pre-stack traces, deconvolution before stack tends to be only partially effective. However, the improved signal-to-noise ratio of the post-stack data provides an opportunity to improve pulse compression.

The stacking process and deconvolution tend to change the noise spectrum and so bandpass filtering of the post-stack data, possibly time and space variant, is necessary to remove unwanted frequencies.

In the case of simple geological structures, where the horizons are near horizontally layered, the processed CMP stacked section is adequate for a geological interpretation. However, if the data are more complicated, a final procedure, migration, is necessary to produce an interpretable section.
Migration is an attempt to image the reflection events on the CMP section back to their correct spatial locations. A CMP section is displayed as though each event recorded on a particular trace was produced by a reflector perpendicularly below the surface, at the Common Mid-point. It can be shown quite easily that this assumption is untrue, for anything other than horizontally layered horizons. Therefore it is important to apply migration in order to display the horizons in their correct spatial locations.

For the purpose of migration the CMP traces are considered as being the recording of the wave field produced with a coincident shot and receiver. In this mode of data collection, upward and downward paths are coincident and so the recording is the same as would be obtained by having a source at the reflector point in a medium with half the true velocity.

Therefore, the CMP section can be regarded as being the recording, at the surface, of the simultaneous initiation of sources, with strength proportional to the reflector strength, at every point in the medium, with the medium having half its true velocity. Mathematical reconstruction of the source strength at every point in space can be obtained by calculating the wavefield at time zero for the entire medium from the wavefield recorded at the surface at later times. Hence the geological structure would be delineated.

This mathematical reconstruction is performed by solving approximations to the acoustic wave equation. There are three methods of approach which are most frequently used; Kirchhoff
Fig 2.11: Diagram to show mis-positioning of seismic events
integral, Finite difference and Frequency wavenumber (F-K) migration, each with its own difficulties. The F-K method of migration provides an accurate solution to the wave equation for all dips of the events in the data, but it is not easy to incorporate anything other than a constant velocity structure. Velocity variation can be accounted for to a certain extent in Kirchhoff migration, and events of quite high dips are migrated accurately, but this method tends to organise the noise in the data into broad "smiles". The finite difference method is the one in which it is easiest to incorporate velocity variations, but it cannot easily be made to cope with events dipping at angles greater than about 45 degrees. The single most important problem with obtaining an accurate result from migration, is in defining an accurate velocity model for it to use. However, if a reasonably accurate model of the velocity structure is available, modern migration methods do produce a reasonable approximation to the geological structure, and enable a reasonably confident interpretation to be made.

Summary

From the description of the seismic reflection method given in this chapter, it is fairly obvious that most of the techniques employed in a routine processing sequence would not be possible without modern computing facilities. Some processes, such as migration, still take a few hours of processing time even with
modern hardware. Also the amounts of data handled in producing a final section are enormous and can only be processed in a reasonable length of time by specialised systems.
Chapter 3

Hardware and Systems Software

Hardware

A brief description of the hardware configuration (fig 1.1) has already been given. This chapter gives a more detailed description of the hardware and the Systems software provided with it.

The hub of the system is the Digital Equipment Corporation pdp 11/34. This is a 16 bit word length minicomputer with 256 Kilobytes (1kbyte = 1024 8 bit units), of MOS memory and an integral memory management unit. Due to the 16 bit wordlength, the processor has an address limit of 64 Kbytes, and therefore the full 256 Kbytes cannot be accessed directly. The UNIBUS on which the pdp 11 series is based has an 18 bit addressing capability which allows a full 256 Kbytes to be attached to the processor. However, the memory mapping unit has to be used to access more than 64 Kbytes of memory. Also the architecture of the processor is such that the highest 8 Kbyte addresses are reserved for the input/output page, and so addresses in this range 56 to 64 Kbyte always refer to these registers, which are used in accessing the peripheral devices. The central processing unit of the 11/34 has a full range of integer arithmetic instructions which take a few microseconds to execute, but floating point arithmetic is performed at a higher level, and so is much more time consuming.
The main peripheral attached to the pdp 11 is the Floating Point Systems AP120B floating point array processor. This is a very fast floating point arithmetic processor, with a parallel pipeline architecture, which allows vector operations to be overlapped and hence completed very quickly. This processor has a separate program source memory and data memory, and also has a ROM table memory containing cosine coefficients for FFT's and other useful constants. The particular AP bought for this system has 8 Kwords of 38bit data memory and a floating point add, subtract or multiply can be initiated every 167 nanoseconds, making this a very powerful processor of floating point data. The AP is connected to the pdp 11 by a Direct Memory Access (DMA) interface. This allows direct transfers of data from the pdp 11 memory to the AP's main data memory without processor intervention, once it has been initiated by software. The conversion from pdp 11, 16 bit integers or 32 bit floating point format to the 38 bit floating point format in the AP is achieved by the interface hardware "on the fly", as the data passes through.

The main storage unit on the system is the Pertec 20 megabyte, moving head disc drive. This consists of 3 fixed platters and a removable cartridge, and therefore has 8 read/write heads. It is interfaced to the pdp 11 through a RK11 compatible DMA interface, so that the drive emulates 8 RK05 disc drives, and the removable cartridge is RK05 compatible. This configuration means that effectively the disc storage is split into 8, units of 2.5 megabytes.
The Versatec electrostatic printer/plotter is 11 inches wide, has 2112 nibs at a density of 200 nibs per inch and serves in the dual roles of system line printer and high quality plotter, being able to use fanfold paper for printing, and roll paper for plotting, as well as film for final good copies of plots. In print mode it is driven by passing it ASCII characters, and in plot mode 128 word wide rasterised data are used to drive the plotter.

The final peripheral attached to the pdp 11 is a VDU which is set up as the system console, and all interaction with the system is made through this device.

The secondary computer in the system is a DEC pdp 8/e. This is a 12 bit word minicomputer with 16 Kword of core memory, which was originally purchased by the department because its relatively simple architecture allows interfaces to other equipment to be designed and constructed fairly easily. An example of this is the ultrasonic acquisition system, designed and built by Mr J H Peacock, used in producing simulations of seismic reflection data, which is interfaced to the pdp 8/e and runs under its control.

The system storage on this machine is provided by twin Calcomp floppy disc drives, built to a format developed in the department. Other peripherals include a 30 channel analogue to digital converter, a Tektronix graphics screen, a fast paper tape reader and a Teletype which is used as the system console.
The most important of all the peripherals attached to the pdp 8, however, are the 3 Cipher 800 bpi tape decks. These are interfaced to the pdp 8 through a DMA interface which has access to 4 Kbytes of semiconductor memory, which acts as a buffer to allow it to read long-record gapped data formats from tape.

These two computer systems are linked by a DR11/L/M 16 bit parallel interface, which allows data transfers between the two machines under program control. Unfortunately, because the pdp 8/e is a 12 bit computer, unlike the pdp 11 with its 16 bit architecture, the interface had to be set up to work on a common data item. Consequently transfers take place 1 byte at a time, with the other 4 bits in the pdp 8/e being used to control the data transfer handshake.

The hardware is set up with the pdp 11/34 as the main controlling computer. The pdp 8/e acts solely as an intelligent peripheral controller when the tape decks are in use. The AP120B is used as a very fast floating point "number cruncher".

Systems Software

A comprehensive package of systems level software was purchased with the hardware, and this is briefly described in this section.

The pdp 11/34 utilises the RT-11 version 3B operating system (Digital Equipment Corporation, 1978d). This is a disc based single user operating system and a side of one of the disc drive platters is used as the Systems Disc. This operating system
provides a full suite of utilities, such as an Editor, Librarian, Linker and file handling utilities, as well as a Macro-11 assembler and a Fortran compiler. The system console is the main means of communicating with RT-11 and apart from command files all operating system commands are entered from this terminal. This operating system provides standard device drivers for the discs, terminal and line printer on the system, but the other interfaces are non-standard.

Communications with the AP1208 are via the AP executive(APEX). This software provides a means of transferring data and microcode to and from the AP and monitoring the execution of AP microcode, in order to return error conditions and check for microcode termination. A full library of microcode routines(Floating Point Systems Inc, 1977) was provided with the machine. These routines have a Fortran callable interface which links into APEX to achieve transfer of the microcode to the AP. This library provides a comprehensive suite of routines for vector operations and it is rare to find an operation which cannot be performed by using a combination of these routines. However should the user find an application he wishes to perform, which cannot be achieved using existing routines, a new microcode routine can easily be developed using the software development tools available for the AP, an assembler (APAL), linker (APLINK), simulator(APSIM) and debugger(APDBUG). A full suite of diagnostic programs were also provided with the AP.
In order to drive the electrostatic printer/plotter as a plotter, the Versaplot library (Versatec, 1978) of plotting routines was purchased. This library provides a suite of Fortran callable subroutines which emulate the standard Calcomp graphics subroutines. They are used at a high level to produce vector type plots, such as graphs and annotation. Also provided, as part of the library, are programs to perform vector to raster conversion, and an input/output package which takes the rasters produced and outputs them to the plotter.

From this description it can be seen that all the peripherals attached to the pdp 11, except the DR11 link to the pdp 8, had systems software of some kind available from the outset.

The systems console on the pdp 8/e is linked into OS/8. This is a reasonably powerful disc based operating system developed for the pdp 8 series of computers. Although it has a rather rudimentary keyboard command language, it does provide a useful suite of utility programs for file and peripheral manipulation. It also provides facilities for program development in pdp 8 assembler PAL-8, and Fortran IV, with a multiple pass Fortran compiler. This compiler converts the Fortran into a pseudo-assembly language RALF, and the RALF assembler is then run to produce an object module. All the peripherals on the pdp 8/e, such as the tape decks and the video screen have OS/8 compatible device drivers and so can be manipulated by the standard utilities.
The hardware link between the pdp 11 and the pdp 8 was the only data pathway for which there was no controlling software once the system was fully configured. All the other peripherals could be manipulated, to a greater or lesser extent, using the facilities of the two operating systems and the additional software provided by the AP microcode library, APEX and the Versaplot library.

Linking the Two Minicomputers

Before any attempt could be made to start planning the seismic software, it was important that the systems software, which it was based on, provided all the utilities necessary for program development and operation.

The obvious starting point in the Software development was therefore to establish a software link between the two minicomputers, as without such a link there was no means of access to the tape decks from the pdp 11.

A need for software links between the two machines was recognised as existing in three different applications. The first objective was to establish device drivers compatible with the operating systems on both machines which would allow files to be passed between them, thus allowing the resources of the two machines to be shared. Secondly, it was most important that software be provided which would allow programs running on the pdp 11 to perform input/output to the tapes as though they were attached to the pdp 11. This would make the pdp 8 act as an
intelligent tape controller for the pdp 11, and in this capacity be able to handle the tape format produced by seismic field recording equipment.

Finally, there was also a need to enable programs running on the two machines to transfer floating point numbers across the interface, with the conversion between the two floating point formats being performed during the transfer. This was necessary to allow data acquired on the ultrasonic tank, in pdp 8 floating point format, to be used on the pdp11 for display and processing as necessary.

**RT-11 to OS/8 transfer**

The link between the two operating systems was achieved by installing new device drivers into them which could control the interface between the two machines.

On the pdp 8 two new device drivers, PIN:, to take the data from the pdp 11, and POUT:, to send data to the pdp 11, were written by Mr J H Peacock in PAL-8 assembler and built into the working version of OS/8. These drivers expect transmissions, of unspecified numbers of bytes, to continue until terminated by a CONTROL Z or another recognised file terminator.

Under RT-11 the author constructed a bidirectional driver DR in Macro-11, which is interrupt driven and follows all the RT-11 Version 3B standards for device drivers. This driver was not built into the Monitor but installed into one of the free device slots originally built into the Monitor. This installation is
performed in the startup command file which is executed when the system is bootstrapped, and so is transparent to the general user. This procedure allows the driver to be updated, without having to reassemble or relink the Monitor.

Having developed these device drivers it was then possible to transfer files between the two machines using keyboard commands, although one drawback is that commands have to be issued at each machine's console to initiate the transfers, e.g.

**RT-11 to OS/8**

RT-11...........COPY DK1:MPDMXA.FOR DR:*.*
OS/8...........R PIP
.............*MTAO:<PIN:/A

**OS/8 to RT-11**

OS/8...........R PIP
.............*POUT:<DD01:SDS10.FT/A
RT-11...........COPY DR:*.* LP:*.*

This software link allows files to be written to tape, in 512 byte blocks, using the standard OS/8 magnetic tape drivers for data transfers to other machines, as above.

**Floating Point Transfer**

The data acquired on the ultrasonic tank are written to tape in pdp8 floating point format, which consists 3 words, or 36 bits per floating point value. However, in order to use the facilities
on the pdp 11 to handle this data, it is necessary to transfer it and simultaneously convert it to the 32 bit floating point representation used on the pdp11.

Therefore two subroutines IN11 and OUT11 were written in pdp8 assembler, which accept numbers in pdp11 format and convert to pdp8 format and vice versa. These two routines were used in a program MPTP11, written in Fortran on the pdp8, which reads ultrasonic data from a tape and then passes it to the pdp11 via these subroutines. Three Macro-11 routines GETNO, GETDAT and SENDAT were written for the pdp11 to take Floating Point numbers from the interface and put them into a memory array and vice versa. One use of these routines was in the program MPUSTR, which reads ultrasonic data from the pdp8, demultiplexes it and writes it to a sequence of disc files in the internal seismic processing data format. However, the assembler subroutines written for both machines allow the transfer of Floating Point data between any two programs running simultaneously on both machines.

Tape Handling

The most important part of the link between the two machines was in providing access to the tape drives for programs running on the pdp11. It was decided that, in order to provide the response required, the pdp8 would have to be dedicated to tape handling when any programs requiring tape usage were being run on the pdp11. The pdp8 would, therefore, become an intelligent tape controller when seismic processing programs were in operation. When design work was begun on this handler, it was realised quite
quickly that there would only be memory space available for one type of tape handling by the program. That is the tapes could either be driven in gapless or a blocked format but not both. As it is essential to be able to operate in gapless mode to be able to read the field tapes, this capability had to be present. Therefore it was decided that just one tape drive handler would be produced and it would work in the gapless format.

As a result of these decisions a standalone tape system monitor, SDS10, was written in pdp8 assembler by Mr J H Peacock. It provided a set of tape manipulations commands, which can be issued from the pdp11 and are then executed by the pdp8. This software also decodes the tape status conditions and returns a status byte to the pdp11 on completion of the tape function.

With the data being read from tape in a gapless format, it streams off tape constantly at whatever tape speed is in operation until end of file is reached. This means that the data has to be moved to its destination at least as quickly as it comes off tape, or data will be lost.

The system is set up on the pdp8 so that, when a read is initiated, data from the tape is transferred by a DMA process into a 4 Kbyte memory buffer. The transfer routine has to be able to pass this data to the pdp11 fast enough to prevent data from the tape overwriting data previously written into the buffer before it has been transferred. There is a similar problem in reverse when writing to the tapes in this mode. Here data must be in place in the buffer before it is required by the tape for writing out.
The software in the pdp11 has to be able to keep up with the tape transfer rate. However, experiments with the interface device driver being used to control the transfers showed that the system overhead was too large, and so the tape buffer was being overwritten during large file transfers. Hence it was decided that specialist low level routines would have to be written to control the data transfers to and from tape.

It was realised that in most circumstances the volumes of data being transferred to and from tape would be too large to fit into the pdp11's lower memory area, meaning that disc files would have to be used as temporary storage. Therefore the transfer routine would have to be responsible for transfers to and from disc during interface transfers. However, there are situations, such as when handling seismic data post-stack, when there is only a small amount of data and it will easily fit into the pdp11 memory. Hence it was also decided to provide a set of routines which could transfer data to and from buffers in pdp11 memory.

The first routine SDS10 was written in Macro-11 and provides the basis for the tape handling. Besides incorporating the capability to read from tape to disc and write from disc to tape, it also passes other commands to the tape handler to allow rewinds and file skipping commands to be executed. The transfers are accomplished in blocks of 2048 bytes, which are buffered in memory before being written to disk, or to the pdp8. This was done to allow the 4096 byte buffer in the pdp8 to be used as though a double buffered transfer were in operation.
To supplement this general purpose routine, two other subroutines, TREAD and TWRI T were written, in Macro-11, to allow transfers to and from memory buffers in the pdp 11, with the tape handler.

The first program to utilise these routines was an interactive program written to allow easy manipulation of the tapes, MPTAPH. This gives the user the capability of extracting files from the tapes and putting them onto the disk and vice versa. This can be particularly useful when test data for filter tests or velocity analyses is being selected and put onto disc. This program also allows the tape to be spooled forwards and backwards, to enable a file to be located on the tape, before it is written to disc and then the tape can be rewound, all through a series of keyboard commands.

**Tape Archiving**

After an early hardware failure on the disc drive, during which some programs were lost, the importance of a reasonable archive system became apparent. Although, using the removable cartridge, copies of programs and data can be put onto a separate disk for archiving, a problem can arise when the disc drive read/write heads are realigned, after a service or repair. Under these conditions it is possible that a realignment of the heads after the backup copies had been made would render these copies unreadable. Therefore it is very important that there should be a capability of archiving programs to tape for later recovery.
It was decided to use the tape handling routines previously described as the basis for a tape backup/restore program. The program which was written MPTPSV, allows files to be written to tape, and in doing so keeps a header block describing the name of the file, its size, the date it was archived and its version number. The version number allows an updated copy of a file to be put onto tape with the same name as a file already present, while preserving the ability to restore it by specifying the higher version number. The program can provide a directory of the tape by reading through the file headers, so the contents of the tape can be easily verified.

In order to recover a tape file, just the name of the file and its version number need to be specified. If the file is already in the program's internal directory, it implies that the tape is already past this file on the tape. Therefore this request is queued until later. Otherwise it searches forward and locates the file to be restored. At the end of a run any restore requests in the queue are executed before the job is terminated. This program is fully interactive and provides a very flexible and easily used system of archiving files on tape for later recovery.

**Processing System Tape Subroutines**

It was realised that the programs in the processing stream would need to have a tape handling capability and that this capability would have to be consistent, from process to process, in its treatment of errors and other processing conditions. Therefore it was decided to produce two Fortran subroutines TAPRED
and TAPSUB to handle the tape programs for the processing routines. TAPRED assumes that the transfers are going to and from the disc and TAPSUB to and from memory buffers.

These subroutines also provide all the error condition handling from the tape drives. This ensures that a particular error condition is treated consistently by each of the processing programs (Fig 3.1). These routines provide the programs with 3 commands: read, write, and wind forward, to allow the end of file record to be skipped when reading. The error handling is based on the expected data sequence being as shown in Fig 3.1.

All the functions necessary to manipulate the tape drives in a seismic processing program are provided by these two routines, and one of the two is used in any application which needs a tape handling capability. Therefore, applications programs which need access to the tape drives can be written without the programmer having to understand how the drives are controlled, by calling one of these two subroutines. Also, the only error handling which need concern the applications programmer is how to treat the fatal errors returned by the routines after retries on read and write have failed. The usual course of action at this point is to close down the job so that it can be restarted with a new tape. End of tape errors are also returned to the user program for handling, so that any special functions which are deemed necessary on an end of tape can be executed.

Therefore these two Fortran subroutines, together with the Fortran callable assembly language routines in the pdp11 and the tape monitor in the pdp8, provide a comprehensive tape
**Tape Data Format**

<table>
<thead>
<tr>
<th>STATUS</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOT</td>
<td>Ignore</td>
</tr>
<tr>
<td>BUSY</td>
<td>Loop until not busy then continue</td>
</tr>
<tr>
<td>OFFLINE</td>
<td>Write a message to the console</td>
</tr>
<tr>
<td>EOT</td>
<td>Write a message to the console</td>
</tr>
<tr>
<td></td>
<td>Set status = -1 and return</td>
</tr>
</tbody>
</table>

**READ ERRORS**

- NORMAL: Return
- SHORT RECORD: Read next record
- PARITY: Wind back one record and retry 3 times and then return with status set if still in error

**WRITE ERRORS**

- NORMAL: Return
- PARITY/SHORT RECORD: Wind back one record, rewrite record with 8 sets of all bits set in header and then rewrite the record. Perform this retry 3 times and then return with status set if still in error

**WIND ERRORS**

- PARITY/SHORT RECORD: Expected so Return
- NORMAL: Wind back 1 file and return

All errors are logged in the file on Fortran unit 2

Fig 3.1: Tape Data Format and Error Handling
manipulation service, which should provide a user transparent means of handling the tape drives and their error recovery.

**Memory Management**

The pdp11/34 is a 16 bit minicomputer whose basic address unit is the byte (8-bits). This means that the physical address capability of the processor is 64 Kbytes. As has been previously mentioned, the UNIBUS has an 18 bit addressing capability which allows up to 256 Kbytes to be accessed. In order to use this capability, a memory management unit between the CPU and the UNIBUS translates virtual addresses into physical addresses by using the relocation information contained in the 8 page address registers inside the unit. Each process's virtual address space is broken up into 4-Kbyte pages, each of which are relocated into physical addresses by one of the page address registers. (Digital Equipment Corporation, 1978)

There are two sets of memory mapping registers. One applies to programs running in KERNEL mode, such as the operating system and device drivers, and the other for programs running in USER mode. This allows the operating systems relocation information to be kept separate from the user's program.

It had been intended originally to operate the pdp11/34 under the Extended Memory Monitor version of RT-11. Fortran programs running under RT-11 are allowed to define a set of variables as virtual. This implies that they are stored in memory other than that directly addressable using the 16 bit word. This allows the
full memory capability of the pdp11/34 to be used from a single program. The Extended Memory Monitor is designed so that two words of address information are passed to the device drivers in order to make up an 18 bit address, so that DMA devices can put their data straight into Virtual memory at the correct address. However, it was discovered that in order to implement this extended virtual address capability, the high 8 Kbyte addresses which are usually mapped into the I/O page are relocated elsewhere in User mode and are only accessible to the system device drivers, running in KERNEL mode. This seemingly minor problem has important side effects. All the non-standard device handlers, such as APEX, the pdp8 transfer routines and the plotter driver, use the I/O page addresses in user mode in order to access their respective devices. This meant that when virtual arrays were in use in Fortran programs under the Extended Memory Monitor, communications with the AP, plotter and the pdp8 were lost.

As virtual arrays are also supported in the less sophisticated Single Job (SJ) Monitor of RT-11, this monitor was investigated. Under this monitor the relocation of the high addresses to the I/O page is unaffected by the virtual arrays option being present, and so this problem is immediately overcome in this environment. Additionally this monitor is much less sophisticated and so has the advantage of occupying much less space in memory than the Extended Memory Monitor. However on further investigation major disadvantages were found in its implementation of the virtual arrays principle.
Fig 3.2: Memory configuration of the pdp 11/34
The Fortran compiler generates a reference to a set of utility subroutines every time a virtual array element is referenced in a Fortran program. These subroutines are given the offset of the start of the virtual array from address 1600 octal (56Kbyte) in 64 Kbyte blocks, and the number of the array element referenced. By dumping the machine code from memory when such an operation was in progress, it was possible to decode the method used to access the data at this extended address.

In normal running under the SJ monitor, the KT-11 memory mapping unit is switched off and the virtual addresses up to 56Kbytes refer to the first 56Kbytes of physical memory. The high 8Kbytes are then mapped into the I/O page which is at addresses 248 to 256 Kbytes in physical memory. In accessing a normal Fortran array element, the address is found by calculating the byte offset from the array's base storage address. A similar method is used in referring to a virtual array element.

As described above, a special subroutine is passed the base address and element number of an array element on the stack when a virtual array is referenced. However, this base address is an offset, in 64 byte blocks, from 1600 (Octal) the 56 Kbyte limit of normal addressing. The subroutine manipulates the two values to generate a byte offset between 0 and 4Kbyte as an element address, while putting the rest of the address into the USER mode page address register 0, which is the register referred to in relocating addresses between 0 and 4 Kbytes. At this point the KT-11 memory mapping unit is switched on, and a special instruction used to fetch the data element from the relocated address and put it onto the stack. When this has been completed
the memory management unit is switched off again, and the subroutine returns to the mainline code which picks up the required value from the stack.

From the example in Fig 3.3 it can be seen that this is a very longwinded process and so there is quite a large time penalty incurred when using virtual arrays in calculations. However, potentially more important than this was that Input/Output with virtual arrays could only be performed via buffers in the lower address memory. This problem is caused by the fact that the device drivers in the SJ monitor are only passed a single word memory address, and so even DMA transfers have to be made into the lower 56 Kbytes of memory. The time penalty involved in transferring data from I/O buffers into data areas in Extended memory would have been intolerably large in a seismic system, where such large quantities of data are handled. Therefore it was decided at this point that some solution to this problem had to be found, whereby DMA transfers between virtual arrays and the disc and AP would be possible

**Virtual Memory Input/Output**

A small test program using virtual arrays was single stepped in execution and areas of memory dumped after each step. From this it was possible to determine that immediately after a reference to a virtual array element, the page address register and register 1 still contain the components of the full 18 bits address. Therefore a Macro-11 routine would be able to access these registers and save the 18 bit address of a virtual array
Enter Subroutine from User Program

System register R0
Start of virtual memory array as an offset from 1600 octal

System register R1
Number of array element to access

Add 1600 octal to offset
Get element offset

Offset as multiple of 4k bytes
Remainder between 0 and 4k bytes

Page address of Virtual memory element
Address of element inside a page

Switch on Mapping

Get data from address (R1) inside the page

Turn off the Mapping

Return to User

USR0 and R1 still contain Virtual Address

Fig 3.3: Virtual Memory access in RT-11 Fortran
The technique used to get a desired 18 bit address was to explicitly reference the particular array element required in a Fortran function call. It was found that this caused the Fortran compiler to generate code which moved this element from virtual memory into local storage. The function which was called was in fact a Macro-11 routine and so at its entry point as the virtual array element had just been referenced the users page address and register 1 still contained the components of the 18 bit address. These two values could be accessed and put into temporary storage within the routine as well as being put into registers 0 and 1 before exiting. These registers are the ones which take the result when a floating point function is called in a Fortran program. Therefore this returned result can be put into a local variable for storage.

This capability of being able to "steal" the 18 bit address from the Fortran system was a very important step forward, as it meant that in principle the full 18 bit address could be given to a DMA peripheral when initiating a data transfer. It had been decided that the two areas where this capability had to be applied to the task of actually transferring data, by DMA transfers, to and from virtual arrays, were the discs and the AP.

The first one to be tackled was the AP as the transfers were already under the control of a non-standard device driver. In initiating data transfers to and from the AP, interface registers were given a 16 bit memory address for the memory buffer. However a further two bits in a different register were used to give the
full 18 bit address. In the standard DMA handler these two bits were just ignored. A Macro-11 function was written to extract the full 18 bit address of the virtual array buffer required and put it into the correct format for the interface registers, before returning it to the calling routine for storage. The standard APEX was then altered so that it always cleared the 2 extended memory bits before it initiated a DMA transfer, so that transfers using the standard routines would always go to lower memory addresses. This allowed a new transfer routine to be written, which expects a full 18 bit extended address as a two word argument for use in initiating transfers to and from the AP. These routines provide a full extended memory DMA transfer capability for the AP.

Solving the same problem with respect to the disc drive was more difficult, because the operating system is based on this device and so it uses the disc driver itself for transferring operating system information in and out of memory. In the SJ monitor disc device driver the two extended memory bits in the interface register are cleared every time a transfer is initiated.

The first step was to stop these two bits being cleared by the device driver by masking them off in the driver code. It was considered necessary still to use the standard driver for initiating the transfers, as the operating system was still in charge of the file structure on the disc. Therefore the operating system was relinked with the altered driver, and as the extended memory bits are not set by any other routine in the system it was considered reasonably safe not clearing them. Also error conditions in the SJ monitor result in a reset instruction being
issued which clears all the device interface registers.

Therefore a Macro-11 function was also written to get the 18 bit address in the correct format for the disc interface and return it as the result of the function for storage. Read and write routines were written which make calls to the system I/O routines giving the low 16 bits as the supposed memory address. However, just before these system calls are made the two extended bits in the interface register are set. On completion of the transfer a completion routine, stated in the transfer call, is executed and this then clears the two extended memory bits in the interface. This call to the completion routine is completely transparent to the user of the routine, which allows the data transfers to be overlapped with program execution in the same way as other DMA routines. Because they manipulate the interface directly, these routines have to wait for all other disc transfers to terminate before they can be initiated, which is not usually a major constraint on their use.

Care has to be taken in the use of these disc transfer routines with respect to the operating system. Usually only the core of the operating system is resident in memory and it reads in its service routines from disc as required. Obviously in this case if the extended memory bits were set it could have disastrous consequences. Therefore, when a program which uses these routines is compiled, a switch has to be specified to the compiler which causes the User Service Routines (USR) to be locked into memory. This causes more space than usual to be taken up by the operating system, but it does mean that besides having a disc DMA capability into virtual memory, there is also a saving of time which would
have been spent reading the USR in from disc. A similar problem occurs with programs which are overlaid. However at least in this case it is known that the overlay handler is not executed until a certain subroutine call is made. Therefore the user has to ensure that all DMA transfers using these routines have finished before making a call to an overlaid routine.

In tests using these routines it was shown that the AP routines work just as well as the normal APEX transfer routines and are not constrained any more than the normal routines. Also it was found that as long as the constraints mentioned above were adhered to, the disc transfer routines ran faultlessly. Therefore this piece of systems programming provided the machine with a great deal more flexibility than it had previously, in allowing DMA transfers to be made from the disc and AP to and from the full 248 Kbytes of useable memory. Also following the principles laid down in writing these two sets of routines it would be possible to provide this capability for any other DMA peripheral devices which might be added to the system.

Plotting Software

The Versaplot software purchased with the electrostatic plotter provides a set of Calcomp compatible graphical subroutines and also some specifically electrostatic routines, which provide the capability to produce shading and patterned lines. This software produces a vector file which has to be processed by vector to raster conversion software before being output to the plotter. This rasterising software and the plotter driver are
also supplied with the Versaplot package. This package is quite sufficient for producing graph type displays and annotation, but does have its drawbacks. It does not provide any high level graphic capabilities, such as contouring, and because of the huge amount of data produced is totally incapable of producing seismic wiggle trace or variable area type displays, the amount of machine time used to display even one or two seismic traces being prohibitively high. Therefore it was decided that more software would have to be developed to complement and add to the Versaplot routines.

**Contouring**

A contouring package CONSYS, which was developed at the University of Michigan, and is considered as being in the public domain, was available on the NUMAC IBM 370/168. Although written in Fortran it was not easily transferable to the pdp11 as, besides using several constructs unique to IBM Fortran it also used operating system calls to allocate dynamic memory for its work space during execution. On the other hand the package produces good quality contours and uses a reasonably time efficient algorithm. Therefore it was decided to convert this package to run on the pdp11. The dynamic memory allocation was rewritten to use static work arrays passed to it by the calling program, in virtual memory. Once this had been accomplished and other parts had been rewritten in standard Fortran it was linked into the Versaplot package for drawing the contour lines. This package provides a full contouring capability, which can easily be used by
any display program which requires contouring.

**Seismic Displays**

The production of seismic trace plots using normal graphic subroutines is a very time consuming process. Each trace is made up of about 2000 points and there may well be a few hundred traces in a section. All these vectors would have to be produced, sorted, and then rasterised by the normal graphical subroutines, producing a very large intermediate plot file.

However, a seismic display is really a quite well ordered dataset, and if the overlap between traces is restricted the possible range of a single trace on the paper can be quite well defined. Therefore it was decided to develop programs which would produce seismic displays by going straight from the input data to a raster output file.

The maximum swing of a trace was limited to plus or minus twice the trace spacing and the maximum trace spacing was limited to 0.1 inches, so that the rasterising buffer would easily fit into memory. Also if the display is longer than 10.24 inches the software produces a second strip later which corresponds to the data off the sheet of paper.

In order to develop the software an interactive section plotting routine was produced, which expected its input to be on the disc and writes the raster output file back to disc. However once developed this algorithm was incorporated into a full section plotting program for the processing system.
Fig 3.4: Example of output from the Seismic Display Package.
It was realised that once a file of rasters have been produced, other plots could be merged with the seismic display, before the rasters are transferred to the plotter, as long as the other plot is also in raster form.

One of the drawbacks of the Versaplot system is that the plot programs produce vector output in particular named output files on the systems disc, and these are then rasterised and put out to the plotter in one run. It would be much more convenient, in case a plot is to be replotted, to store a file of rasters for later retransmission to the plotter. Also, this would allow the Versaplot routines to be used to provide annotation and axes for the seismic displays, which could be rasterised, saved and then merged with the seismic raster file on output. Therefore a set of subroutines were written which emulated the plotter handler routines, but instead of putting the raster output to the plotter, they are transfered to a specified disc file. When linked with a modified version of the vector to raster conversion program MPRASM, it became possible to store the raster images of vector plots.

The next logical step was to develop a post processor which took the raster input from up to 8 files and merges them according to offsets and ranges specified by the user, and even reverses the contrast if required, before using the plotter driver software to put the final rasters onto the plotter. This software allows the seismic image produced by the special display software, to be merged with axes, time scales and annotation which is produced by a program using Versaplot routines, and so allows each approach to be used solely in the mode in which it is most efficient.
It was found that, in general, the inexperienced user found more difficulty developing programs on the pdp11, with its limited software development tools, than on the NUMAC IBM 370 under MTS. Also a large modelling and synthetic seismogram package, AIMS, which was acquired in order to provide an interpretation aid for seismic reflection data, as well as high quality synthetic data for research work, was much too large to fit onto the pdp11, and so had to be installed on the IBM. Examples of the output from AIMS are shown in Fig 3.5 and instructions on how to run it are given in Appendix 2.

The installation of the package was reasonably straightforward as it is written in standard Fortran. The only alterations necessary were to change the input/output unit usage to be compatible with MTS usage and to alter the plotting calls so as to fit in with the *PLOTSYS system on MTS. This package provides a very powerful raytrace modelling and synthetic seismogram tool for use in conjunction with the seismic processing system.

As has already been explained, some users find the limited program debugging facilities available with the RT-11 Fortran extremely difficult after having become accustomed to more powerful facilities on large mainframes such as the NUMAC IBM 370. Hence program development could tend to take longer than usual at first. With the pdp11 being run as a single user system it means that while program development is taking place it cannot be used for processing and vice versa. Therefore it was decided to
Fig 3.59: Example of an AIMS input model
Fig. 3.5b: AIMS output from the model in Fig. 3.5a
provide development facilities on the NUMAC IBM so that the basic algorithm of a new program could be developed offline from the pdp11 and transferred to it, at a later date, for interfacing to the processing system.

At this time FPS, the suppliers of the AP had a Fortran simulator of their mathematics library under development. A copy was acquired and developed to run on the IBM 370, while still looking to the user as though it was running on the pdp11. By writing programs in as near standard Fortran as possible and using this simulator, a program can be developed on the IBM which is easily transferable to the pdp11 when complete. At this point only the disc input/output needs to be changed and the tape access software added to the program, before final tests can be run. In practice this has proved to be an extremely valuable tool, having allowed MSc students to develop algorithms on the IBM, while the pdp11 is in use, transfer the programs to the pdp11 and then run them on the pdp11 on large datasets, which could not be handled in a reasonable time on the multiuser general purpose IBM system.

Summary

Quite a large proportion of the total development time of the seismic processing system had to be spent designing, programming, and testing the systems utility routines described in this chapter. However these routines provided a solid base from which the system could be developed, and without which the processing system would not have been feasible.
Chapter 4
Seismic Processing Software

Overview

In a research project of this type it is important to realise that the targets set for the project have to be accomplished in a limited period of time. Also it is more useful to set realistic, attainable goals than to overreach and leave an unfinished shambles.

Once the systems software had been established, providing the necessary tools for applications programming, the aims of the project, from the seismic processing viewpoint, could be realistically assessed. It was decided that a suite of programs representing a complete seismic processing stream should be attempted. Although this might seem quite ambitious, it was felt necessary to establish software at all stages of the seismic processing stream in order to establish the conventions and standards for the data handling and processing throughout the sequence.

Obviously, for there to be any chance of this grand hope being accomplished, there had to be certain limitations and compromises made in planning the details of the software. As the University of Durham's interests in seismic reflection work had been almost entirely marine, then this original software suite was based on the needs of processing marine data. Also as data acquisition with different equipment and data exchange from
outside was considered unlikely at this time, the input field data was considered as being solely derived from the department's SDS 10/10 digital acquisition system, and no attempt was made to produce the final data in SEG-Y format for data exchange. Given these relatively minor restrictions, the brief was to produce a full and complete seismic processing system.

**Overall Design Considerations**

Most commercially available seismic processing systems have a seismic monitor, which is responsible for taking menu type input for a particular job, structuring the modules required into a run stream and controlling the data flow through the system. This possibility was considered for controlling the operation of the Durham seismic processing system, and a small amount of development work was carried out to evaluate a rudimentary system, based on the RT-11 batch stream monitor. However, after experimentation this idea was rejected, for several reasons. Possibly the most important reason is that on a small 16 bit minicomputer, such as the pdp11, dynamic memory is one of the most important of the limited resources. Once the space taken up by the operating system and input/output system has been taken into account, only about 24 Kbytes of memory are left for program storage, as instructions have to remain in the directly addressable portion of memory, under RT-11. If a monitor system was developed it would leave even less memory for the seismic programs and the lower memory buffers they use. It was also felt that such a high level of control on program execution would incur an unacceptably large
time penalty and increase the complexity of the software unnecessarily, with the treatment of error conditions in several different types of program being a particularly difficult problem. It is quite usual for large jobs such as demultiplex and migration to be run as the only program in the processing stream, even in commercial systems, and it was felt that those processes which normally run together could still be arranged in this way using standalone programs. Finally, the type of user expected to be using the system is more likely to be at home running a standalone program with a given function than attempting to construct the menu type of input under a seismic monitor, which usually involves quite complicated training courses to master. Therefore, bearing in mind all these considerations, it was decided to build the system as a suite of standalone programs.

Data Format

One of the first considerations in designing the processing system was to decide on the internal format of the data. In this case one constraint was placed on the design from the outset. The tape control program running on the pdp8 has to be constantly resident to be ready to answer any tape command which is issued by the pdp11. However because of the size of the program, there is not enough available memory to allow two types of driver for the tapes. When reading field tapes it is necessary to be able to read a gapless format, and so the software for this format has to be used. As the blocked format driver cannot be resident at the same time, any data written back to tape has to use the gapless
format. Therefore it was decided to keep the data flow as simple as possible and adopt the gapless format as being part of the internal data format for tapes.

The data in disc files under RT-11 are stored in sequential files, each comprising a sequence of 512 byte blocks. The data exchange with the disc is most efficient when carried out in multiples of 512 bytes, or 128 samples. Therefore it seemed fairly logical always to keep the amount of data being processed as an integer multiple of 128 samples, which is just over half a second of data at 4ms sampling rate. This is relatively easy to achieve in the present system because the SDS 10/10 always digitises its data so that the number of samples in 1 second is always a power of 2. Hence this fits in well with the disc file organisation.

1 SDS second at 4ms...256 samples...1.024 seconds

As a further aid to simplifying data transfers it was decided to make the header block, containing information on the data, consist of one 512 byte block, which would be written to block 0 in a disc file. It was felt that 512 bytes would provide adequate space for all present, and any foreseeable future, usage of the header block, for storing data information.

It was decided that in order to keep the structure of the data as simple and straightforward as possible, logical units of data would be separated by file marks on tape and put into separate files on disc. Hence, before stack, common shot gathers or CMP gathers would be contained in separate files and after stack each trace is put into a separate file. This structure
enables the header block to be kept quite brief as the channels inside the gathers are stored sequentially in order of increasing offset. Hence as long as the number of channels, the length of the data in each trace and the acquisition geometry are recorded there is no need for more than one header block per file.

The structure of the file format is shown in Fig 4.1 and it can be seen that the header block is occupied from bytes 0 to 50, with quantities describing the data and acquisition geometry. Bytes 50 to 256 are used to store information provided by the user, as an identifier or comment on the data. The second half of the header block is used to keep a brief account of the processing carried out on the data, by entering a set of predetermined values, which uniquely identify each process. This can be valuable for data which has been archived for a long time, with the original notes on its processing having been lost. This set of header entries identify each process applied and the order in which they were applied for a particular data file.

The header is also used to indicate a bad area on tape. When a parity or CRC error is detected during a write, the tape program backs up the tape and then writes 8 bytes with all bits set, followed by up to 32760 padding zeros. During a file read, if this sequence is found the file is assumed dead and the read routine moves on to the next file for the data.

It is felt that 512 bytes should prove quite adequate for future header block usage, but the possibility of changes in the future to increase this amount was considered in programming the system. Therefore the data references were structured, as much as
Header Block Format

<table>
<thead>
<tr>
<th>Byte position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>ASCII values from field tape header block</td>
</tr>
<tr>
<td>4-5</td>
<td>Sampling interval in msecs (from field tape)</td>
</tr>
<tr>
<td>6-8</td>
<td>Equipment serial number (from field tape)</td>
</tr>
<tr>
<td>9</td>
<td>Equipment serial number (from field tape)</td>
</tr>
<tr>
<td>10</td>
<td>Equipment serial number (from field tape)</td>
</tr>
<tr>
<td>11</td>
<td>Recorded data length</td>
</tr>
<tr>
<td>12</td>
<td>Number of channels recorded</td>
</tr>
<tr>
<td>13-14</td>
<td>Number of channels in the file</td>
</tr>
<tr>
<td>15-16</td>
<td>Starting position of first data sample</td>
</tr>
<tr>
<td>17-18</td>
<td>Ending position of last data sample</td>
</tr>
<tr>
<td>19</td>
<td>Gather code: 0-common shot/reciever, 1-CMP, 2-Stacked trace, 3-Single trace</td>
</tr>
<tr>
<td>20</td>
<td>Units code: 1-metric, 2-Imperial</td>
</tr>
<tr>
<td>21-24</td>
<td>Shot to 1st Receiver offset</td>
</tr>
<tr>
<td>25-28</td>
<td>Receiver spacing</td>
</tr>
<tr>
<td>29-32</td>
<td>Shot spacing</td>
</tr>
<tr>
<td>33-36</td>
<td>Shot interval in seconds (Marine)</td>
</tr>
<tr>
<td>37-40</td>
<td>Source depth</td>
</tr>
<tr>
<td>41-44</td>
<td>Receiver Array depth</td>
</tr>
<tr>
<td>45-46</td>
<td>Source code: 0-Airgun, 1-Explosive, 2-Vibroseis, 3-Weights</td>
</tr>
<tr>
<td>47-48</td>
<td>Next usable address in process header</td>
</tr>
<tr>
<td>49-50</td>
<td>Delimiter with all bits set (%01777777)</td>
</tr>
<tr>
<td>51-254</td>
<td>User comments inserted at DEMUX time</td>
</tr>
<tr>
<td>255-256</td>
<td>Delimiter with all bits set (%01777777)</td>
</tr>
<tr>
<td>257-512</td>
<td>Process Header: A process history inserted into Header at the next free slot by each process</td>
</tr>
</tbody>
</table>

Fig 4.1: Processing System Data Format
possible, so as to allow the header block to be enlarged with only minimal software changes, to data offsets in disc files, and equivalence positions in memory references.

Data Input and Output

In order to keep the input to the processing programs orderly and straightforward, it was decided that each program would expect its input parameters in a particular named disc file. This allows the average user to input data to the system without having to understand how to assign logical I/O units in RT-11. It was also decided that each user of the system should keep their files separate by prefixing each filename by a two letter unique identifier. Therefore a user's input parameter files would be created under his own identifier, and then copied across to the correct input file name before the program is run. This has the added advantage that each user has a copy of the job input parameters under his own identifier.

If seismic data files are written to disc, the user is allowed to specify the file names, so allowing these files to be collected under the users identifier. This is obviously especially useful if several users are running data tests on data stored on disc.

It had been intended originally to keep the programs as portable as possible by using Fortran, unformatted direct access I/O for manipulating the data traces. However this was found to be much too slow and unwieldy, and so RT-11 DMA transfer routines
were used, which access the data in block mode from the disc. Although this means that this aspect of the system is operating system dependent, it would be relatively easy to convert the programs to run under a different operating system, by just replacing the RT-11 routine calls with their analogues in the other operating system. One advantage of using these RT-11 routines is that once they have been initiated the CPU can continue execution while the data transfer is completed by DMA operations.

Seismic Processing System

Once the basic principles of the system, as described above, had been decided upon it was necessary to identify the programs which would be needed to produce the complete seismic processing system. The suite of programs which, it was decided, would fulfill the stated requirements of the system is shown below.

Synthetic Seismogram Package
Demultiplex
Sort
General Pre-Stack Processing
Fourier Data Analysis
Velocity Analysis
Gather Plotting
CMP Stack
General Post-Stack Processing
Post-Stack Mix
Section Plotting
It was felt that major processes such as Demultiplex, Sort and Stack should be separate, as they form a natural break in the data processing. However, whenever possible, it is desirable to minimise data transfers by amalgamating functions into a single program. Therefore it was decided that the general processing, such as filtering and deconvolution, should be applied inside one program, so that once the parameters had been decided they could be applied to the data, in order, in one run, before being written back to tape. Therefore these functions are bound into just two programs, one for Pre-Stack application and one Post-Stack.

**Synthetics**

When implementing a new suite of programs it is essential to have access to reliable synthetic data, which can be used to test and evaluate the processes. Also it is useful to have the capability to produce synthetic data so that new applications programs can be easily tested.

It is hoped that eventually the ultrasonic tank, developed by Mr J H Peacock, could be used for the routine production of synthetic seismic reflection data for various acquisition situations. However at the time of this project the tank itself was also being assembled, and so data derived from it could not be used reliably. Therefore one of the first tasks was to generate
programs capable of producing syntetic data.

It was decided that two small programs would provide an adequate source of synthetic data, one to be used to produce simulated CMP gathers and the other synthetic CMP stacked sections.

The CMP gather generator ANSEI was developed with A Nunns (Nunns, 1980) and it generates a set of traces, each displaying a specified number of primary reflections. Each reflection is defined by an arrival time on a zero offset trace and a stacking velocity, which is used to calculate the hyperbolic trajectory of the seismic arrivals. No allowance is made for inversion or transmission energy loss effects, so the seismic pulses are always positive and of the same amplitude. A Ricker wavelet (Ricker, 1953), with a specified frequency, represents the seismic wavelet and band limited random noise is added to all the traces. This noise is generated in the frequency domain by constructing a unit amplitude with random phase, up to the cutoff frequency. A Fourier transform then yields a random noise trace which can be added into the seismic trace. This program provides a simple but effective method of producing pre-stack synthetic data.

A more sophisticated program was required for producing synthetic stacked sections, as these would be used as test data for processes such as Migration where amplitude variations are important. Therefore a program developed by C Godbold (Godbold, 1980), for use on the NUMAC IBM370 as a tool in investigating Kirchhoff migration, was converted to run on the pdp11 and produce
Fig 4.3: A synthetic stacked section showing diffractions
output compatible with the seismic processing system.

This program allows plane dipping layers and point reflectors to be specified with a vertical variation of velocity with depth. It uses a simple ray tracing technique to evaluate the travel times, ignoring multiples and refracted events, and calculates the appropriate impulse response using an approximate wave equation method. The synthetic is completed by convolving a Ricker wavelet with the calculated impulse response to give the seismic waveform for each arrival and random noise is added using the same method as described above.

These two simple programs are extremely useful in producing different types of synthetic data for program testing and evaluation.

Demultiplex

The demultiplex program was designed principally to handle the SEG-A format produced by the departmental SDS 10/10 acquisition system. However the main data flow of the program, and its general logical sequence was designed so that a new version, to handle SEG-B or SEG-C, could be written, using it as a template, around which to build the specific routines.

Demultiplex is fairly obviously a tape to tape process, and so it utilises a modified version of TAPRED to handle the tape input/output. This routine had to be modified slightly because it is necessary to send a byte swap command to the pdp8, to swap the bytes in every 16 bit word, when transmitting the multiplexed
data. This is because data on the tape are written out conforming to IBM data standards, and in the IBM architecture the low address byte is the most significant byte in a word, which is exactly the opposite to the architecture on the pdp11. By getting the byte swap performed by the tape handler in the pdp8 during data transmission, no overhead is incurred in subsequent processing in the pdp11.

Although the demultiplex program is basically designed for tape to tape operations, it is also possible to leave files on disc, as well as putting them out to tape, if required for quality control plots and data tests, and input can also be taken from the disc if required.

The program was designed to operate in two modes; fast with only minimal error checking, and slow with full error checking and attempted error recovery capabilities. Either of the two modes can be selected at the start of a demultiplex run and, if the fast mode has been selected, another option is available which allows the user to specify that it should revert to the slow mode if an error is detected in fast mode.

As the demultiplex is the first program in the processing stream, it is responsible for initialising the file header block for each of the output files it generates. Some of the parameters are extracted from the field tape header, but the geometry values have to be entered by the user and are stored into each file header by the program.
With all the reordering of the data which is taking place during demultiplex, it is a natural point at which to reorder the traces in the output files into ascending order of shot-receiver offset. Therefore the user specifies, in order, the channels on the field tape which correspond to an increasing offset of the receivers, to allow the program to sort the channels into this order. Usually the channels are written out into a common shot point gather, and in fact this is probably the most desirable form for the data to be written back to tape, as these raw demultiplex tapes can be easily used as the starting point for later reprocessing runs if required. However a sorting ability was incorporated (see Sort), so that small datasets could be demultiplexed directly into CMP gathers, to save time and the amount of tape handling required. It was envisaged that this option would be used to select records from the tape for data quality examination and filter tests before the whole of the line had been demultiplexed.

One option in the program which was added in the light of experience at Durham occurs when a change of tape is requested. The SDS 10/10 has twin tape decks so that when one tape is finished the system can switch to use the second drive without any data being lost. However if one tape deck is inoperative it is possible that data will be lost during the tape changeover. Therefore when a new tape is requested, the operator is asked if blank files are required. By this means zeroed channels can be written out to tape and it provides a simple way of padding the data out to the correct lateral scale, from the start of the processing. These zeroed trace would then be sorted into CMP
gathers with "live" traces during the sort.

The basic design principle behind the operation of the demultiplex programs operation, is to demultiplex enough samples in one pass to produce one disc block, 128 samples, of trace sequential data for each trace. In this format 128 scans is equivalent to 4 Kwords of multiplexed data, which is half of the AP's main data memory. Fortunately after demultiplexing this reduces down to slightly less, and so there is also room available for the gain codes. As it is possible to just fit one block per trace of multiplexed data into the AP and demultiplex it, a microcode routine was written for the AP to perform the demultiplexing and the reformatting of the data into floating point numbers from the 15 bit integers and their associated 4 bit gain codes. While performing this operation the microcode routine only checks the start of scan code and the submultiplexed gain information for errors. However if an error is detected, the routine exits and sets an error flag which can be picked up by the main program. When the main program detects the error flag, it can, if the option is set to allow it, restart demultiplexing the file in slow mode in an attempt to overcome the error.

In conjunction with this fast microcode mode of operation all the input and output operations are performed in a double buffered manner, so that the disc and AP transfers are fully overlapped with computations. Once a block of data has been demultiplexed it is written out to its correct place in a disc file, used as temporary storage. If the field data is error free this mode of demultiplex allows the operation to be performed very quickly.
However, there are occasions, when the field data contain many errors, such as data lost or corrupted. In the university environment it is important to use as much of the data as possible. Therefore a lot of effort was put into a slow mode for the demultiplex program which allows a significant amount of error recovery, even from poorly recorded data. In this mode no attempt is made to perform the operation very quickly; rather every piece of information in the multiplexed format, such as the time code and the submultiplexed gains, are checked to ensure no errors have occurred. If an error condition is detected the demultiplex is continued in an attempt to use the redundancy checks so that an output trace may still be produced when a serious error has occurred. The number of errors and lost samples which the user is prepared to tolerate in a trace, before it is declared "dead", is an input parameter to the program. If the number of errors exceeds these limits, or data recovery is not enabled in fast mode, the traces for that shot are zeroed before they are written out to disc.

While the error checking is being carried out in slow mode, the data are also being put into trace sequential order. Therefore when a block has been completed, all that remains is to reformat the data. This operation is carried out in the array processor. The data are transferred as integers and then converted to a floating point integer representation in the AP. A microcoded routine is then used to apply the gain factors to each sample in turn to complete the reformatting. The data are then retrieved from the AP and written to the temporary disc file. Log files of any errors detected are also produced for each shot file.
Once an entire field tape file has been demultiplexed, or as much of it as has been requested, the temporary file is closed and the data is written to tape, a copy being left on the disc for later use, if requested.

The demultiplex program is responsible for producing the data in the form in which they will be used during the remainder of the processing. Therefore a great deal of effort was put into its design and implementation, to provide the program with as much flexibility as seemed desirable. Also in implementing the fast mode, the program was made to be as fast as data transfers would allow, so that if the data was known to be virtually error free, from preliminary tests, large quantities of data can be demultiplexed very quickly. On the other hand, it is hoped that the sophistication of the error recovery capability in the slow mode will allow data of a reasonable quality to be produced even when acquisition malfunctions have gone unnoticed.

In designing the system it was considered that the sorting capability in the demultiplex program should only really be used in producing test CMP gathers on small data sets. When large datasets are being demultiplexed, it is best to produce tapes of common shot gathers so that the demultiplexed data correspond closely to the field tapes. This is more convenient for referring to the data at a later date.
Fig 4.6: CMP Acquisition Geometry
Sort

It was decided that during demultiplex all the data on the field tape should be demultiplexed and written to tape as common shot gathers, as mentioned previously, so that once it has been completed the field tapes would not have to be referred to again. Therefore, in order to select the segment of the data required and reorder it into CMP gathers, a sort facility is required.

The main consideration in designing the sort program was to be able to produce CMP gathers from common shot gathers easily, as this is likely to be the most common sort operation performed. However, it was realised that future work might require the data in different configurations, such as common receiver gathers, and that especially in the marine case, where accurate positioning and speed over the ground can be difficult to control, the acquisition geometry might be less than ideal, such that the shot spacing would not give a true CMP configuration. For example, if the speed over the ground at sea has been too fast or slow, instead of being able to get a CMP gather by sorting as shown in Fig 4.6, a smear, or footprint of midpoints is generated. In this case a different sort procedure than is usual must be adopted in order to minimise the size of the CMP footprint, so as to reduce the distortion of the velocities and structures that would otherwise result.

Therefore the sort program was designed to allow the reordering of the data to be totally user specified. As is shown in Fig 4.8 the repositioning of the input data into output files
Correct CMP positioning

Incorrect CMP positioning

Fig 4.7: The Smear Effect of Incorrect CMP positioning
An example with CMP gathers as input files

Fig 4.8: Generating Different Trace Gathers using SORT
is specified by a coordinate pair for each input channel. At the same time a starting and ending value for the data can be specified, so that the amount of data in each trace can be reduced while the sort is being carried out. As can be seen from the example, almost any new data configuration can be generated, entirely under the user’s control.

The input to the program is usually read from tape into a temporary file on the disc, although input directly from disc is possible. At the start of a run, a set of temporary sort files are created on the disc to collect the output gathers. If a 12 fold CMP gather was being generated from common shot gathers, then 12 temporary files would be required to hold the gathers until all the necessary data had been read in. The part of the data required is then transferred from the input file to its correct position in the correct output file. Once all the output channels have been transferred, at least one of the gather files will be complete, and so can be written to tape or to another disc file for later use. This file is then deleted and another one created in its place, and the process is repeated for the next input file.

It is possible that tape problems may cause the program to close down. If this occurs, the last line in the Log file for the job contains an index to the order of the temporary files which it has written out before terminating. If the program is restarted this can be input, along with the restart, flag and the job will continue from the point in the sort at which it was interrupted, so that the whole job does not have to be resubmitted.
The major features of the sort algorithm were included in the demultiplex program so that reordering into CMP gathers, for test purposes, can be accomplished straight from the field tapes.

It was felt that this sort program provides sufficient flexibility to allow marine data and most land data to be reordered into any configuration desired. The only situation which would be difficult for it to cope with would be in the crooked line sorting of land data, where the optimum sort configuration for CMP gathers, is constantly changing. This type of data would probably prove very tedious to sort as it would need to be performed in short segments, with constant operator intervention. However, apart from this, always complicated situation, it should be possible to handle all the data configurations likely to be encountered.

The sort program sets the header entries of the items it affects, such as the gather type, number of channels, and data start and end positions, so that the header block still carries up to date information on the data.

Pre-Stack Data Analysis

It seems reasonable that once the data has been sorted into the required configuration the user is going to want to examine the gathers, in order to check data quality and determine data characteristics, such as frequency content.
Fig 4.9: Example of data plotted with a large trace.

Spacing
Therefore a suite of interactive display programs was developed in order to facilitate data examination. Two interactive trace plotting programs were developed, using the raster plotting algorithm, to display gathers. One produces plots with the traces spaced at a maximum of 0.1 inch and with a maximum deflection of 0.2 inch, so that reflection events can be picked out by their continuity from trace to trace. The second module, spaces the traces so that each individual trace can be amplified, without overlapping other traces, so that the wavelet characteristics and data quality can be examined more closely.

These programs are fully interactive and expect the input data to be in disc files. They both produce raster output which can be put out to the plotter using one of the postprocessor programs, MPMERP with a merged timing line background, or MPPROC with no merged in background. This also provides the user with a quick method of examining the effect of different processes on the data, by allowing displays on the data after filter tests and other processes.

The other interactive display package is one which produces spectral plots of data traces. This program MPFANL allows several traces within a gather to be spectrally analysed, and a power and phase spectrum to be displayed for each. The spectra are derived by padding the data to a power of two in the AP and performing a Fourier transform. The phase and amplitude spectra can then be calculated from the real and imaginary parts of the transform. This information is used to produce a vector plot using Versaplot routines and once the program has been terminated the plot can be displayed on the electrostatic plotter using the Versaplot post
processing routine RASM.

This Fourier analysis package together with the gather displays, allows the frequency characteristics of the signal and noise to be determined, which can be very useful, in later processing, in allowing filters to be designed more easily.

Pre-Stack Processing

It was decided that all the processes usually applied to the data before stack should be included in a single program, with general subroutines being developed for the filter routines, so that they could also be used in Post-Stack processing. The idea behind this decision was that, once a trace from a gather had been transferred to the AP, for a certain process to be applied, it is more efficient to apply all the other processes to it before returning it to the pdp11, than to have several programs each dedicated to one technique each putting the data in and out of the AP. Similarly this approach cuts down the number of tape transfers needed to carry out Pre-Stack processing.

Another important factor in designing this program was that it must be able to accept input data from either the tape or the disc, so that tests could be easily carried out on data files on disc.

The processes which were included in the Pre-Stack processing package are shown below:-

Edit
Polarity reversal
Gain application
Muting
Bandpass filtering
Bandreject filtering
Spike deconvolution
Prediction error deconvolution
Trace normalisation

These processes can be selected as required and applied in any order, with even the capability of a process being applied more than once if required. The program was designed to be as modular as possible so that other processes which may be required at a later date can be easily slotted into the program. However if many more processes were added, it would probably be necessary to use overlays to provide sufficient room for the executable image in lower memory.

**Edit**

The data editing capability is used to zero very noisy traces, or ones with spikes, which cannot be made useable by further processing. Once a trace has been zeroed by the Edit option it is not passed through the rest of the selected options, and so this is normally the first option applied to the data. If a data trace is known to have been zeroed by the demultiplex program, then this option can be selected for these traces to prevent them passing unnecessarily through the rest of the
processes. This option is most likely to be used to kill traces containing bursts of high energy noise at about the same frequency as the source signal which precludes filtering to remove it. If these traces were left in for further processing they would contaminate the stacked results and so it is best to remove their effect by editing them out at this stage in the processing.

**Polarity Reversal**

It is not unusual for a data channel to be connected into the acquisition system with a different polarity to the other channels. If this situation was not altered it would lead to the stacked results being degraded. This option can be used to allow data traces with the incorrect polarity to be reversed before further processing.

**Gain Application**

Due to the spherical spreading of the source energy and transmission losses on passage through the Earth, the amplitudes of seismic arrivals decrease with increasing travel time. Therefore it is necessary to make some correction to counter this effect, so that reflection events at low travel times can be compared with those further down the trace, and the same event on traces with a greater offset.
As spherical spreading occurs in a predictable 3-d environment, it can be calculated. However the effect of attenuation losses can only be estimated. If 3-d spreading is considered then the decay is directly proportional to the travel time, and so a function which is just a linear ramp in time can be used to correct for this effect. Attenuation factors are usually exponential decay functions of the type $e^{-at}$. Therefore the inverse function $e^{at}$ can be applied. In seismic studies a value of 0.2 has been found empirically to be quite a good approximation for marine data. Therefore two of the gain functions in the program are of the form given below. It is relatively simple to adapt these routines to change the value for the absorption factor, and to apply only a T ramp if required.

$$t \exp(0.2t)$$

A third function of the type $TV^2$ is also available and this has been shown to give good results for near vertical data (Newman, 1973). Therefore this function can be used if an approximate velocity structure is already known.

Gain functions such as these have to be applied before such operations as deconvolution, so that the energy of the wavelet remains approximately constant down the record, in order to preserve the assumption of stationarity of the trace statistics with time.

In experiments with real data it was felt that the function $t \exp(0.2t)$ gave the best results, of the ones available, in producing equalisation of amplitudes down the trace. It was also felt that
these deterministic methods are preferable to AGC functions at this stage in the processing because their effects can be easily removed, which is not possible with AGC. In fact one of the options in the program is to be able to remove one of the specified functions, perhaps to replace it with an alternative, or to remove the ramp after deconvolution.

The ramps are generated by subroutines at the beginning of the run and are then stored in virtual memory, from where they are transferred into the AP to be applied, or removed.

**Muting**

Often the direct arrivals from the shot and the refracted arrivals from near surface events are so large that they tend to swamp the early reflection events. Therefore it is desirable to remove the effect of these unwanted events. This is accomplished by arbitrarily zeroing the traces down to a predetermined level to remove them. This is known as muting and is accompanied, in the algorithm developed, by a tapering of the data from the point of the last zero sample into the "live" data.

At the point where the mute ends there could be a large sudden increase in amplitude, which is equivalent to introducing high frequencies at this point. Therefore a cosine taper, of a user specified length, is applied to the data at the end of the mute zone, to smooth the transition from zero to live data.
Fig 4.11 : Example of Muting a CMP gather
The capability of applying a mute to the end of the data is also available in the program. This is done so that a small cosine taper can be applied at the end of the data, to remove the effect of the implied high frequencies generated by the sudden cutoff in the data.

The range of the mute can be specified for each channel, but the length of the taper is kept constant for all the channels. A subroutine designs the cosine tapers and stores them in virtual memory at the beginning of the run, and they are transferred into the AP to be applied when needed.

Frequency Filtering

Although it is desirable to leave as high a frequency content as possible in the data, quite often the data are dominated by noise, which may well be at a different frequency from the source wavelet. It is quite common for low frequency noise, such as ground roll, or streamer snatch at sea, to swamp the data, and probably the most common source of noise in this country is the 50Hz pickup from electrical supplies, which can completely corrupt the data in some cases (e.g. fig. 4.10).

Therefore it was considered necessary to have both Bandpass and Bandreject filters available in this Pre-Stack processing program.
**Bandpass Filter**

The bandpass filter used in the program is a zero phase filter with tapered ends to the pass region as shown in Fig 4.12. It is specified by giving the ends of the all pass region and the frequency range over which the end taper is to be applied. A cosine taper is used at each end. It is important that a zero phase filter should be used so that reflection events are not time shifted, by delays introduced into the phase spectrum. Also this ensures that the phase components of the source wavelet should not be affected so that the assumption of minimum phase is not affected by the frequency filtering.

The filter is designed in the frequency domain at the start of the run, and stored in virtual memory, and it is then applied in the frequency domain in the AP for each trace. The input traces are padded out to twice their original length before transforming, so as to avoid the possibility of cyclical convolution. Care should be taken not to design too narrow a pass region, or too short an end taper, as these tend to lead to instabilities in the filter (Oppenheim and Schafer, 1975).

**Bandreject Filter**

The type of zero phase bandreject filter which is available in this program is shown in Fig 4.13. The user specifies the two points at which the reject region begins and a sine taper is designed between the two points, falling to zero midway between these two points. Therefore only one frequency component is made
Fig 4.12: Bandpass Filter Representation

\[ F_l = \text{Lower Limit of Bandpass region} \]
\[ F_u = \text{Upper Limit of Bandpass region} \]
\[ F_t = \text{Cosine Taper Length} \]

Fig 4.13: Bandreject Filter Representation

\[ F_l = \text{Lower Limit of Reject region} \]
\[ F_u = \text{Upper Limit of Reject region} \]
identically equal to zero.

This filter is designed and applied in the frequency domain by the same method as described for the bandpass filter. However, when specifying a bandreject filter, which by its nature has a long time-domain representation, the user must be careful not to specify too narrow a bandreject region, as this would tend to produce the equivalent of an infinite filter and so cyclical convolution may be unavoidable.

In the case of both filters described, tapers are used at the ends of the pass regions in order to avoid ringing at the cutoff frequency being generated. The filter is applied in the frequency domain, because with the speed of FFT's in the AP the multiplication to apply the filter is much faster than the convolution in the time domain, and it makes the process much more understandable to the user.

**Deconvolution**

Two types of deconvolution were included in this program, Wiener spiking deconvolution and Prediction error deconvolution. Both have the aim of compressing the source wavelet to improve the resolution of the data.
Wiener Spiking Deconvolution

The aim of spiking deconvolution is to design a filter which when convolved with the source wavelet produces a spike output on the seismic trace. In order to do this exactly an infinite length filter would be required, and so an approximate truncated filter has to be designed using Wiener's least mean square energy criteria (Robinson and Treitel, 1967).

Consider the problem of designing a filter $F_t$ such that when it is convolved with an input wavelet $B_t$, it produces an output $C_t$, which is an approximation to a desired output $D_t$.

$$eqn 1 \quad C_t = \sum_{s=0}^{M} F_s B_{t-s}$$

and the error energy is given by:-

$$eqn 2 \quad I = \sum_{t=0}^{M} (D_t - C_t)^2 = \sum_{t=0}^{M} (D_t - \sum_{s=0}^{M} F_s B_{t-s})^2$$

The error energy is at a minimum when the partial derivatives with respect to the filter coefficients are equal to zero. Therefore:-

$$eqn 3 \quad \frac{\partial I}{\partial F_i} = 0 = \sum_{t=0}^{M} 2(D_t - \sum_{s=0}^{M} F_s B_{t-s})(-B_{t-i})$$

and this reduces to

$$eqn 4 \quad \sum_{s=0}^{M} F_s \sum_{t=0}^{M} B_{t-s} B_{t-i} = \sum_{t=0}^{M} D_t B_{t-i}$$
now

$$\sum_{t=0}^{\text{min}} B_{t-i} B_{t-i} = \phi_{\mathbf{BB}}(s-i)$$

This is the autocorrelation function of the input wavelet

$$\sum_{t=0}^{\text{min}} D_t B_{t-i} = \phi_{DB}(i)$$

This is the crosscorrelation function of the input wavelet with the desired output. Therefore the equations can be specified in matrix form as (see, for example, Robinson and Treitel, 1967):

$$\begin{bmatrix} \phi_{BB} \\ \phi_{DB} \end{bmatrix} \begin{bmatrix} F \end{bmatrix} = \begin{bmatrix} F \end{bmatrix}$$

This set of equations can be solved using the Levinson recursion as $$\begin{bmatrix} \phi_{BB} \end{bmatrix}$$ is a Toeplitz matrix.

In order to apply this to a seismic trace several assumptions have to be made. The purpose of deconvolution is to remove the wavelet and leave behind just a spike at the time of the onset of the wavelet. The assumptions which are made to allow this are:

1. The impulse response of the earth is assumed to be white, stationary and random, as is the noise content.

2. The seismic wavelet is assumed to be minimum phase.

$$\text{Trace} = c(t) = S(t) * r(t) + n(t)$$

where

- \( S(t) \) is the seismic wavelet,
- \( r(t) \) is the impulse response of the earth,
- \( n(t) \) is the noise content,

and the asterisk denotes convolution.
The first assumption is necessary to allow us to assume that
the autocorrelation of the source wavelet is a scalar multiple of
the autocorrelation of the trace, except for the zero-lag coefficient.

\[
\sum_{s=0}^{\text{min}} (s_s \times s_e) (s_s \times s_e + n) = \sum_{s=0}^{\text{min}} s_s s_{s+e}
\]

From this assumption it follows that the autocorrelation
values of the noise and reflection impulses can be considered to
be zero after the zero lag value, and the crosscorrelation of the
noise with the wavelet can be considered to be
negligible (Robinson and Treitel, 1967). Therefore the
autocorrelation function of the trace, after the spherical
divergence correction has been made, can be assumed to be the
autocorrelation function of the source wavelet.

The other assumption was that the wavelet was minimum phase.
This is because the spiking filter will do the best job on the
minimum phase wavelet, of all the wavelets with the same
autocorrelation function.

\[
I = 1 - 2c_0 + \sum_{c_0} c_t^2
\]

\[
= 1 - 2B_c F_0 + \sum_{c_0} c_t^2
\]

Because of the minimum error energy criteria, F must be
minimum phase, as any other filter with the same autocorrelation
function would have a smaller value for \( F_0 \) which would increase
the error energy, \( \sum c_t^2 \) being a constant for B convolved with any
filter having the same autocrrelation function.

In a seismic application the filter is derived as follows:

\[ \Phi_{x\times} \begin{bmatrix} F_k \end{bmatrix} = \Phi_{x\times} \]

\[ \Phi_{x\times} \] is the autocorrelation of the seismic trace for lags 0-M.

\[ F_k \] is the desired filter of length M+1.

\[ \Phi_{x\times} \] is the cross-correlation function between a spike at \( T=0 \) which is the desired output, and the source wavelet.

As the spike series can be represented by 1,0,0,0,0,...0 the crosscorrelation can be seen to be 1,0,0,0,0,...0 and so 1,0,0,0,...0 can be used and still be correct to within the scale factor A.

In the processing system a subroutine SPIKE was written to design and apply a spiking deconvolution filter for a particular data trace. The method used was to first find the autocorrelation function of the data trace. This was performed by padding the trace to double its length with zeros, to avoid cyclical correlation, and then Fourier transforming. The autocorrelation function can then be calculated as the inverse transform of the power spectrum.
A user-specified whitening factor is added to the zero lag value of the autocorrelation function to stabilise the solution of the equations. This is equivalent to adding a small value to each of the frequency components in case any of them are zero.

The crosscorrelation function is generated as a spike at position 0 followed by M zero values. This, together with the first M+1 lags of the autocorrelation function, are input to the Levinson recursion routine which produces the M+1 length desired filter. This filter is transformed into the frequency domain where it is multiplied with the transformed version of the trace. The resultant is then transformed back into the time domain to give the resultant deconvolved trace which can be used in further processing. This procedure is repeated for each trace and the user has the choice of giving the filter unit energy, keeping the input and output trace energies the same or applying no scaling at all.

One other useful input parameter to the deconvolution, is the position of the spike. Although a minimum phase wavelet is the only one which has a realisable inverse if the spike position is at T=0, causal filters can be designed if the occurrence of the spike is delayed from T=0. The approximate spike produced by this method has a tail and a precursor from T=0 to T-t, where t is the spike position, and of course this results in the peak of the output compressed waveform being delayed by t.
Prediction Error Deconvolution

A second deconvolution method based on the statistics of the seismic trace is prediction error deconvolution. The basis of this method is the ability to predict the values of the trace at a future position $t+\Delta t$ from the values at the present position $t$. The error in the prediction between the actual value and the predicted value is then recorded.

In principle random events, such as reflection series, should record high prediction errors, while multiples or bubble pulses which are predictable should produce a low prediction error, if the prediction distance is set to the period of the effect.

The prediction error filter can be derived from the prediction filter, and this is the filter which when convolved with the data predicts the data at a future time.

\[
\sum_{s=0}^{t} \mathcal{R}_s p_{t-s} = \mathcal{R}_{t+\Delta} 
\]

Once again the Wiener least squares criteria can be used to minimise the error energy between the predicted and the actual values. Therefore following the derivations of the previous section the following equation can be derived which has to be solved for $P_m$, the prediction filter.

\[
[\mathcal{O}_{\Delta}] [P_m] = \mathcal{O}_{\Delta}
\]
is the autocorrelation of the input trace

is the crosscorrelation between the input and the desired output. This represents just the alpha lag value of the autocorrelation function, for a prediction distance of alpha.

\[
\begin{bmatrix}
\phi_{\alpha \alpha} \\
\end{bmatrix}
= \begin{bmatrix}
1 & \cdots & N \\
\phi_{\alpha \alpha} \\
\end{bmatrix}
\begin{bmatrix}
P_0 \\
P_1 \\
P_2 \\
\vdots \\
P_m \\
\end{bmatrix}
= \begin{bmatrix}
\phi_{\alpha \alpha} \\
\end{bmatrix}
\]

Therefore the prediction filter coefficients can be derived using the Levinson recursion method. Once the prediction filter has been derived the prediction error filter can be formed.

\text{eqn 4 prediction filter} = P_0, P_1, P_2, \ldots, P_m

\text{eqn 5 prediction error filter} = 1, 0, 0, \ldots, -P_0, -P_1, -P_2, \ldots, -P_m

where there are alpha-1 zeros for a lag of alpha.

It can be shown (Peacock and Treitel, 1969) that when the prediction distance is 1, that the prediction error filter corresponds to the Wiener spiking filter, except for a constant scale factor. Therefore the Wiener spiking filter is a special case of the more general prediction error filter. It has been shown that the prediction filter will be minimum phase for all lags, as long as the input series is minimum phase, which is not known a priori, but should be true for the marine case.

The design of the filter in this program follows similar lines to the spiking filter. The routine written, \text{PRDICT}, generates the autocorrelation function from the transform of the
power spectrum. The lag value and the length of the filter are specified by the user, and from this the input to the Levinson recursion routine is the autocorrelation function from 0 to M and the autocorrelation function from t to t+M, for a filter lag of t. The prediction filter so formed is turned into a prediction error filter by negating the coefficients and inserting the correct number of zeros between the value of 1 and the negated coefficients. This filter is then transformed into the frequency domain and applied to the input trace. The result is then transformed back into the time domain and scaled if required.

The prediction error filter is likely to be used before stack, to compress the source wavelet, if a spiking filter cannot be used successfully because the input wavelet is not minimum phase, as in the case of a single airgun source. If the filter length and filter lag are well chosen this method can be used to reduce the bubble pulse effect of an airgun and so compress the wavelet.

**Trace Normalisation**

The facility was provided for the data to be normalised to unit energy or unit maximum amplitude so that all the traces in a gather would be at about the same energy. This option would not normally be used, as it tends to obscure amplitude variations. However, if such variations have occurred for some reason during acquisition, this option can be used to remove that effect by allowing each trace to have unit energy.
Pre-Stack processing - Summary

Any of the processes previously described can be applied to pre-stack data in any order specified by the user, and processes can be repeated if required. For example bandpass filtering could be applied both before and after deconvolution if specified by the user.

The program reads data from tape to a temporary disc file, or straight from a disc file, for tests, and reads in and operates on one trace at a time from the gather. Once the trace has been passed to the AP by a process, a flag is set to show other processes that the trace is in the AP, so each process then acts on the data in the AP. When the last process has been applied the trace is retrieved from the AP and put into another temporary disc file. When a complete gather has been processed it is written back to tape using TAPRED, or left on disc if necessary.

A complete record of each process being applied is recorded in the header block for each gather, so that the processing carried out on the data can be deduced from the data, without the need of independent records.

The filter, ramp and taper generators were all written as general purpose subroutines so that they could be used in a Post-Stack program too without having to make any changes.
Velocity analysis and Stacking

Possibly the most important step in producing a CMP stacked section is the determination of stacking velocities. If this is not performed correctly then the resultant stack will be poor and all the other processing will have been wasted.

The basic principle of CMP stacking is that a set of reflection traces derived with different shot-receiver offsets, but with a common mid point, also have a common reflection point on horizontal subsurface horizons. It is trivial to show that for a single, homogeneous, horizontal layer the trajectory of a primary reflection across the CMP gather traces is given by:

\[
T_x^2 = T_o^2 + \frac{x^2}{v^2}
\]

where
- \(T_x\) is the observed time on the trace,
- \(T_o\) is the arrival time on a common shot receiver trace,
- \(x\) is the shot-receiver offset,
- \(v\) is the stacking velocity of the event.

In the case of horizontal reflectors overlain by beds of differing velocities the stacking velocity approximates to the RMS velocity (Dix, 1955), and so is often referred to by this name. The difference in the onset time on a particular trace with respect to the zero offset trace is known as the normal moveout.
eqn 2 Normal Moveout \( \Delta T = \left( \frac{T^2}{T_o} - \frac{x^2}{v^2} \right)^{\frac{1}{2}} - T_o \)

Therefore if the stacking velocity is known or can be determined, the normal moveout can be calculated and the normal moveout correction applied to the traces in a gather. This leads to a reflection event occurring at the same time on all the traces in a gather. If this is repeated for all the primary reflections in a gather and the traces are then summed (stacked) to give a single output trace, the multiple reflections should be attenuated and the primary reflections enhanced with respect to the background noise. However for this to be achieved the value of the stacking velocity for each event on the trace has to be determined. This information is usually derived from the trace itself by velocity analysis. There are three common methods of velocity analysis, constant velocity stack panels, constant velocity gather panels and coherence scan analysis.

In the constant velocity stack method of analysis, several CMP gathers centered on the point of interest are taken, a stack is produced for each gather for a particular constant stacking velocity, and the resultant panel of about 20 stacked traces is displayed. This is then repeated for a range of different constant velocities. The intention is that when a primary reflection is stacked at its correct stacking velocity, it will show up most clearly on the stack panels. Therefore a time velocity function is picked by finding the stacking velocities at which the reflection events give the largest stacked amplitude.
A similar approach is used with constant velocity gathers. In this case a single gather is displayed after the NMO correction has been applied for a particular constant velocity. This is again repeated for a range of velocities to produce a range of gathers all with NMO corrections corresponding to different velocities. When the correct stacking velocity for a primary reflection is reached, the event should appear horizontal after NMO correction. Therefore a time velocity function can be derived by picking the velocities at which the reflection events appear horizontal on the NMO corrected gathers.

The methods described above rely on the primary reflections being clearly recognisable, and the correct velocity being one of the ones chosen for the panels. A method which produces a map of goodness of stack for a range of velocities and times would obviously be desirable, and this is what the coherence methods of velocity analysis attempt to do (Taner and Koehler, 1969). At all times down the section a scan along the stacking hyperbolae corresponding to a predetermined range of velocities is made. A measure of the success of stacking along these trajectories is calculated, and can be displayed as function of velocity and time, allowing the stacking velocity function to be picked at the points where this measure is a maximum.

It was decided that the capability of using all three of the methods described should be developed for use within the processing system, for determining stacking velocities.
Coherence Velocity Analysis

In the velocity analysis program designed for use in the processing system it was decided to use semblance as the measure of stacking coherency. It is computed by calculating the moveout trajectory over the CMP gather at a particular velocity \( V \) for each \( T_c \) at the centre of an \( N+1 \) point gate. Therefore an \( N+1 \) point gate is derived for each of the \( M \) traces, of amplitude \( A \), in the gather, from which a value of the semblance \( S(v,t) \) for that time and velocity can be calculated.

\[
eqn 1 \quad S(v,t) = \frac{\sum_{k=M}^{N+1} \left( \sum_{j=1}^{M+1} A_{k+j} \right)^2}{M \sum_{k=M}^{N+1} \sum_{j=1}^{M+1} A_{k+j}^2}
\]

The semblance is a measure of the ratio of energy after stacking to the total signal energy prior to stacking, normalised in the range 0 to 1. These principles are implemented in the program written for the system MPVEL.

This program expects its input, a CMP gather, to be resident on disc, and it produces an unformatted Fortran output file containing the results of the semblance calculation for later display. The user specifies the gate width and gate step size to be used in the semblance calculation, as well as the time and velocity ranges over which the analysis is to be performed. These parameters are checked in the first part of the program for consistency and modified, if necessary, to prevent things like the moveout trajectory going off the end of the data at later analysis.
The obvious way to calculate semblance is to perform the calculation for a full range of velocities at a given zero offset time, and then to move to the next time gate position and repeat the procedure. However it was found that due to the limited pdp11 and AP memory, this algorithm was not possible without placing undue restrictions on the number of channels which could be used in an analysis. Therefore a slightly reworked algorithm had to be used.

In the method adopted all the data required from a particular channel for the full range of velocities specified is read into pdp11 memory from the disc, and then put into the AP in a double buffered scheme allowing calculations and data transfers to be overlapped. The partial semblance contributions, from this channel are then calculated for each velocity. This is repeated for all the channels, and when all the partial semblance contributions are complete, the final semblance $S(t,v)$ vector for the range of specified velocities can be calculated in the AP. These can then be written out to disc, and the time gate moved to the next zero offset time for the procedure to be repeated. All the indices to AP positions, disc positions and data sizes for transfer are calculated in the AP at the beginning of a new time gate, and are used from storage in the pdp11 memory for the rest of the calculation.

The double buffered data transfers allow the program to run at a reasonably quick rate. However the method used means that data is often read in from disc and transferred to the AP more
than once during the analysis. On the other hand no constraints are placed on the number of traces in the analysis using this algorithm. With a larger AP data memory the algorithm could be largely restructured so that each point in trace would only be read in once, by allowing the partial semblance calculations to be carried out on larger segments of the data at once.

**Velocity Analysis Display**

The output from the velocity analysis program is in unformatted standard Fortran output and is used as the input to the display program MPVCON, which gets all its input from this file. All the control parameters needed by MPVCON are written out to the front of the unformatted output file by MPVEL before the initiation of the semblance calculation, so that the display program can be run without any need for user input.

MPVCON is written using the Versaplot graphical subroutines, and the CONSYS contouring package to produce a vector plot file. This file can be displayed on the plotter using the Versaplot post processor RASM or it can be converted to rasters and stored for later replotting, by using the vector to raster intercept routine MPRASM.

The program produces a contoured display of semblance on a grid of time against velocity, and it also marks the position of the maximum semblance for each time gate with a small square. Annotation at the side of the display shows all the parameters used in the velocity analysis program to produce the results, so
Fig 4.14: Contoured Velocity Analysis Display

PROCESSING PARAMETERS

NO. OF CHANNELS = 24
SAMPLES PER CHANNEL = 2048
SAMPLE DELAY = 0
LEVEL OF INTERPOLATION = 1
CHANNEL 1 OFFSET = 260.0
CHANNEL SPACING = 100.0
SAMPLING INTERVAL MS = 4
START OF ANALYSIS MS = 88
END OF ANALYSIS MS = 4096
TIME STEP MS = 24

OPERATOR GATEWIDTH MS = 168
START VELOCITY KM/S = 1.00
END VELOCITY KM/S = 3.00
VELOCITY STEP KM/S = 0.05
that the plot is self documenting.

**NMO corrected gathers**

A program MPCDP was written to produce NMO corrected gathers for use in constant velocity gather production and to examine the quality of stack produced by a particular velocity function. It expects its input to be an N channel CMP gather and it outputs N moveout corrected channels, using a user specified velocity function and the resultant stack channel, so that in total N+1 trace are output from the program.

The seismic trace is composed of a suite of samples with a sample interval DT such that:

\[ T_i = (i-1)DT \text{ for } i = 1\ldots \text{length of trace} \]

The moveout corrected trace is generated by removing the moveout delay on a trace of a particular offset, due to a particular velocity. Therefore for a time \( T_j \) on a moveout corrected trace, the data sample to be placed at this point comes from a position in the original trace defined by the moveout trajectory.

Therefore for a sample \( T_j \) on NMO corrected trace, sample on original trace is \( T_i \) where \( i \) is given by:-
\[ i = 1 + \text{INT}(\sqrt{T_j^2 + X_k^2/V_j^2}/DT) \]

That is, the nearest sample to the moveout hyperbola intersection is used to represent the new sample on the NMO corrected trace. Once the NMO correction has been performed for all traces the stack is simply obtained by summing all the traces and scaling them.

\[ S_j = \sum_{j=1}^{M} C_j \frac{A_{j,j}}{M} \]

**Equireal 1** stacked value

\[ C_j = \left( \sum_{j=1}^{M} A_{j,j}^2 \right)^{-1} \]

**Equireal 2** for constant energy stack

\[ C_j = \left( \sum_{j=1}^{M} A_{j,j}^{-2} \right)^{-2} \]

**Equireal 3** for diversity stack

The problem with this approach is that the removal of the NMO delay, as described, is a time varying non-linear process and it tends to distort the trace, as the correction can only be made to the nearest sample. It can be shown that the moveout corrected signal suffers a power loss which varies with frequency, given by:

\[ 1 - \frac{(\sin(\pi Fdt))/\pi Fdt)^2}{2} \]

The lost power, from all frequencies, is distributed throughout the spectrum as white noise. With a 4 ms sampling rate the cutoff of the acquisition systems anti-aliasing filters is set at 62.5Hz and the power loss calculated for this frequency can amount to about 20%. 

A solution to this problem is to increase the sampling rate prior to applying the NMO correction by resampling the data using interpolation. It can be shown that if the sampling rate is increased to 1ms the loss of energy at 62.5Hz is reduced to only 1.3%, and if the resampling is taken to 0.25ms it is reduced to a negligible 0.1%.

Therefore an interpolation, resampling technique, based on the approach of Lu and Gupta (1978), was implemented as a part of the NMO correction algorithm. This interpolation is performed in the frequency domain, and allows the original trace to be resampled at an arbitrary rate without altering the frequency content of the trace. This is accomplished by Fourier transforming the original trace and multiplying by the factor \( \exp(-2\pi F dt) \) \( e^{-\omega dt} \), where \( dt = DT/2 \) is the time shift needed to generate another sample half way between two previous samples. If this is transformed back into the time domain and merged with the original data a trace with twice the original sampling rate will be produced. This can be continued to higher levels in a straightforward manner. If the rate is to be increased by a factor \( N \) of \( \exp(-2\pi F DT/N) \) \( e^{\omega DT/N} \) the transformed trace is multiplied by the factor and the resulting trace can be transformed back into the time domain and the process repeated \( N-2 \) times with the resulting traces being merged to give the data with the increased sampling rate.

In this program, therefore, the first step is to set up the complex interpolation array in the AP, to allow the interpolation specified by the user, up to 16 times, to be repeatedly applied to the data. Each trace is read from disc into pdp11 memory, and an index array of samples required after interpolation from the
Set up Exponential shift ramps in AP memory

Trace in PDP Memory

Transfer to AP

Apply Forward FFT

Apply Exponential Ramp

Repeat N times

Apply Inverse FFT

Transfer to PDP

N Trace Segments

Sort the Trace segments

New Interpolated Trace

Fig 4.15: Flow of the Interpolation Algorithm
uncorrected trace, to form the NMO corrected trace, is computed in the AP from the velocity information supplied. The trace is then transferred to the AP where it is scaled, muted if required, and has its mean level removed, before it is interpolated. The interpolated portions are returned to the pdp11 memory and the NMO corrected trace is then composed, using the index array previously calculated. This NMO corrected trace is both written out to disc and put into the AP, where it is added into the running stack which is permanently resident in the AP. When all N traces have been NMO corrected the stack trace is returned from the AP and is also written to the output file as the M+1th trace.

These traces can be displayed by using one of the gather plotting programs, allowing the NMO correction and stack quality to be examined. If this is performed for several consecutive gathers, with different constant velocities, the stacked traces can be selected for use in constant velocity stack panels, while the gathers are used in constant velocity gather analysis.

Standalone Interpolation

A program ANINT was written to apply the interpolation algorithm, described in the previous section, in a standalone mode, so that data can be interpolated to a higher rate, independent of the stacking programs. It takes its input from a disc file one channel at a time, produces the interpolated traces, using the AP as described previously, and then writes them out to a user specified output file.
This program can be used to interpolate the data to a higher rate before the semblance velocity analysis is performed. However it was found that the improvements in the semblance output produced are only marginal. The reason for this is probably that, in the velocity analysis, rounding the moveout trajectory to the nearest sample causes the semblance gates to be misaligned by up to one sample, but this tends to occur on a random basis from trace to trace. This effect is then averaged out in the semblance calculation and so the improvement to be derived from resampling is only small. However the program provides the user with a resampling tool if required, for this and other processes.

The basic algorithms for the Velocity analysis, interpolation and NMO correction program were developed in conjunction with A. Nunns (Nunns, 1980).

**CMP Stacking**

Once the Pre-Stack processing has been carried out and the velocity functions have been determined, the data on the line is ready for stacking. The main stacking program, MPSTAK, was developed from the algorithms described in the previous section, and it allows an entire line to be processed in one run.

The important capability of the stacking program is that it can interpolate velocity functions between points at which they are defined. Velocity analysis produces velocity functions at intervals of 20 to 50 CMP positions along the line. At a point in between two defined functions the velocity function has to be
interpolated using the values on either side. The user defines velocity functions at a set of CMP positions along the line, and the program interpolates the time and velocity of a particular event from those on either side, by linear interpolation. Therefore adjacent velocity functions must have the same number of layers defined. However if another layer has to be introduced at some point this can be accommodated by having two functions at consecutive CMP positions, as interpolation is not performed in this case and the program will continue after the second function with the new set of layers. The interpolation of velocities can be turned off if required, when the program continues to use the last defined value until an update position is reached. This can be useful in producing a brute stacked section, using some approximate velocity function for the whole line.

The program was designed as a tape to tape operation, but input and output to disc was also provided so that test stack panels could be easily produced without having to continually read the tapes. Data is read from tape to disc using TAPRED and then each trace in the gather is read into the pdp11 one trace at a time. The NMO correction is applied as in the NMO gather program, except that the NMO corrected traces are only added onto the running stack and are not written out. Once a stacked trace has been accumulated it is written to a temporary file on disc and then transferred back to tape as a single trace and a header block in the internal Post-stack format. The header block is updated by the program to indicate that the file now only contains 1 CMP stacked trace, and to record some of the stacking parameters, such as the number of input channels and the level of interpolation
Therefore once the velocity analyses have determined a suite of velocity functions for the line, this program can be run to produce the stack for the whole line in just one process.

**Post-Stack Processing**

A Post-Stack processing program capable of applying several different processing options to the post-stack data was designed along similar lines to the Pre-Stack processing program, with the user again having complete flexibility in the choice of processes and the order in which to apply them. The processes decided upon for this package were:

1. Edit
2. Gain Ramp Application
3. Mute
4. Spike deconvolution
5. Prediction error deconvolution
6. Bandpass filtering
7. Normalisation

**Edit**

It may have been that on displaying the CMP stacked section, that various traces were seen to be very noisy, and to interfere with events on either side to such an extent as to degrade an interpretation. Therefore this option allows the user to zero
selected traces, and having done so they are not passed through the remaining processes selected.

Gain Ramps

The same range of gain functions which were available in the Pre-Stack program are also included in this program. Therefore if desired the function applied before stack can be removed and another one, which is considered to be more suitable, applied. Therefore the type of gain function required and whether it is to be applied or removed, can be selected by the user.

Mute

In this post stack phase, a space variant early mute can be specified by the user, along with the length of a fixed length cosine taper. This can be useful with marine data, allowing any noise before the sea bottom to be muted out and the front of the data tapered. This should reduce the effects of any noise introduced during stacking, if the mute was not applied low enough prior to stacking.

Deconvolution

Stacking is a non-linear process, and as such leads to a distortion of the frequency spectrum of the data, which in turn can lead to a broadening of the seismic pulses forming the
reflection events. Therefore spiking deconvolution is usually applied to the data after stack in order to reduce the pulse width as much as possible. This type of deconvolution may also be more successful after stack because of the improvement in the signal to noise ratio brought about by the stack. If multiples are present in the post-stack data it may be possible to attenuate them using a prediction error filter, with a suitable gap and length. Also if the source pulses have been broadened and do not respond to spike deconvolution, it may be that a suitable prediction error filter can be found to compress the wavelet, and often this followed by a spike deconvolution produces better results.

**Frequency Filtering**

It is usual to perform as little frequency filtering as possible before stack so as to leave the data with as broad a band of frequencies as possible. As the stack process distorts the frequency content of the data it is possible that noise will be put into frequencies previously filtered, and increase the noise levels in unwanted frequency components. Therefore the unwanted frequencies are usually removed by bandpass filtering after stack, although care must be taken not to remove too much energy by filtering or the resolution of the primary wavelet can be reduced.
Normalisation

If amplitude variations across the section are such that the events are difficult to follow, and the absolute amplitudes are not considered important, or the variation is known to be caused by some acquisition malfunction, then it can be useful to normalise each of the post stack traces to unit energy. This will tend to minimise amplitude variations across the section making it easier to follow events laterally. Of course this must not be applied if the data is to be migrated.

Post-Stack program:- Summary

The post-stack processing program MPPOST uses the subroutines for frequency filter design, gain ramp design, and deconvolution design and application, which were designed for the pre-stack program. It also uses a similar type of data flow with only the tape transfers being really different.

As the post-stack data comprises only one trace per CMP position, with less than 2048 samples, it can be read straight into the pdp11 memory using TAPSUB, with no need to use temporary disc files. Input and Output to the disc is designed into the program as well, to allow processing tests to be carried out on data on disc.
Once the data has been read into memory, its header block is updated to give a record of the processes which are to be applied. The trace is then passed to the first process, which puts it into the AP, and all subsequent processes check and find that the data is in the AP, and so operate on it in place in the AP, as in the pre-stack program. This procedure cuts data transfers to a minimum, as only when all the processes have been applied is the trace retrieved from the AP. The processed trace, together with its updated header block are written back to tape using TAPSUB, or put on disc if required. The use of tape to memory transfers and the absence of disc reads in a production run, means that the post-stack program processes data at about the maximum throughput rate of the system, with the tape transfer times being the dominant factor.

**Post-Stack Mix**

A program MPTMIX was developed to provide a simple spatial filtering capability, in order to clean up sections for interpretation if migration was not going to be performed.

The design of the program is quite straightforward, in that it brings data into memory from disc, or tape, using TAPSUB, and mixes three input traces in the ratio 1 to 2 to 1, to produce a single output trace, which is then written back to tape or disc. This procedure tends to reinforce horizontal events while limiting steeply dipping events, such as refractions and diffractions, and so produces a more easily interpretable section with greater event continuity.
It was envisaged that this program could be easily upgraded to apply more complicated spatial filters if they are required. However in most cases of sub horizontal primary horizons and steeply dipping noise, even this simple process can produce a significant improvement in data quality.

**Stacked Section Plotting**

The interactive program MPSPLI can be used to plot sections, if the data are collected into one large gather type file on disc, and this is often a useful way of producing quick plots of small parts of a section when processing tests are being performed. However for producing the final section plot a more general package had to be designed.

The program MPSPLT was developed using the raster plotting algorithm to produce raster images of the final stacked section, from data on tape. The traces are read from tape using TAPSUB, into pdp11 memory and buffers of rasters, containing the seismic plot, are written back to tape. The program allows the time scale to be such that two strips of paper are needed to form the plot. This is accomplished by writing the rasters for the two portions of the plot to different tape drives. These can then be combined later or left on separate tapes for later processing.

The background grid for the section plot is produced by a program MPPLBK, which runs interactively, to get the parameters controlling the formation of the grid, and reads annotation and velocity functions from a disc file if they are required. The
plot output is generated using Versaplot routines, and this can be written either to disc or tape as rasters using MPRASM.

The raster images produced by MPSPLT and MPPLBK can be merged and put out to the plotter, from disc or tape, using the post processor MPMERG which expects just one background and one seismic plot as its input.

Header Interrogation

All the main processing stream programs, besides updating the fixed header values to reflect the changing state of the data, also put a set of codes into the free part of the header block, showing the type of processes which have been applied to the data, and the order in which they were applied. In order to convert these codes into an understandable format it was necessary to write a program to interrogate the header block and translate the processing codes.

The program MPHIST is run interactively, and expects its input to come from a disc file, whose name is specified by the user. The header block is read from the file and decoded. The program then produces an output listing giving all the acquisition parameters, as put into the header block by demultiplex, and all the processes applied to the data during the processing run. (Fig 4.16)
ENTER NAME OF FILE TO BE EXAMINED:
FILE DK3TSPRSDAT TO BE ANALYSED, LENGTH= 145 BLOCKS
FILE NO:484 TAPE NO:02282
SAMPLING INTERVAL SET AT 4 MSEC
AND RECORDING LASTED 12 SECS
ON 12 ACTIVE CHANNELS
THE FILE CONTAINS A COMMON MIDPOINT GATHER
CONSISTING 12 CHANNELS, STARTING AT SAMPLE NO: 8
AND ENDING WITH SAMPLE NO: 1536
TYPE OF SOURCE= AIRGUNS
AND THE UNITS OF LENGTH USED= METRES
RECIEVER DEPTH= 18.73
SOURCE DEPTH= 7.38
SOURCE-RECIEVER OFFSET= 297.38
RECIEVER SPACING= 185.88
SHOT SPACING= 58.38
SHOT REPITITION RATE= 28.83 SECS
USER INFO PUT INTO HEADER BLOCK
IS GIVEN BELOW
DURHAM UNIVERSITY 1988 CARRIBBEAN CRUISE LAST LINE
THIS IS A PIECE OF DATA DEMULTIPLEXED FOR A TEST
AIRGUN SOURCE 12 CHANNEL RECIEVER
THE FOLLOWING PROCESSES HAVE BEEN PERFORMED ON THE DATA
PRE-STACK PROCESSING CONSISTING OF
8 OPERATIONS IN THE FOLLOWING ORDER
TRACE EDITING
TIMEEXP(-0.27) AMP RECOVERY
MUTING
BANDPASS FILTERING
AMPLITUDE/ENERGY NORMALISATION
PREDICTION ERROR DECONVOLUTION
BANDPASS FILTERING
MUTING

Fig 4.16: Processing History of a Seismic Trace
as shown by MPHIST
Migration

In most cases where the reflecting horizons are relatively horizontal, and there is little faulting or folding to produce diffractions, a stacked section is usually of a good enough quality to be used for interpretation. However if there are dipping beds, faults and other disturbances producing diffraction events, then it is probably necessary to migrate the final section, in order to collapse the diffraction patterns and image the dipping events to their correct spatial locations.

The aim of migration is to produce a display corresponding to the subsurface geology from the seismic reflection data. The most useful model on which to base the migration process is that put forward by Loewenthal (Loewenthal et al, 1976). It is assumed that the CMP stacked traces represent coincident shot-receiver recordings and that reflection coefficients are small enough for multiple events to be neglected. The recorded traces can then be considered to be the same as those which would be recorded if a series of shots were simultaneously exploded, with strengths equal to the respective reflection coefficients, at every subsurface point, with the medium having half its true velocity. Hence a reconstruction of the wavefield for all depths at time zero, the shot instant, would give the geological structure.

If the two dimensional wavefield produced by the experiment described is represented by \( U(x,z,t) \) where \( x \) is the horizontal location, \( z \) is the depth and \( t \) is the time from the shot instant, then the seismic trace can be considered to be, \( U(x,0,t) \), the
recording of the wavefield at the surface \( z=0 \) for all time. The migrated section can therefore be considered as \( U(x,z,0) \), the wavefield at all depths at the time \( t=0 \).

In all modern applications it is considered that the wavefield \( U(x,z,t) \) satisfies the scalar wave equation, given by:

\[
\frac{1}{v^2} \frac{\partial^2 U}{\partial t^2} = \frac{1}{v^2} \frac{\partial^2 U}{\partial z^2} + \frac{1}{\partial x^2}
\]

The task of migration is to obtain the values \( U(x,z,0) \) from the recorded values of \( U(x,0,t) \) by solving the acoustic wave equation.

It was decided to design algorithms for and implement migration for two different approaches to the problem, Kirchhoff summation, and Finite Difference Migration. These methods use solutions to the wave equation to perform migration, but approach this solution from two different viewpoints.

**Kirchhoff Migration**

Starting from the scalar wave equation it is possible to develop the mathematics from several approaches. It is instructive to first consider the development of a solution to the problem in the Fourier domain. Therefore if we consider \( U'(Kx,t,Wt) \) to be the Fourier counterpart of \( U(x,z,t) \), then the scalar wave equation can be written as:
This can be solved in the case of upgoing waves to give the solutions in equation 2. The second of the two solutions is for evanescent waves, and as can be seen this is an exponential function which would tend to blow up under downward continuation. Therefore the evanescent energy has to be treated as noise and just the first equation used for the wave energy. This equation is in fact the basis of F-K migration.

\[ u'(K_x, z, \omega_c) = U'(K_x, 0, \omega_c) e^{i \omega \sqrt{(\omega_c^2 - K^2)} t} \]

Now if one considers the solution to the wave equation from the integral approach, the starting point is Kirchhoff's integral solution to the scalar wave equation. It can be shown from this equation that the solution for \( U(x, z, t) \) is given, in integral formulation by (Godbold, 1980):

\[ U(x, z, t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U((x-x'), (0, (t-t')) M_2(x,t) \, dx' \, dt' \]

\[ M_2 = -\frac{1}{2\pi} \frac{\partial}{\partial t} \left[ \frac{H(t' - r'/v)}{(t' - r'/v)^2} \right] \]

\[ H(t') = \begin{cases} 1 & t' > 0 \\ 0 & t' < 0 \end{cases} \]

\[ \gamma' = (x' + z^2)^{3/2} \]
It is interesting to transform the operator $Mz$ into the Fourier domain and examine its counterpart $M'z$, which is given below.

$$M' \equiv e^{-2 \frac{\omega \cos^2 (\omega_0 - \omega_1) + \omega_1}{\omega_2}} \frac{\omega - \omega_1}{\omega} \left| \omega > V |K| \right|$$

Eqn 4

It is interesting to note that this operator now has the same form as the phase shift operators used in the F-K solution to the wave equation. Therefore the Kirchhoff approach can be seen to be providing the same type of solution to the wave equation, but is expressing it in the time distance domain rather than in the F-K domain. The Kirchhoff representation can be developed to allow the wavefield to be downward continued, to reconstruct the wave values in the earth at time $t=0$, which according to our model should provide an illumination of the geological structure.

$$M \equiv \int \int \int U(x, z, c) \frac{\partial}{\partial c} \left[ \frac{H(t, \tau)}{(c^2 - c_0^2)^2} \right]$$

Eqn 5

$$M' \equiv e^{-2 \frac{\omega \cos^2 (\omega_0 - \omega_1) + \omega_1}{\omega_2}} \frac{\omega - \omega_1}{\omega} \left| \omega > V |K| \right|$$

Eqn 4

It is usual not to actually migrate from time into depth coordinates, but rather to use a migrated two-way travel time coordinate, so that errors in velocity estimation do not result in too large a distortion in the migrated data. Therefore a two-way travel time $\tau$ is defined as $\tau = z/v$ and so the migration equations
can be couched in terms of x and \( \tau \) rather than x and z.

\[
U_m(x, \tau) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U((x-x'), 0, \tau) M_t \; dx' dx
\]

Eqn 6

\[
M_t = \frac{1}{\sqrt{\pi}} \frac{d}{d\tau} \left[ \frac{H(\tau+\tau_c)}{(\tau^2-\tau_c)^{\frac{3}{2}}} \right]
\]

\[
t_c = \left( \frac{x^2}{v^2} + \tau^2 \right)^{\frac{1}{2}}
\]

In practice, of course, the data is not continuous in time and space, but defined over a grid of surface and time positions. Therefore the integral formulation of the equations has to be replaced by finite sums and the operator \( M_t \) must also be digitised. In order to derive a digital representation of the operator it has to be expressed in terms other than the ones shown above in order to avoid the singularity inherent in the expression. This reevaluation of the operator can be performed following the treatment by Berryhill (Berryhill, 1979), as follows.

\[
t_c = \left[ \left( \frac{\Delta t}{v} \right)^2 + \tau^2 \right]^{\frac{1}{2}}
\]

Eqn 7

\[
t_c \psi_n^k = H(k\Delta t - t_c) \; t_c \left[ \left( \frac{k\Delta t}{t_c} \right)^2 - 1 \right]^{\frac{1}{2}}
\]

\[
k_0 = \left( \frac{t_c}{\Delta t} \right)
\]

\[
\psi_n^k = 0 \quad \text{for} \; k \leq k_0
\]

\[
= \left[ \left( \frac{k\Delta t}{t_c} \right)^2 - 1 \right]^{\frac{1}{2}} \quad k > k_0
\]
this gives the alternative

$$M_{nK} = \frac{2}{\pi \nu t_0} \frac{\partial^2}{\partial \varepsilon^2} \left[ H(t - t_0) (\varepsilon^2 - \varepsilon^2) \right] \partial \varepsilon \partial t$$

using the standard central difference notation this allows it to be expressed in discrete terms by:

$${\text{eqn } 8} \quad M_{nK} = \left( 2 \psi_{nK} - \psi_{n,K-1} - \psi_{n,K+1} \right) \frac{\partial \varepsilon}{\partial \varepsilon \partial t}$$

Hence the discrete expression for the Kirchhoff migration expression is given by:

$${\text{eqn } 9} \quad U_m(\Delta x, \varepsilon) = \sum_{n=-N}^{N} \sum_{k=0}^{K} U((n \Delta x, 0, k \Delta \varepsilon) \cdot M_{nK}$$

The above equation was used as the basis for an implementation of Kirchhoff migration for the processing system, and it was implemented as two separate routines. One program MPOGEN is responsible for generating a series of migration operators, for a particular migration model, according to the definitions in eqns 7 and 8, and the second program MPKMIG uses the operators generated by the previous program to perform the migration, as defined in eqn 9 on the data.

In designing the implementation of this procedure for the processing system, the results of work performed by C Godbold (Godbold, 1980), on the effect of approximating the operators used in the migration, was used to allow realistic limits to be placed on some of the migration parameters, so that the programs could be designed to run within the limited memory
and disc resources of the pdp11. As a result of this work it was decided to limit the operator to 5 samples, as this had been shown to provide quite adequate accuracy in the migrated output. It was also decided to use the operator upgrade criteria in order to limit the number of operator values which have to be calculated. This basically works out the positions at which the previously defined operator is no longer a reasonable approximation to the actual operator at a particular point, given a certain acceptable percentage error in the operator evaluation. It was found that with a specified acceptable error of about 1% quite reasonable results were obtained and the number of calculated operators was quite drastically reduced. If a new operator had to be calculated at each sample position, then for an 8 second record the operator would have to be recalculated 2000 times, whereas with only 0.5% allowable error this number is reduced to about 350, which is obviously a considerable saving. However the update has to be faster at low travel times than at later ones, and so with deep marine data with the water bottom a few seconds deep even larger savings can be made with only about 50 operator calculations being necessary.

The constraints on the migration parameters were determined from the amount of available disc space and AP memory. The operator values obviously have to be written out to a disc file once they have been calculated, and this file is limited in size by the storage available on a single disc platter. Also from an evaluation of how the algorithm could be programmed to use the AP, it was decided that in order to fit into the AP memory the half-width of the operators, assuming a 5 sample operator, would
have to be limited to 512 traces. A suitable value for the operator half-width can be calculated from an estimate of the dip of the events at the deeper part of the section, or even at shallower positions if the dips are larger, using the relation given below:

\[
\text{migration aperture} = X = Z \tan \phi \text{ where } z=\text{depth}, \phi = \text{angle of dip}
\]

Obviously for a particular application the half-width may well be less than the maximum value and the number of operators which can be calculated will be a function of the halfwidth, so that it will fit in the available disc space. If a reasonably large half-width is specified for the operator, then a full trace migration would be limited to about 1% error in the operator update calculation. However the top and bottom of the data could be migrated separately in two passes so that the operators were calculated at separate times, if a greater degree of accuracy were required.

The user inputs the operator half-width, its time range for application, and a velocity model of RMS velocity against two-way travel time to the operator generation program. Given the percentage acceptable operator error, the program evaluates the operator update positions necessary for this degree of error inside the specified migration time range. The program uses this information to evaluate the 5 point migration operator at each of the update positions, over the specified half-width. This is written out, with its associated positioning information to the user specified disc file in the format shown in Fig 4.17.
Each different set of values starts on a disc block boundary

NOP = Number of operators to be applied

NRANGE = Range of validity of the operator as a sample value on the central trace; number of values = Number of Operators

NLEAD = Starting position for each operator on each trace; number of values = Operator half-width × Number of operators

OPVAL = Operator values; number of values = Number of operators × Operator Length × Operator Halfwidth

Fig 4.17: Kirchhoff Migration Operator storage
The operator data can then be used to migrate any data with this particular structure. By giving the migration program MPKMIG the data files and the operator file, the operator can be convolved with the input data one trace at a time, in the AP, across the full width of the operator to produce an output trace, fully migrated in the specified time range, which is written out to another disc file.

The input data are expected to be in a trace sequential form in a single disc file and are put back to another disc file in the same form. These files can be easily generated by reading data onto disc using MPTFDK and MPSORT can be used to put the data back to tape. The number of traces which can be handled in one pass is dependent only on the number of traces which can be put into a single file. If it was necessary to migrate a long line, this would have to be performed in short sections as a roll on roll off type of sequence, with enough traces at each side of the active block to accommodate the half-width of the operator.

This type of migration is very useful, as its range of application can be easily limited, allowing it to be applied only in regions of interest, rather than over the entire section. If the dips of events on the section can be estimated the aperture of the operator can often be limited as well, again reducing the number of calculations to be performed. Therefore these programs allow the user to tailor the migration to his own particular needs.
Fig 4.18: The Diffractor Model of Fig 4.3 after Kirchhoff Migration using MKKMG
Kirchhoff migration has been shown to be often quicker than the finite difference approach and it tends to migrate even high dips relatively accurately. However, on the other hand it does tend to produce organised noise, which in areas of low signal to noise ratio can make interpretation after Kirchhoff migration quite difficult.

Finite Difference Migration

The finite difference approach to migration, first popularised by Claerbout (Claerbout and Johnson, 1971), is the one undergoing the most research at the present.

\[ \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2} = \frac{1}{V^2} \frac{\partial^2 u}{\partial t^2} \]  

(eqn 1)

This method is also based on the scalar wave equation, but here the aim is to represent the equation by a finite difference formulation to allow the wavefield to be downward continued. For computational purposes, to keep the wavefield on the computational grid, it is usual to express the equation in terms of a retarded time system as shown below:

\[ t' = t - \int \frac{dz}{V} \]

(eqn 2)

\[ \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2} = -\frac{2}{V} \frac{\partial^2 u}{\partial z \partial t} \]
For small dips it is possible to neglect the terms in $\frac{\partial^2 u}{\partial z^2}$ and so we are left with the well known 15 degree approximation to the wave equation.

$$\text{eqn 3} \quad \frac{1}{2} \frac{\partial^2 u}{\partial z^2} + \frac{\partial^2 u}{\partial x \partial t} = 0$$

If there are larger dips in the data than about 15 degrees then this approximation is too severe and a less limited approximation must be developed. Therefore by differentiating eqn 2 with respect to $z$ and substituting for $\frac{\partial^2 u}{\partial z^2}$ we can derive:

$$\text{eqn 4} \quad \frac{\partial^3 u}{\partial x^3} + \frac{\partial^3 u}{\partial z^3} = \frac{2}{V} \frac{\partial^3 u}{\partial x \partial t} + \frac{4}{V^2} \frac{\partial^3 u}{\partial z \partial t}$$

Again a further approximation can be made by dropping the term in $\frac{\partial^3 u}{\partial z^3}$ to give the following:

$$\text{eqn 5} \quad \frac{\sqrt{\frac{1}{4}} \frac{\partial^3 u}{\partial x^3}}{\partial t} - \sqrt{\frac{1}{2}} \frac{\partial^3 u}{\partial x \partial t} - \frac{\partial^3 u}{\partial z \partial t} = 0$$

This is known as the 45 degree approximation to the scalar wave equation. Once again, as the migration in $z$ depends on knowing the velocity model reasonably accurately, it is better to use a migrated two-way travel time $\tau$ which is not so susceptible to errors in the velocity specification. Hence the equations below can be formed.
15 degree equation \[ \frac{V^2}{2} \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial t} = 0 \]

45 degree equation \[ \frac{V^2}{4} \frac{\partial^3 u}{\partial x^3 \partial t} - \frac{V^2}{2} \frac{\partial^3 u}{\partial x^2 \partial t} - \frac{\partial^3 u}{\partial t^3} = 0 \]

These equations can be used to derive the finite difference representations which will allow the wavefield to be downward continued.

For 15 degree given the following:-

\[ V = \frac{V}{2} \]
\[ t = n \Delta t \]
\[ z = \frac{1}{2} \Delta t \]
\[ x > \frac{1}{2} \Delta x \]

\[ \Delta = 0.5 \frac{V^2 \Delta t \Delta z}{10 \Delta x^2} = \frac{V^2 \Delta t \Delta z}{32 \Delta x^2} \]

\[ \frac{\partial^2}{\partial x^2} = \frac{1}{\Delta x^2 (1 + bT)} \]

\[ \frac{\partial}{\partial t} = \frac{2}{\Delta t} \left( \frac{(E_t - I)}{(E_t + I)} \right) \]

\[ \frac{\partial}{\partial z} = - \frac{2}{\Delta t} \left( \frac{(E_z - I)}{(E_z + I)} \right) \]

\[ E_z = \text{Advance Operator} \quad \therefore E_z P_z = P_{z+\Delta z} \]

\[ [(b - \Lambda)T + I] U_{k+1,j} = [(b + \Lambda)T + I] U_{k+1,j} + [(b + \Lambda)T + I] U_{k,j} - [(b - \Lambda)T + I] U_{k,j} \]

* c.f. Berkhout (1979)

† c.f. Loewenthal et al. (1976)
It can be seen that this equation is a tridiagonal matrix equation in which the solution for $U_{k,n,j}^n$ can be found if the right hand side is known. The right hand side can be determined by specifying boundary conditions, that the wavefield is zero at all points outside the recorded data area.

Similarly for the 45 degree equation:-

The difference operators:

$$A = \frac{0.259 \sqrt{\Delta \xi}}{4 \Delta x^2} = \frac{v^2 \Delta t}{4 \Delta x^2}$$

$$B = \frac{\sqrt{\Delta t \Delta \xi}}{32 \Delta x^2} = \frac{v^2 \Delta t \Delta \xi}{2 \Delta x^2}$$

$$\frac{\partial^2}{\partial x^2} = \frac{T_x}{\Delta x^2 (I + bT_x)}$$

$$\frac{\partial^2}{\partial t^2} = \frac{T_t}{\Delta t^2}$$

$$\frac{\partial}{\partial \xi} = \frac{2}{\Delta \xi} \left( \frac{(E_t - \xi)}{(E_t + \xi)} \right)$$

$$\frac{\partial}{\partial t} = \frac{1}{2\Delta t} \left( E_t - E_t^{n-1} \right)$$

The centering of the time time difference operator is different in this case to provide stability in the solution of the equations. Using the above difference operators it is possible to derive the following for the 45 degree equation:-

45 degree finite difference expression

$$\left[ I + (b \cdot \beta)T_x \right] E_{\xi} E_{\xi}^{-1} = \left[ I + (b \cdot \beta)T_x \right] E_t - \left[ I + (b \cdot \beta)_x \right] E_{\xi} E_{\xi} - \left[ Z + (2b + A) \right] T_x \left[ E_{\xi} E_{\xi}^{-1} \right]$$

$$\left[ I + (b \cdot \beta)_x \right] U_{k,n,j}^{n-1} = \left[ I + (b \cdot \beta)_x \right] U_{k,n,j}^{n+1} - \left[ I + (2b + A) \right] T_x \left[ U_{k,n,j}^{n+1} \right]$$

$$+ \left[ Z + (2b + A) \right] U_{k,n,j}^{n-1} + \left[ I + (b \cdot \beta)_x \right] U_{k,n,j}^{n+1}$$
Once again given the boundary conditions the factors on the right hand side of the equation can be evaluated and so we are left with an tridiagonal matrix equation with the vector in \( x \) \( U_n^{n+1} \) as the unknown.

From the equations it can be seen that the downward continuation formula is expressed in terms of vectors in the \( x \) plane. Therefore before finite difference migration can be performed the data must be multiplexed into \( x \) vectors, or "time sliced". Two programs, MPSLIC and MPUSLC, were written to perform the "time slicing" and trace sequential sorting respectively. So as to keep the data in integer numbers of disc blocks the traces are padded, equally on either side, with zero traces to make an integer number of 128 traces. This has an added advantage of providing a zone at the edge of the real data for the edge effects of the migration operation to be dispersed into, tending to prevent reflections from the side of the computational grid. The trace sequential resorting program assumes the number of traces specified will be padded on either side by zero traces so that this is all transparent to the user.

Once a time sliced data file has been produced this can be used as the input to one of the two finite difference programs. Both the 15 and 45 degree algorithms were implemented because, for shallow dips the 15 degree algorithm works quite adequately and takes about half the time to run as the 45 degree approximation. On the other hand when large dips are present in the data the 45 degree algorithm has to be used if a reasonable result is to be produced.
In order to fit the algorithms into the AP, it was decided to limit the number of traces which can be migrated in one run to 1024. An LU factorisation method was used to solve the tridiagonal matrix equation. This is a two pass algorithm which factorises the left hand side coefficient matrix in the first pass before using the factorised results to solve the equation in the second pass. This algorithm has distinct advantages over the method proposed by Claerbout (1976).

If we have an N length vector

Claerbout approach...3N mult, 2N div, 3N add/subtract

LU factorisation step 1...N mult, N div, N sub
    step 2...2N mult, N div, 2N add/sub

Total...3N mult,2N div, 3N add/subtract

It can be seen that the two methods involve the same number of calculations in solving for one vector. However in the case of downward continuation the left hand side coefficient matrix remains constant for a particular Δt step and so the factorisation need be performed only once, for each downward step. This results in a considerable saving over the other method which has to perform the complete solution at each step. Therefore the LU factorisation algorithm was microcoded for the AP into two Fortran callable subroutines to provide a quick solution of the equations at each step.
Fig 4.19: The Diffractor Model of Fig 4.3 after 15° Finite Difference Migration with MPFDIS
A velocity model can be input to the programs, which can vary in velocity with two-way travel time, and small lateral variations are also allowed, but these must be made to be gradual in their nature or else the solution to the equations can become unstable. The functions can be defined at different trace positions and are then interpolated in a similar way to the stack program. The depth variation is not constrained in the same way, although velocity variations, greater than one downwards step are averaged out by the program.

The user has to specify the downward continuation step size $\Delta z$ and also how many downward steps to perform. The program converts the velocity functions, which are input as RMS velocities, into interval velocities for each depth step internally. The program then downward continues the entire dataset one $\Delta z$ step at a time to determine the values of $U(x, n\Delta z, 0)$ for each value of $n$ down the section. The downward continued $x$ vectors are written back to the input file so that at any stage this file contains a mix of fully migrated data and that which has been downward continued to position $n\Delta z$.

This method of migration is very time consuming as the remaining non-migrated data is handled for every downwards step, although, of course the number of remaining non-migrated data samples is reduced at each $\Delta z$ step by $M$, where $M$ is given by $M=\Delta z/\Delta t$, as the step $\Delta z$ can be larger than $\Delta t$.

The two algorithms give good results, although neither can migrate data containing large dips as well as the Kirchhoff algorithm. On the other hand the finite difference approach tends
to cause noise to be dissipated rather than organised, so that the results are often clearer, with less background noise than a comparable Kirchhoff output. The choice of migration approach has to be based on several criteria, dips, signal to noise and run time. The Kirchhoff program takes less time to execute than the 45 degree finite difference algorithm but is slower than the 15 degree method and so for simple low dip structures the 15 degree finite difference algorithm is probably the best choice.

Summary

With the descriptions in this chapter it has been shown, that a full suite of seismic processing programs have been developed, fulfilling the original aims. Also, perhaps more importantly, a working data structure has been developed, which makes the system more than just a collection of programs. In addition the basis behind the entire structure has been to provide a template for future development, a foundation on which further procedures can be built. Although there are, no doubt, some areas where improvements could be made, a working, complete seismic processing system has been designed and implemented, fulfilling the original design aims.
In order to assess the system and to show others how it was intended that it should be used, two pieces of real data were processed by the Author. The first line from the Norwegian Sea-Jan Mayen area was acquired with the departments SDS10/10 system in SEG-A format in 1977, and consists of 11 channel, 7 second data recorded at 4 milliseconds sample interval. This line was processed when the system was partially complete in order to fully test those components thought to be working and to indicate any shortcomings in the system at that stage. The second line was from the 1980 Caribbean experiment, and was also recorded using the SDS10/10. This line was processed when the system was virtually complete. Unfortunately, the migration programs were still undergoing their final development at this time, and could not therefore be included in the trial. The processing of this line, besides acting as a thorough test for the system, was basically designed as a demonstration for future users of the capabilities of the system, and how it should be used.

A brief description of the processing of these two pieces of data is included, in order to illustrate the data flow through the system, as well as the geophysical factors which have to be considered when processing seismic reflection data.
Data Flow

The programs within the processing system are basically set up for tape to tape processing operations. That is data is read from tape, processed, perhaps using temporary files on the disc, and the final product is written back to tape for storage. However, all of the programs allow data input and output to be directed to the disc, so that small amounts of data, such as those being used in filter tests or velocity analyses, can be easily accessed without having to use the tapes all the time. Data can be extracted from the tapes and put onto the disc by utility programs, such as MPTAPH (see Appendix 2), for use in this mode.

It was envisaged that the data flow through the system would be very much like that shown in Fig 5.1, and this was in fact confirmed by the experience gained in processing the two test lines. After Demultiplex and Sort, which are both definitely tape to tape processes, data are frequently dumped to disc to allow data tests to be performed before the next tape to tape process is initiated, using the parameters decided upon during the tests. It is important to realise that a lot of time must be spent performing thorough tests on a wide range of data segments along each line, if the best possible final section is to be produced. When the input parameters have been determined for a particular process, they can be put into its input file, and once it is started the only interaction with the operator will be for tape changing and error reports.
Fig 5.1 : Processing System data flow
Jan Mayen Data Processing

The Jan Mayen data were recorded with an 11-channel streamer, each channel separated by 100m and with a 228m offset from the seismic source. The source was composed of three airguns, two of 160cuin, and one 300 cuin. Data records of 7 seconds in length with a sampling rate of 4 milliseconds were recorded, with a 62.5 Hz antialiasing filter applied before digitising. These data were recorded in SEG-A on 9 track magnetic tape using the SDS10/10.

As it was a test run, the data were demultiplexed and sorted at the same time, so that CMP gathers were written back to tape. The fast mode of the demultiplex program was used, as earlier tests had shown that there were no obvious problems with the field tapes. About 600 shot points were demultiplexed, although only the first 300 or so CMP points were intended for the final display, this being the most interesting portion of the line on the monitor displays. However, it was decided to put the whole dataset into the processing system, to see how well it stood up to handling large quantities of data.

Data were chosen for tests and velocity analyses and brought down onto disc from the CMP gather tapes. These were spaced at about every 40 CMP positions over the first part of the line and then about every 100 over the area of lesser interest at the end of the line. When these gathers were displayed it was found that channel 9 was dead on some of the records, presumably due to some acquisition fault, and channel 10 had been recorded with the wrong
Fig. 5.2a: An Example of a Jan Mayen Shot Record
polarity with respect to the rest of the data. On the whole, the data quality was quite reasonable with only minimal noise corruption.

The gathers on disc were used as the input to the pre-stack processing program, and different filter and deconvolution parameters were tried out until the best results were obtained. Velocity analysis was then performed on each of these gathers, after the pre-stack processing had been applied. These analyses allowed the velocity functions to be determined at every 40th CMP point along the zone of interest.

At this point, a stream of processing was set up which attempted to minimise the amount of tape access. The pre-stack processing was performed on short segments of data read from tape, and its output was written to the disc. These pre-stack processed gathers were then used as the input to the stack program, whose output was written back to tape. Although this was quite quick in terms of processing time, in comparison with two tape to tape operations, it required much more operator intervention, and it cannot be recommended as a viable method for anything other than small datasets.

After the stack was complete, a display of the stacked section was produced. The section was reasonably good and had a high signal to noise ratio, with very good suppression of multiples. However, the bubble pulse of the airguns was still present as two distinct pulses, and this was hindering the resolution of the structure. Therefore, prediction error deconvolution was applied to the data followed by a bandpass
filter to produce the final display seen in Fig 5.2b. It can be seen that the post-stack deconvolution has improved the resolution of the data, and the bandpass filter has helped to keep the signal to noise ratio at reasonable levels.

The processing of this line showed that single tape to tape operations are far easier than complicated sequences which use the disc as temporary storage between different modules. It also indicated the need for the spectral analysis program which was not available at the time, and without which bandpass filtering was difficult to setup. However, the system in its then still rudimentary form coped relatively easily with the processing of this dataset and produced a reasonably high quality final section.
Fig. 5.2b: A Portion of the Final Section for the Jan Mayen Line
Jan-Mayen Processing Details

1. Demultiplex and Sort into 11 channel CMP gathers
2. Amplitude correction - $te^{0.2t}$
3. Polarity reversal channel 10
4. 150ms Spike Deconvolution, 5% prewhitening
5. Bandpass filter 5-10/40-45 Hz
6. Velocity analysis, every 40th CMP
7. NMO correction with 8 fold interpolation
8. Stack, 11 fold CMP
9. Prediction error deconvolution, 100ms gap,
   100ms filter 5% prewhitening
10. Bandpass filtering 5-10/40-45
Caribbean Arc 1980

The data from the Caribbean region were acquired aboard RRS Discovery on cruise 109 during April 1980. The seismic reflection data were acquired using a 12-channel streamer with a 3-airgun array composed of two 160cuin and one 300cuin guns, used as the seismic source. The data were recorded in SEG-A using the departmental SDS10/10 recording system, and over the area in question 12 seconds of data was recorded, due to the fact that there was over 6 seconds of recording before the water bottom arrival was received.

As well as acting as a demonstration of the system's capabilities, the processing of this line was a very good test for the system in its nearly-finished form. Only the migration algorithms in their final stages of testing could not be applied to this data.

As the data were recorded in such deep water, it was decided to setup the demultiplex to only keep the last 6 seconds of data. Therefore, 460 shot points were demultiplexed into 12-channel common shot point gathers and written back to tape. The first field tape was processed with the demultiplex in the fast mode, but it became apparent that the program was frequently switching to the slow mode because of inconsistencies in the redundancy checks. Therefore, the remainder of the line was demultiplexed in the slow mode, with very lenient error allowances. Even so several files were declared dead, and zeroed by the program when it was unable to recover from serious data errors. An analysis of
the detected errors indicated that a malfunction in the timing code generator probably caused by dropping a bit, was the cause of most of the problems. The files where several errors occurred also had parity errors recorded even after three retries, so these were most likely caused by a bad piece of tape.

From the monitor record it could be seen that the sea bed was quite undulating, and so it was decided that velocity analyses would be carried out every 20 CMP positions in order to give as good a stack as possible. Therefore the data were sorted into CMP gathers in a tape to tape operation, and then every 20th file was brought down to disc for data tests and velocity analyses. On examining the CMP records, the data were seen to be of a very poor quality. All the channels were contaminated by high frequency noise, with channel 7 being completely immersed. Also on some records several of the channels contained quite large amounts of low frequency noise, especially channel 2. On the other hand the two noisy channels, 2 and 7, were the ones with the most recorded energy, implying that the pre-amplifier gains on the recording system had not been set correctly. Also, on further examination it was possible to see that on the remaining channels the waveform had a clipped appearance, which was presumably caused by a fault in the acquisition system's gain ranging logic.

Spectral analysis plots showed quite clearly that the high frequency noise was at 50Hz, presumably due to pick up from the ship's electrical supply. The low frequency noise was centered on about 11Hz, and may have been due to the ships propellers or cable snatch on the streamer. The clipped data showed the presence of energy in the traces well above the 62.5Hz cutoff of the
Fig 6.3: Display of a Caribbean Arc Shot Record
antialiasing filter, indicating that the clipping of the data and the high frequencies it produces in the data were introduced during digitising. In order to analyse this problem, sample shot points were completely demultiplexed and displayed. The high amplitude first break events were correctly digitised, implying that the low amplitude response of the acquisition system was at fault or that the A-D converter was losing gain information.

From this preliminary analysis of the data it was fairly obvious that bandpass filtering had to be applied before any other processes, because in its raw state the signal to noise ratio of the data was so poor. The filter chosen was one with a complete cut below 10Hz and above 45Hz, with a 5Hz taper at each end. This filter succeeded in diminishing both major sources of noise. Because of the wide range of amplitudes across the channels it was decided to normalise the traces to unit energy after applying the exponential gain recovery curve. At this point, there was just sufficient detail to allow deconvolution tests to begin. However with the distortion of the waveform on most channels and the quite heavy filtering which had been applied, little hope was held for good results from the deconvolution. After many trials it was felt that the best results were given by a 200ms spike deconvolution with 5% prewhitening, which seemed to sharpen up the waveform to an acceptable level.

Once decided upon, this suite of pre-stack parameters was applied to the test gathers on disc and then to the data on tape, in one tape to tape operation. The results of this processing were seen by sorting out channel 2 for a single channel display of the pre-stack data.
Fig.5.5: A single channel display of a portion of the Caribbean line after Pre-Stack Processing.
The velocity analyses were carried out on the data on disc to produce a contour plot for each test point. Using these displays and the single channel section it was possible to pick a set of velocity functions for the line. In performing the velocity analyses several different gate widths were tried in order to give as clear a contour display as possible. In the end a reasonably large gate of 184ms was used, which was very similar to the length of the deconvolution filter, suggesting that this was the effective length of the airgun waveform.

The set of picked velocity functions were put into the stack program and the data for the whole line was stacked in one tape to tape operation. The stack tape was then used to produce a display, which was also saved to tape. A segment of the stacked data was put onto the disc from the tape to enable testing of post-stack processing parameters. It was found difficult, even after a full range of tests to find a deconvolution operator which would adequately improve the data. In the end a short spiking deconvolution operator was used in an attempt to increase the resolution of the data as much as possible. A bandpass filter with the same cutoff as in the pre-stack processing was applied after deconvolution to provide an improvement in the signal to noise ratio in the post-stack data. Once again, these parameters were entered into the program and the entire line was post-stack processed in one tape to tape operation.

Due to the large number of diffractions on this line it would normally have been desirable to perform migration, and probably because of the depth of the data, Kirchhoff migration applied over the 7 to 9 seconds range of the traces with an aperture of about
Fig. 5:6 A Portion of the Caribbean line after stack
50 traces would have sufficed with very few operator updates being
required. Probably about 40 operators would be required and the
limited time range would allow it to be applied quite quickly.

However, the migration programs were not available and so a
three-trace mix was run, again tape to tape, in order to clear up
some of the diffractions and reinforce the subhorizontal events in
order to aid interpretation. In fact, the two major dipping
events were rendered much clearer by this process and a major part
of the diffraction energy was removed, making the display much
clearer than before.

The final sections present a very clear picture of oceanic
crust dipping under an accretionary prism, and considering the
original data quality the final results are very pleasing. It was
felt that the ability to process data to this standard indicated
the flexibility and capability of the processing system, and,
apart from the migration programs not being available, there was
probably no other process which could have been applied to
significantly improve the data quality of the final sections.
Fig 5.7: A portion of the Caribbean line after all processing
Caribbean Processing Parameters

- Shot-receiver spacing: 304m/297m
- Channel spacing: 100m
- Array Depth: 10.7m
- Source Depth: 7.3m

1. Demultiplex SEG-A to 12 channel 6-12 seconds common shot gathers for 460 shots.
2. Sort, common shot to CMP gathers
3. Amplitude correction $t \cdot e^{0.2t}$
4. Mute 200ms with 80ms cosine taper
5. Bandpass filter 10/15-40/45 Hz
6. Spiking deconvolution, 240ms operator, 5% prewhitening
7. Velocity analysis, every 20
8. NMO correction with 8 level interpolation
9. CMP stack
10. Spike deconvolution, 72ms, 5% prewhitening
11. Bandpass filtering 10/15-40/45 Hz
12. Three trace mix
Summary

From the experience gained processing these two test lines, one method of use of the system can be recommended as the easiest and most flexible. It was found that by far the most straightforward method of working was to carry out all major processing runs as tape to tape operations, and only to use the disks as temporary storage for test data. It is also fairly obvious that this means a reasonable amount of time has to be spent running tests on the data before embarking on a major processing run. It is advisable to pick representative data from many positions on a line, and if the parameters required to process them are very different, to break the tape runs at certain points to enter a new set of parameters, rather than trying to process the entire dataset with the same average values.

In its present state of development, the processing system should allow the routine processing of marine seismic reflection data without any further additions to the software suite. However, further developments might be necessary to accommodate other types of data. At the time of writing, the system is in routine use for the processing of the main bulk of the Caribbean seismic data.
Chapter 6
Conclusions

As this project was the first one involved with the development of the Durham University Seismic Processing System, it was very important that, while the basic system was under development, the possibilities for future improvement, in both hardware and software, should be critically assessed. Also the software was developed throughout with the concepts of flexibility and portability uppermost, so that if the operating hardware of the system should change, the number of software changes required would be small, and those that were necessary would be straightforward to make, which should also allow new programs to be added to the system fairly easily. The first part of this chapter is an attempt to evaluate those software elements which could be added to the processing system in order to complete it in its present configuration, and the second part is an evaluation of the hardware upgrades considered necessary to improve the system.

Future Software Developments

Due to the project being constrained by a time limit, several restrictions were placed on the aims of the system at its conception. The system as designed and implemented was intended for processing marine data acquired in SEG-A using the departmental acquisition system, and be able to handle data in the processing stream's internal format. However, it was always envisaged that these restrictions would be purely temporary, and would be removed by the addition of further software to the system.
by other projects in the future. Therefore, in designing the system this was taken into account so that the addition of further modules, allowing the above limitations to be removed, should be relatively easy. Also as the structure of the system, its data format and data handling conventions are well established, the development of newer and possibly more sophisticated techniques for inclusion in the system should be relatively straightforward. New techniques could be developed on the NUMAC IBM 370, while the system is being used for processing, and, once tested and refined, tied into the processing system using the data handling subroutines already present, and following the conventions used by the other modules already in the system.

Demultiplex

In any future project one of the first items which should be considered is the development of a demultiplexing capability for SEG-B and SEG-C. The existing demultiplex program for SEG-A could be easily used as the basis for two new programs, one for each format, as the basic pattern of data flow should be the same. The header blocks of the 3 formats are virtually identical, as according to the format specifications the first 16 bytes for each format should be the same, and this would contain all the information that would need to be transferred into the internal format trace header, as is done for the present SEG-A demultiplex. All three formats are based on 30 channel recording blocks which should also simplify the conversion.
SEG-C would probably be the easier to produce a new program for, as the data is in 4 byte IBM floating point format. As one of the capabilities of the AP is to convert IBM floating point numbers into its own internal format "on the fly" through the interface, the data conversion would be straightforward once the demultiplex had been performed. Most of this logic would also be shared with the SEG-A program, as the two share the same 3 byte start of scan code.

SEG-B is a slightly more complicated format, but the samples are in the same representation of a 15 bit mantissa and a 4 bit gain code as SEG-A, and so the same data conversion routines could be utilised.

It should be realised that even if these two routines were produced, it may be necessary to produce slightly altered versions from time to time to suit the exact format of any data received, because most recording equipment, though remaining close to the standard, usually uses a variation on one of the three standards.

Data Exchange Format

Another drawback of the system is that, at the present, there is no capability for reading or writing data in SEG-Y, which is the accepted data exchange format. However one problem with SEG-Y is that it is a gapped format, having inter-record gaps between traces on tape. With the present hardware configuration it would be a quite longwinded process to read or write SEG-Y tapes. The method employed would basically have to be a two pass method,
involving storage of quite large quantities of data on disc between the passes.

It would be possible to produce a program to run on the pdp 8/e under OS/8, which would transfer data to and from the tape drives, from and to the pdp11 respectively, in a gapped format. A program of this type is already in existence to allow blocked transfers of ultrasonic tank data. However this program could not run at the same time as the gapless read/write program, used for the seismic systems internal format, because of memory limitations, as previously mentioned. Therefore two programs would be needed. In the case of producing SEG-Y tapes from internal format tapes, data would have to be transferred from tape into a program for converting to SEG-Y, and written to disc. On filling the disc, a gapped tape write program could be run on the pdp8 to allow the data to transferred to tape in the correct format. Obviously the procedure could be reversed for reading SEG-Y tapes. However it is clear that this slower procedure is a consequence of hardware limitations.

Land Data

Although the system was designed with marine processing as the primary target, a conscious effort was made not to exclude the possibility of processing land seismic data. It would be quite feasible to process land data on the system, especially if the one glaring omission from the normal suite of land processing techniques were added to the system. This is, of course, the capability to handle static corrections.
In processing marine data, the importance of static effects is negligible. However, in order to successfully process land data the ability to apply static corrections to the data is of paramount importance, if a section of interpretable quality is to be produced.

The writing of a static correction module should not prove too difficult if a need to process land data arises. The basic principle of static corrections is to apply a time shift to each trace, so that the revised start-time is that which would have been observed had source and receiver both been on a chosen datum, assuming that the seismic velocity in the region below the datum is the elevation velocity. The implementation of a static correction module would not be too different from the application of the NMO correction, except that the time shift is constant for the entire trace. As in the NMO correction, the data should be interpolated up to a higher sampling rate, using the same routine, and then once the time shift has been converted to a sample shift at this new rate, the correct samples can be extracted, to reduce the trace back to the original sample rate, with the static shift applied.

Also useful in the processing of land data would be a residual statics package. However, the problem of determining residual statics is probably a large enough problem to be dealt with as a full project in its own right.
Other Possible Software Additions

The improvements in the system proposed in the previous sections, especially the improved demultiplex and statics capability, are necessary to complete the all round capabilities of the system. However, as well as these it is possible to identify several techniques which could be added to the system in its present form, and so increase the range of possible techniques in the system available for processing data.

Vibroseis

Vibroseis is of increasing importance in the acquisition of land data. After the data has been demultiplexed the Vibroseis sweep has to be removed by performing a correlation between the recorded sweep and the data trace, which can be up to 30 seconds in length if a long source sweep has been used. At present a Vibroseis correlation capability is not available within the system. However research work into Vibroseis sweeps has been carried out within the department, and so it may become desirable to design a tape to tape process to perform Vibroseis correlation on the data after demultiplex.
Improved Filtering

The application and design of filters within the system is an area where greater flexibility could be provided, both in deconvolution and frequency filtering.

Due to the selective attenuation of high frequencies in the source wavelet on its passage through the earth, the frequency spectrum of the trace at longer travel times tends to have less high frequency components than the earlier arrivals, and because of this the source wavelet is usually a slightly different shape. This effect is clearly seen in land data where the change in frequency characteristics down the record can be quite marked.

As a result of this phenomenon, a band-pass filter designed for the trace as a whole tends not to remove enough of the high frequency noise at longer travel times, while leaving unwanted low frequency effects in the trace at early arrival times. One way to get round this problem would be to enable time variant band-pass filtering to be applied. This could be done, in the system, by allowing, say, 3 different bandpass gates to be specified which relate to 3 different areas down the trace. The actual application of the 3 different filters could be performed in either the frequency or time domain, but the method basically consists of applying the filters separately and merging the resultant, filtered traces with appropriate scale factors, to give the final resultant time variant filtered trace. (Fig 6.1)
Fig 6.1: Application of Time Variant Filters

Fig 6.2: Example of AGC for Trace scaling
A similar effect is evident in the effectiveness of deconvolution operators if designed on the trace as a whole, and a better result may be obtained if deconvolution operators are designed over different time gates, to allow for the differing characteristics of the source waveform down the trace. The resulting filters could then be applied in the same manner as described above for the time-variant bandpass filters.

The successful deconvolution of seismic data, to remove the effects of the source wavelet, is always a problem because the assumption of a minimum phase waveform for the source wavelet is often not valid in practice, especially in the case of reverberatory sources, such as maxipulse or airguns. This is the reason why attempts to remove the source wavelet due to airguns are often unsuccessful, and why so much effort is expended on the design of airguns and airgun arrays, in order to try to produce an impulsive minimum phase waveform.

Therefore, a particularly useful addition to the Durham system, as it is biased towards marine work with only small airgun arrays, would be a wavelet estimation package, to allow deterministic signature deconvolution. The ability to perform signature deconvolution would also be useful in experiments where the far field source signature was actually recorded.

One method of wavelet estimation, homomorphic deconvolution, has in fact been investigated by MSc research projects in the department and so could probably be implemented reasonably easily. However, reasonable success at wavelet estimation is possible by simple methods, such as stacking together time gates, identified
as containing the wavelet, such as the sea bed arrival on a marine seismic record (Stone, 1979). Once the wavelet is known, the wiener shaping filter can be designed to turn the source wavelet into a spike, as both the source autocorrelation function and source/desired output cross correlation function can be directly calculated. If the source wavelet is estimated for different time gates down the trace the method can be applied as for the time variant bandpass filter.

Amplitudes

Although it is desirable to display the seismic section with a minimum of amplitude manipulation, other than the application of a spherical divergence correction, so as to allow comparisons in the amplitude of various events down the trace, this can lead to small, low-amplitude events being missed. Therefore it is probably desirable to have the capability to apply some form of AGC (Automatic Gain Control) to the data before display, in order to produce a more even amplitude down the trace so that even small events can be easily detected.

Summary

The Software improvements described above fall into two categories, those which are necessary in order for the system to be viewed as complete, and those latter suggestions which, based on the experience gained in processing the test lines, it would have been desirable to add to the system in order to improve its
performance. It is felt that these are software improvements which could readily be included in the present system with the present hardware configuration.

**Hardware Evolution**

During the course of the project the possibility of future hardware upgrades was continually assessed. Three reasons for hardware changes were identified.

1) **Necessity** - Some hardware changes were viewed as necessary for the future development of the system to remain viable, in terms of the volume of data processed.

2) **Desirability** - Some hardware changes would allow algorithms already produced to run more efficiently, with restructuring where necessary. Other algorithms could then be performed on larger quantities of data, and some algorithms which are not realistic at the present time could become possible with future hardware upgrades.

3) **Long Term Evolution** - Developments in electronics are continually bringing more sophisticated pieces of equipment within the budget range even of bodies such as Universities, and at the same time older equipment becomes obsolete and difficult to maintain. Therefore a long term hardware evolution path has to be identified and updated in the light of new product announcements. At all times, however, hardware upgrading must be considered only in the light of software compatibility.
The hardware evolution of the system which was envisaged as being the best compromise between necessity, software compatibility and cost is shown in Fig 6.3. It can be seen from this diagram that the provision of a tape subsystem attached directly to the pdp11 is the most important hardware upgrade, and is probably the only one which could be described as being absolutely necessary.

The total reliance on the pdp8 for access to the tape drives makes the reliability of the system wholly dependent on the pdp8, which is the oldest and least reliable component in the system. Also, the passage of data to and from the two processors to the tape drives places two major constraints on the system. Firstly, as the transfer is performed under processor control, the tape read/write time is the limiting factor on how fast any processing module can execute, because computations cannot be overlapped with the data transfers. Secondly, the implementation, of the tape read/write program on the pdp8 dictates that the tape format is always gapless, which prevents SEG-Y being generated easily and prevents the data being input to general purpose computer systems, such as the NUMAC IBM370.

Ideally, a tape subsystem which allows gapless reads should be purchased to allow field tapes to be read, using these drives, so that the pdp8 would no longer need to be an integral part of the processing system. However, if this solution proved to be initially too expensive, the drives and formatter of a system, which could later have the gapless facility added, could be purchased. This would allow the drives on the pdp8 to be used solely for reading the field tapes, all subsequent tape
Present system

1. Add standard Tape Drives to PDP

2. Upgrade Tape Controller to read Field Tapes which ends reliance on PDP 8

3. Attach a 36 Inch Electrostatic Plotter

4. Upgrade AP120b to 32 Kwords of Data Memory

5. Attach a larger capacity Disc unit

6. VAX System

Fig 6.3: Envisaged System upgrade path
manipulation being performed on the drives interfaced to the pdp11.

The software changes needed to accommodate such a change would be small, assuming the device driver for the tapes was provided by the vendor. If a gapless read facility was provided, the transfers would still be performed much as they are at present, with the transfers being to and from disc and tape. Hence this tape routine would be modified to accept data from the tape interface and put it to disc, rather than from the pdp8 interface.

The main software difference would be that the internal format would be changed to be the same as the format on disc. That is, the contents of the format would remain the same, but the files written to tape would be in blocks of 512 bytes, just like the disc files. The tape handling routines would then be changed to perform the skipping and error checking functions for the new interface directly, while the read and write functions in the subroutines would be performed using the tape read/write functions in RT-11, allowing the data transfers and computations to be overlapped, as is done in accessing disc files, resulting in a massive reduction in processing time.

A secondary gain from this upgrade would be that a more flexible file transfer capability in RT-11 and other standard formats would be possible, and the tape backup/restore facility would be much simpler to use.
This improvement in the system is possibly more important than any other foreseeable update, and any resources available to the system should really be used to get the system to stage 2 in Fig 6.3 before further upgrades are considered.

Once the system is fully independent of the pdp8, with its own tape subsystem, the next most important upgrade would be to the plotting hardware. The final output of all the time and effort spent in a processing system is always a plotted section used for visual interpretation, and so it is only sensible to produce plotted output of as high a quality as possible. In the present system the plotter is only 11 inches wide, and so a stripping algorithm has to be used to display most sections at a reasonable scale. It is therefore proposed that a 36 inch, 200 dots/inch electrostatic plotter be added to the system, which would be used to produce final sections which would not have to be stuck together. The 11 inch printer/plotter would still be used as the line printer and for small plots, and the 36 inch plotter need not have a printer capability.

The only changes needed to the software would be to make the number of dots at which stripping is to occur an input parameter to the section plotting program, to allow plotting on both devices. As stripping involves the use of more than one output tape drive, this would also reduce the number of tape drives needed in plotting operations. This upgrade would produce a vast increase in the quality of final plots, and make the management of plot tapes much easier, with very little alteration to the software already present.
An upgrade which is desirable rather than necessary, and would not effect the system configuration, would be to add more Main Data Memory to the AP to bring it up to 32 Kwords, with a corresponding upgrade in the Table Memory to allow bigger FFT’s. This would involve no immediate software changes, but it would remove the 2048 sample data length restriction for single channel filtering operations. However, by increasing the amount of data which can be held in the AP at any one time, programs can be restructured so that the number of data transfers in programs such as demultiplex, velocity analysis and Finite Difference Migration could be drastically reduced.

The most important gain derived from this upgrade would be that the algorithms used by processes such as velocity analysis stack, and migration are based on the assumption that only 8 Kwords of memory are available. This makes the method used a little convoluted and long winded, with many data transfers to and from the AP. With a larger AP memory the algorithms could be rewritten to use the AP more efficiently, and would probably make it worthwhile for more algorithms to be microcoded to run almost entirely in the AP, which would result in a vast improvement in data throughput.

The final upgrade envisaged, of equal merit to the increase in the AP memory, is to attach a bigger disc system to the pdp11. The limiting factor on processes such as finite difference migration is the size of the largest disc file it can create. Therefore if a disc system with more overall storage, and more importantly a bigger maximum file size, could be added to the system, it would enable processes such as finite difference
migration to be applied to bigger working sets. Also it would enable the processing of larger pieces of data to be carried out from disc for filter tests, and perhaps allow small lines to be processed almost entirely from disc. The present disc drive would be retained for data file and program storage, and for such things as velocity analysis files. The software changes would only involve altering the disc driver and producing virtual memory read/write routines as was done for the present disc drive. The actual total size of this disc subsystem need not be enormous as long as the maximum file size is appreciably larger than the present system's disc, although a very large disc would obviously vastly increase the flexibility of the system.

Future Evolution path

The upgrades described above are about as far as it is reasonable to go while retaining the basic configuration of the system. Once the system reaches the stage of advancement described, it is no longer the peripherals which are the limiting factor but the controlling processor, the pdp11.

Fortunately, recent developments in computing hardware provide the logical upgrade from the pdp11 at a comparatively low cost, as shown in Fig 6.4. The obvious development is to replace the pdp11 with a VAX system. The VAX is manufactured by DEC and is fully compatible with the pdp11. In fact VAX-11 is an acronym for Virtual Address eXtension to the pdp11.
Fig 6.4: Suggested Future Computer System
The VAX is a 32-bit word computer and, depending on model, has up to 4 Megabytes of physical memory. However, the virtual address limit is that provided by the 32 bit word, which is about 4 Gigabytes, and is unlikely to be exceeded by present data processing requirements. One of the great advantages of the VAX is that it uses the same peripheral buses as the pdp11 family, and so the peripherals on the pdp11 could be put straight onto the VAX. Also pdp11 Fortran is compatible with VAX Fortran and even Macro-11 instructions can be executed in compatibility mode, although the native mode Macro-32 is similar enough for conversions to be trivial, with the VAX using the same conventions for its data types. Hence the processing software would require no conversion, other than to replace the RT-11 system calls with their VAX/VMS analogues, which should be reasonably easy. Device drivers for the peripherals to allow them to run on the VAX should be available from the original suppliers.

The Virtual memory system on the VAX means that there are no realistic limits to data length or number of channels per gather, or window width for migration, due to the main processor, although these things would still be regulated by the AP limitations. With the purchase of a machine as powerful as the VAX, it would be sensible to provide a reasonable amount of disc space to allow full use of its facilities to be made by processes such as migration.

As the VAX is a multi-user machine, several terminals could be attached to it to allow it to perform an educational function as well as seismic processing. So, although it may seem to be a rather extravagant upward step, a machine such as this could
easily provide a service for the whole department as well as performing seismic processing. Program development could also take place at the same time as processing in this sort of environment.

At present there are two machines in the VAX family. The VAX11/780, which is the most powerful and expensive, is probably out of reach economically and unnecessary from a power point of view. Therefore the VAX 11/750 would seem to be the one to choose. However, a smaller VAX 11/730 is about to be released which would have adequate performance for this application.

**Summary**

In summary, it is felt that this project has provided a working system which forms an easily useable tool for the processing of seismic reflection data. Also, with the simulator capabilities on the NUMAC IBM, along with the capability of AIMS in providing synthetic data, program development for the system should be reasonably straightforward.

An assessment of the system has shown those areas where future work could be usefully directed, and from the experience gained working on the project a critical assessment is given of the evolution of the system considered most apt for the future.

With more development the system should be able to provide an even better educational service, by producing demonstrations of data processing techniques in action, and form a starting point for future research projects. Hopefully, if the work begun in
this project is continued, the department can continue to be at the forefront of seismic reflection experience in Universities.
References

Barry K M, Cavers D A, and Kneale C W 1975
Recommended standards for digital tape formats
Geophysics 40 344-352

Bergland G D 1969
A guided tour of the Fast Fourier Transform
IEEE Spectrum 6 41-52

Berkhout A J 1977
Least squares inverse filtering and wavelet deconvolution
Geophysics 42 1369-1383

Berkhout A J 1979
Steep dip finite difference migration
Geophysical Prospecting 27 196-213

Berkhout A J and Wulfften Palthe D W van 1979
Migration in terms of spatial deconvolution
Geophysical Prospecting 27 261-291

Berryhill J R 1979
Wave equation datuming
Geophysics 44 1329-1344
Bolondi G, Rocca F and Savelli S 1978

A frequency domain approach to two-dimensional migration

Geophysical Prospecting 20 750-772

Claerbout J F 1970

Coarse grid calculations of waves in inhomogeneous media with application to delineation of complicated seismic structure

Geophysics 35 407-418

Claerbout J F 1971

Toward a unified theory of reflector mapping

Geophysics 36 467-481

Claerbout J F 1976

Fundamentals of geophysical data processing with applications to petroleum prospecting

New York McGraw-Hill

Claerbout J F and Doherty S M 1972

Downward continuation of moveout corrected seismograms

Geophysics 37 741-768

Claerbout J F and Johnson A G 1971

Extrapolation of time dependant waveforms along their path of propagation

Geophy J R astr Soc 26 285-293
Digital Equipment Corporation 1974

OS/8 Handbook
Maynard Mass

Digital Equipment Corporation 1977

PDP11 FORTRAN - Language reference manual
Maynard Mass

Digital Equipment Corporation 1978

PDP11/34 Processor Handbook
Maynard Mass

Digital Equipment Corporation 1978b

PDP11 Peripherals Handbook
Maynard Mass

Digital Equipment Corporation 1978c

RT-11 Advanced Programmers Manual
Maynard Mass

Digital Equipment Corporation 1978d

RT-11 System users guide
Maynard Mass

Digital Equipment Corporation 1978e

RT-11/RSTS/E FORTRAN IV Users guide
Maynard Mass
Dix C H 1955

Seismic velocities from surface measurements
Geophysics 20 68-86

Dobrin M B 1977

Geophysical Prospecting

Dunkin J W and Levin F K 1973

Effect of normal moveout on a seismic pulse
Geophysics 38 635-642

Embree P, Burg J P and Backus M M 1973

Wide band velocity filtering - The pie-slice process
Geophysics 28 948-974

Floating Point Systems Inc 1977

AP-120B Math Library Parts 1 and 2
Beaverton Oregon

Floating Point Systems Inc 1977b

AP-120B Processor Handbook
Beaverton Oregon

Floating Point Systems Inc 1977c

AP-120B Software Development Package
Beaverton Oregon
French W S 1975

Computer migration of oblique seismic reflection profiles

Geophysics 40 961-980

Gardner G H F, French W S and Matzuk T 1974

Elements of migration and velocity analysis

Geophysics 39 811-825

Gazdag J 1978

Wave equation migration with the phase shift method

Geophysics 43 1342-1351

Godbold C C 1980

Convolutional Migration of Seismic Reflection Data

MSc Thesis University of Durham

Hagedoorn J G 1954

A process of seismic reflection interpretation

Geophysical Prospecting 2 85-127

Hood P 1978

Finite difference and wave number migration

Geophysical Prospecting 26 773-789

Levin F K 1971

Apparent velocity from dipping interface relations

Geophysics 36 510-516
Loewenthal D, Lu L, Roberson R and Sherwood J 1976

The wave equation applied to migration
Geophysical Prospecting 24 380-399

Lu C H and Gupta S C 1978

A multirate digital filtering approach to interpolation-
Application to common depth point stacking
Geophysics 43 877-885

Mayne W H 1962

Common reflection point horizontal data stacking techniques
Geophysics 27 927-938

Mayne W H 1967

Practical considerations in the use of common reflection point
techniques
Geophysics 32 225-229

Meiners E P, Lenz L L, Dalby A E and Hornby J M 1972

Recommended standard for digital tape formats(Digital format C)
Geophysics 37 45-58

Meyerhoff H J 1966

Horizontal stacking and multichannel filtering
Geophysical Prospecting 14 441-454
Neidell N S and Taner M T 1971
Semblance and other coherency measures for multichannel data
Geophysics 36 482-497

Newman P 1973
Divergence effects in a layered Earth
Geophysics 38 481-488

Northwood E J, Weisinger R C and Bradley J J 1967
Recommended standards for digital tape formats
Geophysics 32 1073-1084

Nunns A G 1980
Marine Geophysical Investigations in the Norwegian-Greenland Sea between the latitudes of 62 N and 74 N
PhD Thesis University of Durham

O’Brien P N S 1977
New techniques in seismic exploration for oil
Sci Prog Oxf 64 487-519

Oppenheim A V and Schafer R W 1975
Digital Signal Processing
Engelwood Cliffs New Jersey, Prentice Hall

Otnes R K and Enochsen L 1972
Digital time series analysis
New York Wiley
Pann K, Shin Y and Eisner E 1979

A collocation formulation of wave equation migration

Geophysics 44 712-721

Peacock K L and Treitel S 1969

Predictive deconvolution - Theory and Practice

Geophysics 34 155-169

Ricker N 1953

Wavelet contraction, wavelet expansion and the control of seismic resolution

Geophysics 18 769-792

Robinson E A 1967

Statistical communication and detection with special reference to digital processing of Radar and Seismic signals

London Griffin

Robinson E A 1967b

Predictive decomposition of time series with application to seismic exploration

Geophysics 32 418-484

Robinson E A and Treitel S 1967

Principles of digital Wiener filtering

Geophysical Prospecting 15 310-333
Schneider W A 1978

Integral formulation for migration in two and three dimensions

Geophysics 43 49-76

Sheriff R E 1973

Encyclopaedic dictionary of exploration geophysics

Society of Exploration Geophysicists Tulsa Ok USA

Stone D G 1979

Pulse shaping methods

Developments in geophysical exploration methods

London Applied Science Publishers 239-270

Stolt R H 1978

Migration by fourier transform

Geophysics 43 23-48

Taner M T, Cook E E and Neidell N S 1970

Limitation of the reflection seismic method - lessons from computer simulations

Geophysics 35 551-573

Taner M T and Koehler F 1969

Velocity spectra - Digital computer derivation and applications of velocity functions

Geophysics 34 859-881
Telford N M, Geldart L B, Sheriff R E and Key S D A 1976

Applied Geophysics
Cambridge University Press

Treitel S and Robinson E A 1969

Optimum digital filters for signal to noise ratio enhancement
Geophysical Prospecting 17 248-288

Treitel S, Shanks J L and Frasier C W 1967

Some aspects of fan filtering
Geophysics 32 789-800

Tribolet J M 1979

Seismic applications of Homomorphic signal processing
Engelwood Cliffs New Jersey Prentice Hall

Versatec 1978

Versaplot graphics programming manual
Santa Clara California

Waters K H 1978

Reflection Seismology - A tool for energy resource exploration
New York Wiley


Residual statics analysis as a general linear inverse problem
Geophysics 41 922-938
Wood L C and Treitel S 1975

Seismic Signal Processing

Proc IEEE 63 649-661

Wood L C, Heiser R C, Treitel S and Riley P L 1978

The debubbling of marine source signatures

Geophysics 43 715-729
Appendix 1

This appendix contains the description of the input parameters, and the source listings for each of the main processing programs.
**Demultiplex:** MPDMXA

Input file ....... DK1:MPDMXD.SPF

Log file .......... DK1:MPDMXD.LOG

**Input Parameters**

READ(1,1001) NCHAN, NFILES, ITSIZ, IHSTRT, NROW, TPDRR, TPDRW, VELNUM

1001 FORMAT(12I5)

NCHAN .... Number of channels to demultiplex
NFILES .... Number of input files to demultiplex
ITSIZ .... Last half second to be demultiplexed
IHSTRT .... First half second to be demultiplexed
NROW .... Number of rows in sort matrix ... at least 1
TPDRR .... Input tape drive number
TPDRW .... Output tape drive number
VELNUM .... Number of files to save on disc

READ(1,1001) USEFLG, OUTFLG, INFLG, IERFLG, NRECOV, NUERR, NALOW

USEFLG .... New start/Restart flag
   0 - new job
   1 - restart of old job with old sort files

OUTFLG .... Output flag
   0 - output to tape
   1 - output to disc
INFLG....Input flag
  0 - input from tape
  1 - input from disc
IERFLG....Demultiplex mode switching flag
  0 - fast mode demultiplex
  1 - slow mode demultiplex
NUERR....Number of different logged demultiplex errors allowed
  before a file is declared dead
NALOW....Number of consecutive frames in error allowed before
  a file is declared dead

READ(1,1000) FNBUF

1000 FORMAT(3A4)

FNBUF....Input file name
  If INFLG=0.....Temporary file for tape read
  If INFLG=1.....NFILS input files to demultiplex

READ(1,1001)(INDEX(I),I=1,NROW)

INDEX....Sequence of sort buffer files
  If USEFLG=1....Input sequence from last line of
  previous log file
  If USEFLG=0....Input sequence, 1....NROW

READ(1,1001)(ICHAN(I),I=1,NCHAN)

ICHAN....Position of output channels in order of increasing
offset, on input to demultiplex

\texttt{READ(1,1001)(FPOS(I),I=1,NCHAN)}

\texttt{FPOS....Output sort position for each input channel}

\texttt{READ(1,1001)(VELAN(I),I=1,VELNUM)}

\texttt{VELAN....File numbers to be saved on disc}

\texttt{READ(1,1000)(FNBUF(I),I=1,NROW)}

\texttt{FNBUF....File names for sort file buffers, always at least 1}

\texttt{READ(1,1000)(VELNAM(I),I=1,VELNUM)}

\texttt{VELNAM...File names for files to be saved on disc}

\texttt{READ(1,1001)NOCHAN,IGCODE,IUNITS,ISCODE}

\texttt{NOCHAN...Number of active channels recorded in field data}

\texttt{IGCODE...Output gather code}

\begin{itemize}
  \item 0 - Common shot gather
  \item 1 - CMP gather
  \item 2 - Single channel stacked
  \item 3 - Single channel unstacked
\end{itemize}

\texttt{IUNITS...Units of measurement}

\begin{itemize}
  \item 1 - Metres
  \item 2 - Feet
\end{itemize}
ISCODExAcquisition source code

0 - Airgun
1 - Explosives
2 - Vibroseis
3 - Weight drop/ Hammer

READ(1,1003)SROFF,RSPAC,SLSPAC,STSPAC,SDEPT,RDEPT

1003 FORMAT(6F10.0)

SROFF....Shot to channel 1 offset
RSPAC....Receiver spacing
SLSPAC...Shot spacing(distance)
STSPAC...Shot spacing(Time at sea)
SDEPT....Shot depth
RDEPT....Receiver depth

READ(1,1000)(HSBLK(I),I=51,254)

HSBLK(51)....HSBLK(254)....Free area of header block, used for user comments
THIS PROGRAM TAKES DATA OFF TAPE (VIA PDP-8) OR DISC IN THE MULTIPLEXED FORMAT USED AT DURAM UNIV AND MULTIPLIXES THE CHANNELS AND REFORMATS THE DATA VALUES. THESE ARE THEN WRITTEN TO DISC FOR STORAGE BEFORE BEING ASSEMBLED INTO A STACK POINT GATHER. THEY ARE THEN EITHER LEFT ON DISC FOR A VELOCITY ANALYSIS OR WRITTEN (VIA THE PDP-8) TO TAPE FOR STORAGE.

DATA STORAGE DECLARATIONS.

VIRTUAL BUFF(8448), FNAMES(30), VELNAM(60), RTNAM(60)
REAL*8 FNAMR, FNAMES, VELNAM, DBL(2), FMBUF, RTNAM
REAL*4 DEVNAM, BUFOUT(256), FMBUF(3)
INTEGER*2 FNUM, CHOFF(30), FPPOS(30), NBLKOF(30), IBLKOF(30), INDEX(31), %USEFLG, QNUM, VNUM, BLK, BSST, GAINS(30), GCNT, %SPOS, FST, BSST1, BLST, OLAP, RST, ICHAN(30), %SAVE(30), EOFFLG, MASK, SYNC, BFF, HSBLK(W), VELAN(60), GAINS(3840), %OUTFLG, INFILG, FLEN, VELNUM
LOGICAL*1 STATUS, RC, ITLEN, IGCODE, IUNIT, NOCHAN, %LEN, TPDRR, TPDRW, HSBLK(256)
COMMON /SUBS/GAINS, GSAVE, NSMPIN, EOFFLG, IFDIR, IERR, %IERFLG, SPOS, RC
COMMON /DECOM/BUFOUT, CHOFF, ICHAN, FPOS, NCHAN, NBLKOF, INDEX, GCNT, #HSTR, INSEC
COMMON /BFSCOM/FBSST1, FBLST, F256, F256D, FBSST, F1, F4097, %BSST1, BLST, OLAP, RST, BLK
COMMON /BUFFS/NUERR, NALOW, ITIC
EQUIVALENCE(HSBKLK(1), HSBKLK(11), (SRoff, HSBKLK(21)), %ISLSPAC, HSBKLK(29)), (STSPAC, HSBKLK(33)), (SDEPT, HSBKLK(37)), %INDEPT, HSBKLK(41)), (NOCHAN, HSBKLK(12)), (IGCODE, HSBKLK(45)), %IUNIT, HSBKLK(20)), (ISVT, HSBKLK(45)), (NBFREE, HSBKLK(47)), %IFSPAC, HSBKLK(25))
DATA DEVNAM/3RDK /
DATA CHOFF/4352, 4460, 4660, 4736, 4884, 4992, %5120, 5248, 5576, 5504, 5632, 5760, 5888, 6016, 6144, 6272, %5400, 5628, 6056, 6784, 6912, 7040, 7168, 7296, 7424, 7552, %7680, 7808, 7936, 8064/ DATA MASK, SYNC/"17,"177777/ DATA HSBLK(49)/"377/, HSBLK(50)/"377/, HSBLK(255)/"377/, #HSBLK(256)/"377/

IPADNO=0 IOTR=0 EOTR=0 BSST=496+256
IBIAS=15 FNUM=1 IAD=BSST+1
C SET UP THE RT-11 INPUT-OUTPUT PROCEDURES
C
C CALL APINIT
C READ IN THE NECESSARY INPUT DATA
C
C CALL ASSIGN(1,'DK1:MPDMXD.SPF',14)
C CALL ASSIGN(2,'DK1:MPDMXD.LOG',14)
C
READ(1,1001)NCHAN,NFILES,ITSIZ,IHSTRT,NROW,TPDRR,TPDRW,VELNUM
1001 FORMAT(12I5)
C READ(1,1001)USEFLG,OUTFLG,INFLG,IERFLG,NRECOV,NUERR,NALOW
C
IF(INFLG.NE.0)GOTO 1
CALL READ(1,1002)FNUF
1002 FORMAT(3A4)
C CALL IRADS0(12,FNUF,FNAMR)
GOTO 2
2 CONTINUE
DO 3 IRD=1,NFILES
READ(1,1002)FNUF
CALL IRADS0(12,FNUF,FMBUF)
RTNAM(IRD)=FMBUF
3 CONTINUE
READ(1,1201)INDEX(I),I=1,NCHANDO 5 IPL=1,NROW
READ(1,1002)FNUF
CALL IRADS(12,FNUF,FMBUF)
5 FNAMES(IPL)=FMBUF
DO 6 IPL=1,NROW
READ(1,1002)FNUF
CALL IRADS(12,FNUF,FMBUF)
6 CONTINUE
READ IN HEADER INFO
C
READ(1,1001)NOCHAN,IGCODE,UNITS,ISCODE
1001 FORMAT(6F15.0)
READ(1,1003)SROFF,RSPAC,SLSPAC,STSPAC,SDEPT,RDEPT
1003 FORMAT(8GA1)
SBLK(I)=NOCHAN
SBLK(I)=ISTRT*128-128
SBLK(I)=ITSIZ*128
SET UP CONSTANTS AND REST OF RT-11

INPUT OUTPUT ROUTINES.

C
ITSIZE=ITSIZE-IHSTRT+1
IBLKOF(1)=1
DO 10 J=2,NCHAN
10 IBLKOF(J)=IBLKOF(J-1)+ITSIZE
LSTBLK=(ITSIZE*NCHAN)
IFSZ=LSTBLK+1
ITLEN=ITSIZE/2
QNUM=NCHAN+2
IFIQSET(QNUM).NE.0STOP 'QSET ERROR'
IFET=IFETCH(DEVNAM)
IF(IFET.NE.0'TYPE 1009,IFET
1009 FORMAT(' FETCH RETURN=',I2)
IFET=IFETCH(DEVNAM)
STOP 'BAD HANDLER FETC'
C

STACK FILE ORGANISATION

C
IF(USEFLG.NE.0GOTO 20
DO 15 JJ=1,NROW
FMBUF=FILENAME(JJ)
15 CONTINUE
C
IF(INFLG.NE.0GOTO 45
CALL CLOSEC(22+JJ)
C
START OF THE MAIN DEMULTIPLEX LOOP

C
DO 999 I=1,NFILES
999 CONTINUE
C
FILE ORGANISATION ON A NORMAL RUN

C
DO 35 M=1,NCHAN
35 IBLKOF(M)=IBLKOF(M)
C
INDEX(NROW+1)=INDEX(1)
DO 40 MM=1,NROW
40 INDEX(MM)=INDEX(MM+1)
C
SEE IF FILES TO BE ZEREOED

C
IF(IPADNO.GT.0GOTO 100
IF(INFLG.NE.0'GOTO 45
C
IF THE DATA IS TO BE READ FROM TAPE THE ROUTINE TAPRED
IS USED IN ORDER TO COMMUNICATE WITH THE PDP-8 AND ALSO
TO DO A FAST FILE TRANSFER BOTH 8->11 AND 11->8.
THE ROUTINE ALSO RETURNS THE STATUS BYTE FOR ERROR
ANALYSIS BY THE ROUTINE TAPRED.

OPEN FILE ON CH20 FOR SDS10

IF(IN.LT.0) WRITE(7,'(A)')
IF(IN.LT.0) STOP 'FNAMR ENTER ERROR'

CHECK ARNT AT EOT

IF (I.EOTR.GE.0) GOTO 43
WRITE(7,1062) TPDRR, IFNUM
1060 FORMAT (' EOT ENCOUNTERED ON DRIVE:', I2, ' FILE NO:', I4)
WRITE(7,1061)
1061 FORMAT (' ENTER NEW READ DRIVE NO:', S)
READ(5,1062) TPDRR
1062 FORMAT(I1)
I.EOTR=0
IF(TPDRR.GT.2) GOTO 265

CHECK IF ZERO FILES TO BE ADDED AT END OF TAPE

WRITE(7,1063)
1053 FORMAT (' ENTER NO OF ZERO FILES TO BE ADDED(I2):', S)
READ(5,1064) IPADNO
1064 FORMAT(I2)
IF(IPADNO.GT.Z) GOTO 1052

DO A READ

CALL TAPRED(-1, TPDRR, STATUS, TLEN, FLEN, IFNUM, I.EOTR)
IF (STATUS.LT.0) WRITE(2, 1050) IFNUM
1050 FORMAT(' WARNING FILE NO ', I4, ' RETRIES FAILED')

DO A WRITE

IF(I.EOTR.LT.0) GOTO 46
CALL TAPRED(D, TPDRR, STATUS, , IFNUM, I.EOTR)

START OF MAIN BUSINESS

46 CALL IWAIT(20)
45 CALL CLOSEC(20)
BLK=0
GCHT=0
IHSEC=0
EOFFLG=0
RC=.FALSE.
IF(INFLG.EQ.0) IOPEN=LOOKUP(29, FNAMR)
IF(INFLG.NE.0) FMBUF=RTNHAM(IFNUM)
IF(INFLG.NE.0) IOPEN=LOOKUP(29, FMBUF)
IF (IOPEN.LT.0) WRITE(7,'(A)')
IF (IOPEN.LT.0) STOP 'FNAMR LOOKUP ERROR'
READ IN FIRST DATA BLOCK FROM DISC AND EXTRACT THE
HEADER INFORMATION

IF(IREADA(20,BSST,BLK,F1),LT.0)STOP 'READ ERROR'
IF(IWAIT(20),LT.0)STOP 'WAIT ERROR'
BLK=BLK+17
IF(IREADA(20,4096,BLK,FBSST).LT.0)STOP 'READX ERROR'
BLK=BLK+16

END OF INITIAL READS
BEGINNING OF HEADER BLOCK SCANS

DO 50 LL=1,4
50 HSBLK(LL)=ISWAP(BUFF(LL)).OR."30060"
HSBLK(5)=ISWAP(BUFF(5))
HTIC=2*HSBLK(9)
IGJL=1
DO 70 JL=1,30
EGAINS(JL)=BUFF(5+JL).AND.MASK
GSAVE(JL)=EGAINS(JL)+IBIAS
GAINS(IGJL)=GSAVE(JL)
IGJL=IGJL+128
CONTINUE
SPOS=35
70 SPOS=SPOS+1
IF(BUFF(SPOS).EQ.0)GO TO 70

BUFFER CONSTANTS SET UP AFTER END OF HEADER BLOCK IS LOCATED

FST=SPOS
BSST1=4096+FST
BLST=8192+FST
OLAP=257-FST
RST=4096-OLAP

CHECK SYNC WORDS FOR ERRORS AND EXTRACT THE GAINS READY
FOR USE AS INTEGERS.

IF(INDIR=1DIRG(BUFF(SPOS+1))
IF(IERRFLG.NE.0.OR.IERR.NE.0)CALL BUFSCN(BUFF,0,IFNUM)
IF(IERR.LT.0)GOTO 100

PUT FIRST BUFFER INTO AP

F=APGAD(BUFF(FST))
CALL APPUT(0,4096,1)

FIND ADDRESS PAIRS FOR XM OPERATIONS

FBST1=APGAD(BUFF(0))
FDLST=APGAD(BUFF(0))
IF(IERRFLG.NE.0.OR.IERR.NE.0)GOTO 20
C MAIN DEMUX LOOP FOR A SINGLE FILE
C
C COME HERE IF WANT TO DEMUX ALL THE FILE
C WITHOUT FULL ERROR CHECKING SWITCHED ON?
C
0196 210 CONTINUE
0197 INSEC=INSEC+1
0198 CALL DMX
0199 IF(IERR.NE.0)GOTO 215
0201 IF(INSEC.GE.ITSIZ)GOTO 230
0203 IF(EOFILT.EQ.2)GO TO 230
0205 CALL DBLUF(BUFF)
0206 GO TO 210
0207 215 CALL IWAT(20)
0208 WRITE(2,103B)IFNUM
0209 103 FORMAT(' ERROR ON FILE: ',I4, ' FOUND GOING INTO RECOVERY MODE')
0210 IF(NRECQV.EQ.0)GOTO 100
0212 DO 216 JJ=1,NCHAN
0213 216 NBLKOFCJZ)=IBLKOF(CJZ)
0214 GOTO 46
C
C COME HERE IF WANT TO DEMUX ALL THE FILE
C WITH FULL ERROR CHECKING SWITCHED ON
C
0215 220 CONTINUE
0216 INSEC=INSEC+1
0217 IF(INSEC.GE.IHSTRT)CALL DEMUX
0219 IF(INSEC.GE.ITSIZ)GOTO 230
0221 IF(EOFILT.EQ.2)GOTO 230
0223 CALL BUFSCN(BUFF,1,IFNUM)
0224 IF(IERR.GE.0)GOTO 220
C END OF MAIN DEMUX LOOP
C
C BLANK PARTS OF FILES WITH FATAL ERRORS
C
0225 100 CALL VCLR(0,1,128)
0227 CALL APWR
0228 CALL APGET(BUFOUT,8,128,2)
0229 C WRITE(2,102B)IFNUM
0230 102 FORMAT(' FILE NUMBER ',I4,' DELETED')
0231 CALL APWD
0232 DO 235 ZZ=1,NCHAN
0233 NCH=INDEX(PPSS(JZ))
0234 IBLK=IBLKOF(JZ)
0236 DO 238 LZ=1,ITSIZ
0237 IF(IWRITB(256,BUFOUT,IBLK,22+NCH).LT.0)STOP 'CLEAR ERR' 
0238 IBLK=IBLK+1
0239 235 CONTINUE
0240 IPADNO=IPADNO-1
C
C WRITE OUT HEADER BLOCK AND CLOSE DOWN COMPLTED GATHER FILE
FORTRAN IV

0241 FMBUF=NAME$INDEX(I))
0242 IF(IWRITE(128,H$BLK,O,22+INDEX(NROW)).LT.O)STOP 'HSBLK ERROR'
0244 IF(USEFLG.EQ.O.AND.FNUM.LT.NROW)GO TO 998
0246 CALL CLOSEC(22+INDEX(I))
0247 IF(OUTFLG.NE.O)GOTO 240

C

WRITE OUT GATHER FILE TO TAPE

0249 FLEN=LOOKUP(21,FMBUF)
0250 IF(FLEN.LT.O)STOP' FMBUF LOOKUP ERR'
0252 CALL TAPRED(1,TPDRW,STATUS,FLEN,FNUM,IEOTW)
0253 CALL CLOSEC(21)
0254 IF(IEOTW.GE.O)GOTO 235
0256 WRITE(7,1080)TPDRW,IFNUM
0257 1080 FORMAT( 'EOT ON DRIVE: ',12,' FILE NO: ',I4)
0258 WRITE(7,1081)
0259 1081 FORMAT(' ENTER NO OF NEW WRITE DRIVE: ',$)
0260 READ(5,1082)TPDRW
0261 IEOTW=$
0262 IF(TPDRW.GT.2)GOTO 265
0264 235 IF(STATUS.GE.O)GOTO 240
0265 WRITE(2,1070)IFNUM
0267 1070 FORMAT(' WRITE ON FILE ',I4,' FATAL ERROR')
0269 GOTO 265
0269 240 IF(FNUM.NE.VELANCVNUM)GOTO 250

C

SET UP A VELOCITY ANALYSIS FILE

0271 DBLK(1)=FMBUF
0272 DBLK(2)=VELNAME(VNUM)
0273 IF(RENAME(21,21,21).GT.O)STOP 'RENAME ERROR'
0275 VNUM=VNUM+1
0276 IF(ENTER(22+INDEX(I),FMBUF,IFSIZ).LT.O)STOP'ENTER ERROR'
0278 IF(IWRITE(256,BUFOUT,LSTBLK,22+INDEX(I)).LT.O)STOP'WRITE ERROR'
0279 CALL CLOSEC(22+INDEX(I))
0281 250 CONTINUE

C

CLOSE DOWN FILE 20 AND GO TO NEXT INPUT FILE IF REQU'D

0282 IF(LOOKUP(22+INDEX(I),FMBUF).LT.O)STOP'LOOKUP IND ERROR'

C

DO 260 LL=1,NROW

C

STOP 'NORMAL TERMINATION'

END
SUBROUTINE DEMUX

INTEGER*2 CHOFF(32), GANADD, FPOS(32), NBLKOF(32),
%EOFFLG, GAINS(384), ICHAN(32), GSVE(32), INDEX(31), GCNT

REAL*4 BUFOUT(256)
LOGICAL*1 RC

COMMON /SUBS/GAINS, GSAVE, NSMPIN, EOFFLG, IFDIR, IERR,
%IERFLG, ISPOS, RC

COMMON /DECOM/ BUFOUT, CHOFF, ICHAN, FPOS, NCHAN, NBLKOF, INDEX, GCNT,
%IHSTRT, IHSEC

DO DEMUX AND BRING IN THE SAVED GAINS

NCH=0
NIN=128
IF (EOFFLG.EQ.2) NIN=NSMPIN
NOUT=2*NIN
IF (NOUT.EQ.0) RETURN
CALL APWD
CALL VFLT(4352,1,4352,1,384)
CALL APWR
CALL APPUT(GAINS,0,384,1)

FORM THE DEMUXED NOS INTO R*4 REREPRESENTATION
FOR EACH CHANNEL IN TURN AND THEN WRITE THEM
OUT TO DISC. THIS IS DONE FOR 128 SAMPLES OF
EACH CHANNEL WHICH ARE EXPECTED TO BE IN THE A.P.

CALL APWD
DO 19 NJ=1, NCHAN
CHADD=CHOFF(ICHAN(NJ))
GANADD=NCHADD-4352
CALL VBINSCCNCHADD,1,NCHAN,1,GANADD,1,NIN
CALL APWR

CALL IWAIT(22+NCH)
NCH=INDEX(FPOS(NJ))
CALL APGET(BUFOUT, NCHADD, NIN, 2)
CALL APWD
IF (WRITE(NOUT, BUFOUT, NBLKOF(NJ), 22+NCH).LT.0) STOP 'DEMUX ERROR'
NBLKOF(NJ)=NBLKOF(NJ)+1
CONTINUE

CALL IWAIT(22+NCH)
RETURN
END
SUBROUTINE DMX
#892 INTEGER*2 CHOFF, FPOS, NBLKOF, GCNT, EOFFLAG,
#893 GAINS, ICHAN, GSAVE, INDEX
#894 REAL*4 BUFOUT
#895 LOGICAL*1 RC

COMMON /SUBS/ GAINS, GSAVE, NSMPIN, EOFFLAG, IFDIR, IERR,
#896 XERFLAG, ISPOS, RC
COMMON /DECOM/ BUFOUT, CHOFF, ICHAN, FPOS, NCHAN, NBLKOF, INDEX, GCNT,
#897 !IHOST, IHSEC

DO DEMUX AND BINARY SCALING

NCHAN=0
NIN=128
IF (EOFFLAG.EQ.2) NIN=NSMPIN
NOUT=2*NIN
IF (NOUT.EQ.0) RETURN
CALL APVD
CALL APPUT(GSAVE, 4996, 0, 1)
CALL APWD
CALL DMX(G996, 4352, 128, GCNT, IFDIR, NIN)
CALL APW
CALL APSP(IERR, 15)
IF (IERR.NE.0) IERR=1
IF (IERR.NE.0) RETURN
CALL APSP(GCNT, 3)
CALL APSP(IFDIR, 4)
CALL APGET(GSAVE, 4996, 0, 1)
CALL IWAIT(20)
IF (IHSEC.LT.IHOST) GOTO 30
NCHADD=CHOFF(ICHAN(1))
CALL APGET(BUFOUT(1), NCHADD, NIN, 2)
IIN=129
IOUT=1
IF (NCHAN.EQ.1) GOTO 20

EXTRACT EACH WANTED CHANNEL AND PUT ON DISC

DO 10 NJ=2, NCHAN
NJ1=NJ-1
NCHAN=CHOFF(ICHAN(NJ))
CALL IWAIT(22+NCHAN)
CALL APGET(BUFOUT(IIN), NCHAN, NIN, 2)
NCHAN=INDEX(FPOS(NJ1))
IF (IWRITE(NOUT, BUFOUT(IOUT), NBLKOF(NJ1), 22+NCHAN).LT.0) STOP 'DMX'
NBLKOF(NJ1)=NBLKOF(NJ1)+1
IT=IIN
IIN=IOUT
IOUT=IT
10 CONTINUE
20 NCHAN=INDEX(FPOS(NCHAN))
CALL APVD
IF (IWRITE(NOUT, BUFOUT(IOUT), NBLKOF(NCHAN), 22+NCHAN).LT.0) STOP 'DMX'
NBLKOF(NCHAN)=NBLKOF(NCHAN)+1
SUBROUTINE DBLBUF(BUFF)

VIRTUAL BUFF(8448)

INTEGER*2 BUFF,EOFFLG,BST,BLS,T,SPOS,BST1,
%RST,SYNC,OLAP,BLK,GSAVE(384),GAINS(384)

LOGICAL*1 RC

COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR.IERR,
%MVF,GVAR,RC

COMMON /BUFFCOM/BST1,BLS,OLAP,RST,BLK

DATA SYNC/"177777/

C THIS ROUTINE CONTROLS THE DOUBLE BUFFERING SCHEME USED
C TO TAKE DATA FROM DISC AND PUT IT IN TO THE AP
C WHEN IT IS NEEDED. IT IS ALSO RESPONSIBLE FOR CHECKING
C THE SYNC WORDS AND EXTRACTING THE GAINS AS INTEGERS FOR USE

C SET UP THE START OF THE BUFFER

SPOS=BST1

IF(.NOT.RC)SPOS=BLST

C DO A BUFFER SCAN WHEN THE INPUT PROCEDURE HAS NOTIFIED EOF

IF(EOFFLG.LE.0)GOTO 50

ISMPIN=0

IPOS=SPOS

DO 40 L=1,NSMPIN

IF(ISUFF< IPOS).NE.SYNCGO TO 60

IPOS=IPOS+32

IF(IPOS.GT.8448)IPOS=IPOS-8192

ISMPIN=ISMPIN+1

40 CONTINUE

GOTO 50

60 NSMPIN=ISMPIN

50 CONTINUE

C PUT A BUFFER INTO THE A.P. AND START THE READ TO FILL
C THE SECOND BUFFER FOR USE NEXT TIME

C

IF(RC)CALL APPUTA((4996,1,FBSST1)

IF(.NOT.RC)CALL APPUTA((3,OLAP,1,FBLST)

IF(.NOT.RC)CALL APPUTA((OLAP,RST,1,F256)

IF(EOFFLG.GT.0)EOFFLG=2

IF(EOFFLG.GT.0)RETURN

IF(EOFFLG.LT.0)RETURN

FINP=F256D

IF(.NOT.RC)FINP=BSSST

IIN=IREADA((4996,BLK,FINP)

BLK=BLK+16

C CHECK INFO RETURNED FROM INPUT ROUTINE
C FOR ERRORS AND AN EOF SITUATION

C

IF((IN.EQ.4996)RETURN

IF((IN.GT.RST)GOTO 140

IF((IN.EQ.-1))IN=0

IF((IN.LT.0)WRITE(7,"*"))IIN,BLK

IF((IN.LT.0)STOP 'READ ERROR'

EOFFLG=1

NSMPIN=OLAP+1IN)/32

RETURN

EOFFLG=0

IF((IN.RST)

NSMPIN=IN/32

END
SUBROUTINE BUFSCN(BUFF,ICODE,FDONE)
VIRTUAL BUFF,BUFF(848)
INTEGER*2 BUFF,GAIN,GSAVE(3840),SPOS,EOFFLG,DMXBUF(3840),
%FRAME(33),FDONE
LOGICAL*1 RC,IF(2),IBYT(2),IBYTE(2),FRAME(66),IBSYNC
EQUIVALENCE (IVORD,IF(1)),(IWORD,IBYT(1)),(IWORD,IBYTE(1)),
%FRAME(1),FRAME(1))
COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
%IERFLG,SPOS,RC
COMMON/BUFFS/NERR,NALOW,ITIC
DATA ISYNC/"177777/,IBIAS/15/
DATA IBSYNC/"377/
IF(ICODE.GT.10)GOTO 1
ICHCK=1
ICD=0
LPINT=0
FRAME(33)=BUFF(SPOS+1)
SPOS=SPOS+2
ITBIAS=BUFF(SPOS+24)/2*2
ITCONT=0
NERR=0
GSAVE(30)=30
CALL FRAMFL(BUFF,FRAME,1,33,-1)
DATA CHECK AND GAIN PREPARATION SUBROUTINE
IGSCHK=GSAVE(ICHCK)
LDONE=0
LDONE=LDONE+1
IF(LPINT.GT.0)GOTO 48
FRAME(1)=FRAME(33)
CALL FRAMFL(BUFF,FRAME,2,33,ICD)
SYNC TEST
IF(EOFFLG.EQ.2)GOTO 90
IF(FRAM(32).NE.ISYNC)GOTO 20
IF(FRAM(66).EQ.ISYNC)GOTO 30
WRITE(2,10)FDONE
10 FORMAT(' FILE NO.','I4,' ERROR DETECTED')
TEST FOR TYPE OF DATA CORRUPTION
IF(LDONE.EQ.128)GOTO 90
IF(NERR.GT.NERR)GOTO 160
NERR=NERR+1
NERR=0
DO 40 I=1,32
40 IF(IWORD.EQ.FRAME(1))
45 IF(IWORD.NE.ISYNC)GOTO 50
46 IF(IWORD.EQ.FRAME(1))
47 IF(IBYTE(2).NE.ISYNC)GOTO 50
C DETECT PATTERN OF BYTES LOST
C
IND=1
DO 60 L=1+1,33
FRAME(IND)=FRAME(L)
IND=IND+1
60 CONTINUE
IF(ICD.EQ.3)GOTO 86
CALL FRAMFL(BUFF,FRAME,IND,33,ICD)
ICD=0
IF(EOFFLG.EQ.2)GOTO 90
GOTO 6
5 IF(IBYTE(1).NE.1BSYNC)GOTO 70
IWORDF=FRAME(1+1)
IF(IWORDF.NE.ISYNC)GOTO 70
C ODD BYTE LOSS OR GAIN DETECTED
IND=1
DO 60 L=1+2,33
IWORDB=FRAME(L)
IF(2).NE.IBYTE(1)
IF(1).NE.IBYTE(2)
FRAME(IND)=IWORD
IWORDF=IWORDB
IND=IND+1
60 CONTINUE
IF(32)GOTO 85
DO 90 L=1+2,33
IF(EOFFLG.EQ.2)GOTO 90
GOTO 6
7 IF(IBYTE(2).NE.1BSYNC)GOTO 40
IWORDF=FRAME(1+1)
IF(IBYTE(1).NE.1BSYNC)GOTO 40
COMMUNICATION ERR POSSIBLE
WRITE(2,1229)FDONE
1229 FORMAT(' POSSIBLE COMMUNICATION LOSS FILE NO:',I4)
C CHECK TO SEE IF OK TO SEARCH FURTHER
C AHEAD IN ATTEMPTING TO REESTABLISH CONTACT
C
I NFER+1
IF(NFER.GT.NALOW)GOTO 160
IF(IFER.EQ.33)GOTO 160
CALL FRAMFL(BUFF,FRAME,2,33,0)
IF(EOFFLG.EQ.2)GOTO 150
GOTO 25
C TIME CHECK SECTION

C

39 ITN=FRAME(26)/2*2
314 ITP=ITCONT.OR.ITBIAS
315 IF(ITN.IEQ.ITP)GOTO 90
317 NERR=NUERR+1
318 INTVAL=(ITN-ITP)/ITIC
319 IF(INTVAL.GT.0.0)GOTO 105
320 WRITE(2,1(33))FDONE
3110 FORMAT(133) TIME CHECK ERROR ON FILE NO: ',I4)
3111 IF(NERR.NE.NUERR)GOTO 160
3112 INTVAL=(ITN-ITP)/ITIC
3113 IF(INTVAL.NE.0.0)GOTO 105
3115 LPINT=INTVAL
316 IGPOS=LDONE
317 DO 110 I=1,29
318 LDONE=LDONE+1
319 LPINT=LPINT-1
320 GAINS(IGPOS)=GSAVE(I)
321 CONTINUE
322 GAINS(IGPOS)=30
323 LDONE=LDONE+1
324 LPINT=LPINT-1
325 GAIN=ITCONT+ITIC
326 GICH=ICHCK+1
327 IF(IICHCK.GT.0.0)ICHCK=1
328 IF(LDONE.GE.120)GOTO 150
329 IF(LPINT.GT.0.0)GOTO 150
330 CONTINUE
331 IGN=ITCONT+ITIC
332 GAINS CORRECTION
333 ISAVE=ICHECK(FRAME(1),IDIR)+ITBIA
334 IF(ISAVE.EQ.ISAVE)GOTO 120
335 WRITE(2,1(34))FDONE
336 FORMAT(134) GAIN CHECK ERROR FILE NO: ',I4)
337 IF(NERR.GT.NUERR)GOTO 160
338 NERR=NERR+1
339 GSAVE(ICHCK)=ISAVE
340 ICHCK=ICHCK+1
341 IF(IIFDIR.EQ.IDIR)GOTO 130
342 WRITE(2,1(35))FDONE
343 FORMAT(135) GAIN DIRECTION ERROR FILE NO: ',I4)
344 IF(NERR.GT.NUERR)GOTO 160
345 NERR=NERR+1
346 IFDIR=IDIR
347 CONTINUE
348 GAINS(IGPOS)=GSAVE(L)
349 GAINS(IGPOS)=129
340 GAINS(IGPOS)=GSAVE(L)
341 ICHCK=ICHCK+1
342 IF(IICHCK.GT.0.0)ICHCK=1
343 IGPOS=LDONE
344 DO 140 L=1,29
345 LDONE=LDONE+1
346 DO 145 L=1,29
347 DMBUF(IGPOS)=GAIN(FRAME(L+1),GSAVE(L),IFDIR)
348 C
349 GAINS(IGPOS)=GSAVE(L)
350 IGPOS=IGPOS+128

C NORMAL WORK

352 ICHCK=ICHCK+1
353 IF(IICHCK.GT.0.0)ICHCK=1
354 IGPOS=LDONE
355 DO 140 L=1,29
356 LDONE=LDONE+1
357 DMBUF(IGPOS)=GAIN(FRAME(L+1),GSAVE(L),IFDIR)
358 GAINS(IGPOS)=GSAVE(L)
359 IGPOS=IGPOS+128

C 140 CONTINUE

C

GAINS(IGPOS)=30
360 IGCHK=GSAVE(ICHCK)
361 IFDIR=IFDIR
362 IF(LDONE.LT.120.AND.EOFFLG.NE.2)GOTO 5
363 NSMIN=LDONE
364 CALL APPUT(45,1482,3840,1)
365 RETURN
366 WRITE(2,1(36))FDONE
367 FORMAT(136) FILE NO ',I4,' DECLARED DEAD')
368 IF=1
369 RETURN
370 END
SUBROUTINE FRAMFU(BUFF,FRAME,IST,IFIN,ICODE)

VIRTUAL BUFF(8448)

INTEGER*2 FRAME(33),SPOS,BUFEND,EOF,BUFF,GAINS(3840),GSAVE(30),
MEOFFLG,BBST1,BLST,OLAP,RST

LOGICAL*1 RC,IF(2),IBYT(2),BYTE(2)

EQUIVALENCE (IWORD,IF(1)),(IWORD,IBYTE(1)),(IWORD,IBYTE(1))

COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
MEROFL,SPOS,RC

COMMON/BUFCON/FBSST1,FBLST,F256,F256D,FBSST,F1,F497,
FBSST1,FBLST,OLAP,RST,IBLK

IF (ICODE.GE.30) GOTO 1

BUFEND=8449

IF (ICODE.GE.0) GOTO 1

EOF=0

RETURN

C FILL FRAME IN NORMAL CIRCUMSTANCES

1 IF (EOFFLG.EQ.2) RETURN

IF (ICODE.GT.1) GOTO 10

IF (ICODE.EQ.2) FRAME(IST)=IWORD

DO 20 I=IST,IFIN

FRAME(I)=BUFF(SPOS)

SPOS=SPOS+1

20 IF (SPOS.LT.BUFEND) GOTO 20

EOF=0

GOTO 10

EOF=0

CONTINUE

RETURN

C BYTE LOST PATTERN FRAME FILL

IF (ICODE.EQ.2) IWORD=FRAME(33)

DO 40 I=IST,IFIN

IWORD=BUFF(SPOS)

IF (I.EQ.2) BYTE(1)

IF (I.GT.2) BYTE(1)

FRAME(I)=IWORD

IF (ICODE.EQ.2) IWORD=IWORD

SPOS=SPOS+1

BUFEND=SPOS+IN

EOF=EOF+4

CONTINUE

RETURN

EOF RETURN

EOF=EOF+2

RETURN

END
SUBROUTINE TAPRED(ICOM, IDRV, ISTAT, ITLEN, ILEN, IFNUM, IEDT)

C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ, 0 IS A WRITE, 1 IS AWRITE
C IDRV IS THE DRIVE BEING USED
C ISTAT IS THE STATUS ON RETURN
C ITLEN IS THE TIME LENGTH OF A FILE READ
C ILEN IS THE BLOCK LENGTH OF A FILE READ OR WRITTEN

INTEGER*2 MASKI8, ESTATI
LOGICAL*1 ISTAT, COM(4), SDSCOM(8), IDRV, ITLEN, ECOM(4), ZIFLEN, ESTAT, ERRS(8)

DATA MASK/"1","2","4","10","20","40","100","200/"
DATA SDSCOM/"0","1","2","3","4","6","7/"
DATA ERRS/"377","377","377","377","377","377","377/"

0007 ITRY=0
0008 IF(ICOM) 10,30,20

SECTION CONTROLLING A READ
CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED

10 ITRY=ITRY+1

SET UP COMMAND FOR READ

11 COM(1)=SDSCOM(4)
12 COM(2)=0
13 COM(3)=IDRV
14 COM(4)=-1
15 CALL SDSCOM(COM, ISTAT, ITLEN, ILEN)
16 IF(ISTAT.EQ.0)RETURN

ERROR DETECTED ON READ
GOTO 40

IF SHORT RECORD FOUND REREAD TAPE

50 ITMP=ISTATI.AND.MASK(6)
51 IF(ITMP.NE.0)GOTO 10
52 ITMP=ISTATI.AND.MASK(2)
53 IF(ITMP.EQ.0)RETURN

IF CRC ERROR FOUND REWIND TAPE AND RETRY

WRITE(2,2010)IFNUM
5510 FORMAT(' FILE NO ',14,' CRC ERROR REWINDING'
5520 IF(ITRY.GE.2)GOTO 10
5520 ECOM(1)=SDSCOM(4)
5530 ECOM(2)=1
WRITE SECTION

20 ITRY=ITRY+1
20 IF(ITRY.GT.3)GOTO 130
20 COM(1)=SDSCOM(7)
20 ILEN=(ILEN+3)/4
30 COM(2)=ILEN
30 COM(3)=IDRV
60 COM(4)=1
30 CALL SDS10(COM,ISTAT, )
60 IF(ISTAT.EQ.0)RETURN

WRITE ERROR DETECTED

70 ISTAT=ISTAT
30 GOTO 40
70 ITMPI=ISTATI.AND.MASK(6)
70 IF(ISTAT.EQ.0.AND.ITMPI.EQ.0)RETURN

REPORT AND RETRY

70 WRITE(2,2020)IFNUM
70 FORMAT(1,14,'WRITE CRC ERR RETRY PROPOSED')
70 ECOM(1)=SDSCOM(6)
70 ECOM(2)=2
70 ECOM(3)=IDRV
70 ECOM(4)=6
70 CALL SDS10(ECOM,ESTAT, )
70 NBUF=0
70 ILEN=16
70 IPAD=32768
70 CALL TWRITE(ENNS,NBUF,ESTAT,IPAD,ILENE,IDRV)
70 GOTO 20

WIND FORWARD ONE FILE

30 COM(1)=SDSCOM(5)
30 COM(2)=1
30 COM(3)=IDRV
30 COM(4)=6
30 CALL SDS10(COM,ISTAT, )

CLEAR IRRELEVANT BITS FROM ERROR BYTE

30 ISTAT=ISTATI.AND..NOT.MASK(6)
30 IF(ISTAT.EQ.2)RETURN
30 ISTAT=ISTAT
30 IF(ISTAT.NE.0)GOTO 40
IF ISTAT=0 REWIND AND SET UP FOR NEXT READ 
AS THIS WAS A DATA FILE NOT A SHORT RECORD

IF ISTAT=1 SET UP FOR NEXT READ

C THIS SECTION THE MAIN TAPE ERRORS ARE 
C HANDLED SUCH AS: TAPE BUSY, TAPE OFFLINE 
C BOT,EOT 
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG

C HAVING EXAMINED STATUS IF TAPE STILL 
C BUSY, LOOP AGAIN IF NOT TRY COMMAND AGAIN

C TAPE OFFLINE

C HAVING ANOUNCED ERROR SKIP UNTIL CORRECTED

FORTRAN IV
.TITLE ISWAP

.globl ISWAP, IGCHK, IGAIN, IDIRG

ISWAP:
    mov 02(R5), R0
    SWAB R0
    RTS PC

IGCHK:
    mov 02(R5), R0
    TSTB R0
    BPL 1S
    mov @1, 04(R5)
    3r
    1S:
    mov $-1, 04(R5)
    2S:
    BIC @177760, R0
    RTS PC

IDIRG:
    TST 02(R5)
    BPL 3S
    MOV @1, R0
    RTS PC
    3S:
    MOV @-1, R0
    RTS PC

IGAIN:
    mov 02(R5), R0
    ASR R0
    BCC 4S
    ADD @6(R5), 04(R5)
    4S:
    TST R0
    BPL 5S
    ADD @1, R0
    5S:
    ASL R0
    RTS PC

.END
Sort :- MPSORT

Input file......DK1:MPSoRT.DAT
Log file........DK1:MPSoRT.LOG

Input Parameters

READ(1,1000)NFILES,NCHANI,NCHANO,NROW,TPDRR,TPDRW

1000 FORMAT(12I5)

NFILES...Number of input files for sorting
NCHANI...Number of channels in input files
NCHANO...Number of channels to be output
NROW.....Number of rows in sort matrix
TPDRR....Input tape drive
TPDRW....Output tape drive

READ(1,1000)ISECIN,ISBLKO,IFBLKO,USEFLG,INFLG,OUTFLG,IGCODE

ISECIN...Number of half second(128 sample) blocks in input data
ISBLKO...First half second block to output
IFBLKO...Last half second block to output
USEFLG...New run, restart flag
  0 - New run
  1 - restart of previous run using old temporary sort files
INFLG...Input flag
    0 - Input from tape
    1 - input from disc

OUTFLG...Output flag
    0 - output to tape
    1 - output to disc

IGCODE...Gather code for type of gather formed by this sort run

READ(1,1000)(INDEX(I),I=1,NROW)

INDEX....Sort file sequence
    If USEFLG = 0 input sequence 1...NROW
    If USEFLG = 1 Input sequence from last line of previous log file

READ(1,1000)(ICHANO(I),I=1,NCHANO)

ICHANO...Input channels which are to correspond to the output channels 1 to NCHANO in order.

READ(1,1000)(IPOS(I),I=1,NCHANO)

IPOS.....Sort position of each of the output files
    Can take values from 1 to NROW

READ(1,1001)FBUF

1001 FORMAT(3A4)
FBUF....If INFLG = 0 Temporary file for tape input
    INFLG = 1 Input files from 1 to NFILES

READ(1,1001)FBUF

FBUF....If OUTFLG = 0 Not present
    OUTFLG = 1 Output files from 1 to NFILES

READ(1,1001)(TPNAM(I),I=1,NROW)

TPNAM....Temporary files for sort
& NPSORT.FOR ... THIS IS A GENERAL PURPOSE SORTING PROGRAM FOR SEISMIC DATA

C VIRTUAL RDNAM(200), WRTNAM(200), TPNAM(24)
C REAL *8 FSPECWR, FSPECW, FMBUF, RDNAM, WRTNAM, TPNAM
C INTEGER*4 DEV, FMBUF(3), SEISC(2448)
C INTEGER*2 IBLK(256), IBLKO(24), IBLKI(24), ICHANO(24), IPOS(24), USEFLG, INFLG, OUTFLG, INDEX(25), FLEN
C LOGICAL*1 STAT, TLEN, TPDRR, TPDRW, LBLK(512), ICODE
C EQUIVALENCE (LBLK(ll), IHBLK(lll
C DATA DEV/3RRK

C SET UP I/O
C IF (ICDFN(5).NE.0) STOP 'CHANNEL SET ERROR'
C IF (IFETCH(DEV).NE.0) STOP 'FETCH ERROR'
C CALL ASSIGN(1, 'DK2:MPSORT.DAT', 14)
C CALL ASSIGN(2, 'DK2:MPSORT.LOG', 14)
C IRD=20
C IER=21
C IEOV=0
C IEOV=0

C READ INPUT DATA
C READ(1,1000) NFILES, NCHAN1, NCHAN0, NROW, TPDRR, TPDRW
C FORMAT(1215)
C READ(1,1000) IFSECIN, ISBLKO, IFBLKO, USEFLG, INFLG, OUTFLG, ICODE
C READ(1,1000) (INDEX(I), I=1,NROW)
C READ(1,1000) (ICHANO(I), I=1,NCHAN0)
C READ(1,1000) (IPOS(I), I=1,NCHAN0)

C READ FILE SPECS
C IF (INFLG, NE.0) GOTO 10
C READ(1,1001) FMBUF
C FORMAT(3A4)
C CALL IRAD50(12, FMBUF, FSPECW)
C GOTO 15
C DO 20 J=1, NFILES
C READ(1,1001) FMBUF
C CALL IRAD50(12, FMBUF, FMBUF)
C READ(3A4)
C CONTINUE
C CONTINUE
C IF (OUTFLG,.EQ.0) GOTO 30
C DO 35 J=1, NFILES
C READ(1,1001) FMBUF
C CALL IRAD50(12, FMBUF, FMBUF)
C CONTINUE
C CONTINUE
C READ ARRAY SORT FILE SPECS

C
39 DO 40 J=1,NROW
30 READ(1,1001)FNBUF
34 CALL IRADS0(12,FNBUF,FMBUF)
355 TPNAME(J)=FMBUF
35 CONTINUE

C
C SET UP DATA CONSTANTS
C
947 ISV=.0
948 NBLKR=INCHAN*ISEC/NL+9
949 IFSIZO=(NBLK*INCHAN)+1
951 LSTBLK=IFSIZO-1
952 NSAMP=W=NBLK*256
953 NBE=*(ISBLK)*128
954 NFIN=(ISBLK)*128

C
C SET UP BLOCK POSITIONS IN FILES
C
955 DO 45 J=1,NCHAN
956 IBLK(J)=(ICHANO(J)-1)*ISEC+ISBLK
957 IBLKO(J)=1
958 DO 50 J=2,NCHAN
959 IBLKO(J)=IBLKO(J-1)+NBLK

C
C SET UP ARRAY SORT FILES
C
960 IF(USEFLG.NE.0)GOTO 60
961 DO 55 L=1,NROW
962 FMBUF=TPNAME(L)
963 ITHM=22+L
964 IF(IWRITE(IITM,FMBUF,IFSIZO).LT.0)STOP'ENTER ERR'
965 IF(IWRITE(IITM,FMBUF,IFSIZO).LT.0)STOP'WRITE ERR'
966 CALL CLOSE(IITM)
967 CONTINUE
971 DO 70 L=1,NROW
972 ITHM=22+L
973 FMBUF=TPNAME(L)
974 IF(LLOOKUP(IITM,FMBUF).LT.0)STOP'LOOKUP ERR'
975 CONTINUE

C
C C START OF MAIN WORK LOOP
C
977 DO 999 IFIL=1,NFILES
978 IFNUM=IFIL
979 INDEX(NROW+1)=INDEX(1)
980 DO 100 J=1,NROW
981 IFINDEX(J)=INDEX(J)+1
982 IF(INEFLG.NE.0)GOTO 105

C
C C TAPE READ CONTROL
C
IF (IENET (IRD, FSPEC, NBLKR).LT.0) STOP 'ENTER ERR'

C EOT CONTROL

120 IF (ITRY.GT.3) GOTO 125

110 IF (IENET.GE.0) GOTO 110

125 WRITE (7, 1070) TPDRR, IFNUM

1070 FORMAT(' EOT ON READ DRIVE:' , 'I2,' FILE NO:' , 'I5')

WRITE (7, 1070)

1020 FORMAT(' ENTER NEW READ TAPE DRIVE:' , 'S')

READ (5, 1030) TPDRR

1030 FORMAT (11)

IEOTR = 0

1031 IF (TPDRR.GT.2) STOP 'EOTR TERMINATE'

C TAPE READ

110 CALL TAPRED (-1, TPDRR, ISTAT, TLEN, FLEN, IFNUM, IEOTR)

1070 IF (ISTAT.LT.0) WRITE (2, 1070) IFNUM

1070 FORMTA(' WARNING RETRY FAILED FOR FILE:' , 'I5')

115 IF (IEOTR.LT.0) GOTO 115

115 CALL TAPRED (0, TPDRR, ISTAT, , IFNUM, IEOTR)

C CHECK IF READ A BAD FILE

115 CALL IWAIT (IRD)

116 ERR = 0

117 ITRY = ITRY + 1

118 IF (IREADM (1, IERR, I, IRD)).LT.0) STOP 'ERR READ ERR'

119 IF (IERR.EQ.'177777') GOTO 120

120 CALL CLOSEC (IRD)

C OPEN UP INPUT FILE FOR USE

125 FMBUF = FSPEC

125 IF (INFLG .NE. 0) FMBUF = RDNUM (IFNUM)

125 FLOOKUP (IRD, FMBUF).LT.0) STOP 'LOOKUP ERR'

C HEADER BLOCK MANIPULATION

125 ICH = 22 + INDEX (INROW)

125 IF (IREADM (256, IBLK, I, IRD)).LT.0) STOP 'READ ERR'

125 LBLK (19) = ICODE

125 IBLK (7) = NCHANO

125 IBLK (9) = IBLK (8) + HBEG

125 IBLK (9) = IBLK (9) + HFIN

125 IF (IVRITW (256, IBLK, I, ICH)).LT.0) STOP 'WRITE ERR'

C MAIN TRANSFER LOOP

125 DO 132 ICH = 1, NCHANO

132 ICH = 22 + INDEX (IPOS (ICH))

132 IBLK I = IBLK (ICH)
ITBLKO=ITBLK(I)

IF(IREADW(NSAMPW,SEIS,ITBLKO,IRD).LT.0) STOP 'READ ERR'

IF(IWRITE(NSAMPW,SEIS,ITBLKO,ITCH).LT.0) STOP 'WRITE ERR'

CONTINUE

IF(USEFLG.EQ.0.AND.IFNUM.LT.NROW) GOTO 210

ITCH=INDEX(1)+22

FBUFF=TPNAME(INDEX(1))

IF(OUTFLG.NE.0) GOTO 220

CALL CLOSEC(ITCH)

IFLEN=LOOKUP(IFWRT,FMBUF)

IF(IFLEN.LT.0) STOP 'FMBUF LOOKUP ERR'

CALL TAPRED(1,TPDRW,ISTAT,IFLEN,IFNUM,IEOTW)

CALL CLOSEC(IFWRT)

IF(LOOKUP(ITCH,FMBUF),LT.0) STOP 'LOOKUP ERR'

TAPE OUTPUT

IF(IEOTW.EQ.0) GOTO 230

WRITE(7,1043)TPDRW,IFNUM

1043 FORMAT(' TPE DRV:',I2,' FL: ',I5)

WRITE(7,1053)

1053 FORMAT(' ENR: ',S)

READ(5,1033)TPDRW

1033 IF(EOF=.T.) STOP 'EOF TERMINATION'

CALL TAPRED(1,TPDRW,ISTAT,IFLEN,IFNUM,IEOTW)

READ(5,1033)TPDRW

1033 IF(EOF=.T.) STOP 'EOF TERMINATION'

STORAGE ON DISC

CALL CLOSEC(IFWRT)

END OF MAIN LOOP

CLOSE DOWN CODE

END OF MAIN LOOP

CALL CLOSEC(IFWRT)

END OF MAIN LOOP

CLOSE DOWN CODE
Pre-Stack Processing :- MPPRST

Input File.....DK2:MPPRST.DAT
Log File.......DK2:MPPRST.LOG

Input Parameters

READ(1,1000)NFILES,NCHAN,NSAMP,NSTART,INFLG,OUTFLG

1000 FORMAT(12I5)

NFILES...Number of files to process
NCHAN....Number of channels per file
NSAMP....Number of samples per channel
NSTART...Starting sample number, from time 0
INFLG....Input flag
  0 - Tape input
  1 - Disc input
OUTFLG...Output flag
  0 - Tape output
  1 - disc output

READ(1,1000)TPDRR,TPDRW

TPDRR....Input tape drive
TPDRW....Output tape drive

READ(1,1100)FSAMP

1100 FORMAT(F10.0)
FSAMP....Sampling frequency, samples per millisecond

READ(1,1200)FBUF

1200 FORMAT(3A4)

FBUF.....If INFLG = 0, Temporary file for tape read
        INFLG = 1, Input files, from 1 to NFFILES

READ(1,1200)FBUF

FBUF.....If OUTFLG = 0, Temporary file for tape write
        OUTFLG = 1, Output files, from 1 to NFFILES

READ(1,1000)NPROC

NPROC....Number of processes to be applied. Including any process applied twice.

READ(1,1000)(UTLFLG(I),I=1,NUTIL)

UTLFLG...On/Off flag for each process
         1 - Process is to be applied
         0 - process not to be applied

READ(1,1000)(IORD(I),I=1,NPROC)

IORD.....Order in which processes to be applied
         Input code number for process in position in which it is wished to apply it
For each of the processes which is to be applied, the specific input is input next, in the UTLFLG bit set order.

1....Trace Edit

READ(1,1000)NFILED

READ(1,1000)(IFILED(I),ICHAND(I),I=1,NFILED)

NFILED...Number of channels to be edited out
IFILED...edit channel file number
ICHAND...edit channel, channel number in above file

2....Polarity Reversal

READ(1,1000)NCHANP

READ(1,1000)(ICHANP(I),I=1,NCHANP)

NCHANP...Number of channels in each gather with incorrect polarity
ICHANP...Number of channel with incorrect polarity

3....Gain Ramps

e0.2t Ramp

READ(1,1000)IAPLX

IAPLX....Application flag
 0 - apply ramp
 1 - remove ramp
te0.2t Ramp

READ(1,1000)IAPLTX

IAPLTX...Application flag

0 - apply ramp

1 - remove ramp

TV**2 Ramp

READ(1,1000)IAPLTV,NLYR

READ(1,1100)(TOLYR(I),VLYR(I),I=1,NLYR)

IAPLTV...Application flag

0 - apply ramp

1 - remove ramp

NLYR.....Number of time/velocity pairs

TOLYR....Zero offset two-way travel time

VLYR.....RMS velocity at above time

4....Mute

READ(1,1000)NTAP

READ(1,1000)(MUTE(I),I=1,NCHAN)

READ(1,1000)(MUTET(I),I=1,NCHAN)

NTAP.....Number of points in cosine taper

MUTE.....Sample value at which to end mute, for early mute.

MUTET....Sample value to mute from, for late mute

5....Spiking Deconvolution
READ(1,1000)NFILT,ISPIKE,INORM

READ(1,1100)WHITE

NFILT....Number of samples in the filter

ISPIKE....Spike position

INORM....Normalisation flag

0 - no normalisation
1 - Filter unit energy
2 - constant input/output energy

WHITE....Fractional pre-whitening

6....Bandpass Filtering

READ(1,1100)FL,FU

READ(1,1100)FTPR1,FTPR2

FL.......Starting frequency for lower cutoff position Hz
FU.......Starting frequency for upper cutoff position Hz
FTPR1....Length of lower cosine taper Hz
FTPR2....Length of upper cosine taper Hz

7....Bandreject filtering

READ(1,1100)FLR,FUR

FLR......Lower frequency cutoff position Hz
FUR......Upper frequency cutoff position Hz

8....Prediction Error Deconvolution
READ(1,1000)NPFILT,NLAG,IPNORM

READ(1,1100)PRWHIT

NPFILT...Filter Length in samples
NLAG.....Prediction distance, samples
IPNORM...Normalisation flag
    0 - no normalisation
    1 - Filter normalised to unit energy
    2 - Constant input/output energy
PRWHIT...Fractional prewhitening

9....Normalisation

READ(1,1000)NRMFLG

NRMFLG...Normalisation flag
    0 - normalise to unit energy
    1 - normalise to unit maximum amplitude
C THIS INVOLVES THE FOLLOWING
  1: EDIT
  2: POLARITY REVERSAL
  3: EXP(-0.2T) AMP RECOVERY
  4: T*EXP(-0.2T) AMP RECOVERY
  5: TV**2 AMP RECOVERY
  6: MUTING
  7: DECONVOLUTION
  8: BANDPASS FILTERING
  9: BANDREJECT FILTERING
  10: PREDICTION ERROR FILTERING
  11: NORMALISATION TO UNIT ENERGY OR AMPLITUDE

REAL*8 FSPECR, FSPECW, FNAMR, FNAMO
VIRTUAL FNAMR(180), FNAMO(180), EXPT(2048), TEXPT(2048),
%TVSQ(2048), TAPER(512), BPASS(2049), BRJCT(2049)
REAL*4 FBUF(3), VLYR(20), VLYR(20), CONST(3), SEISM(2048)
INTEGER*2 IORD(11), IUTFLG(11), IFILED(180), ICAND(180),
%ICHANP(24), MUTE(24), MUTET(24), IHBK(256), OUTFLG
LOGICAL*1 TPDRR, TPDRW, ISTAT, ITLEN
DATA DEV/3RRK /

SET UP VIRTUAL ADDRESSES
IEOTR=0
IEOTW=0
ATAP=APGAD(TAPER(1))
ATBP=APGAD(BPASS(1))
ATVSQ=APGAD(TVSQ(1))
ATEXPT=APGAD(TEXPT(1))
ATEXPT=APGAD(TEXPT(1))
ATBR=APGAD(BRJCT(1))

SET UP I/O CHANNELS AND READ IN CONTROL DATA
IF(ICDFN(25).NE.0) STOP 'CHANNEL OVERFLOW'
CALL ASSIGN(1,'DK2:MPRST.DAT',14)
CALL ASSIGN(2,'DK2:MPRST.LOG',14)
IDCH=20
IDCH=21
READ(1,1000) NFIL, NCHAN, NSAMP, NSTART, INFLG, OUTFLG
1003 FORMAT(12I5)
1022 FORMAT(2F10.0)
READ(1,1000) TPDRR, TPDRW
READ(1,1100) FSAMP
1103 FORMAT(2F10.0)
C READ IN FILE SPECS FOR INPUT
IF(INFLG.NE.0) GOTO 10
READ(1,1200) FBUF
1200 FORMAT(3A4)
CALL IRADS(12,FBUF,FSPECR)
GOTO 20
10 DO 30 I=1,NFILES
20 READ(1,1200)FBUF
30 CALL IRADS00(12,FBUF,FSPECW)
35 FNAMR(I)=FSPECW
36 30 CONTINUE

READ IN FILE SPECS FOR OUTPUT

20 IF(OUTFLG.NE.0)GOTO 40
30 READ(1,1200)FBUF
40 CALL IRADS00(12,FBUF,FSPECW)
50 GOTO 50
40 DO 50 I=1,NFILES
50 READ(1,1200)FBUF
60 CALL IRADS00(12,FBUF,FSPECW)
65 FNAMO(I)=FSPECW
70 50 CONTINUE

READ IN JOB SPECIFIC DATA AND SET UP THE FILTERS TO BE USED

50 NUTIL=11
60 ITX=0
70 CONST(1)=FLOAT(NSTART)
80 CONST(2)=0.2
90 CONST(3)=1.0/(1.000*0.8*FSAMP)
100 CALL APINIT
110 NSAMP2=2
120 IF(NSAMP2.GE.NSAMPl)GOTO 52
130 NSAMP2=NSAMP2*2
140 52 GOTO 52
150 CONTINUE

READ IN FLAGS FOR PROCESSES AND EXECUTION ORDER

60 PROC
70 READ(1,1000)NPROC
80 READ(1,1000)(UTLFLG(I),I=1,NUTIL)
90 READ(1,1000)(IORD(I),I=1,NPROC)

TRACE EDIT DATA

50 IF(UTLFLG(I).EQ.0)GOTO 55
60 READ(1,1000)NFILED
70 READ(1,1000)(IFILED(I),ICHAND(I),I=1,NFILED)

POLARITY REVERSAL DATA

55 IF(UTLFLG(2).EQ.0)GOTO 70
65 READ(1,1000)NCHANP
70 READ(1,1000)(ICHANP(I),I=1,NCHANP)

EXP(0.2T) RAMP DATA

GOTO 20
FORTRAN IV

0778 7.5 IF (UTLFLG(3).EQ.0) GOTO 87
0779 CALL TEVRMP(EXPT, ITX, NSAMP, CONST, AEXPT)
0780 ITX=1

0785 85 IF (UTLFLG(4).EQ.0) GOTO 90
0786 READ(1,1000) IAPLTX
0787 CALL TEVRMP(TEXPT, ITX, NSAMP, CONST, ATEXPT)
0788 ITX=1

0788 86 IF (UTLFLG(5).EQ.0) GOTO 100
0789 READ(1,1000) IAPLV, NLVR
0790 READ(1,1000) (TBLYR(I), VLYR(I), I=1, NLYR)
0791 CALL TVRMP(TVSQ, TBLYR, VLYR, NLVR, ITX, NSAMP, NSTART, CONST, %FSAMP, ATVSQ)

C MUTE DATA

0795 1.87 IF (UTLFLG(6).EQ.0) GOTO 110
0796 READ(1,1000) NTAP
0797 READ(1,1000) (MUTE(I), I=1, NCHAN)
0798 READ(1,1000) (MUTET(I), I=1, NCHAN)
0799 CALL COTAP(TAPER, NTAP, ATAP)

C DECON INPUT

0805 1.10 IF (UTLFLG(7).EQ.0) GOTO 120
0806 READ(1,1000) NFILT, ISPIKE, INORM
0807 READ(1,1000) WHITE

C BANDPASS FILTER

0805 1.25 IF (UTLFLG(8).EQ.0) GOTO 130
0807 READ(1,1100) FL, FU
0809 READ(1,1100) FTPR1, FTPR2
0801 DFI=FLOAT(NSAMP/2+1)/(FSAMP*500.0)
0802 FL1=FL-FTPR1
0803 FU4=FU+FTPR2
0804 CALL BANDPSIATBP, BPASS, FL1, FL, FU4, DFI, NSAMP2)
0805 NTRANF=2*NSAMP2
0806 NBEXP=NSAMP2+1

C BANDREJECT FILTER

0810 1.33 IF (UTLFLG(9).EQ.0) GOTO 140
0812 READ(1,1100) FLR, FUR
0814 DFI=FLOAT(NSAMP/2+1)/(FSAMP*500.0)
0815 NTRANF=2*NSAMP2
0816 NBfilt=NSAMP2+1
NBEXP = (2 * NSAMP) + 2 - NSAMP
CALL BANDRJ(ATBR, BRCT, FLR, FUR, DFI, NSAMP2)

C PREDICTION ERROR FILTER

14J IF (UTLFLG(16), EQ, 0) GOTO 150
READ (1, 1000) NPFIL, LT, NLAG, IPNORM
READ (1, 1000) PRWHIT

C TRACE NORMALISATION

150 IF (UTLFLG(11), EQ, 0) GOTO 160
READ (1, 1000) NRMFLG

C BLOCKING PARAMETERS

16J CONTINUE
IFED = 1
NBLKW = NSAMP / 128 * NCHAN + 1
NBLKR = NBLKW + 5
NBLKTR = NSAMP / 128

C START OF LOOP ON DIFFERENT FILES

DO 200 IFE = 1, NFILES
IFIL = IFNUM

C TAPE INPUT HANDLING

IF (ENTER(IDCH, FSPEC, NBLKR), LT, 0) STOP 'ENTER ERROR'
ITYR = 1

C EOT CHECK

225 IF (ITYR, GE, 3) GOTO 227
226 IF (IEOTR, GE, 0) GOTO 220
227 WRITE (7, 1000) TPDRR, IFIL
1000 FORMAT (' EOT ON DRIVE:', I2, ', FILE NO:', I4)
228 WRITE (7, 1001)
1001 FORMAT (' ENTER NEW READ DRIVE NO:', S)
229 READ (5, 1002) TPDRR
1002 FORMAT (11)
230 IEOTR = 0
231 IF (TPDRR, GT, 2) STOP ' EOT TERMINATION'

C READ FROM TAPE

233 CALL TAPRED(-1, TPDRR, ISTAT, ITLEN, IFILEN, IFIL, IEOTR)
234 IF (ISTAT, LT, 0) WRITE (2, 1500) IFIL
1500 FORMAT (' RETRIES ON READ FAILED ON FILE', I5)
235 IF (IEOTR, LT, 0) GOTO 230

C WIND OVER EOF MARK
CALL TAPRED(Ø, TPDRR, ISTAT, , IFIL, IEOTR)
230 CALL IWAIT(IDCH)

199 ITRY=ITRY+1
200 IF(IREADW+1, IERR, Ø, IDCH), LT. Ø) STOP 'ERR READ ERR'
201 IF(IERR.EQ."177777") GOTO 225
202 CALL CLOSE(IDCH)

210 IF(INFLG.NE.Ø) FSPECW=FNAMO(IFNUM)
211 IF(LOOKUP(IDCH, FSPECW), LT. Ø) STOP 'LOOKUP ERR'

214 

215 IF(OUTFLG.NE.Ø) FSPECW=FNAMO(IFNUM)
216 IF(ENTER(IDCH1, FSPECW, NBLKW), LT. Ø) STOP 'ENTER ERR2'

HEADER BLOCK MANAGEMENT
218 IF(IREADW(256, IBLK, Ø, IDCH), LT. Ø) STOP 'READW ERR'

PUT IN CORRECT ORDER OF PROCESSING
219 IBFREE=IBLK(24)
220 IBLK(129+IBFREE)=1
221 IBFREE=IBFREE+1
222 IBLK(129+IBFREE)=NPROC
223 IBFREE=IBFREE+1
224 DO 215 J=1, NPROC
225 IBLK(129+IBFREE)=IORD(J)
226 IBFREE=IBFREE+1
227 CONTINUE
228 IBLK(24)=IBFREE

WRITE OUT UPDATED HEADER
229 IF(IWRITE(256, IBLK, Ø, IDCH1), LT. Ø) STOP 'WRITE ERR'
230 JBLK=1
231 DO 300 ICHNUM=1, NCHAN
232 IF(IREADW(2*NSAMP, SEISM, JBLK, IDCH), LT. Ø) STOP 'READW ERR2'
233 CALL APPPUT(SEISM, Ø, NSAMP, 2)
234 CALL APWD
235 DO 310 IPCNUM=1, NPROC
236 GOTO(410, 420, 430, 440, 450, 460, 470, 480, 490, 500, 510) IORD(IPCNUM)

PROCESS EDIT COMMANDS
237 IF(IFED.GT.NFILED) GOTO 400
238 IF(IFILED(IFED), NE. IFIL) GOTO 400
239 IF(ICHAND(IFED), NE. ICHNUM) GOTO 400
240 IFED=IFED+1
CALL VCLR(0,1,NSAMP)
CALL APWR
CALL APGET(SEISM,0,NSAMP,2)
CALL APWR
GOTO 310

POLARITY REVERSAL

DO 421 IP=1,NCHANP
IF(ICHANP(IP).EQ.ICHNUM)GOTO 425
CONTINUE
GOTO 4/18
CALL VNEG(0,1,0,1,NSAMP)
CALL APWR
GOTO 480

AMP RECOVERY FILTERS APPLICATION AND REMOVAL

CALL APPUTA(NSAMP,NSAMP,2,AEXPT)
GOTO 455

CALL APPUTA(NSAMP,NSAMP,2,ATEXPT)
GOTO 455

CALL APPUTA(NSAMP,NSAMP,2,ATVSQ)

COMMON CODE

CALL APWD
IF(IAPL.EQ.0)CALL VMLU(0,1,NSAMP,1,0,1,NSAMP)
IF(IAPL.NE.0)CALL VDIV(NSAMP,1,0,1,0,1,NSAMP)
GOTO 480

MUTE APPLICATION

CALL APPUTA(NSAMP,NTAP,2,ATAP)
CALL APWD
LMUT=MUTE(ICHNUM)
CALL VCLR(0,1,LMUT)
CALL VMUL(LMUT,1,NSAMP,1,LMUT,1,NTAP)
CALL APWR
LMUT=MUTET(ICHNUM)
IF(LMUT.GE.NSAMP)GOTO 480
FORTN IV VS2.84 THU 08-JAN-81 00:23:54 PAGE 887

0233  NMUT=NSAMP-LMUT
0236  CALL VCLR(NSAMP-1,-1,NMUT)
0237  CALL VMUL(LMUT,-1,NSAMP,1,LMUT,-1,NTAP)
0238  CALL APWR
0239  GOTO 400

C DECON
C
0241  478 CALL SPIKE(NSAMP,NSAMP2,NFILT,WHITE,INORM,ISPIKE)
0242  GOTO 400

C BANDPASS FILTER
C
0244  488 CALL APPUTA(4100,NBFILT,2,ATBP)
0245  CALL APWD
0246  CALL VCLR(NSAMP,1,NBEXP)
0247  CALL RFFT(0,NTRANF,+1)
0248  CALL RFFTSC(0,NTRANF,3,1)
0249  CALL VMUL(0,2,4100,1,0,2,NBFILT)
0250  CALL VMUL(1,2,4100,1,1,2,NBFILT)
0251  CALL RFFTSC(0,NTRANF,-3,0)
0252  CALL RFFT(0,NTRANF,-1)
0253  CALL APWR
0254  GOTO 400

C BANDREJECT FILTER
C
0256  498 CALL APPUTA(4100,NRFILT,2,ATBR)
0257  CALL APWD
0258  CALL VCLR(NSAMP,1,NBEXP)
0259  CALL RFFT(0,NTRANF,+1)
0260  CALL RFFTSC(0,NTRANF,3,1)
0261  CALL VMUL(0,2,4100,1,0,2,NRFILT)
0262  CALL VMUL(1,2,4100,1,1,2,NRFILT)
0263  CALL RFFTSC(0,NTRANF,-3,0)
0264  CALL RFFT(0,NTRANF,-1)
0265  CALL APWR
0266  GOTO 400

C PREDICTION ERROR FILTER
C
0268  508 CALL PREDICT(NSAMP,NSAMP2,NPfilt,PRWHIT,IPNORM,NLAG)
0269  GOTO 400

C NORMALISATION
C
0271  518 IF(NRMFLG.EQ.0)GOTO 515
0272  CALL MAXMGV(0,1,2050,NSAMP)
0273  GOTO 516
0274  515 CALL SVESQ(0,1,2050,NSAMP)
0275  CALL VSORT(2050,1,2050,1,1)
0276  515 CALL VDIV(2050,0,0,1,0,1,NSAMP)
0277  CALL APWR
0278  GOTO 400
$275$ 420 CONTINUE

C END OF PROCESS LOOP

C$276$ CALL APWAIT

C$277$ CALL APGET(SEISM, SE, NSAMP, 2)

C$278$ CALL APVD

C$279$ IF(IWRITW(2*NSAMP, SEISM, JBLK, IDCH1) .LT. 0) STOP 'WRITW ERR2'

C$280$ JBLK = JBLK + NBLKTR

C END OF CHANNEL LOOP

C$281$ CALL CLOSEC,IDCH)

C$282$ CALL CLOSEC,IDCH1)

C OUTPUT TO TAPE

C$283$ IF(OUTFLG .NE. 0) GOTO 286

C$284$ IFLEN = LOOKUP(IDCH1, FSPECW)

C$285$ IF(IFLEN .LT. 0) STOP 'LOOKUP ERR3'

C$286$ CALL TAPRED(1, TPDRW, ISTAT, ITLEN, IFLEN, IFIL, IEOFW)

C$287$ CALL CLOSEC,IDCH1)

C CHECK FOR ERRORS

C$288$ IF(IEOFW .GE. 0) GOTO 289

C$289$ WRITE(7, 1600) TPDRW, IFIL

C$290$ 1600 FORMAT(' EOT ON DRIVE: ', I2, ' FILE NO: ', I4)

C$291$ WRITE(7, 1601)

C$292$ 1601 FORMAT(' ENTER DRIVE NO FOR NEW WRITE TAPE: ', $)

C$293$ READ(5, 1802) TPDRW

C$294$ IEOFW = 0

C$295$ IF(TPDRW .GT. 2) STOP ' EOT WRITE TERMINATION'

C$296$ 500 IF(ISTAT .GE. 2) STOP 'WRITE TERMINATION'

C$297$ 250 IF(ISTAT .GE. 0) GOTO 286

C$298$ WRITE(2, 1700) IFIL

C$299$ 1700 FORMAT(' FATAL ERROR ON WRITE FILE NO: ', I5)

C$300$ STOP 'WRITE ERROR'

C$301$ CONTINUE

C$302$ CALL CLOSEC,IDCH)

C$303$ CALL CLOSEC,IDCH1)

C$304$ STOP 'NORMAL TERMINATION'

C$305$ END
SUBROUTINE TEXRMP(FILT, IFTYP, IFLG, NSAMP, CONST, AFILT, FS)
C IF IFTYP=0 THIS ROUTINE PRODUCES AN EXP(0.2T) ARRAY
C NSAMP LONG IN FILT
C IF IFTYP=1 T*EXP(0.2T) PRODUCED
C
CONST(1)=NSART
CONST(2)=0.2
CONST(3)=1.0/(1000.0*FSAMP)

VIRTUAL FILT(2048)
DIMENSION CONST(3)

IF(IFLG.NE.1) GOTO 10

FORM THE T RAMP

CALL APWAIT
CALL APPUT(CONST, 8189, 3, 2)
CALL APWD
CALL VCLR(0, 1, NSAMP)
CALL VRAMP(8189, 8191, 0, 1, NSAMP)

FORM EXP(0.2T)

CALL VSMUL(0, 1, 8190, NSAMP, 1, NSAMP)
CALL VEXP(NSAMP, 1, NSAMP, 1, NSAMP)
11 GOTO 20

FORM T*EXP(0.2T)

CALL VMUL(NSAMP, 1, 0, 1, NSAMP, 1, NSAMP)
20 CALL APWR
CALL APGETA(NSAMP, NSAMP, 2, AFILT)
RETURN
END

SUBROUTINE COTAP(TAPER, NTAP, ATAP)

THIS ROUTINE PRODUCES A COSINE TAPER NTAP

VIRTUAL TAPER(512)
CALL APWAIT
CALL APPUT(1.0/FLOAT(NTAP), 0, 1, 2)
CALL VCLR(1, 1, NTAP)
CALL VSADD(1, 1, 2306, 1, 1, 1)
CALL VSMUL(0, 1, 2306, 0, 1, 1)
CALL VRAMP(1, 0, 0, 1, NTAP)
CALL VCOS(0, 1, 0, 1, NTAP)
CALL VSADD(0, 1, 2409, 0, 1, NTAP)
CALL VSMUL(0, 1, 2327, 0, 1, NTAP)
CALL APWR
CALL APGETA(0, NTAP, 2, ATAP)
CALL APWD
RETURN
END
SUBROUTINE TVRMP(FILT, TLVR, VLVR, NLYR, IFLG, NSAMP, NSTART, 
%CONST, FSAMP, AFILT)

C THIS ROUTINE PRODUCES A TV**2 RAMP
C FROM VELOCITY INFO IN TLVR, VLVR

VIRTUAL FILT(2048)
DIMENSION TLVR(20), VLVR(20), CONST(3)

CHECK IF TV RAMP ALREADY FORMED

IF(IFLG.NE.0)GOTO 10
CALL APWAIT
GOTO 50
CALL APWD
CALL VRC(1, NSAMP)
CALL VRAMP(8189, 8191, 0, 1, NSAMP)

FORM VELOCITY RAMP IN FILT

10 N1 = 1
N2 = IFIX(FSAMP*TLVR(1))-NSTART
V1 = VLVR(1)
DO 14 I = N1, N2
14 FILT(I) = V1
15 IF(NLYR.EQ.1)GOTO 40
DO 20 J = 2, NLYR
20 N1 = N2 + 1
N2 = IFIX(FSAMP*TLVR(J))-NSTART
DELV = (VLVR(J)-VLVR(J-1))/(N2-N1+2)
V = VLVR(J-1)
DO 25 I = N1, N2
25 FILT(I) = V
26 V = V + DELV
30 CONTINUE
20 CONTINUE
40 NI = N2 + 1
50 N2 = NSAMP
VN = VLVR(NLYR)
DO 55 I = N1, N2
55 FILT(I) = VN

PUT TV RAMP IN AP AND FORM TV**2

CALL APWAIT
CALL APPUTA(NSAMP, NSAMP, 2, AFILT)
CALL APWD
CALL VSO(NSAMP, 1, NSAMP, 1, NSAMP)
CALL VMUL(0, 1, NSAMP, 1, NSAMP, 1, NSAMP)
CALL APVR
CALL APGETA(NSAMP, NSAMP, 2, AFILT)
CALL APWD
RETURN
END
SUBROUTINE BANOPS(ATBP,BPASS,Fl,F2,F3,F4,DFI,NSAMP)

C SUBROUTINE WHICH CREATES A BANDPASS FILTER
C Fl=BOTTOM CUT OFF FREQUENCY
C F2=START OF FULL PASS
C F3=END OF FULL PASS
C F4=TOP CUT OFF FREQUENCY
C BETWEEN Fl,F2 AND F3,F4 A COSINE TAPER IS APPLIED

C VIRTUAL BPASS(2049)
C
C SET UP CONSTANTS
C
DO 10 I=1,2
  DO 10 NTAP=NTP1
  IF(I.EQ.1) NTAP=NTP2
  RNTAP=1.0/FLOAT(NTAP)
  CALL APPUT(RNTAP,0,1,2)
  CALL VCLR(1,1,NTAP)
  CALL VTSADD(0,1.0,0,1,1,1)
  CALL VTSMUL(0,1.0,0,1,1,1)
  CALL VRAMP(1,0,0,1,NTAP)
  CALL VCOS(0,1.0,1,NTAP)
  CALL VTSADD(0,1.0,2049,0,1,NTAP)
  CALL VTSMUL(0,1.0,2327,0,1,NTAP)
  IF(I.EQ.1) CALL VMOV(0,1,2049,1,NTAP)
  IF(I.EQ.2) CALL VMOV(0,1,4100,1,NTAP)
  10 CONTINUE
C NOW HAVE TAPERS FORM REST OF FILTER
C
CALL VCLR(0,1,NFILT)
CALL VADD(N1,1.050,1,N1,1,NTP1)
CALL VADD(N4,-1.410,1,N4,-1,NTP2)
CALL VTSMUL(N2,1.2049,N2,1,NOK)
CALL APWR
CALL APGETA(0,NFILT,2,ATBP)
CALL APWD
RETURN
END
SUBROUTINE BANDRJ(ATBR,BRJCT,F1,F2,DFI,NSAMP)

C SUBROUTINE TO CREATE A BANDREJECT FILTER
C
F1=LOWER CUTOFF POSITION
F2=UPPER CUTOFF POSITION

THE FILTER TAKES THE FORM OF
A SINE BELL CENTERED ON THE FREQUENCY TO BE REMOVED COMPLETELY

VIRTUAL BRJCT(2849)

SET UP CONSTANTS

N1=2.*F1*DFI
N2=2.*F2*DFI
NTAP=N2-N1

CALL APWAIT

SET UP FILTER IN AP

CALL VCLR(0.,1.,NTAP)
NFILT=NSAMP+1
FTAP=1./FLOAT(NTAP)
CALL APPUT(FTAP,1.,1.,2.)
CALL APWD
CALL VTSMUL(0.,1.,2317,1.,1.,1.)
CALL VRAMP(0.,1.,0.,1.,NTAP)
CALL VCS(0.,1.,0.,1.,NTAP)
CALL VTSADD(0.,1.,2849,0.,1.,NTAP)
CALL VTSMUL(0.,1.,2327,0.,1.,NTAP)
CALL VMOV(0.,1.,4096,1.,NTAP)

SET UP FULL FILTER NOW TAPER FINISHED

CALL VCLR(0.,1.,NFILT)
CALL VTSMUL(0.,1.,2849,0.,1.,NFILT)
CALL VNUL(N1,1.,4096,1.,N1,1.,NTAP)
CALL APWR
CALL APGETA(0.,NFILT,2.,ATBR)
CALL APWD
RETURN
END
SUBROUTINE SPIKE(NSAMP, NSAMP2, ILENTH, WHITE, IFLAG, ISPIKE)

C WIENER SPIKING FILTER ROUTINE
C NSAMP=DATA LENGTH
C NSAMP2=NEAREST POWER OF 2 TO NSAMP
C ILENTH=FILTER LENGTH
C IFLAG=TRACE NORMALISATION FLAG
C 0 NO NORMALISATION
C 1 FILTER UNIT ENERGY
C 2 EQUAL INPUT-OUTPUT ENERGY
C ISPIKE SPIKE POSITION
NTRAN=2*NSAMP2
NCLR=NTRAN-NSAMP
NFCLR=NTRAN-ILENTH
NTRAN2=NTRAN+2
I6=NTRAN+ILENTH
I7=I6+ILENTH
I8=I7+ILENTH
ISP=I6+ISPIKE

HAVING SET UP CONSTANTS GET INPUT TRACE
ENERGY IF REQD FOR NORMALISATION

IF(IFLAG.NE.2)GOTO 10

CALL SVE SQ(0,1,8191,NSAMP)
CALL VSQRT(8191,1,8191,1,1)
CALL APWR
CALL APGETIEN,8191,1,2)
CALL APWD

GET AUTOCORRELATION FUNCTION

CALL VCLR(NSAMP,1,NCLR)
CALL RFFT(0,NTRAN,1)
CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
CALL CVMAGS(NTRAN,2,NTRAN2,2,NM1)
CALL VCLR(NTRAN+3,2,NM1)
CALL RFFTSC(NTRAN,NTRAN,-1,-1)
CALL RFFT(NTRAN,NTRAN,1)

AUTO FUNCTION NOW 2N LONG FROM NTRAN
ORIGINAL FUNCTION TRANSFORMED 8-NTRAN

NOW WHITEN

CALL APWR
CALL APPUT(WHITE,8191,1,2)
CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)

SET UP SPIKE CC FUNCTION
CALL VCLR(I6, ILENTH)
CALL VTSADD(ISP, 1, 2049, ISP, 1, 1)

C SOLVE EQNS
CALL WIENER(ILENTH, NTRAN, I6, I7, I8, 1)
CALL APCHK(IER)
IF (IER.NE.0) STOP 'LEVINSON FAILURE'
CALL VMOV(I7, 1, NTRAN, 1, ILENTH)

C NORMALISE FILTER IF ASKED FOR
IF (IFLAG.NE.1) GOTO 20
CALL SVESQ(NTRAN, 1, I6, ILENTH)
CALL VSORT(I6, 1, I6, 1, 1)
CALL VDIV(I6, 0, NTRAN, 1, NTRAN, 1, ILENTH)

C APPLY FILTER
20 CALL VCLR(I6, 1, NFCLR)
CALL RFFT(NTRAN, NTRAN, 1)
CALL VMUL(0, 1, NTRAN, 1, 0, 1, 2)
CALL CVMUL(2, 2, NTRAN2, 2, 2, 2, NM1, 1)
CALL RFFTSC(0, NTRAN, 0, -1)
CALL RFFT(0, NTRAN, -1)

C DO SCALING IF REQD
30 IF (IFLAG.NE.2) GOTO 30
CALL SVESQ(0, 1, 8191, NSAMP)
CALL VSQRT(8191, 1, 8191, 1, 1)
CALL APWR
CALL APPUT(EN, 8190, 1, 2)
CALL APWD
CALL VDIV(8191, 1, 8190, 1, 8190, 1, 1)
CALL VSMUL(0, 1, 8190, 0, 1, NSAMP)
30 CALL APWR
RETURN
END
SUBROUTINE PREDICT(NSAMP, NSAMP2, ILENTH, WHITE, IFLAG, NLAG)

THIS ROUTINE DESIGNS AND APPLIES
A PREDICTION ERROR FILTER
NSAMP=DATA LENGTH
NSAMP2=NEAREST POWER OF 2 TO NSAMP
ILENTH=FILTER LENGTH
WHITE=FRACTION PREWHITENING
IFLAG=0 NO NORMALISATION
=1 FILTER UNIT ENERGY
=2 INPUT/OUTPUT TRACE ENERGY CONSTANT
NLAG= LAG OFFSET OF PREDICTION

NTRAN=2*NSAMP
NCLR=NTRAN-NSAMP
NM1=NSAMP2-1
NTRAN2=NTRAN+2
NLG=NTRAN+NLAG
I7=NLG+ILENTH
I8=I7+ILENTH

GET INPUT TRACE ENERY

IF(IFLAG.NE.2)GOTO 10
CALL SVESQ(8191,1,NSAMP)
CALL VSQRT(8191,1,8191,1,1)
CALL APGETCEN(8191,1,2)
CALL APWD AUTOCORRELATION FUNCTION

CALL VCLR(NSAMP,1,NCLR)
CALL RFFT(Ø,NTRAN,1)
CALL VMOV(Ø,1,NTRAN,1,NTRAN)
CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
CALL VCLR(NTRAN+3,2,NM1)
CALL RFFTSC(NTRAN,NTRAN,-1,-1)
CALL RFFT(NTRAN,NTRAN,-1)

NOW WHITEN IT

CALL APWR
CALL APPUT(WHITE,8191,1,2)
CALL VSMAT(NTRAN,1,8191,1,NTRAN,1,NTRAN,1,1)

NOW SOLVE EQNS

CALL WIENER(ILENTH,NTRAN,NLG,17,18,1)
CALL APCHK(IER)
IF(IER.NE.0)STOP 'LEVINSON FAILURE'
IF(IFLAG.NE.1)GOTO 20
CALL SVESQ(I7,1,18,ILENTH)
CALL VSORT(18,1,18,1,1)
CALL VDIV(18,17,17,1,1,ILENTH)

CALL VCLR(NTRAN,1,NFILT)
CALL VTSADD(NTRAN,1,0.49,NTRAN,1,1)
CALL VSUB(17,1,NLG,1,NLG,1,ILENTH)
CALL VCLR(17,1,NTRAN-NFILT)
CALL RFFT(NTRAN,NTRAN,1)
CALL VMUL(1,NTRAN,1,1,1,1,2)
CALL CMUL(2,2,NTRAN2,2,2,2,1,1)
CALL RFFTS(NTRAN,NTRAN,-1)
CALL RFFT(NTRAN,1,1,1,1)

DO SCALING

IF (IFLAG.NE.2) GOTO 30
CALL VSORT(8191,1,8191,1,1)
CALL APWR
CALL APPUT(EN,8190,1,2)
CALL APWD
CALL VDIV(8191,1,8190,1,8190,1,1)
CALL VSMUL(1,1,8190,1,1,NSAMP)
30 CALL APWR
RETURN
END
SUBROUTINE TAPRED(icom, idrv, istat, itlen, ilen, ifnum, ieor)

C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ, 0 IS A WRITE
C IDRV IS THE DRIVE BEING USED
C ISTAT IS THE STATUS ON RETURN
C ITLEN IS THE TIME LENGTH OF A FILE READ
C ILEN IS THE BLOCK LENGTH OF A FILE READ OR WRITTEN

INTEGER*2 mask(8), estati
LOGICAL*1 istat, com(4), sdscom(8), idrv, itlen, ecom(4),

DATA mask/"1", "2", "4", "10", "20", "100", "200/
DATA sdscom/"0", "1", "2", "3", "4", "5", "6", "7/

ITRY = 0

IF (ICOM) 18, 38, 28

C SECTION CONTROLLING A READ

C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED

18 ITRY = ITRY + 1

C SET UP COMMAND FOR READ

COM(1) = SDSCOM(4)
COM(2) = 1
COM(3) = IDRV
COM(4) = -1
CALL SDSC10(COM, ISTAT, ITLEN, ILEN)

C ERROR DETECTED ON READ

48 ISTAT = ISTAT
GOTO 48

C IF SHORT RECORD FOUND REREAD TAPE

58 ITMP = ISTAT & MASK(6)
68 IF (ITMP .NE. 0) GOTO 19

C IF CRC ERROR FOUND REWIND TAPE AND RETRY

98 WRITE(2, 2810) IFNUM
2810 FORMAT(' FILE NO ', I4, ' CRC ERROR REWINDING')
108 IF (ITRY .GE. 2) GOTO 138

C ECOM(1) = SDSCOM(6)
138 ECOM(2) = 1
WRITE SECTION

035   20  ITRV=ITRV+1
036   IF (ITRV.GT.3) GOTO 130
037   COM(1)=SDSCOM(7)
039   ILEN=(ILEN+3)/4
040   COM(2)=IFLEN
041   COM(3)=IDRV
042   COM(4)=1
043   CALL SDS10(ECOM,ISTAT, , )
044   IF (ISTAT.EQ.0) RETURN

WRITE ERROR DETECTED

046   ISTAT=ISTAT
047   GOTO 40
048   ITMP=ISTAT.AND.MASK(6)
049   IF (ITMP.EQ.2.AND.ITMPI.EQ.2) RETURN

REPORT AND RETRY

052   WRITE(2,2020)IFNUM
2020  FORMAT( ' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED' )
055   ECOM(1)=SDSCOM(6)
056   ECOM(2)=2
057   ECOM(3)=IDRV
058   ECOM(4)=8
059   CALL SDS10(ECOM,ESTAT, , )
060   NBUF=8
061   ILEN=16
062   IPAD=32768
063   CALL TWRIT(ERROS,NBUF,ESTAT,IPAD,ILENE,IDRV)
064   GOTO 20

WIND FORWARD ONE FILE

064   30  COM(1)=SDSCOM(5)
065   COM(2)=1
066   COM(3)=IDRV
067   COM(4)=8
068   CALL SDS10(COM,ISTAT, , )

CLEAR IRRELEVANT BITS FROM ERROR BYTE

072   ISTAT=ISTAT.AND..NOT.MASK(6)
072   IF (ISTAT.EQ.2) RETURN
074   ISTAT=ISTAT
077   IF (ISTAT.NE.8) GOTO 40
ISTAT = rewind and set up for next read

As this was a data file not a short record

ECOM(1) = SDSCOM(6)
ECOM(2) = 1
ECOM(3) = IDRV
ECOM(4) = 8

CALL SDST0(ECOM, ESTAT, , )
35 RETURN

In this section the main tape errors are handled such as: TAPE BUSY, TAPE OFFLINE

Tape busy section...after clearing bot flag

WRITE(2,1010) ISTATI, IFNUM
WRITE(1,1910) FORMAT(' STATUS=',13,' FILE NO=',I4)
ISTATI = ISTATI .AND. .NOT. MASK(4)
ITMP = ISTATI .AND. MASK(5)

If(ITMP .EQ. 0) GOTO 90
90 ECOM(1) = SDSCOM(1)
ECOM(2) = 0
ECOM(3) = IDRV
ECOM(4) = 0

CALL SDST0(ECOM, ESTAT, , )

Having examined status if tape still busy. Loop again, if not try command again

ESTATI = ESTAT
ITMP = ESTATI .AND. MASK(5)
If(ITMP .NE. 0) GOTO 90
If(IOM) 10, 30, 20

Tape offline

If(ITMP .EQ. 0) GOTO 90
If(IOM) 1001, IDRV
1001 1211 FORMAT(' TAPE DRIVE ',11,' OFFLINE')

Having announced error skip until corrected

ECOM(1) = SDSCOM(1)
ECOM(2) = 0
ECOM(3) = IDRV
ECOM(4) = 0

CALL SDST0(ECOM, ESTAT, , )
ESTATI = ESTAT
ITMP = ESTATI .AND. MASK(1)
If(ITMP .EQ. 0) GOTO 100

Having EOT

If(IOM) 10, 30, 20

If(ITMP .EQ. 0) GOTO 100
If(IOM) 1002, IDRV
1002 FORMAT(' EOT ON DRIVE ',11)
IEOT = -1
RETURN
125 IF(IOM) 50, 35, 70

Error exit return

ISTATI = -1
RETURN
END
Interactive Spectral Analysis Program : - MPFANL

MPFANL is totally interactive, and it expects the input data to be in a disc file in the processing system internal format.

Plots are produced on the electrostatic plotter by running RASM after this program terminates.

Input Parameters

NSAMP(I4) ... Number of samples/trace
NCHAN(I2) ... Number of channels to analyse
ICHAN(I), I=1,NCHAN(I3) ... Channel numbers of those to be analysed
TSEC(F10.0) ... Sampling period of data in seconds.
FNAMR(3A4) ... Name of data file containing the data to be analysed
THIS IS A SPECTRAL ANALYSIS PROGRAM

READ IN ICHR=IGETC()

IF(IFETCH(DEV).NE.0)STOP 'FETCH ERR'

IF(LOOKUP(ICHR,FSPECR).LT.0)STOP 'LOOKUP ERR'

NWDS=2*NSAMP

NBLKS=NSAMP/128

PROCESS FIRST CHANNEL

NBLKST=(ICHAN(1)-1)*NBLKS+1

IF((ICHAN(1)).LT.0)STOP 'READ ERR'

CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(1))

IF(ICHAN.EQ.1)GOTO 10

DO MOST OF CHANNELS

DO 20 J=2,NCHAN

NBLKST=(ICHAN(J)-1)*NBLKS+1

IF((ICHAN(J)).LT.0)STOP 'READW ERR'

IFLAG=1

CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(J))

20 CONTINUE

IEND
SUBROUTINE SPEC(AVAL, INUM, I PLOT, TSEC, ICHAN)

DIMENSION AVAL(INUM)

DATA MASK/01£1421£1/

C**************************************************~**

C AVAL=SOURCE DATA TO BE PADDED TO NEAREST 2**N
C INUM=DIMENSION AVAL
C I PLOT=DISPLAY CONTROL
C TSEC=SAMPLE PERIOD
C ICHAN=CHANNEL NO BEING ANALYSED
C THIS PROGRAM USES PEPLIB AND FPSLIB.
C THE DISPLAY OF AMPLITUDE SPECTRUM IS NORMALISED
C TO UNITY WITH THE INDICATED SCALING FACTOR.
C PHASE SPECTRUM STILL CONTAINS THE SAMPLING RAMP
C EXPAND INUM TO 2**N

C*****************************************************************************

IF(I PLOT.EQ.0)GOTO 999

N=2
ICNTRL=0

100 CONTINUE

IF(N.GE.INUM) GOTO 99
N=N*2
GOTO 100

N=LOWEST 2**N GT INUM

99 IF(N.GT.4096) STOP 'AP OVERFLOW ON SPECTRUM'

THIS FOR AP'S SAKE

DELF=1.0/(N*TSEC)
DELF IS RETURNED TRANSFORM FREQUENCY

PAD OUT AVAL WITH ZEROS

IF(INUM.EQ.N) GOTO 1011
ISTART=N+1
DO 101 I=1,N
IF(I.EQ.0)AVAL(I)=0.0
FLIM=MAX TRANSFORM FREQUENCY

SEEK FREQUENCY DECADE OF PLOT

FBASE=0.1
IBASE=-1

102 CONTINUE

IF(FLIM.LE.FBASE) GOTO 103
FBASE=FBASE*10.
IBASE=IBASE+1
GOTO 102

103 FBASE=FBASE*0.1
IBASE=IBASE-1
FMAX=FLIM/FLIM
IFMAX=IFIX(FMAX)
FMAX=FLOAT(IFMAX)

FMAX=NEXT HIGHEST INTEGER DECADE=1 TO 9
FORM PLOTTING MASK

VSIZE=9.0
ASPECT=0.75
XSIZE=VSIZE*ASPECT
XOFST=XSIZE/20.0
YOFST=VSIZE/20.0
CSIZE=VSIZE/60.0
DELX=XSIZE/100.0
IF( ICNTRL.NE.0 ) DELX=XSIZE/36.0
V=FIX( Y )
DELV=VSIZE/IV
IF( IPLOT.GT.0 ) CALL PLOTS(X,Y,Z)
IF( IPLOT.LE.0 ) CALL PLOT(X,Y,Z,-999)
CALL SETMSG(0)
NX=100
IF( ICNTRL.NE.0 ) NX=36
CALL GRID( XOFST, YOFST, NX, DELX, IV, DELV, MASK )
CALL PLOT( XOFST, YOFST, 3 )
CALL NEWPEN(3)
X=XOFST+XSIZE
V=YOFST+VSIZE
CALL PLOT( X, YOFST, 2 )
CALL PLOT( X, Y, 2 )
CALL PLOT( XOFST, Y, 2 )
CALL PLOT( XOFST, YOFST, 2 )
CALL NEWPEN(1)
CALL PLOT( X, YOFST, -3 )
CALL PLOT( X, 0.0, 3 )
CALL PLOT( X, VSIZE, 2 )
CALL NUMBER( X, Y, CSIZE, XNUM, 9.0, -1 )
XNUM=XNUM+DELX
X=XINC
Y=YINC+0.1
YINC=VSIZE/IV
Y=IV+1
YINC=0.0
X=1.5*CSIZE
XINC=XSIZE
DO 105 I=1,IY
CALL PLOT(Ø,Ø,Y,3)      CALL PLOT(XINC,Y,2)
CALL NUMBER(X,Y,CSIZE,YNUM,Ø,Ø,-1)
YNUM=YNUM+1
105 Y=Y+YINC
SSIZE=CSIZE*1.5
X=X+2.Ø*SSIZE
Y=YSIZE*Ø.5-7.Ø*SSIZE
CALL SYMBOL(X,Y,SSIZE,12HFREQUENCY*Ø,Ø,Ø,12)
Ø=Ø*12.Ø*SSIZE
X=X-SSIZE
DELX=FLOAT(IBASE)
CALL NUMBER<X,V,CSIZE,DELX,Ø,Ø,-1)
V=Ø.65*YSIZE
X=X+SSIZE
SSIZE=SSIZE*1.33
CALL NEWPENCIL
111 IF(INCTRL.EQ.B) CALL SYMBOL<X,V,SSIZE,15ENERGY SPEC TRUM,Ø,Ø,15)
111 IF(INCTRL.NE.B) CALL SYMBOL<X,V,SSIZE,15PHASE SPECTRUM,Ø,Ø,15)
FEND=FMAX*FBASE
DFPLOT=VSIZE*DELF/FEND
IF(INCTRL.NE.B) GOTO 1051
DPLOT=PLLOT UNITS PER FREQUENCY SAMPLE
Do AP processing
MN=N+2
NN=NN/2
LN=LN+1
NN=NN+1
CALL APCLR
CALL APPUT(AVAL,Ø,NX,2)
CALL APWD
CALL VMOV(N,1,8189,1,3)
CALL RFFT(Ø,N,1)
CALL RFFTS(Ø,N,3,1)
NN=COMPLEX TRANSFORM LENGTH
CALL POLAR(Ø,2,Ø,2,NN)
CALL MAXVC(Ø,2,NN,NN)
CALL VFILL(NN,LN,1,NNN)
CALL VDIV(NM,1,Ø,2,Ø,2,NN)
SCALE SPECTRUM FOR PLOTTING
CALL VSMUL(Ø,2,8189,Ø,2,NN)
SCALE PHASE FOR PLOTTING
CALL VSMUL(1,2,8190,1,2,NN)
DO AP processing
CALL VSADD(1,2,8191,1,2,NN)
CALL APWR
CALL APG(T,AVAL,0,LN,2)
CALL APVD
C) AVAL(1,LN)=AMPLITUDE:PHASE
C) AVAL(LN)=SCALING FACTOR FOR AMPLITUDE
C) PLOT AMPLITUDE SPECTRUM
1051 CALL PLOT(0.,0.,3)
1052 CALL NEWPEN(3)
1053 Y=0.
1054 ISTART=1
1055 IF(INCTRL.NE.0.) ISTART=2
1056 DO 1057 I=ISTART,MN,2
1057 CALL PLOT(AVAL(I),Y,2)
1058 Y=Y+DFPLOT
1059 IF(INCTRL.NE.0.) GOTO 206
C) MARK IN SCALING FACTOR
1060 CALL NEWPEN(1)
1061 X=4.5*CSIZE
1062 Y=0.
C) FLOAT SCALING FOR DISPLAY
1063 ASCALE=1.0
1064 BSCALE=0.
1065 INDEX=0.
1066 IF(AVAL(LN).LT.1.0) ASCALE=1.0
1067 IF(AVAL(LN).GT.1.0) ASCALE=0.1
1068 IF(AVASCALC.EQ.1.0) INDEX=-1.0
1069 IF(AVASCALC.EQ.0.1) INDEX=1.0
1070 CONTINUE
1071 IF(AVAL(LN).GE.1.0.AND.AVAL(LN).LT.10.0) GOTO 207
1072 CALL PLOT(AVAL(LN)*ASCALE,ALN)
1073 BSCALE=BSCALE+INDEX
1074 GOTO 206
1075 CALL SYMBOL(X,Y,CSIZE,21HSCALING FACTOR=
1076 Y=Y+15.0*CSIZE
1077 CALL NUMBER(X,Y,CSIZE,AVAL(LN),90.0,1)
1078 Y=Y+6.0*CSIZE
1079 DELX=0.5*CSIZE
1080 X=X-DELX
1081 CALL NUMBER(X,Y,DELX,BSCALE,90.0,-1)
1082 ICNTRL=1
1083 CALL PLOT(B.,B.,-999)
1084 CALL SYMBOL(X,V,CSIZE,'CHANNEL NUMBER= ',9B.B,17>
1085 V=V+16.0*CSIZE
1086 DELX=FLOAT(ICCHAN)
1087 CALL NUMBER(X,Y,CSIZE,RCHAN,90.0,-1)
1088 ICNTRL=0
1089 IF(IPLOT.EQ.0) CALL PLOT(B.,B.,999)
1090 RETURN
END
Velocity Analysis :- MPVEL

Input file.....DK2:ANINV.DAT
Output file....DK2:ANOUTV.DAT

Input Parameters

READ(1,1000)L,NCHAN,NSTART,M

1000 FORMAT(I5)

L........Number of Samples per trace
NCHAN....Number of Channels in gather
NSTART...Starting sample number, from time zero
M........Level of interpolation carried out on data

READ(1,1200)FBUF

1200 FORMAT(12X,3A4)

FBUF.....Input data file name

READ(1,1100)XSTART,XSTEP,FSAMP

1100 FORMAT(6F10.0)

XSTART...Shot/First receiver offset
XSTEP.....Receiver spacing
FSAMP.....Sampling frequency, samples/millisecond
**READ(1,1100)T01,T02,TOSTEP,TGATE**

T01......Start time for analysis, milliseconds
T02......End time for analysis, milliseconds
TOSTEP...Gate step size, milliseconds
TGATE....Semblance calculation gate size, milliseconds

**READ(1,1100)VSTEP,V1,V2MIN,V2MAX,VICPT,VGRAD**

VSTEP....Velocity step in semblance calculation
V1.......Start velocity in analysis
V2MIN....Minimum value of end velocity
V2MAX....Maximum value of end velocity
VICPT....Intercept on V-axis of line joining V2MIN and V2MAX
VGRAD....Gradient of line joining V2MIN and V2MAX
DIMENSION NMDFS(4096),TWSQ(512),NV(512),VINSQ(170),SMBLCE(170),
TAB(1024),S1(1024),XSQ(24),
2JSTCH(24),JST(24),NVDS(24),NEL(24),JBLK(24),CONST(7),FBUF(3),
LOGICAL*1 SW
REAL*8 FSPEC
DATA K1,K2,KNCHN,KBLK,KBLKIN,KADD2,KADD1,KHF,KFS,NBUF
/1/1.2,8185,8186,8187,8188,8189,8190,8191,1.024/
DATA DEV/3RDK /
C ASSIGN INPUT AND OUTPUT CHANNELS
CALL ASSIGN(1,'DK2:ANINV.DAT',13)
CALL ASSIGN(2,'DK2:ANOUTV.DAT',14)
IF(IFETCH(DEV).NE.0)STOP 'FETCH ERROR'
IDCH=IGETC(
C READ IN REQUIRED INPUT PARAMETERS
READ(1,100) L,NCHAN,NSTART,M
READ(1,120) FORMAT(415)
READ(1,110) FORMAT(10F10.0)
READ(1,100) T01,T02,T0STEP,TGATE
READ(1,110) VSTEP,V1,V2MIN,V2MAX,VICPT,VGRAD
C VALIDATE DATA AND CALCULATE REQUIRED ARRAYS AND CONSTANTS.
C FORM ARRAY JSTCH CORRESPONDING TO THE BEGINNING BLOCK
C FOR EACH TRACE ON DISK.
JSTCH(1)=1
NREAD=L*M/128
X=XSTART
XSQ(1)=X**2
DO 160 JCHAN=2,NCHAN
JSTCH(JCHAN)=JSTCH(JCHAN-1)+NREAD
X=X+XSTEP
XSQ(JCHAN)=X**2
160 CONTINUE
C CALCULATE FSAMP,THE SAMPLING FREQUENCY AFTER INTERPOLATION
FSAMP=FLOAT(M)*FSAMP
C CALCULATE TSAMP THE SAMPLE INTERVAL
TSAMP=1.0/FSAMP
C CALCULATE TSTART,THE BEGINNING TIME
TSTART=NSTART*TSAMP
C CALCULATE NSTM THE INITIAL SAMPLE NUMBER AFTER INTERPOLATION.
NSTM=M*NSTART
C TRUNCATE T0STEP TO BE INTEGRAL MULTIPLE OF TSAMP
T0STEP=TSAMP*IFIX(T0STEP*FSAMP)
C ROUND TGATE TO BE EVEN INTEGRAL MULTIPLE OF TSAMP
NGATE=IFIX(TGATE*FSAMP-1.0)/2*2
TGATE=FLOAT(NGATE)*TSAMP
NGATE=NGATE+1
NGTM=M*(NGATE/2)
C ENSURE THAT TB1 IS A LEAST TSTART+TGATE/2
T01=AMAX1(T01,TSTART+0.5*TGATE)
C ROUND T01 UP TO BE INTEGRAL MULTIPLE OF TSAMP
T01=TSAMP*(IFIX(T01/TSAMP)+1)
C TRUNCATE T02 IF NECESSARY SO THAT THERE IS SUFFICIENT
DATA TO DO VELOCITY ANALYSIS AT T02
T02MAX=SQRT((TSTART+(L-1)*TSAMP-0.5*TGATE)**2-
1*XSQ(NCHAN)/V1**2)
T02=AMIN1(T02,T02MAX)
CALCULATE NT0, THE NUMBER OF TWO WAY TIMES CONSIDERED
NT0=IFIX((T02-T01)/TSTEP)+1
T02=T01+(NT0-1)*TSTEP
IF(NT0.LE.0) STOP 'NT0 NOT POSITIVE'
FORM ARRAYS T0SQ (SQUARED TWO WAY TIME) AND NV (NUMBER
OF VELOCITY POINTS AT EACH TWO WAY TIME)
T0J=T01
DO 20 JT0=1,NT0
20 T0SQ(JT0)=T0J**2
V2=AMAX1(V2MIN,V2)
V2=AMIN1(V2,V2MAX)
NV(JT0)=IFIX((V2-V1)/VSTEP)+1
T0J=T0J+TSTEP
CONTINUE
VINSQ IS THE ARRAY OF INVERSE SQUARED VELOCITIES
NVMX=NV(NT0)
IF(NVMX.GT.170) STOP 'NVMX GT 170'
V=V1
DO 30 JV=1,NVMX
30 VINSQ(JV)=1.8/V**2
V=V+VSTEP
CONTINUE
YOU CALCULATE NVMX, AN UPPER BOUND ON THE SIZE OF THE
DATA WINDOW, IF THIS EXCEEDS NBUF PROGRAM STOPS.
NVMX=IFIX(FSAMP**2*(T0SQ(1)+XSQ(NCHAN)*VINSQ(1)+0.5)+
1-IFIX(FSAMP**2*(T0SQ(1)+XSQ(NCHAN)*VINSQ(NVMX)+0.5)+2*NGTM2+1
IF(NVMX.GT.NBUF-127) STOP 'NVMX TOO LARGE'
CONST IS ARRAY OF CONSTANTS USED IN AP
CONST(1)=1.8/FLOAT(NCHAN)
CONST(2)=128.
CONST(3)=1.8/128.
CONST(4)=FLOAT(NGTM2-NSTM+1)
CONST(5)=FLOAT(NGTM2+NSTM)
CONST(6)=0.5
CONST(7)=FSAMP
INITIALISE AP AND PLACE CONSTANTS IN TOP 7 LOCATIONS
CALL APINIT
CALL APWR
CALL APPUT(CONST,KNCHIN,7,K2)
CALCULATE THOSE ADDRESSES WHICH ARE CONSTANT IN NMO
AND SEQUENCE COMPUTATIONS.
IVINSQ=NCHAN+1
ISQ=NVMX
NEN=IVSQ+1
LOOKUP FILE TO BE READ FROM
THE PURPOSE OF NMO IS TO CALCULATE ALL THE NORMAL MOVEOUTS AND
THE REQUIRED RANGE OF VELOCITIES AT A GIVEN VERTICAL INCIDENCE
AND AUXILIARY INFORMATION NECESSARY TO CALCULATE THE SEMBLANCE FOR

OUTPUT ARGUMENTS AND METHOD OF COMPUTATION

NMOFS IS ARRAY CONTAINING ALL THE REQUIRED NORMAL MOVEOUT OFFSETS.
ALTHOUGH IT IS A LINEAR ARRAY IT IS USED IN A 'TWO DIMENSIONAL'
MANNER TO STORE ALL THE OFFSETS AS A FUNCTION OF BOTH SEISMIC
CHANNEL NUMBER AND VELOCITY. THUS THE FIRST NVD VALUES ARE
THE OFFSETS FOR CHANNEL 1, IN ORDER OF INCREASING VELOCITY, THE NEXT
NVD VALUES THE OFFSETS FOR CHANNEL 2 AND SO ON. THE OFFSETS
ARE ROUNDED TO THE NEAREST (INTERPOLATED) SAMPLE AND, FOR A GIVEN
SEISMIC CHANNEL, ARE RELATIVE TO THE BEGINNING OF THE DATA WINDOW
THAT IS TO BE TAKEN INTO THE AP TO DO THE PARTIAL SEMBLANCE
CALCULATIONS FOR THAT CHANNEL, AT TIME JT0.

AT A TIME JT0, AND FOR A CHANNEL JCHAN THE DATA WINDOW THAT IS
TAKEN INTO THE AP (IN ROUTINE SEMB) MUST COVER ALL TIMES FROM
THE CROSSING OF THE SHALLOWEST ARRIVAL TRAJECTORY (VELOCITY OF 
V(NVD) TO THE CROSSING OF THE STEEPEST ARRIVAL TRAJECTORY
VELOCITY OF V(NV1(1))). IN ADDITION A HALF GATEWIDTH OF DATA IS
REQUIRED AT EITHER END.

SET UP INITIAL ADDRESSES

SET UP INITIAL ADDRESSES FOR RESULTS
IRES = INMO
IRNV1 = IRES + NVD - 1
I10 = 11
I20 = 12
CALL APWD
C ITERATE THROUGH CHANNELS
DO 40 JCHAN = 1, NCHAN
C FORM KSQ(JCHAN)*VINSQ(JV) + T0SQ(JT0) IN (RES, NVD)
C CALL VMSA(IVINSEQ, K1, JCHAN, 5, IRES, K1, NVD)
C FORM SQRT(IRES, NVD)
C CALL VSQRT(IRES, K1, IRES, K1, NVD)
C FORM INT(FSAMP*SQRT(T0SQ(JT0) + KSQ(JCHAN)*VINSQ(JV)) + 0.5) USING FSAMP
C IN KFS AND 0.5 IN KHF
C CALL VINT(IRES, K1, KFS, KHF, IRES, K1, NVD)
C ADD IADD WITH VALUE FROM IRNV1 AND NEGATE IT
C CALL VFILL(IRNV1, IADD, K1, K1)
C CALL VNEG(IADD, K1, IADD, K1, K1)
C FILL I1J WITH VALUE FROM IRNV1 (VELOCITY V2)
C AND I2J WITH VALUE FROM IRES (VELOCITY V1)
C CALL VADD(IRES, I1J, K1, K1, K1)
C ADD VALUE IN IADD TO (IRES, NVD), I.E. SUBTRACT MOVEOUT FOR VELOCITY V2
C TO GIVE NMOFS
C CALL VSADD(IRES, K1, IADD, IRES, K1, NVD)
C REDEFINE ADDRESSES SO THAT RESULTS FOR NEXT CHANNEL FOLLOW ON
C IRES = IRES + NVD
C IRNV1 = IRNV1 + NVD
I10 = I10 + 1
I20 = I20 + 1
40 CONTINUE
C ADD -NSTM - NGTM2 TO (I1, NCHAN) USING VALUE IN KADD1
C CALL VSADD(I1, K1, KADD1, I1, K1, NCHAN)
C ADD -NSTM - NGTM2 TO (I2, NCHAN) USING VALUE IN KADD2
C CALL VSADD(I2, K1, KADD2, I2, K1, NCHAN)
C SUBTRACT (I1, NCHAN) FROM (I2, NCHAN) TO FORM (I3, NCHAN)
C CALL VSUB(I1, K1, I2, K1, I3, K1, NCHAN)
C MULTIPLY (I1, NCHAN) BY 1.8/NSBLK FROM KBLKIN TO FORM (I4, NCHAN)
C CALL VSMUL(I1, K1, KBLKIN, I4, K1, NCHAN)
C TRUNCATE (I4, NCHAN) TO INTEGER VALUES
C CALL VINT(I4, K1, I4, K1, NCHAN)
C MULTIPLY (I4, NCHAN) BY -NSBLK FROM KBLK TO FORM (I5, NCHAN)
C CALL VSMUL(I4, K1, KBLK, I5, K1, NCHAN)
C ADD (I5, NCHAN) TO (I1, NCHAN)
C CALL VADD(I1, K1, I5, K1, I1, K1, NCHAN)
C ADD 1.0 FROM TM TO (I1, NCHAN)
C CALL VTADD(I1, K1, 2.0, I1, K1, NCHAN)
C ADD (I5, NCHAN) TO (I2, NCHAN) AND MULTIPLY BY 2.0 FROM TM
C CALL VADD(I2, K1, I5, K1, I2, K1, NCHAN)
C CALL VSMUL(I2, K1, 2.0, I2, K1, NCHAN)
C FIXED VALUES FROM INMO TO IS PREPARATORY T
C transfer TO HOST
C CALL VFIX(INMO, K1, INMO, K1, NTOT)
C transfer TO HOST ARRAYS
CALL APWR
CALL APGET(NMOFS,INMO,NVNC,K1)
CALL APGET(JST,I1,NCHAN,K1)
CALL APGET(NWDS,I2,NCHAN,K1)
CALL APGET(NEL,I3,NCHAN,K1)
CALL APGET(JBLK,I4,NCHAN,K1)

SET UP INITIAL ADDRESSES FOR SEMBLANCE CALCULATIONS
ISEMB=IEN+NVD
ISEMB1=ISEMB-1
ISMAX=ISEMB+NVD
ISMAX1=ISMAX1+1
ISTK=ISMAX+2
CALL APWD
C CLEAR SECTIONS OF AP MEMORY THAT WILL HAVE DATA ADDED.
CALL VCLR(IEN,K1,NVNG)
NVNG=NVNG*NGATE
CALL VCLR(JSTK,K1,NVNG)
CALL VCLR(JST,JCHAN,1)
IF<IREAD<NWDS<1,J,BLK<1,JCHAN1.IDCH>,LT.0>STOP'READ ERROR'
SW=.TRUE.,READ FROM S$J INTO S1
SW=.FALSE.,READ FROM S1 INTO S$
SW=.TRUE.
C ITERATE THROUGH CHANNELS
JNMO=1
DO 200 JCHAN=1,NCHAN
JCHAN1=JCHAN+1
C TRANSFER DATA FROM BUFFER TO AP
CALL APWR
CALL IWAIT(JCHAN)
IF(SW) CALL APPUT(S$,JST,JCHAN),NEL,JCHAN1,K2)
1.1 IF(.NOT.SW) CALL APPUT(S1,JST,JCHAN),NEL,JCHAN1,K2)
IF(JCHAN1.GT.NCHAN)GOTO200
C TRANSFER NEW DATA FROM DISK TO BUFFERS
IF(SW) IERR=IREAD(NWDS,JCHAN1),S$1,JSTCH,JCHAN1,JBLK,JCHAN1,JCHAN1,JCHAN1,1
1.3 IF(.NOT.SW) IERR=IREAD(NWDS,JCHAN1),S$1,JSTCH,JCHAN1+JBLK,JCHAN1,JCHAN1,1
IF(IERR.LT.0)STOP'READ ERROR'
SW=.NOT.SW
C NOW DO SEMBLANCE CALCULATIONS ON DATA WITHIN AP
CALL APWD
DO 100 JV=1,NVD
C ADD APPROPRIATELY MOVED OUT GATE ONTO STACK USING ONLY EVERY MTH POINT
CALL VADD(JSTK,K1,NMOFS,JNMO),M,JSTK,K1,NGATE)
C FORM SUM OF SQUARES OF ELEMENTS ADDED ONTO STACK
CALL SVESQ(NMOFS,JNMO),M,ISSQ,NGATE)
1.7 ENJ=ISSQ+JV
C DEFINE JSTK SO THAT NEXT STACK FOLLOWS ON
ISTK=ISTK+NGATE
I. NOW ACCUMULATE SEMBLANCE
   DO 390 JV=1,NVD
   C FIND MEAN SUM OF SQUARES FOR EACH STACK
   CALL SVESS(ISTK,K1,ISEMB1+JV,NGATE)
   ISTK=ISTK+NGATE
   CONTINUE
C DIVIDE MEAN STACK ENERGIES BY SUMMED ENERGY
   CALL VDIV(IEN,K1,ISEMB,K1,ISEMB,K1,NVD)
C DIVIDE BY NCHAN TO NORMALISE SEMBLANCE AND ITS MAX VALUE
   NV1=NVD+1
   CALL VSMUL(ISEMB,K1,KNCHI,ISEMB,K1,NV1)
C WRITE OUT SEMBLANCE AND ITS MAXIMUM VALUE TO SMBLCE
   CALL APWR
   CALL APGET(SMBLCE,ISEMB,NV1,K2)
C WRITE OUT ADDRESS OF MAXIMUM VALUE TO IMAX
   CALL APGET(IMAX,ISMAX,K1,K1)
C CALCULATE JVSMX, VALUE OF JV FOR WHICH SEMBLANCE IS MAXIMUM
   JVSMX=IMAX-ISEMB1
C WRITE OUT SEMBLANCE TO OUTPUT DEVICE 2
   WRITE(2) NVD,JVSMX,(SMBLCE(JV),JV=1,NVD)
   CONTINUE
   CALL CLOSE(IDCH)
   STOP
   END
NMO corrected gathers :- MCDP

Input File......DK2:ANCDP.DAT

Input Parameters

READ(1,1000)L,NCHAN,NSTART,NLYR,M

1000 FORMAT(12I5)

L........Number of samples per trace
NCHAN....Number of channels per gather
NSTART...Sample number of start of trace
NLYR.....Number of time/velocity pairs
M........Level of interpolation

READ(1,1100)XSTART,XSTEP,FSAMP

1100 FORMAT(6F10.0)

XSTART...Shot-First receiver offset
XSTEP....Receiver spacing
FSAMP....Sampling rate in samples/millisecond

READ(1,1200)(TOLYR(I),VLYR(I),I=1,NLYR)

1200 FORMAT(2F10.0)

TOLYR....Two-way travel time milliseconds
VLYR.....RMS velocity down to this two-way travel time
READ(1,1000)INDEN,INDMUT

INDEN....Scaling flag
   <0 - no scaling
   =0 - Unit RMS energy
   >0 - Inverse energy scaling, diversity stack

INDMUT....Mute flag
   >0 - apply mute
   <0 - no mute

If INMUT is set for a mute option then read the following

READ(1,1000)(MUTE(I),I=1,NCHAN)

MUTE.....Sample position to mute down to for each channel

READ(1,1300)FSPECR

1300 FORMAT(3A4)

FSPECR....File containing data for input

READ(1,1300)FSPECW

FSPECW....Output file, to contain NCHAN NMO corrected channels

and 1 stacked trace
C-------------------

REAL & FSPECR, FSPECW

VIRTUAL T$SQ(2048), VINSQ(2048)

DIMENSION MUTE(24), XSQ(24), FBUF(3)

COMMON /LAYER/TOLYR(99), VLYR(99)

DATA DEV/1RRK /

CALL ASSIGN('DK1:ANCDP.DAT',13)

IF(IFETCH(DEV).NE.0) STOP 'FETCH ERROR'

IDCH=IGETC()

IDCH1=IGETC()

READ(1,1000) NCHAN, NSTART, NLYR, M

READ(1,1200) XSTART, XSTEP, FSAMP

READ(1,1200) TOLYR(I), VLYR(I), I=1,NLYR

READ(1,1200) IDEN, INDMUT

IF(INDMUT.GE.0) READ(1,1000)(MUTE(I), I=1,NCHAN)

READ(1,1300) FBUF

READ(1,1400) FSPECW

IF(LOOKUP(IDC1,FSPECW).LT.0) STOP 'LOOKUP ERROR'

NBLKS=L*(NCHAN+1)/128+1

IF(ENTER(IDC1,FSPECW,NBLKS).LT.0) STOP 'ENTER ERROR'

L2INT=2

DO 5 K=1,1000

5 CONTINUE

X=0.1*XSTART

DO 20 JCHAN=1,NCHAN

XSQ(JCHAN)=X**2

X=X+XSTEP

CONTINUE

TSAMP=1./FSAMP

T0=NSTART*TSAMP

DO 30 JT0=1,L

30 CONTINUE

CALL SETAP(L2INT,M)

CALL INVSQ(FSAMP,NSTART,NLYR,L,VINSQ)

CALL CDP(L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE, IDEN, INDMUT, IDCH, IDCH1, T$SQ, VINSQ, XSQ)

STOP

END
SUBROUTINE INVSQ(FSAMP,NSTART,NLYR,L,VINSQ)

USES INPUT DATA TO GENERATE ARRAY OF INVERSE SQUARED VELOCITIES
VELOCITIES ARE CONSTANT UP TO FIRST REFLECTOR AND BEYOND
LAST AND ARE LINEARLY INTERPOLATED IN BETWEEN.

VIRTUAL VINSQ(2048)

COMMON /LAYER/T0LYR(99),VLYR(99)
N1=1
N2=IFIX(FSAMP*T0LYR(1))-NSTART
VINSQ1=1.0/VLYR(1)**2
DO I1=N1,N2
VINSQ(I1)=VINSQ1
CONTINUE
IF(NLYR.EQ.1) GOTO 40
DO J2=2,NLYR
N1=N2+1
N2=IFIX(FSAMP*T0LYR(J2))-NSTART
DELV=(VLYR(J2)-VLYR(J-1))/(N2-N1+2)
V=VLYR(J-1)
DO I1=N1,N2
VINSQ(I1)=1.0/V**2
V=V+DELV
CONTINUE
CONTINUE
40 N1=N2+1
L2=20
VINSQN=1.0/VLYR(NLYR)**2
DO 5O I1=N1,N2
VINSQ(I1)=VINSQN
CONTINUE
RETURN
END

SUBROUTINE SETAP(L2INT,M)

USES TECHNOLOGY TO COMPUTE EXPONENTIAL TABLE IN D1
APPUTS 1.0 IN ID
COMMON /KONST/ K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
IIA,IB,IC,ID,ITOP
CALL APINIT
CALL APADD
CALL APWD
FORM VECTOR RAMP AND TAKE SIN AND COS OF IT
CALL VTSMUL(K9,K1,L2121)
CALL VRAMP(K9,K1,IB,K1,L2121)
CALL VCVS(K1,K9,IB,K1,L2121)
CALL VCMK(IN,IB,K1,K1,2569)
RETURN
END
SUBROUTINE CDP(L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,
        IINDEX,INDMUT,IDCH,TSQ,VINSQ,XSO)

VIRTUAL SEISM(32767),TSQ(2048),VINSQ(2048)

DIMENSION INDEX(2048),MUTE(24),STACK(2048),
        IDUM(16),CONST(16),FSTAP(20),XSO(24)

COMMON /KONST/ K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
                  IA,IB,IC,ID,ITOP

C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS

IB=IB+1
IC1=IC+1
IC2=IC+2
ID1=ID+1
L1=L+1
L2121=L2INT/2-1
CONST(2)=M*FSAMP
CONST(3)=.6
CONST(4)=-M*NSTART
CONST(5)=.6
CONST(6)=L*M
CONST(7)=L+1
CONST(8)=.6/M
CONST(9)=(L+1)*M-1
CONST(10)=L/128
FSTD=ADGET(SEISM(1))
FTSQ=APGAD(TSQ(1))
FVINSQ=APGAD(VINSQ(1))
ISTART=1
DO 58 K=1,M
     FSTAP(K)=APGAD(SEISM(ISTART))
      ISTART=ISTART+L1
58 CONTINUE

C SET SEISM(L+1) TO 0.0 TO COPE WITH TIME OVERFLOW AND CLEAR A IN AP
SEISM(L)=0.0
CALL VCLR(I1A,K1,L)

C ITERATE THROUGH CHANNELS
DO 500 JCHAN=1,NCHAN

C READ IN TRACE SEISM FROM UNIT 2
CALL IWAIT(IDCH)
IF(IREAD(IDCH,JBLK,FSTD).LT.0)STOP 'READA ERROR'

C COMPUTE INDEX ARRAY
CONST(1)=XSO(JCHAN)
CALL APWR

C TRANSFER (K1,K10) TO HOST DUMMY ARRAY DUM AND REPLACE BY CONST
CALL APGET(DUM,K1,K10,K2)
CALL APPUT(CONST,K1,K10,K2)

C TRANSFER TSQ TO B AND VINSQ TO IC1
CALL APPUTA(IB,L,K2,FTSQ)
CALL APPUTA(IC1,L,K2,FVINSQ)

C FORM XSO(JCHAN)*VINSQ IN C1
CALL VSMULCIC1,K1,K1,IC1,K1,L1 
C FORM SQRT((TQS+XSQ*(JCHAN)*VINSQ)}*8.5)-NSTART*M IN B 
CALL VMSA(1B,K1,K2,K3,1B,K1,L1) 
CALL VSADDC(1B,K1,K4,1B,K1,L1) 
C CLIP B BETWEEN 0.0 AND L*M 
CALL VCLIP(1B,K1,K5,K6,1B,K1,L1) 
C MULTIPLY B BY L+1,ADD 1.0 AND PLACE RESULT IN C1 
CALL VMSA(1B,K1,K7,K8,IC1,K1,L1) 
C MULTIPLY B BY 1.0/M AND TAKE INTEGER PART 
CALL VSMUL(1B,K1,K9,1B,K1,L1) 
C MULTIPLY B BY (L+1)*M-1 
CALL VSMUL(1B,K1,K10,1B,K1,L1) 
C SUBTRACT B FROM C1 AND PUT RESULT IN B 
CALL VSUB(1B,K1,IC1,K1,1B,K1,L1) 
C FIX B AND TRANSFER BACK AS HOST ARRAY INDEX 
CALL VFIX(1B,K1,1B,K1,L1) 
CALL APWR 
CALL APGET(INDEX,1B,L1,K1) 
C TRANSFER DUM BACK TO K1 
CALL APPUT(DUM,1B,L1,K1) 
CALL APWD 
C HAVING COMPUTED INDEX ARRAY NOW START INTERPOLATION OF TRACE 
C CLEAR B IN AP 
CALL VCLR(1B,K1,L2INT) 
C TRANSFER TRACE TO B 
CALL APWR 
CALL APPUTA(1B,L,K2,FSTAP(1)) 
CALL APWD 
C FIND MEAN VALUE OF TRACE AND PLACE IN AP(ITOP) 
CALL MEANV(1B,K1,ITOP,L1) 
C SUBTRACT MEAN FROM TRACE 
CALL VNEG(1B,K1,ITOP,K1,K1) 
CALL VSADD(1B,K1,ITOP,1B,K1,L1) 
C IF INDEM IS NEGATIVE, NO SCALING OF TRACE 
C IF INDEM IS ZERO SCALE TO UNIT RMS VALUE 
C IF INDEM IS POSITIVE USE INVERSE ENERGY SCALLING (DIVERSITY STACK) 
IF(INDEM.LT.0) GOTO 180 
CALL RMSQV(1B,K1,ITOP,L1) 
IF(INDEM.GT.0) CALL VSQITOP(1B,K1,ITOP,K1) 
CALL VDIVITOP(1B,K1,1B,K1,ITOP,K1,K1) 
CALL VSMUL(1B,K1,ITOP,1B,K1,L1) 
C IF INDMUT IS NEGATIVE USE NO MUTING 
IF(INDMUT.LT.0) GOTO 280 
CALL VCLR(1B,K1,MUTE(ICHAN)) 
C TRANSFER MODIFIED TRACE BACK TO SEISM 
CALL APWR
ENTRY FROM 2 TO M
CALL APGETA(1B, L, K2, FSTAP(K))
CALL APWR
CONTINUE
TRANSFER REQUIRED ELEMENTS OUT OF SEISM AND PLACE IN STACK
DO 63 K = 1, L
STACK(I) = SEISM(INDEX(I))
CONTINUE
TRANSFER STACK INTO BI AND ADD TO STACK IN A
CALL APWR
CALL APPUT(STACK, IB1, L, K2)
CALL APWDO
IF (WHITE(2L, STACK, JBLK, IDCH1).LT.0) STOP 'WRITE ERROR'
JBLK = JBLK + NBLKTR
CALL ADDC(IA, K1, IB1, K1, IA, K1, L)
CONTINUE
TRANSFER INTO 1.B/NCHAN
EXEC:NV=1.8/NCHAN
CALL APWR
CALL ASPUT(FNCINV, ITOP, K1, K2)
CALL APWDO
EXEC:... 28.07
CALL VSMUL(IA, K1, ITOP, IA, K1, L)
CONTINUE
TRANSFER SCALED STACK BACK TO STACK
CALL APWR
CALL WAIT(IDC1)
CALL APGETI(STACK, IA, L, K2)
CALL APWDO
IF (WHITE(2L, STACK, JBLK, IDCH1).LT.0) STOP 'WRITE ERROR 2'
CALL (VALY, IDCH1)
CALL CLOSEC(IDCH1)
RETURN
CMP Stacking : MPSTAK

Input file........DK1:MPSTAK.DAT

Log file..........DK1:MPSTAK.LOG

Input Parameters

READ(1,1000)NFILES,NVEL,L,NCHAN,NSTART,N2LYR,M

1000 FORMAT(12I5)

NFILES........Number of files to be stacked
NVEL..........Number of velocity functions to be used in stack
L...........Number of samples per channel
NCHAN........Number of channels per gather
NSTART........Starting sample number
N2LYR.........Number of Time/Velocity pairs in first velocity function
M............Level of interpolation

READ(1,1000)TPDRR,TPDRW,INTSW

TPDRR........Input tape drive
TPDRW.........Output tape drive
INTSW.........Interpolation switch

0 - no velocity function interpolation
1 - linear interpolation between velocity functions
READ(1,1000)(IVELAN(I),I=1,NVEL)

IVELAN...File positions at which velocity functions are defined

READ(1,1100)XSTART,XSTEP,FSAMP

1100 FORMAT(3F10.0)

XSTART...Shot/First receiver offset
XSTEP....Receiver spacing
FSAMP....Sampling rate in samples per millisecond

READ(1,1200)(T02LYR(I),V2LYR(I),I=1,N2LYR)

1200 FORMAT(2F10.0)

T02LYR...Two-way travel time in milliseconds
V2LYR....RMS velocity at the above two-way travel time

READ(1,1000)INDEN,INDMUT,INFLG,OUTFLG

INDEN....Scaling flag
  <0 - no scaling
  =0 - Unit RMS energy scaling
  >0 - Inverse energy scaling:- Diversity stack

INDMUT...Mute flag
  >0 apply mute
  <0 no mute applied

INFLG....Input flag
0 - Input from tape  
1 - Input from disc  

OUTFLG...Output flag  
0 - Output to tape  
1 - Output to disc  

IF(INDMUT.GE.0)READ(1,1000)(MUTE(I),I=1,NCHAN)  

MUTE.....Sample position to mute down to for each channel  

READ(1,1300)FSPECR  

1300 FORMAT(3A4)  

FSPECR...If INFLG = 0 Temporary file for tape read  
INFLG = 1 Input files..1 to NFILES  

READ(1,1300)FSPECW  

FSPECW...If OUTFLG = 0 Temporary file for tape write  
OUTFLG = 1 Output files..1 to NFILES  

There are then NVEL velocity functions in the following format:-  

READ(1,1000)N2LYR  

READ(1,1200)(T02LYR(I),V2LYR(I),I=1,N2LYR)  

N2LYR.....Number of pairs in following analysis  
T02LYR...Two-way travel time in milliseconds
V2LYR....RMS velocity at the above two-way travel time
C===========================================================================
C MAIN PROGRAM
C===========================================================================

C THIS IS A STACK PROGRAM CAPABLE OF VELOCITY INTERPOLATION

C===========================================================================

REAL*8 FSPECR,FSPECW,FNAMR,FNAMO
VVIRTUAL TSQ(256),VINSCQ(256),FNAMR(100),FNAMO(100),
XTLYR(20),VTLYR(20),TQINT(20),VINT(20),IVELAN(100),
XT2LYR(20),VT2LYR(20)
DIMENSION MUTE(24),XSQ(24),FBUF(3),IMBLK(256)
INTEGER*2 OUTFLG
LOGICAL*1 TPDRR,TPDRW,STATUS,ITLEN,LBLK(512)
COMMON/STK/ L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,INDEN,
XINDMUT,INDCH,INDCH1,XSQ
DATA DEV/3RRK

C SET UP I/O CHANNELS AND READ IN CONTROL DATA

IF(ICDFN(25).NE.0)STOP 'CHANNELS FULL'
CALL ASSIGN(1,'DK1:MPSTAK.OAT',14)
CALL ASSIGN(2,'DK1:MPSTAK.LOG',14)
IF(IFETCH<DEV>.NE.0)STOP 'FETCH ERROR'
IEOTR=0
IEOTW=0
IDCH=2
IDCH1=21
READ<1,1>(T.0'.0')(T.0'2LVR<I>,V2LVR<I>, I=1,N2LVR>
READ<1,1>(T.0'.0')(MUTE(I), I= 1,NCHAN>
READ<1,1>(T.0'.0')(MUTE(I), I= 1,NCHAN>

C READ IN FILESPECS FOR INPUT DEPENDING ON
C IF IT IS FROM TAPE OF DISC

IF(INFLG.NE.0)GOTO 40
READ<1,1>(T.0'.0')FBUF
FORMAT(3A4)
CALL IRAD50(12,FBUF,FSPECR)
GOTO 50
DO 60 I=1,NFILES
READ<1,1>(T.0'.0')FBUF
CALL IRAD50(12,FBUF,FSPECR)
FNAMR(1)=FSPECR
60 CONTINUE
C
C READ IN OUTPUT FILE SPECS AGAIN THIS
C IS DEPENDANT ON IF IT GOES TO TAPE OR STAYS ON DISC.

C

50 IF(OUTFLG.NE.0)GOTO 70
04 READ(1,13000)FBUF
044 CALL IRA05(12,FBUF,FSPECW)
045 GOTO 80
3946 70 DO 90 I=1,NFILES
047 READ(1,13000)FBUF
048 CALL IRA05(12,FBUF,FSPECW)
045 FNAME(I)=FSPECW
3950 90 CONTINUE
C
C SET UP CONSTANTS ARRAYS
C
80 L2INT=2
085 DO 5 K=1,L
085 IF(L2INT.GE.L)GOTO 10
085 L2INT=2*L2INT
385 5 CONTINUE
087 X=XSTART
8858 DO 2 JCHAN=1,NCHAN
8859 XSQ(JCHAN)=X**2
8860 X=X+XSTEP
8861 CONTINUE
086 TSAMP=1./FSAMP
086 T0=LASTART*TSAMP
086 DO 30 J=1,L
086 T0SQ(JT0)=T0**2
086 T0=T0+TSAMP
086 30 CONTINUE
C
C START UP AP CONSTANTS AND CDP SUBROUTINE
C OUTSIDE OF LOOP
C
085 CALL SETAP(L2INT,M)
085 CALL CDP(T0SQ,VINSQ,0)
087 NBLKR=L*NCHAN/128+6
087 NBLKW=L/128+1

C
C START OF MAIN PROCESSING LOOP
C
087 DO 999 IFNUM=1,NFILES
087 IF(INFLG.NE.0)GOTO 100
C COME HERE IF INPUT FROM TAPE
087 IF(IENTER(IDCCH,FSPECR,NBLKR).LT.0)STOP 'ENTER ERR'
C EOT CHECK
087 ITRY=1
08888 IF(ITRY.GT.3)GOTO 210
IF(IEOTR.GE.0)GOTO 165
210 WRITE(7,166)TPDRR,IFIL
160 FORMAT(’ EOT ON DRIVE: ’,I2,’ FILE NO.: ’,I4)
161 WRITE(7,160)
162 IF(1601 FORMAT(’ ENTER DRIVE NUMBER: ’,I2,’ FILE NO.: ’,I4))
WRITE(7,160)
163 IF(IEOTR.GT.0)STOP’ EOT READ TERMINATE’

DO A TAPE READ
165 CALL TAPRED(-1,TPDRR,STATUS,ITLEN,IFLEN,IFIL,IEOTR)

FATAL ERROR DETECTION
170 IF(STATUS.LT.0)WRITE(2,1500)IFNUM
171 1500 FORMAT(’ WARNING FILE ’,I3,’ RETRIES FAILED’) 166 GOTO 106

WIND OVER EOF MARK
172 IF(3.170)CALL TAPRED(0,TPDRR,STATUS, , , IFIL, IEOTR)
173 166 CALL IWAIT(IDCH)
174 IERR=0
175 ITRY=ITRY+1
176 IF(IREADW(1,IERR,IDCH).LT.0)STOP’ ERR READ ERR’
177 IF(IERR.EQ.’77777’)GOTO 208
178 CALL CLOSE(IDCCH)

OPEN UP FILES FOR READING IN
179 IF(INFLG.NE.0)FSPECW=FNAMR(IFNUM)
180 180 IF(ENTER(IDCCH,FSPECW,NBLKW).LT.0) STOP’ ENTER ERR’

OPEN UP OUTPUT FILES
181 IF(OUTFLG.NE.0)FSPECR=FNAMR(IFNUM)
182 IF(IENTER(IDCCH,FSPECW,NBLKW).LT.0)STOP’ ENTER ERR’

HEADER BLOCK MANAGEMENT
183 IF(IREADW(256,1HBLK,IDCH).LT.0)STOP’HBLK ERR’

UPDATE HEADER
184 IBFREE=1HBLK(24)
185 1HBLK(7)=1
186 LBLK(19)=3
187 1HBLK(129+IBFREE)=2
188 IBFREE=IBFREE+1
189 1HBLK(129+IBFREE)=NCHAN
190 IBFREE=IBFREE+1
191 1HBLK(129+IBFREE)=NVEL
```
C WRITE OUT BLOCK
C
IF(IWRITEGT 256, IBLK, J, IDCH1).LT.0) STOP 'HBLKW ERR'
C SEE IF NEW VELOCITY ANALYSIS REQUIRED
C
IF(IVELAN(IVEL).NE.IFNUM) GOTO 108
IVEL = IVEL + 1
NLVR = NLVR
DO I = 1, NLVR
   T02LVR(I) = T02LVR(I)
END
C SET UP NEW VINSQ TABLE
C
CALL INVSQ(FSAMP, NSTART, NLVR, L, VINSQ, T0LYR, VLYR)
IF(IVEL.LT.IVEL) INTSW = 0
IF(IVEL.LT.IVEL) GOTO 120
C READ IN NEXT ANALYSIS AND SET UP INTERPOLATION
C
READ(1, 1000) NLVR
READ(1, 1200) (T02LVR(I), V2LVR(I), I = 1, NLVR)
IF(INTSW.EQ.0) GOTO 120
INT = IVELAN(IVEL) - IVELAN(IVEL - 1)
IF(INT.LE.1) GOTO 120
RINT = FLOAT(INT)
DO I = 1, NLVR
   T0INT(I) = (T02LVR(I) - T0LYR(I))/RINT
   VINT(I) = (V2LVR(I) - VLYR(I))/RINT
END
GOTO 120
C COME HERE WHEN NOT A NEW ANALYSIS
C
IF(INTSW.EQ.0) GOTO 120
C ADD ON INTERPOLATING PARAMETERS
C
DO I = 1, NLVR
   T0LYR(I) = T0LYR(I) + T0INT(I)
   VLYR(I) = VLYR(I) + VINT(I)
END
C
```
C SET UP VINSQ TABLE AND THEN DO STACK
C
0160 CALL INVSQ(fsamp,nstart,nlvr,l,vinsq,twlvvr,vlyr)
0169 123 CALL CDP(TSQ,VINSQ,1)
0170 CALL CLOSEC(IDCH)
0171 CALL CLOSEC(IDCH1)
0172 IF(OUTFLG.NE.0)GOTO 999
C
C COME HERE IF OUTPUT TO GO TO TAPE
C
0174 IFLEN=LOOKUP(IDCH1,FSPECW)
0175 IF(IFLEN.LT.0)STOP 'LOOKUP ERR'
C
C DO A TAPE WRITE
C
0177 CALL TAPRED(1,tpdwr,status,itlen,iflen,ifil,ieotw)
0178 CALL CLOSEC(IDCH)
0179 IF(IEOTW.GE.0)GOTO 150
0181 WRITE(7,1600)TPDWR,IFIL
0183 1600 FORMAT(' EOT ON WRITE DRIVE:',I2,' FILE NO:',I4)
0184 WRITE(7,1601)
0185 1601 FORMAT(' ENTER NUMBER OF NEW DRIVE:',$)
0186 READ(5,1802)TPDWR
0187 IF(TPDWR.GT.2)STOP 'WRITE EOT TERMINATE'
0189 150 IF(Status.GE.0)GOTO 160
0191 WRITE(2,1700)IFNUM
0192 1700 FORMAT(' FATAL ERROR ON WRITE FILE','I3')
0193 STOP 'FATAL ERR'
0194 160 CONTINUE
0195 999 CONTINUE
0196 CALL CLOSEC(IDCH)
0197 CALL CLOSEC(IDCH1)
0199 STOP 'NORMAL TERMINATION'
0190 END
SUBROUTINE INVSO(FSAMP,NSTART,NLVR,L,VINSQ,TZLVR,VLYR)
C----------------------------------------·----------------------
C USES INPUT DATA TO GENERATE ARRAY OF INVERSE SQUARED VELOCITIES
C VELOCITIES ARE CONSTANT UP TO FIRST REFLECTOR AND BEYOND
C LAST AND ARE LINEARLY INTERPOLATED IN BETWEEN.
VINSQ(I)=VINSQ1
N1=1
N2=IFIX(FSAMP*TZLVR(I))-NSTART
VINSQ1=1.0/VLYR(I)**2
DO 10 I=N1,N2
VINSQ(I)=VINSQ1
10 CONTINUE
IF(NLVR.EQ.1) GOTO 40
DO 30 J=2,NLVR
N1=N2+1
N2=IFIX(FSAMP*TZLVR(J))-NSTART
VLYR(J)-1)
DELV=(VLYR(J)-VLYR(J-1))/(N2-N1+2)
V=VLYR(J-1)
DO 10 J=N1,N2
VINSQ(I)=1.0/V**2
V=V+DELV
30 CONTINUE
40 N1=N2+1
N2=L
VINSQ(I)=VINSQ1
DO 50 I=N1,N2
VINSQ(I)=VINSQ1
50 CONTINUE
RETURN
END

IMPLICIT INTEGER*2(I-N)
COMMON /KONST/ K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
IA,IB,IC,ID,ITOP
DATA K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,IA,IB,IC,ID,ITOP
/-1.0,1.2,3.4,5.6,7.8,9,10.0,2848,4096,6144,8191/
SUBROUTINE SETAP(L2INT, M)

C SETS UP COMPLEX EXPONENTIAL TABLE IN D+1

COMMON / KONST/ K1M, K8, K1, K2, K3, K4, K5, K6, K7, K8, K9, K10,
11A, 1B, IC, ID, ITOP

C INITIALISE AP

CALL APINIT

C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS.

ID1 = ID + 1
ID2 = ID + 2
L2I2I = L2INT/2 - 1

C TRANSFER 1.0/(L2INT*M) TO K8 AND MULTIPLY BY 2PI FROM TM.

CONST = 1.0/FLOAT(L2INT*M)

CALL APWR

CALL APPUT(CONST, K8, K1, K2)

CALL APWD

C FORM VECTOR RAMP AND TAKE SIN AND COS OF IT

CALL VTSMUL(K8, K1, 2317, K8, K1, K1)

CALL VRAMP(K8, K8, IB, K1, L2I2I)

CALL VCO(1B, K1, ID1, K2, L2I2I)

CALL VSIN(1B, K1, ID2, K2, L2I2I)

C PUT 1.0 IN ID

CALL VCLR(ID, K1, K1)

CALL VTSADD(ID, K1, 2049, ID, K1, K1)

RETURN

END
SUBROUTINE CDP(TSQQ,VINSQ,ICODE)

C MAIN STACKING SUBROUTINE

C VIRTUAL SEISM32767,TSQQ(2048),VINSQ(2048)

DIMENSION INDEX(2048),MUTE(24),STACK(2048),
1IDUM(18),CONST(18),FSTAP(20),XSQ(24)

COMMON /STK/L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,INDEN,
XINDUT,IDCH1,IDCH,XSQ

COMMON /KONST/ K1M,K2,K3,K4,K5,K6,K7,K8,K9,K10,
KAA,IB,IC,ID,ITOP

SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS

IF(ICODE.GT.0)GOTO 20

IB1=IB+1
IC1=IC+1
IC2=IC+2
ID1=ID+1
L1=L+1
L2121=L2INT/2-1
CONST(2)=M*FSAMP
CONST(3)=6.5
CONST(4)=-M*NSTART
CONST(5)=6.6
CONST(6)=L*M
CONST(7)=L+1
CONST(8)=1.6
CONST(9)=1.6/M
CONST(10)=(L+1)*M-1
NBLKTR=L/128
FSTD=ADGET(SEISM(1))
FTSQQ=APGAD(TSQQ(1))
FVINSQ=APGAD(VINSQ(1))
ISTART=1
DO 50 K=1,M
FSTAP(K)=APGAD(SEISM(ISTART))
ISTART=ISTART+L
50 CONTINUE
RETURN

C SET SEISM(L+1) TO $ to cope with time overflow and clear A in AP

JBLK=1
IBLK=1
SEISM(L1)=$.6
CALL VCLR(A,K1,L)

C ITERATE THROUGH CHANNELS

DO 50 CHAN=1,NCHAN
C READ IN TRACE SEISM FROM UNIT 2

IF(IREADA(IDCH,2*L,JBLK,FSTD).LT.0)STOP 'READA ERROR'

JBLK=JBLK+NBKTR
CALL IWAIT(IDCH)

C COMPUTE INDEX ARRAY

CONST(1)=XSQ(JCHAN)
CALL APWR
C TRANSFER (K1,K1) TO HOST DUMMY ARRAY DUM AND REPLACE BY CONST
CALL APGET(DUM,K1,K1,K2)
CALL APPUT(CONST,K1,K1,K2)
C TRANSFER T0SQ TO B AND VIN SQ TO IC1
CALL APPUTA(IB,L,K2,FB0SQ)
CALL APPUTA(IC1,L,K2,FVIN SQ)
CALL APWD
C FORM XSQ(JCHAN)*VINSQ IN CI
CALL VSMUL(IC1,K1,K1,IC1,K1,L)
C FORM SQRTP(T0SQ+XSQ(JCHAN)*VINSQ) IN B
CALL VADD(IB,K1,IC1,K1,IB,K1,L)
CALL VSORT(IB,K1,IB,K1,L)
C FORM IFIX(FSAMP*F*SQRT(T0SQ+XSQ(JCHAN)*VINSQ)*S.5)-NSTART*M IN B
CALL VSMSA(IB,K1,K2,K3,IB,K1,L)
C VINT(IN B,K1,IB,K1,L)
C VSADD(IB,K1,K4,IB,K1,L)
CALL VSADD(IB,K1,K4,IB,K1,L)
C CLIP B BETWEEN 0.0 AND L*M
CALL VCLIP(IB,K1,K5,K6,IB,K1,L)
C MULTIPLY B BY L+1, ADD 1.0 AND PLACE RESULT IN CI
CALL VSMSA(IB,K1,K7,IB,K1,L)
C MULTIPLY B BY 1.0/M AND TAKE INTEGER PART
CALL VSMUL(IB,K1,K9,IB,K1,L)
C SUBTRACT B FROM CI AND PUT RESULT IN B
CALL VSUB(IB,K1,IC1,K1,IB,K1,L)
C FIX B AND TRANSFER BACK AS HOST ARRAY INDEX
CALL VFIXCIB,K1,IB,K1,L
CALL APWR
C HAVING COMPUTED INDEX ARRAY NOW START INTERPOLATION OF TRACE
C CLEAR B IN AP
CALL VLIR(IB,K1,L2INT)
C TRANSFER TRACE TO B
CALL APWR
CALL APPUTA(IB,L,K2,FSTAP(1))
CALL APWD
C FIND MEAN VALUE OF TRACE AND PLACE IN AP(ITOP)
CALL MEANV(IB,K1,ITOP,L)
C SUBTRACT MEAN FROM TRACE
CALL VNEG(IB,K1,ITOP,K1,K1)
CALL VSADD(IB,K1,ITOP,IB,K1,L)
C IF INDEN IS NEGATIVE, NO SCALING OF TRACE
C IF INDEN IS ZERO SCALE TO UNIT R.M.S VALUE
C IF INDEN IS POSITIVE, USE INVERSE ENERGY SCALING (DIVERSITY STACK)
IF(INDEN.LT.UL) GOTO 100
CALL RMSQV(IB,K1,ITOP,L)
IF(IDMUT .GT. 0) CALL VSQCITOP,K1,ITOP,K1,K1
C USE 1.0 RESIDING ID (FROM SETAP)
CALL VDIV(ITOP,K1,ID,K1,ITOP,K1,K1)
CALL VSMUL(IB,K1,ITOP,IB,K1,K1)
C IF INDMUT IS NEGATIVE USE NO MUTING
IF(IDMUT.LT.0) GOTO 200
C TRANSFER MODIFIED TRACE BACK TO SEISM
CALL VCLR(IB,K1,MUTE(/CHAN/))
200 CALL APWR
CALL APGETA(IB,L,K2,FSTAP(1))
CALL APVD
IF(M.EQ.1) GOTO 350
C TAKE TRANSFORM OF TRACE AND PUT IN C
CALL RFFTB(IB,IC,L2INT,K1)
CALL RFFTSC(IC,L2INT,K1,K1)
CALL VCLR(IC,K1,K2)
C ITERATE FROM 2 TO M
DO 300 K=2,N
C MULTIPply TRANSFORM BY COMPLEX EXPONENTIAL ARRAY
CALL CVMUL(IC2,K2,ID,I2,K2,I2,K2,I2K1)
C MOVE C TO B AND TAKE IN PLACE IFFT
CALL VMOVIC,K1,IB,K1,L2INT)
CALL RFFT(IB,L2INT,K1M)
C TRANSFER SHIFTED TRACE BACK TO SEISM
CALL APWR
CALL APGETA(IB,L,K2,FSTAP(K))
CALL APVD
300 CONTINUE
C PICK REQUIRED ELEMENTS OUT OF SEISM AND PLACE IN STACK
DO 350 I=1,L
STACK(I)=SEISM(INDEX(I))
350 CONTINUE
C TRANSFER STACK INTO BI AND ADD TO STACK IN A
CALL APWR
CALL APPUT(STACK,IB1,L,K2)
CALL APWD
CALL VADD(I1,K1,IB1,K1,IA,K1,L)
500 CONTINUE
C SCALE STACK BY 1.0/NCHAN
FNCINV=1.0/NCHAN
CALL APWR
CALL APPUT(FNCINV,ITOP,K1,K2)
CALL APWD
CALL VSMUL(IA,K1,ITOP,IA,K1,L)
C TRANSFER SCALED STACK BACK TO STACK
CALL APWR
CALL APGET(STACK,IA,L,K2)
CALL APWD
IF(IWRITW(2*L,STACK,IBLK,IDCH1,LT.0) STOP 'WRITE ERROR 2'
RETURN
SUBROUTINE TAPRED (ICOM, IDRV, ISTAT, ITLEN, ILEN, IFNUM, IEOF)

TAPE HANDLING SUBROUTINE
ICOM is the command signal
-1 is a read, 0 is a wind, 1 is a write
IDRV is the drive being used
ISTAT is the status on return
ITLEN is the time length of a file read
ILEN is the block length of a file read or written

INTEGER*2 MASK(8), ESTAT
LOGICAL*1 ISTAT, COM(4), SDSCOM(8), IDRV, ITLEN, ECOM(4), IFLEN, ESTAT, ERRS(8)

DATA MASK/'1, 2, 4, 10, 28, 48, 100, 200/
DATA SDSCOM/'1, 2, 3, 4, 5, 6, 7/

ITRV = 0
IF (ICOM) 10 30, 20

SECTION CONTROLLING A READ
CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
10 ITRY = ITRY + 1
SET UP COMMAND FOR READ
COM(1) = SDSCOM(4)
COM(2) = 1
COM(3) = IDRV
COM(4) = -1
CALL SDSTOM(COM, ISTAT, ITLEN, ILEN)
IF (ISTAT.EQ.0) RETURN

ERROR DETECTED ON READ
ISTAT = ISTAT
GOTO 40
IF SHORT RECORD FOUND REREAD TAPE
50 ITMP = ISTAT .AND. MASK(6)
IF (ITMP.NE.0) GOTO 10
ITMP = ISTAT .AND. MASK(2)
IF (ITMP.EQ.0) RETURN

IF CRC ERROR FOUND REWIND TAPE AND RETRY
WRITE(2, 2010) IFNUM
2010 FORMAT(' FILE NO ' , I4, ' CRC ERROR REWINDING')
IF (ITRY .GE. 2) GOTO 130
ECOM(1) = SDSCOM(6)
ECOM(2) = 1
FORTRAN IV

331 ECOM(3)=IDRV
332 ECOM(4)=0
333 CALL SDS10(ECOM,ESTAT,)
334 GOTO 18

WRITE SECTION

335   ITRY=ITRY+1
336   IF(ITRY.GT.3)GOTO 130
338   COM(1)=SDSCOM(7)
339   IFLEN=(IFLEN+3)/4
340   COM(2)=IFLEN
341   COM(3)=IDRV
342   COM(4)=1
343   CALL SDS10(ECOM,ISTAT,)
344   IF(ISTAT.EQ.0)RETURN

WRITE ERROR DETECTED

346   ISTAT=ISTAT
347   GOTO 40
348   ITMP=ISTAT.AND.MASK(6)
349   IF(ITMP.EQ.6.AND.ITMP.EQ.2)RETURN

REPORT AND RETRY

352   WRITE(2,2B2)IFNUM
353   2B2F FORMAT(' FILE NO ',14,' WRITE CRC ERR RETRY PROPOSED')
355   ECOM(1)=SDSCOM(6)
356   ECOM(2)=2
357   ECOM(3)=IDRV
358   CALL SDS10(ECOM,ESTAT,)
359   NBUF=B
360   IPAD=(IFLEN*2)/8-NBUF
361   CALL TWrit(Errs,NBUF,ESTAT,IPAD,IFLEN,IDRV)
362   GOTO 20

VIND FOWARD ONE FILE

363   COM(1)=SDSCOM(5)
364   COM(2)=1
365   COM(3)=IDRV
366   CALL SDS10(ECOM,ISTAT,)

CLEAR IRRELEVANT BITS FROM ERROR BYTE

368   ISTAT=ISTAT.AND.NOT.MASK(6)
369   IF(ISTAT.EQ.2)RETURN
370   ISTAT=ISTAT
372   IF(ISTAT.NE.0)GOTO 40
C IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
COM(1)=SDSCOM(6)
COM(2)=1
COM(3)=IDRV
COM(4)=&
CALL SDS18(ECOM,ESTAT,
35 RETURN
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS: TAPE BUSY, TAPE OFFLINE
C BOT,EOT
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
48 WRITE(2,1018)ISTAT,IFNUM
1010 FORMAT( ' STATUS=',I3,' FILE NO=',I4)
ISTAT=ISTATI.AND..NOT.MASK(4)
ITMP=ISTATI.AND.MASK(5)
IF(ITMP.EQ.99)GOTO 90
90 ECOM(1)=SDSCOM(1)
ECOM(2)=&
ECOM(3)=IDRV
ECOM(4)=&
CALL SDS18(ECOM,ESTAT,
C HAVING EXAMINED STATUS IF TAPE STILL
C BUSY, LOOP AGAIN, IF NOT TRY COMMAND AGAIN
ESTATI=ESTAT
ITMP=ESTATI.AND.MASK(5)
IF(ITMP.NE.99)GOTO 90
IF(ICOM).EQ.38,29
C TAPE OFFLINE
ITMP=ISTATI.AND.MASK(1)
IF(ITMP.NE.99)GOTO 90
TYPE 1001,IDRV
1001 FORMAT( ' TAPE DRIVE ',I1,' OFFLINE'
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
110 ECOM(1)=SDSCOM(1)
ECOM(2)=&
ECOM(3)=IDRV
ECOM(4)=&
CALL SDS18(ECOM,ESTAT,
ESTATI=ESTAT
ITMP=ESTATI.AND.MASK(1)
IF(ITMP.NE.99)GOTO 110
IF(ICOM).EQ.38,29
C EOT
103 ITMP=ISTATI.AND.MASK(3)
104 IF(ITMP.NE.99)GOTO 120
105 TYPE 1002,IDRV
106 EOT=-1
107 RETURN
108 120 IF(ICOM).EQ.35,78
C ERROR EXIT RETURN
109 ISTATI=-1
110 RETURN
111 END
Post-Stack Processing :- MPPOST

Input file......DK2:MPPOST.DAT

Log file.......DK2:MPPOST.LOG

Input Parameters

READ(1,1000)NFILES,NSAMP,NSTART,INFLG,OUTFLG

1000 FORMAT(12I5)

NFILES...Number of files to process
NSAMP....Number of samples per trace
NSTART...Starting sample number of trace
INFLG....Input flag
          0 - Input from tape
          1 - Input from disc
OUTFLG...Output flag
          0 - Output to tape
          1 - Output to disc

READ(1,1000)TPDRR,TPDRW

TPDRR....Input Tape drive
TPDRW....Output tape drive

READ(1,1100)FSAMP

1100 FORMAT(2F10.0)
FSAMP....Sampling rate in samples/millisecond.

If INFLG = 1 READ(1,1300)FSPECR

1300 FORMAT(3A4)

FSPECR...NFILES input files

If OUTFLG = 1 READ(1,1300)FSPECW

FSPECW...NFILES Output files

READ(1,1000)NPROC

NPROC....Number of processes to be applied

READ(1,1000)(UTLFLG(I),I=1,NUTIL)

UTLFLG...Flag showing if each process is to be applied

0 - Do not apply
1 - do apply

READ(1,1000)(IORD(I),I=1,NPROC)

IORD.....Process numbers in order of application

This is then followed by input data to each chosen process, in the UTLFLG order

1....Edit

READ(1,1000)NFILED
READ(1,1000)(IFILED(I),I=1,NFILED)

NFILED...Number of stacked traces to edit

IFILED...Trace numbers of traces to be edited, in ascending order.

2....Gain Ramps

e0.2t Ramp

READ(1,1000)IAPLX

IAPLX....Application flag
   0 - apply
   1 - remove

te0.2t Ramp

READ(1,1000)IAPLTX

IAPLTX...Application flag
   0 - apply
   1 - remove

TV**2 Ramp

READ(1,1000)IAPLTV,NLYR

READ(1,1000)(TOLYR(I),VLYR(I),I=1,NLYR)

IAPLTV...Application flag
0 - apply
1 - remove

NLYR.....Number of Time/Velocity pairs to be entered
TOLYR....Two-way travel time in milliseconds
VLYR.....RMS Velocity down to specified time

3....Mute

READ(1,1000)NTAP,NMPTS

READ(1,1000)(MNPOS(I),MSAMP(I),I=1,NMPTS)

NTAP.....Number of points in the cosine taper
NMPTS....Number of defined mute positions
MNPOS....File number of defined mute
MSAMP....Sample number to mute down to

4....Spiking Deconvolution

READ(1,1000)NFILT,IDUM,ISPIKE,INORM

READ(1,1100)WHITE

NFILT....Number of filter points
ISPIKE....Spike position
INORM....Scaling flag
  0 - no scaling
  1 - unit filter energy
  2 - equal input/output energy
WHITE....Fractional pre-whitening
5....Bandpass Filtering

READ(1,1100)FL,FU
READ(1,1100)FTPR1,FTPR2

FL.......Lower cutoff frequency Hz
FU.......Upper cutoff frequency Hz
FTPR1....Length of lower cutoff taper Hz
FTPR2....Length of upper cutoff taper Hz

6....Bandreject Filtering

READ(1,1100)FLR,FUR

FLR......Lower cutoff frequency Hz
FUR......Upper cutoff frequency Hz

7....Prediction Error Deconvolution

READ(1,1000)NPFILT,NLAG,IPNORM
READ(1,1100)PRWHIT

NPFILT...Number of samples in filter
NLAG.....Prediction distance, in samples
IPNORM...Scaling flag
  0 - no scaling
  1 - Filter unit energy
  2 - equal input/output energy
PRWHIT...Fractional prewhitening
8....Trace Normalisation

READ(1,1000)NRMFLG

NRMFLG...Normalisation flag

0 - normalise to unit energy
1 - normalise to unit maximum amplitude
C POST STACK UTILITY PROGRAM
C THIS INVOLVES THE FOLLOWING
C
C 1: EDIT
C 2: EXP(0.2T) AMP RECOVERY
C 3: T*EXP(0.2T) AMP RECOVERY
C 4: TV**2 AMP RECOVERY
C 5: MUTING
C 6: DECONVOLUTION
C 7: BANDPASS FILTERING
C 8: BANDREJECT FILTERING
C 9: PREDICTION ERROR FILTERING
C 10: NORMALISATION TO UNIT ENERGY/AMPLITUDE

REAL*8 FSPECR, FSPECW, FNAMR, FNAMO
VIRTUAL FNAMR(100), FNAMO(100), EXPT(2048), TEXPT(2048),
XTVSO(2048), TAPER(512), BPASS(2049), BRJCT(2049), MUTE(2049),
REAL*4 BUF(3), T0LYR(20), VLYR(20), CONST(3), SEISM(2049)
INTEGER*2 IORDC81, UTLFLG£81, IFIELD< l/6/6l, DBUFC435.2),
XIHBLK<256l, OUTFLG, MNPOS<3/6l, MSAMP<3.0'l, MINC(30)
LOGICAL*1 TPDRR, TPDRW, ISTAT
EQUIVALENCE (XIHBLK(1), DBUF(1)), ,(SEISM(1), DBUF(257))
DATA DEV/3RRK /

C SET UP VIRTUAL ADDRESSES
C
IEOTR=0
IEOTW=0
IATAP=APGAD(TAPER(1))
IATBP=APGAD(BPASS(1))
IATBR=APGAD(BRJCT(1))
IATVSQ=APGAD(TVSQ(1))
IATEXPT=APGAD(TEXPT(1))

READ< 1, 1/6/6/6> TPDRR, TPDRW
READ< 1, 1/6/6> TPDRR
READ< 1, 1/6/6> IFIELD, DBUFC1215,
FORMAT<13I2>
READ< 1, 1/6/6> TL\020.
FORMAT<2F10.0>

C READ IN FILE SPECS FOR INPUT
C
READ< 30, 1> INFLG
FORMAT<3A4>
IF(INFLG.EQ.0)GOTO 20
DO 30 I=1, NFILES
READ< 1, 1/6/6> BUF
CALL IRAD60(I2,FBUF,FSPECR)
FNAMR(I)=FSPECR
30 CONTINUE

READ IN FILE SPECS FOR OUTPUT
20 IF(OUTFLG.EQ.0)GOTO 50
DO 60 I=1,NFILES
   READ(1,1200)FBUF
   CALL IRAD60(I2,FBUF,FSPECW)
   FNAMO(I)=FSPECW
60 CONTINUE

READ IN JOB SPECIFIC DATA AND SET UP
THE FILTERS TO BE USED

50 NUTIL=10
ITX=0
CONST(1)=FLOAT(NSTART)
CONST(2)=0.2
CONST(3)=1.0/(1000.0*FSAMP)
CALL APINIT
NSAMP2=2
51 IF(NSAMP2.GE.NSAMP)GOTO 52
NSAMP2=NSAMP2*2
GOTO 51
52 CONTINUE

READ IN FLAGS FOR PROCESSES AND EXECUTION ORDER
READ(1,1000)NPROC
READ(1,1000)UTLFLG(I),I=1,NUTIL
READ(1,1000)IORED(I),I=1,NPROC

TRACE EDIT DATA
IF(UTLFLG(1).EQ.0)GOTO 78
READ(1,1000)NFLLED
READ(1,1000)IFILED(I),I=1,NFILED

EXP(0.2T) RAMP DATA
70 IF(UTLFLG(2).EQ.0)GOTO 88
READ(1,1000)IAPLTX
CALL TEXRMP(EXPT,0,ITX,NSAMP,CONST,AEXPT)
ITX=1

T*EXP(0.2T) RAMP DATA
80 IF(UTLFLG(3).EQ.0)GOTO 98
READ(1,1000)IAPLTX
CALL TEXRMP(EXPT,1,ITX,NSAMP,CONST,AEXPT)
ITX=1
C TV**2 RAM

C 90 IF(UTLFLG(4).EQ.0)GOTO 100
     85 READ(1,1000)TOLTV,NLVR
  874 READ(1,1000)(TOLYR(I),VLYR(I),I=1,NLVR)
  875 CALL TVRMP(TVSQ,TOLYR,VLYR,NLVR,ITX,NSAMP,NSTART,CONST,
               XFSAMP,ATVSQ)

C MUTE DATA

  877 IF(UTLFLG(5).EQ.0)GOTO 100
  878 READ(1,1000)NTAP,NMPTS
  875 READ(1,1000)(MNPOS(I),MSAMP(I),I=1,NMPTS)
  873 CALL COTAP(TAP,NTAP,ATAP)
  881 DO 101 J=2,NMPTS
  901 MINC(J)=(MSAMP(J)-MSAMP(J-1))/(MNPOS(J)-MNPOS(J-1))
  902 MUTE(J)=MSAMP(J)
  904 IP0S=2
  905 DO 102 J=2,NMPTS
  915 IF(J.LT.MNPOS(IPOSO)GOTO 103
  916 MUTE(J)=MSAMP(IPOS)
  917 IP0S=POSO+1
  918 GOTO 102
  921 MUTE(J)=MUTE(J-1)+MINC(IPOS)

  922 CONTINUE

C DECON INPUT

  923 IF(UTLFLG(6).EQ.0)GOTO 120
  924 READ(1,1000)NFILT,ICONT,ISPIKE,INORM
  925 READ(1,1000)WHITE

C SANDPASS FILTER

  927 IF(UTLFLG(7).EQ.0)GOTO 130
  928 READ(1,1100)FL,FU
  929 READ(1,1100)FTP1,FTP2
  931 DF1=FLOAT(NSAMP2/2+1)/(FSAMP*500.)
  932 FL1=FL-FTP1
  933 FU4=FU+FTP2
  934 CALL BANDPS(A,ATBP,FPASS,FL1,FL,FU,FU4,DF1,NSAMP2)
  935 NTRANF=2*NSAMP2
  936 NFILT=NSAMP2+1
  937 NBEXP=(2*NSAMP2)+2-NSAMP

C SANDREJECT FILTER

  939 IF(UTLFLG(8).EQ.0)GOTO 140
  940 READ(1,1100)FLR,FUR
  941 DF1=FLOAT(NSAMP2/2+1)/(FSAMP*500.)
  942 NTRANF=2*NSAMP2
  945 NFILT=NSAMP2+1
  944 NBEXP=(2*NSAMP2)+2-NSAMP
  945 CALL BANDRJ(ABR,BRJCT,FLR,FUR,DF1,NSAMP2)
PREDICTIVE DECONVOLUTION

140 IF(UTLFLG(9).EQ.0)GOTO 150
110 READ(1,150)MPFILT,NLAG,IPNORM
119 READ(1,110)PRWHIT

TRACE NORMALISATION
150 IF(UTLFLG(18).EQ.0)GOTO 160
120 READ(1,120)NRMFLG

BLOCKING PARAMETERS
150 CONTINUE
124 IF(ED=1)
125 NBLKTR=NSAMP/128
126 NBLKW=NBLKTR+1
127 NBYTR=NBLKW*512
128 NWDR=NBYTR/2

START OF LOOP ON DIFFERENT FILES
125 DO 200 IFNUM=1,NFILES
134 IF(IL=I)
135 IF(NINFLG.NE.0)GOTO 118

TAPE INPUT HANDLING
EOT CHECK
140 IF(IFIL.GE.0)GOTO 210

CALL TAPSUB(<-1,TPDRR,ISTAT,IFIL,ISTR,IFNUM,NBYTES,NBYTESR)<

READ FROM TAPE TO MEMORY
147 CALL TAPSUB(-1,TPDRR,ISTAT,IFIL,NBYTES,TPDRR,ISTR,IFNUM,NBYTESR)<
148 IF(ISTR.GT.0)WRITE(2,150)IFIL
150 IF(ISTR.LT.0)WRITE(2,150)IFIL
151 IF(ISTR.GT.0)GOTO 230

WIND OVER EOF MARK
152 CALL TAPSUB(0,TPDRR,ISTAT,IFIL,ISTR,IFNUM,NBYTESR)
C OPEN UP READING FILES

C HEADER BLOCK MANAGEMENT

C OPEN UP OUTPUT FILES

C PROCESS EDIT COMMANDS

C AMP RECOVERY FILTERS APPLICATION AND REMOVAL

C EXP(0.2T) FILTER
FOR1AN IV V92.84 THU 28-JAN-81 08:17:19 PAGE 596

@19 430 IAPL=IAPLX
@19  CALL APPUTA(NSAMP,NSAMP,2,AEXPT)
@19  GOTO 455

C T*EXP(0.2T) FILTER

@20 440 IAPL=IAPLTX
@20  CALL APPUTA(NSAMP,NSAMP,2,ATEXPT)
@20  GOTO 455

C TV**2 FILTER

@22 450 IAPL=IAPLTV
@22  CALL APPUTA(NSAMP,NSAMP,2,ATVSQ)

COMMON CODE

@20 455 CALL APWD
@20  IF(IAPL.EQ.0)CALL VMUL(0,1,NSAMP,1,0,1,NSAMP)
@20  IF(IAPL.NE.0)CALL VDIV(NSAMP,1,0,1,0,1,NSAMP)
@21 CALL APWR
@21 GOTO 480

MUTE APPLICATION

@21 460 CALL APPUTA(NSAMP,NTAP,2,ATAP)
@21  CALL APWD
@21  MNCLR=MUTE(IFIL)
@21  CALL VCLR(0,1,MNCLR)
@21  CALL VMUL(MNCLR,1,NSAMP,1,MNCLR,1,NTAP)
@21 CALL APWR
@21 GOTO 480

SPIKE DECON

@21 470 CALL SPIKE(NSAMP,NSAMP2,NFILT,WHITE,INORM,ISPIKE)
@22 GOTO 480

BANDPASS FILTER

@22 480 CALL APPUTA(410B,NBFILT,2,ATBP)
@22 CALL APWD
@22 CALL VCLR(NSAMP,1,NBEXP)
@22 CALL RFFT(S,NTRANF,1)
@22 CALL RFFTSC(S,NTRANF,3,1)
@22 CALL VMUL(0,2.410B,1,0,2,NBFILT)
@22 CALL VMUL(1,2,410B,1,1,2,NBFILT)
@22 CALL RFFTSC(0,1,NTRANF,-3,0)
@22 CALL RFFT(0,NTRANF,-1)
@22 CALL APWR
@22 GOTO 480

BANDREJECT FILTER
CALL APPUTA(4100,NRFILT,2,ATBR)
CALL APWD
CALL VCLR(NSAMP,1,NBEXP)
CALL RFFT(0,NTRANF,1)
CALL RFFTC(0,NTRANF,3,1)
CALL VMUL(0,2,4100,1,0,2,NRFILT)
CALL VMUL(1,2,4100,1,1,2,NRFILT)
CALL RFFTC(0,NTRANF,-3,0)
CALL RFFT(0,NTRANF,-1)
CALL APWR
GOTO 488
CALL PROICT(NSAMP,NSAMP2,NRFILT,PRVHIT,IPNORM,NLAG)
GOTO 488
CALL APWR
GOTO 516
CALL MAXMGV(0,1,2058,NSAMP)
GOTO 516
CALL SVESQ(0,1,2058,NSAMP)
CALL VSQRT(2058,1,2058,1,1)
CALL VDIV(2058,8,8,1,B,1,NSAMP)
CALL APWR
CONTINUE
CALL APWAIT
CALL APGET(SEISM,B,NSAMP,2)
CALL APWD
IF(OUTFLG.EQ.0)GOTO 328
IF(IWRITE(2*NSAMP,SEISM,JBLK,IDCH1).LT.0)STOP 'WRITE ERR2'
CALL close(IDCH1)
GOTO 288
CALL APWR
END OF PROCESS LOOP

CALL TAPSUB(1,TPDRW,ISTAT,IFLEN,IFIL,DBUF,NBVTR,IEOTW)
CALL CHECK FOR ERRORS
IF(IEOTW.GE.8)GOTO 258
WRITE(7,1600)TPDRW,IFIL
1600 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
WRITE(7,1601)
1601 FORMAT(' ENTER NEW WRITE DRIVE NO:',$)
READ(5,1452)TPDRW
IF(IEOTW.GT.2)STOP 'EOT TERMINATE'
GOTO 258
IF(ISTAT.GE.0)GOTO 288
WRITE(2,1708) IFIL
1708 FORMAT(' FATAL ERROR ON WRITE FILE NO',I5)
STOP 'WRITE ERROR'
CONTINUE
CLOSE(IDCCH)
CALL close(IDCCH)
STOP 'NORMAL TERMINATION'
END
SUBROUTINE TEXRMP(FILT, IFITYP, IFLG, NSAMP, CONST, AFILT, FS)

C IF ITVYP=0 THIS ROUTINE PRODUCES AN EXP(0.2T) ARRAY
C NSAMP LONG IN FILT
C IF ITVYP=1 T*EXP(0.2T) PRODUCED
C C
C C C CONST(1)=NSAMP
C CONST(2)=0.2
C CONST(3)=1.0/(1800.0*FSAMP)

VIRTUAL FILT(2#4B)
DIMENSION CONST(3)
IF(IFLG.NE.1)GOTO 1

FORM THE T RAMP
CALL APWAIT
CALL APPUT(CONST, 8189, 3, 2)
CALL APWD
CALL VCLR(0, 1, NSAMP)
CALL VRAMP(8189, 8191, 0, 1, NSAMP)

FORM EXP(0.2T)
CALL VSMUL(0, 8190, NSAMP, 1, NSAMP)
CALL VEXP(NSAMP, 1, NSAMP, 1, NSAMP)
IF(IFITYP.EQ.0)GOTO 2

FORM T*EXP(0.2T)

10 CALL VMUL(NSAMP, 1, NSAMP, 1, NSAMP)
20 CALL APWR
CALL APGETA(NSAMP, NSAMP, 2, AFILT)
RETURN
END
SUBROUTINE TVRMP(FILT,TOLVR,VLVR,NLYR,IFLG,NSAMP,NSTART,%CONST,FSAMP,AFILT)

C THIS ROUTINE PRODUCES A TV**2 RAMP

C FROM VELOCITY INFO IN TOLVR,VLVR

C VIRTUAL FILT(2048)

DIMENSION TOLVR(20),VLVR(20),CONST(3)

C CHECK IF T RAMP ALREADY FORMED

IF(IFLG.NE.0)GOTO 18

CALL APWIT

CALL APPUT(CONST,8189,3,2)

CALL APWD

CALL VCLR(81,1,NSAMP)

CALL VRAMP(8189,8191,81,1,NSAMP)

C FORM VELOCITY RAMP IN FILT

18 N1=1

N2=IFIX(FSAMP*TOLVR(1))-NSTART

V=VLVR(1)

DO 15 I=N1,N2

15 FILT(I)=V

IF(NLYR.EQ.1)GOTO 48

DO 28 J=2,NLYR

N1=N2+1

N2=IFIX(FSAMP*TOLVR(1))-NSTART

DELV=(VLVR(J)-VLVR(J-1))/(N2-N1+2)

V=VLVR(J-1)

DO 38 I=N1,N2

38 FILT(I)=V

V=V+DELV

38 CONTINUE

28 CONTINUE

48 N1=N2+1

N2=NSAMP

V=VLVR(NLYR)

DO 58 I=N1,N2

58 FILT(I)=V

C PUT V RAMP IN AP AND FORM TV**2

CALL APWIT

CALL APPUTA(NSAMP,NSAMP,2,AFILT)

CALL APWD

CALL VSQ(NSAMP,1,NSAMP,1,NSAMP)

CALL VMUL(81,1,NSAMP,1,NSAMP)

CALL APWR

CALL APGETA(NSAMP,NSAMP,2,AFILT)

CALL APWD

RETURN

END
SUBROUTINE COTAP(TAPER, NTAP, ATAP)

C THIS ROUTINE PRODUCES A COSINE TAPER NTAP

C SAMPLES LONG

VIRTUAL TAPER(512)

CALL APWAIT

CALL APPUT(1.0/FLOAT(NTAP), 1, 2)

CALL VCLR(1, 1, NTAP)

CALL VTSADD(1, 1, 2306, 1, 1, 1)

CALL VTSMUL(1, 1, 2306, 1, 1, 1)

CALL VRAMP(1, 1, 1, NTAP)

CALL VCOS(1, 1, 1, NTAP)

CALL VTSADD(1, 1, 2049, 1, NTAP)

CALL VTSMUL(1, 1, 2327, 1, NTAP)

CALL APWR

CALL APGETA(1, NTAP, 1, ATAP)

CALL APWD

RETURN

END
SUBROUTINE BANDPS(ATBP, BPASS, F1, F2, F3, F4, DFI, NSAMP)

C SUBROUTINE WHICH CREATES A BANDPASS FILTER
C
C F1= BOTTOM CUT OFF FREQUENCY
C F2= START OF FULL PASS
C F3= END OF FULL PASS
C F4= TOP CUT OFF FREQUENCY
C
C BETWEEN F1, F2 AND F3, F4, A COSINE TAPER IS APPLIED

VIRTUAL BPASS(2049)

SET UP CONSTANTS

N1=2*F1*DFI
N2=2*F2*DFI
N3=2*F3*DFI
N4=2*F4*DFI
NFILT=NSAMP+1
NTP1=N2-N1
NTP2=N4-N3

OK=N3-N2

CALL APWAIT

SET UP THE FILTER IN THE AP

DO 10 I=1,2
NTAP=NTP1
NTAP=NTAP
RNTAP=1.0/FLOAT(NTAP)
CALL APPUT(RNTAP, O, 1, 2)
CALL VCLR(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VMOV(O, 1, 2049, NTAP)
CALL VMOV(O, 1, 2049, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VTSADD(O, 1, 2049, O, 1, NTAP)
CALL VRAMP(O, 1, NTAP)
CALL VCOS(O, 1, NTAP)
SUBROUTINE BANDRJ(ATBR, BRJCT, F1, F2, DFI, NSAMP)

SUBROUTINE TO CREATE A BANDREJECT FILTER

F1 = LOWER CUTOFF POSITION
F2 = UPPER CUTOFF POSITION

THE FILTER TAKES THE FORM OF
A SINE BELL CENTERED ON THE FREQUENCY TO
BE REMOVED COMPLETELY

VIRTUAL BRJCT(249)

SET UP CONSTANTS

N1 = 2.0 * F1 * DFI
N2 = 2.0 * F2 * DFI
NTAP = N2 - N1
CALL APWAIT

SET UP FILTER IN AP

CALL VCLR(0,1,NTAP)
NFILT = NSAMP + 1
FTAP = 1.0 / FLOAT(NTAP)
CALL APPUT(FTAP,1,1,2)
CALL VTSMUL(1,1,2317,1,1,1)
CALL VRAMP(0,1,0,1,NTAP)
CALL VCOS(0,1,0,1,NTAP)
CALL VTSADD(0,1,249,0,1,NTAP)
CALL VTSMUL(0,1,2327,0,1,NTAP)
CALL VMUL(0,1,4096,1,NTAP)

SET UP FULL FILTER NOW TAPER FINISHED

CALL VCLR(0,1,NFILT)
CALL VTSADD(0,1,249,0,1,NFILT)
CALL VMUL(N1,1,4096,1,N1,1,NTAP)
CALL APWR
CALL APGETA(0,NFILT,2,ATBR)
CALL APWD
RETURN
END
SUBROUTINE SPIKE(NSAMP,NSAMP2,ILENTH,WHITE,IFLAG,ISPIKE)

C WIENER SPIKING FILTER ROUTINE
C NSAMP=DATA LENGTH
C NSAMP2=NEAREST POWER OF 2 TO NSAMP
C ILENTH = FILTER LENGTH
C IFLAG = TRACE NORMALISATION FLAG
C 0 NO NORMALISATION
C 2 EQUAL INPUT-OUTPUT ENERGY
C ISPIKE SPIKE POSITION

J922 NTRAN=2*NSAMP2
J932 NCCLR=NTRAN-NSAMP
J944 NFCLR=NTRAN-ILENTH
J955 NTRAN2=NTRAN+2
J966 NM1=NSAMP2-1
J977 I6=NTRAN+ILENTH
J988 I7=I6+ILENTH
J999 I8=I7+ILENTH
J000 ISP=I6+ISPIKE

C HAVING SET UP CONSTANTS GET INPUT TRACE
C ENERGY IF REQD FOR NORMALISATION
C
J001 IF(IFLAG.NE.2)GOTO 10
J023 CALL SVESQ(8,1,8191,NSAMP)
J034 CALL VSQRT(8191,1,8191,1,1)
J045 CALL APWR
J056 CALL APGET<EN,8191,1,2>
J067 CALL APWD

C GET AUTOCORRELATION FUNCTION
C
J078 CALL VCLR(NSAMP,1,NCLR)
J089 CALL RFFT(8,NTRAN,1)
J090 CALL VMOV(8,1,NTRAN,1,NTRAN)
J091 CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
J092 CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
J093 CALL VCLR(NTRAN+3,2,NM1)
J094 CALL RFFTS(NTRAN,NTRAN,-1,-1)
J095 CALL RFFT(NTRAN,NTRAN,-1)

C AUTO FUNCTION NOW 2N LONG FROM NTRAN
C ORIGINAL FUNCTION TRANSFORMED 8-NTRAN
C
C NOW WHITEN
J098 CALL APWR
J109 CALL APPUT(WHITE,8191,1,2)
J110 CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)

C SET UP SPIKE CC FUNCTION
CALL VCLR(I6,1,ILENGTH)
CALL VTSADD(ISP,1,2,S9,ISP,1,1)

SOLVE EONS

CALL WIENER(ILENTH,NTRAN,I6,I7,I8,1)
CALL APCHK(IER)
IF(IER.NE.0)STOP 'LEVINSON FAILURE'
CALL VMOV(I7,1,NTRAN,1,ILENTH)

NORMALISE FILTER IF ASKED FOR

CALL SVESQ(NTRAN,1,I6,ILENTH)
CALL VSQR(I6,1,I6,1,1)
CALL VDIV(I6,8,NTRAN,1,NTRAN,1,ILENTH)

APPLY FILTER

CALL VCLR(I6,1,NFCLR)
CALL RFFT(NTRAN,NTRAN,1)
CALL VMUL(8,1,NTRAN,1,8,1,2)
CALL CVMUL(2,2,NTRAN2,2,2,2,NM1,1)
CALL RFFTSC(8,NTRAN,8,-1)
CALL RFFT(8,NTRAN,-1)

DO SCALING IF REQD

IF(IFLAG.NE.2)GOTO 30
CALL SVESQ(8,1,8191,NSAMP)
CALL VSQR(8191,1,8191,1,1)
CALL APWR
CALL APPUT(EN,8199,1,2)
CALL APWD
CALL VDIV(8191,1,8190,1,8190,1,1)
CALL VSMUL(8,1,8190,8,1,NSAMP)
30 CALL APWR
RETURN
END
SUBROUTINE PROICT(NSAMP, NSAMP2, ILENTH, WHITE, IFLAG, NLAG)

C THIS ROUTINE DESIGNS AND APPLIES 
C A PREDICTION ERROR FILTER 
C NSAMP=DATA LENGTH 
C NSAMP2=NEAREST POWER OF 2 TO NSAMP 
C ILENTH=FILTER LENGTH 
C WHITE=FRACTION PREWHITENING 
C IFLAG=0 NO NORMALISATION 
C =1 FILTER UNIT ENERGY 
C =2 INPUT/OUTPUT TRACE ENERGY CONSTANT 
C NLAG= LAG OFFSET OF PREDICTION 

NTRAN=2*NSAMP2 
NCLR=NTRAN-NSAMP 
NM1=NSAMP2-1 
NLAG=NTRAN+NLAG 
NFILT=ILENTH+NLAG 
I7=NLG+ILENTH 
I8=I7+ILENTH 

GET INPUT TRACE ENERGY

IF( IFLAG.NE.2 )GOTO 10
CALL SVESQ(0,1,8191,NSAMP)
CALL VSQRTC8191,1,8191,1,1J
CALL APGETCEN,8191,1,2J
CALL APWD

GET AUTOCORRELATION FUNCTION

CALL VCLRCNSAMP,1,NCLR)
CALL RFFT(0,NTRAN,1)
CALL VMOV(0,1,NTRAN,1,NTRAN)
CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
CALL VCLR(NTRAN+3,2,NM1)
CALL RFFTSC(NTRAN,NTRAN,-1,-1)

NOW WHITEN IT

CALL APWR
CALL APPUTCWHITE,8191,1,2)
CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)

NOW SOLVE EONS

CALL WIENER(ILENTH,NTRAN,NLG,17,18,1)
CALL APCHK(IER)
IF( IER.NE.0 )STOP 'LEVINSOIN FAILURE'
IF( IFLAG.NE.1 )GOTO 20
CALL SVESQ(17,1,18,ILENTH)
CALL VSORT(18,1,18,1,1)
CALL VDIV(18,0,17,1,17,1,1,ILENTH)

C APPLY FILTER

CALL VCLR(NTRAN,1,NFILT)
CALL VTSADD(NTRAN,1,2#49,NTRAN,1,1)
CALL VSUB(17,1,NLG,1,NLG,1,ILENTH)
CALL VCLR(17,1,NTRAN-NFILT)
CALL RFFT(NTRAN,NTRAN,1)
CALL VMUL(0,1,NTRAN,1,0,1,2)
CALL CVMUL(2,2,NTRAN2,2,2,2,NM1,1)
CALL RFFTSC(0,NTRAN,0,-1)
CALL RFFT(0,NTRAN,-1)

C DO SCALING

IF(IFLAG.NE.2)GOTO 38
CALL SVESQ(0,1,8191,NSAMP)
CALL VSQRT(8191,1,8191,1,1)
CALL APWR
CALL APWD
CALL VDIV(8191,1,8190,1,8190,1,1)
CALL VSMUL(0,1,8190,0,1,NSAMP)
38 CALL APWR
RETURN
END
SUBROUTINE TAPSUB(ICOM, IDRV, ISTAT, ILEN, IFNUM, BUF, NBYTE, IEO)

C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ, 0 IS A WRITE
C IDRV IS THE DRIVE BEING USED
C ISTAT IS THE STATUS ON RETURN
C ILEN IS THE BLOCK LENGTH OF A FILE READ OR WRITTEN

INTEGER*2 MASK(8), ESTAT, BUF(1)
LOGICAL*1 ISTAT, COM(4), SDS(8), IDRV, ECOM(4), XIF(1), ESTAT, ERRS(8)
DATA MASK/1, 2, 4, 10, 20, 40, 100, 200/
DATA SDS/0, 1, 2, 3, 4, 5, 6, 7/
ENER = 0
IF (ICOM) 10, 30, 20

SECTION CONTROLLING A READ
CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
10 ITRY = ITRY + 1
SET UP COMMAND FOR READ
IF (ICOM) 10, 30, 20
NB = NBYTE
CALL TREAD(BUF, NBYTE, ISTAT, IDRV)
ERROR DETECTED ON READ
IF (ISTAT) RETURN
IF SHORT RECORD FOUND REREAD TAPE
50 ITP = ISTAT .AND. MASK(6)
17 IF (ITP .NE. 0) GOTO 10
IF (ITP .EQ. 0) RETURN
IF CRC ERROR FOUND REWIND TAPE AND RETRY
WRITE(2, 2010) IFNUM
2013 FORMAT(' FILE NO ', 14, ' CRC ERROR REWINDING')
IF (ITP .EQ. 2) GOTO 10
ECOM(1) = SDS(6)
ECOM(2) = ISTAT .AND. MASK(2)
IF (ITP .EQ. 0) RETURN
WRITE(2, 2010) IFNUM
2013 FORMAT(' FILE NO ', 14, ' CRC ERROR REWINDING')
IF (ITP .EQ. 2) GOTO 10
ECOM(1) = SDS(6)
ECOM(2) = IDRV
ECOM(3) = 0
CALL SDS10(ECOM, ESTAT, 18)
WRITE SECTION

0001:   NBUF=NBYT
0002:   IFLEN=(ILEN+3)/4
0003:   IF(IFLEN.LT.2)IFLEN=2
0004:   IPAD=(IFLEN*2048)-NBUF
0005:   CALL TWRIT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
0006:   IF(ISTAT.EQ.0)RETURN

WRITE ERROR DETECTED

0340:   ISTAT=ISTAT
0341:   GOTO 40
0342:   ITMP=ISTAT.AND.MASK(6)
0343:   ITPMP=ISTAT.AND.MASK(2)
0344:   IF(ITMP.EQ.1.AND.ITPMP.EQ.2)RETURN

REPORT AND RETRY

3049:   WRITE(2,282)IFNUM
3050:   282 FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
3051:   ECOM(1)=SDSCOM(6)
3052:   ECOM(2)=2
3053:   ECOM(3)=IDRV
3054:   ECOM(4)=0
3055:   CALL SDS10(ECOM,ESTAT, )
3056:   NBUF=8
3057:   IPAD=(IFLEN*2048)-NBUF
3058:   CALL TWRIT(ERRS,NBUF,ESTAT,IPAD,IFLEN,IDRV)
3059:   GOTO 20

WIND FORWARD ONE FILE

3062:   COM(1)=SDSCOM(5)
3063:   COM(2)=1
3064:   COM(3)=IDRV
3065:   COM(4)=0
3066:   CALL SDS10(COM,ISTAT, )

CLEAR IRRELEVANT BITS FROM ERROR BYTE

3068:   ISTAT=ISTAT.AND..NOT.MASK(6)
3069:   IF(ISTAT.EQ.2)RETURN
3070:   ISTAT=ISTAT
3071:   IF(ISTAT.NE.0)GOTO 40

IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
AS THIS WAS A DATA FILE NOT A SHORT RECORD

3077:   ECOM(1)=SDSCOM(6)
IN THIS SECTION THE MAIN TAPE ERRORS ARE HANDLED SUCH AS: TAPE BUSY, TAPE OFFLINE.

C TAPE BUSY SECTION...AFTER CLEARING B0T FLAG

C HAVING EXAMINED STATUS IF TAPE STILL BUSY, LOOP AGAIN, IF NOT TRY COMMAND AGAIN

C TAPE OFFLINE

C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED

C ERROR EXIT RETURN

C ERROR EXIT RETURN

C ERROR EXIT RETURN

C ERROR EXIT RETURN
Post-Stack Trace Mix :- MPTMIX

Input file......DK1:MPTMIX.DAT

Log file.......DK1:MPTMIX.LOG

Input Parameters

READ(1,1000)NFILES,NSAMP,TPDRR,TPDRW,INFLG,OUTFLG

1000 FORMAT(6I5)

NFILES....Number of files to process
NSAMP....Number of samples per trace
TPDRR....Input tape drive
TPDRW....Output tape drive
INFLG....Input flag
   0 - Input from tape
   1 - Input from disc
OUTFLG....Output flag
   0 - Output to tape
   1 - Output to disc

IF(INFLG.NE.0)READ(1,1100)FSPECR

1100 FORMAT(3A4)

FSPECR....Input files, 1 to NFILES

IF(OUTFLG.NE.0)READ(1,1100)FSPECW
FSPECW...output files, 1 to NFILES
C THIS IS A PROGRAM WHICH PRODUCES
C A WEIGHTED MIX OF THREE INPUT
C TRACES TO GIVE ONE OUTPUT TRACE
C
C REAL*8 FSPECR(256), FSPECW(256), FBUF
C REAL*4 SEIS(2048), FNBUF(3)
C INTEGER*2 BUF(4352), IBLK(256), IBLKS(256), OUTFLG
C EQUIVALENCE (BUF(1), IBLK(1)), (BUF(257), SEIS(1))
C LOGICAL*1 ISTAT, TPDRR, TPDRW
C DATA DEV/3RRK/

C SET UP I/O PARAMETERS
C
C IEOTR=I
C IEOTW=I
C IRD=IGETC()
C IWR=IGETC()
C IF(IFETCH(DEV).NE.I) STOP 'FETCH ERR'
C CALL ASSIGN(1, 'DK2:MPTMIX.DAT', 14)
C CALL ASSIGN(2, 'DK2:MPTMIX.LOG', 14)

C GET INPUT DATA
C
C READ(1, I000) NFILES, NSAMP, TPDRR, TPDRW, INFLG, OUTFLG
C 1000 FORMAT(6I5)
C 017 IF(INFLG.EQ.0) GOTO 10
C DO 20 J=1, NFILES
C IF(OUTFLG.EQ.0) GOTO 30
C 20 CONTINUE
C 10 CONTINUE
C CALL IRAO58(12, FNBUF, FBUF)
C FSPECR(J)=FBUF
C 20 CONTINUE
C DO 30 J=1, NFILES
C 30 CONTINUE
C READ(1, I001) FNBUF
C CALL IRAO58(12, FNBUF, FBUF)
C FSPECW(J)=FBUF
C 30 CONTINUE

C SET UP CONSTANTS AND INIT AP
C
C CALL APINIT
C CALL VCLR(18, 1, 3*NSAMP)
C NBLKTR=NSAMP/128
C NBLKW=NBLKTR+1
C NBYTR=NBLKWN512
C NWDR=NBYTR/2
C NOPS=NFILES+1
C IOUT=0
C SET UP AP ADDRESSES
C
C SET UP AP ADDRESSES
C
C MAIN OPS LOOP
C
C SWITCH VECTOR POSITIONS IN AP
C
C TAPE READ INPUT
C
C READ TO MEMORY
C
C WIND OVER EOF MARKER
C
C INPUT FROM DISC
C
C CONTINUE
C HEADER BLOCK MANAGEMENT

C PUT DATA IN AP

C SAVE HEADER BLOCK AND REPLACE WITH PREVIOUS ONE

C DO WEIGHTED MIX IN AP

C OUTPUT RESULTANT TRACE

C CHECK FOR ERRORS

C DISC OUTPUT
Trace sequential-Time slice :- MPSLIC

Input file......DK2:MPGLIC.DAT

Input Parameters

READ(1,1000)NCHAN,NSAMP

1000 FORMAT(2I5)

NCHAN....Number of input channels
NSAMP....Number of samples per channel

READ(1,1100)FSPECR

1100 FORMAT(3A4)

FSPECR....Input File - Trace sequential

READ(1,1100)FSPECW

FSPECW....Output File - Time sliced
THIS PROGRAM TAKES TRACE SEQUENTIAL DATA AND TIME SLICES IT FOR INPUT TO THE FD MIGRATION PROGRAM MPFMIG.

VIRTUAL BUFFER(16384), REAL*4 BUFFER, INBUF(128), OUTBUF(128), FBUF(3)
REAL*8 FSPECW, FSPECR
INTEGER*2 IAD(128), IWRTH(128)
DATA DEV/3RRK/

INPUT SET UP

IF(IFETCH(DEV).NE.0) STOP 'FETCH ERROR'
IRD=IGETC()
IWRTH=IGETC()
CALL ASSIGN(1, 'DK2:MPSLIC.DAT', 14)

READ IN DATA

READ(1, 100) NCHAN, NSAMP
100 FORMAT(2I5)
READ(1, 100) FBUF
101 FORMAT(3A4)
CALL IRADS(12, FBUF, FSPECR)
102 READ(1, 100) FBUF
103 CALL IRADS(12, FBUF, FSPECW)

SET UP CONSTANTS

NCHANW=0
NCHANW=NCHANW+128
IF(NCHAN.GT.NCHANW) GOTO 18
NBLANK=NCHANW-NCHAN
NBST=NBLANK/2
TIM=NCHANW/128
NBLKR=NSAMP/128
NFILW=(NCHANW*NBLKR)+1

SET UP BUFFER ADDRESSES

IT=1
DO 20 J=1, 128
IAD(J)=IT
20 IT=IT+128

CLEAR STORE BUFFER

DO 30 J=1, 16384
30 BUFFER(J)=0.0

SET UP I/O FILES
C

0033 IF(LOOKUP(IRD,FSPECW).LT.0)STOP 'LOOKUP ERROR'

0035 IF(IENTER(IWRT,FSPECW,NFILW).LT.0)STOP 'ENTER ERROR'

C

0037 START OF TRANSFER LOOP

0038 IBLRK=J

0039 IST=NBST

C

0040 SET UP OUTPUT DISC ADDRESSES

0041 DO 118 JJ=1,128

118 IPOS=IADCIS'+W)+IST

0042 SORT CODE

0043 DO 218 L=1,NCHAN

218 CONTINUE

0044 PUT DATA IN INT STORE

0046 DO 300 ISW=1,128

300 CONTINUE

0047 IPOS=IAD(ISW)+IST

0048 BUFFER(IPOS)=INBUF(ISW)

0049 OUTPUT CODE

0050 IST=IST+1

0051 IF(L.EQ.NCHAN)GOTO 218

0052 IF(IST.LT.128)GOTO 200

0053 200 CONTINUE

0054 WRITE OUT

0055 IST=0

0056 DO 220 LL=1,128

220 CONTINUE

0057 IPOS=IAD(LL)

0058 IBLK=IWTB(LL)

0059 DO 230 LS=1,128

230 CONTINUE

0060 OUTBUF(LL)=BUFFER(IPOS)

0061 BUFFER(IPOS)=0.0

0062 IPOS=IPOS+1

0063 230 CONTINUE

0064 IF(IWRTW(256,OUTBUF,IBLK,W,IWRT).LT.0)STOP 'WRITE ERROR'

0065 IWRTB(LL)=IBLK+1

0066 220 CONTINUE

0067 IWRTB(LL)=IBLK+1

0068 200 CONTINUE

0069 180 CONTINUE

0070 CALL CLOSE(IRD)

0071 CALL CLOSE(IWRT)

0072 STOP 'NORMAL TERMINATION'
**Time Slice to Trace Sequential :- MPUSLC**

Input file......DK2:MPUSLC.DAT

**Input Parameters**

```
READ(1,1000)NCHAN,NSAMP
```

1000 FORMAT(2I5)

NCHAN....Number of channels
NSAMP....Number of samples per channel

```
READ(1,1100)FSPECR
```

1100 FORMAT(3A4)

FSPECR...Input file - Time sliced

```
READ(1,1100)FSPECW
```

FSPECW...Output file - Trace sequential
C THIS PROGRAM TAKES THE OUTPUT FROM FD MIGRATION AND PUTS IT BACK INTO TIME SEQUENTIAL DATA

VIRTUAL BUFFER(16384)
REAL*4 BUFFER, INBUF(128), OUTBUF(128), FBUF(3)
REAL*8 FSPECW, FSPECR
INTEGER*2 IADC(128), IWRITB(128)
DATA DEV/3RRK/

INPUT SET UP

IF (IFETCH.DEV).NE.0 STOP 'FETCH ERROR'
IRD=IGETC();
IWRIT=IGETC();
CALL ASSIGN(1,'DK2:MPUSLC.DAT',14)

READ IN DATA

READ(1,1000) NCHAN, NSAMP
READ(1,1001) FBUF
READ(1,1002) FORMAT(215)
CALL IRAD50(12, FBUF, FSPECR)
READ(1,1003) FBUF
CALL IRAD50(12, FBUF, FSPECW)

SET UP CONSTANTS

NCHANW=NCHAN+128
IF (NCHAN.GT.NCHANW) GOTO 10
NBST=NBLANK/2
TIM=NCHANW/128
MBLK=NBLKW/128
MFILW=(NCHANW*NBLKW)+1

SET UP BUFFER ADDRESSES

IT=1
DO 20 J=1,128
IAD(J)=IT
20 IT=IT+128

CLEAR STORE BUFFER

DO 30 J=1,16384
30 BUFFER(J)=.0

SET UP I/O FILES
C START OF TRANSFER LOOP
C
INST=NBST+1
NCHLST=0
NCHAN=128-NBST
IST=0
NLEFT=NCHAN
NCHAN=NLEFT-NCHAN
NCHAN=128-NCHAN
NLEFT=NLEFT-NCHAN

C LOOP ON CHANNELS
C
DO 180 JL=1,ITIM
IBLKR=JL
DO 11 JJ=1,NCHAN
IWRTB(JJ)=((J-1)*NBLKW)+(NCHAN*NBLKW)+1
C LOOP ON SAMPLES
C
DO 216 L=1,NSAMP
IF(IREADW(256,INBUF,IBLKR,IRD).LT.6)STOP'READ ERROR'
IBLKR=IBLKR+ITIM
IOUT=INST
C SORT DATA
C
DO 300 LL=1,NCHAN
IPOS=IAD(LL)+1
BUFFER(IPOS)=INBUF(IOUT)
IOUT=IOUT+1
CONTINUE
IST=IST+1
IF(IST.LT.128)GOTO 200
C WRITE OUT CODE
C
DO 210 JS=1,NCHAN
IPOS=IAD(JS)
IBLKW=IWRTB(JS)
DO 220 LS=1,128
OUTBUF(LS)=BUFFER(IPOS)
BUFFER(IPOS)=0.0
CONTINUE
IF(IWRTW(256,OUTBUF,IBLKW,IWRT).LT.6)STOP'WRITE ERROR'
IWRTB(JS)=IBLKW+1
210 CONTINUE
IST=0
220 CONTINUE
INST=1
Finite Difference Migration: - MPFD15 and MPFD45

Input files....DK2:MPFD15.DAT or DK2:MPFD45.DAT

Input Parameters

READ(1,1000) NSAMP, MSAMP, NTRACE, NV, NVELS

1000 FORMAT(10I5)

NSAMP....Number of time samples per channel
MSAMP....Sample number to migrate down to
NTRACE....Number of "live" data traces
NV.......Number of velocity definition points
NVELS....Number of time/velocity pairs

READ(1,1001) DX, DT, DTOR

1001 FORMAT(3F10.0)

DX.......Distance in km between traces
DT.......Interval in seconds between samples
DTOR.....Migration interval in seconds

READ(1,1000)(IV(I),I=1,NV)

IV.......Positions at which velocity functions are defined

READ(1,1002) FSPECR
FSPECR...Data file, used in both input and output

There are then NV sets of velocity functions each with NVELS layers

READ(1,1003)(TLYR(I),VLYR(I),I=1,NVELS)

TLYR.....Two-way travel time
VLYR.....RMS velocity to this point
C M J POULTER NOV 81
C THIS IS A PROGRAM FOR
C 15 DEGREE FINITE DIFFERENCE MIGRATION
C
C VIRTUAL ASAVE(1024), BSAVE(1024), CSAVE(1024),
# RHS(6144), APSAVE(2048), IBLK(2048), VSAVE(2048),
# XSTOR(4096)
C REAL*8 FSPECR, FSVEL
C REAL*4 VINT(100), ADXST(4), VM(100), VRMS(20), FBUF(3),
# ADRHS(6), ADAPSV(2), ADXDIS(4)
C INTEGER*2 IV(100), IST(6), A0, A1, A2, A3, A4, A5, A6, A7,
# IT(20)
# A5/5120/, A6/6144/, A7/7168/
C DATA DEV/3RRK/, FSVEL/12ROK4MPVTMPDAT/
C
C SET UP CONSTANTS AND READ IN DATA
C
C CALL APINIT
C CALL ASSIGN( 'DK2:MPFMIG.DAT', 14)
C IF (IFETCH(.DEVS).NE.0) STOP 'FETCH ERROR'
C
C IVRT=IGETC()
C IRD=IGETC()
C
C READ IN DATA
C
C READ(1,1000) NSAMP, MSAMP, NTRACE, NV, NVELS
C 1000 FORMAT(1I5)
C 1015 READ(1,1001) DX, DT, DTOR
C 1016 1001 FORMAT(3F10.6)
C 1017 READ(1,1002)(IV(I), I=1,NV)
C 1018 READ(1,1002) FBUF
C 1002 FORMAT(3A4)
C 1020 CALL IRAD50(12, FBUF, FSPECR)
C
C SET UP VM ADDRESSES
C
C IPOS=1
C DO 10 J=1,6
C ADRHS(J)=APGAD(RHS(IPOS))
C 10 CONTINUE
C IPOS=IPOS+1024
C ADVEL=ADGET(VSAVE(1))
C ADAPSV(1)=APGAD(APSAVE(1))
C ADAPSV(2)=APGAD(APSAVE(1025))
C ADSV=APGAD(CSAVE(1))
C ADSV=APGAD(BSAVE(1))
C ADXS=APGAD(CSAVE(1))
C IB=1
C DO 15 I=1,4
C ADXST(I)=APGAD(XSTOR(1B))
C 15 CONTINUE
C ADXDIS(I)=ADGET(XSTOR(1B))
SET UP CONSTANTS

NCHANW = 0
NCHANW = NCHANW + 128
IF (NTRACE.GT.NCHANW) GOTO 2B
IF (NCHANW.GT.1024) STOP ' TOO MANY TRACES TO MIGRATE'

ITIM = NCHANW/128
NBST = (NCHANW - NTRACE)/2
NEND = NCHANW - NTRACE - NBST

SET UP CONSTANTS FOR LOOPS

ITOR = DTOR/DT
ITORM = MSAMP/ITOR
LIMIT = NSAMP
NV = NV - 1
ACOF = (1.5/800.25) * DTOR * DT / (16.0 * DX * DX)

L = 1
DO 3S L = 2, NSAMP
3B = IBLKCL + IBLKCL - L + ITIM
NVSIZ = 1
NVSIZ = NVSIZ + 128
IF (ITORM.GT.NVSIZ) GOTO 4S

NCHWM1 = NCHANW - 1
NCHWR2 = NCHANW * 2
ITRLM2 = ITORM * 2
IA01 = A0 + 1
IA31 = A3 + 1
IA41 = A4 + 1

SET UP FILES

IF (LOOKUP (IRD, FSPEC), LT.0) STOP 'LOOKUP ERROR'
IF (FINTER (IVRT, FSVEL, IFILV), LT.0) STOP 'ENTER ERROR'

SET UP I AND I/12

CALL VCLRCA4, 1, NCHANW
CALL VTSADD (A4, 1, 2B49, A4, 1, NCHANW)
CALL VMOVCA4 (A4, 1, A5, 1, NCHANW)
CALL VTSMULCA4 (A4, 1, 2531, A4, 1, NCHANW)
CALL VTSMULCA4 (A4, 1, 4427, A4, 1, NCHANW)
CALL APUR
CALL APUR
CALL APGETA (A4, NCHANW, 2, ADAPSV(1))
CALL APGETA (A5, NCHANW, 2, ADAPSV(2))

SET UP INT VELOCITIES AT EACH DTOR VALUE BY INTERPOLATING
THEN CONVERTING THE RMS VALUES
C AND THEN WRITE OUT TO A TEMP FILE
C

IBLVK=0
DO 55 LV=1,NV
IV(LV)=IV(LV)+NBST
C
READ IN VELS
DO 65 LL=1,NVELS
READ(1,1883)T,VRMS(LL)
FORMAT(2F10.0)
IT(LL)=T/DTOR+1
65 CONTINUE
C
DO LINEAR INTERP ON RMS VELS
N1=1
N2=IT(1)
DO 55 LI=N1,N2
VSAVE(LI)=VRMS(LI)
IF(NVELS.EQ.1)GOTO 55
DO 55 LI=2,NVELS
N1=N2+1
N2=IT(LL)
VT=VRMS(LL-1)
DELV=(VRMS(LL)-VT)/(N2-N1-1)
DO 75 LT=N1,N2
VSAVE(LT)=VT
VT=VT+DELV
75 CONTINUE
55 CONTINUE
N1=N2+1
N2=ITORLM+1
DO 65 LL=1,N2
VSAVE(LL)=VRMS(NVELS)
C
CHANGE INTO INT VELS
VTP1=0.0
VTP2=VSAVE(1)*VSAVE(1)
DO 80 LINT=1,ITORLM
VTP1=VTP2
VTP2=VSAVE(LINT+1)*VSAVE(LINT+1)*(LINT+1)
VSAVE(LINT)=SQRT(VTP2-VTP1)
80 CONTINUE
IF(IWRIT(IVRT,ITRLM2,IBLVK,ADVEL).LT.0)STOP 'WRITV ERROR'
IBLVK=IBLVK+NSIZE
CALL IWAIT(IVRT)
CONTINUE
C
START MAIN LOOP
DO 100 ITORCT=1,ITORLM
LIMIT=LIMIT-ITOR
C
C ZERO X ARRAYS
C
CALL VCLR(AB,1,NCHANW)
CALL APWR
DO 110 I=1,4
IST(I)=I
FAD=ADXST(I)
CALL APGETA(AB,NCHANW,2,FAD)
110 CONTINUE
C
C READ IN VELOCITIES
C
IBLV=1
DO 128 L=1,NV
IF (IREAD(IVRT,ITRLM2,IBLV,ADVEL).LT.0) STOP 'READV ERROR'
IBLV=IBLV+NVSIZ
CALL IWAIT(IVRT)
V(L)=VSAVECT=ITORCT)
128 CONTINUE
C
C GEN V SLICE
C
CALL APPUT(V,AB,NV,2)
DO 138 I=1,NVM
VINT(I)=(V(I+1)-V(I))/(IV(I+1)-IV(I))
CALL APPUT(VINT,A1,NVM,2)
CALL APWR
DO 148 I=1,NVM
NVD=IV(I+1)-IV(I)+1
CALL VRAMP(AB+I-1,A1+I-1,A2+IV(I)-1,1,NVD)
CONTINUE
CALL VFILL(A2+IV(NV)-1,A2+IV(NV),1,NEND)
CALL VSQ(A2,1,1,NCHANW)
C
C GEN A
C
CALL APPUT(ACOF,AB,1,2)
CALL APWR
CALL VSMUL(A2,1,AB,AB,1,NCHANW)
C
C SET UP NECESSARY COEFFS FROM SAVE
C
CALL APWR
CALL APPUTA(A1,NCHANW,2,ADAPSV(1))
CALL APPUTA(A2,NCHANW,2,ADAPSV(2))
C
C GEN COE EF S
C
C \( (I+(B-A)T) \)
C
$\texttt{156}$ CALL VSUB($A_0, A_1, A_2, A_3, 1, NCHANW)$
$\texttt{157}$ CALL VTSMUL($A_3, 1, 2B_0, A_4, 1, NCHANW)$
$\texttt{158}$ CALL VNEG($A_4, 1, A_4, 1, NCHANW)$
$\texttt{159}$ CALL VADD($A_2, 1, A_4, 1, A_4, 1, NCHANW)$

C
C \( (I+(A+B)T) \)
C
$\texttt{160}$ CALL VADD($A_0, A_1, A_1, A_5, 1, NCHANW)$
$\texttt{161}$ CALL VTSMUL($A_5, 1, 2B_0, A_6, 1, NCHANW)$
$\texttt{162}$ CALL VNEG($A_6, 1, A_6, 1, NCHANW)$
$\texttt{163}$ CALL VADD($A_2, 1, A_6, 1, A_6, 1, NCHANW)$
$\texttt{164}$ CALL APWR
$\texttt{165}$ CALL APGETA($A_5, NCHANW, 2, ADRHS(3))
$\texttt{166}$ CALL APGETA($A_6, NCHANW, 2, ADRHS(4))
$\texttt{167}$ CALL APGETA($A_5, NCHANW, 2, ADRHS(5))
$\texttt{168}$ CALL APGETA($A_6, NCHANW, 2, ADRHS(6))
$\texttt{169}$ CALL APWD
$\texttt{170}$ CALL VNEG($A_3, 1, A_5, 1, NCHANW)$
$\texttt{171}$ CALL VNEG($A_4, 1, A_6, 1, NCHANW)$
$\texttt{172}$ CALL APWR
$\texttt{173}$ CALL APGETA($A_5, NCHANW, 2, ADRHS(11))
$\texttt{174}$ CALL APGETA($A_6, NCHANW, 2, ADRHS(2))
$\texttt{175}$ CALL APGETA($A_3, NCHANW, 2, ACSV)$
$\texttt{176}$ CALL APWD
$\texttt{177}$ CALL VMOV($A_3, 1, A_5, 1, NCHANW)$

C
C PARTIALLY SOLVE AND SAVE RES
C
$\texttt{178}$ CALL FACTOR($A_3, 1, A_4, 1, A_5, 1, A_6, 1, A_7, 1, NCHANW)$
$\texttt{179}$ CALL APWR
$\texttt{180}$ CALL APGSP(15, IER)
$\texttt{181}$ IF(IER.NE.0) STOP 'FACTOR ERROR'
$\texttt{183}$ CALL APGETA($A_6, NCHANW, 2, ACSV)$
$\texttt{184}$ CALL APGETA($A_7, NCHANW, 2, ACSV)$
$\texttt{185}$ CALL APWD

C
C START ON RHS
C
C
C LOOP ON SAMPLES
C
$\texttt{186}$ ISTB=NSAMP
$\texttt{187}$ DO 200 ITR=1,LIMIT
$\texttt{188}$ IBLK=IBLK(ISTB)
$\texttt{189}$ FAD=ADXDIS(IST(3))
$\texttt{190}$ CALL IWAIT(IRD)
$\texttt{191}$ IF(IREAD(IRD, NCHNR2, IBLK, FAD).LT.0) STOP 'READ ERROR'
$\texttt{193}$ CALL VCLR($A_7, 1, NCHANW)$
$\texttt{194}$ IA1=-1
$\texttt{195}$ IA2=3
$\texttt{196}$ CALL IWAIT(IRD)
DO 210 I=1,3
IA1=IA2+1
IA2=IA1+1
CALL APWR
CALL APPUTA(A0,NCHANW,2,ADRHS(IA1))
CALL APPUTA(A1,NCHANW,2,ADRHS(IA2))
FST=ADXST(IST(I))
CALL APPUTA(A3,NCHANW,2,FST)
CALL APVD
CALL VMUL(A1,1,A3,1,A4,1,NCHANW)
CALL VMA(AA0,1,AA3,1,AA4,1,IA41,1,IA41,1,NCHWM1)
CALL VMA(AA0,1,IA31,1,A4,1,A4,1,NCHWM1)
CALL VADD(AA4,1,AA7,1,AA7,1,NCHANW)
CONTINUE
DO SOLUTION TO EQN
CALL VCLR(A3,1,NCHANW)
CALL VCLR(A4,1,NCHANW)
CALL APWR
CALL APPUTA(A0,NCHANW,2,ADASV)
CALL APPUTA(A1,NCHANW,2,ADBSV)
CALL APPUTA(A2,NCHANW,2,ADCSV)
CALL APVD
CALL SOLVE(A0,1,AA1,1,AA7,1,AA2,1,AA3,1,AA4,1,NCHANW)
CALL APWR
CALL APGSP(IER,15)
IF(IER.NE.0)STOP 'SOLVE FAILURE'
FST=ADXST(IST(4))
CALL APGETA(A4,NCHANW,2,FST)
FAD=ADXDIS(IST(4))
CALL APVD
IF(IWRITE(IRD,NCHWR2,IBLKR,FAD).LT.0)STOP 'WRITE ERROR'
TURN AROUND VECTORS
IST(5)=IST(1)
IST(6)=IST(2)
DO 220 I=1,4
ISTB=ISTB-1
END OF LOOP
CONTINUE
CALL IWAIT(IRD)
CONTINUE
CALL CLOSEC(IRD)
STOP 'NORMAL TERMINATION'
END
THIS IS A PROGRAM FOR
45 DEGREE FINITE DIFFERENCE MIGRATION

VIRTUAL ASAVE(1024), BSAVE(1024), CSAVE(1024),
RHS(1024), APSAVE(2048), IBLK(2048), VSAVE(2048),
XSTOR(6144)

REAL*8 FSPEC, FSVEL
REAL*4 VINT(100), ADXST(6), VRMS(20), FBUF(3),
ADRH(100), ADAPS(2), ADXDIS(6)
INTEGER*2 IV(100), IST(8), A0, A1, A2, A3, A4, A5, A6, A7,
IT(20)

DATA DEV/3 RRRK/, FSVEL/12 RRRK MP VMP DAT/

SET UP CONSTANTS AND READ IN DATA

CALL APINIT
CALL ASSIGN(1, 'DK2: MPMIG MP DAT', 14)
IF(IFETCH(DEV).NE.0) STOP 'FETCH ERROR'
IVRT = IGETC()
IRD = IGETC()

READ IN DATA
READ(1, 1001) NSAMP, MSAMP, NTRACE, NV, NVELS
1001 FORMAT(10I5)
READ(1, 1001) DX, DT, DTOR
1001 FORMAT(3F10.8)
READ(1, 1002) IV(I), I = 1, NV
1002 FORMAT(3A4)
READ(1, 1002) FBUF
CALL IREAD(12, FBUF, FSPEC)

SET UP VM ADDRESSES
IPOS = 1
DO 10 J = 1, 10
ADRHS(J) = APGAD(RHS(IPOS))
IPOS = IPOS + 1024
10 CONTINUE
ADVEL = ADGET(VSAVE(1))
ADPSV(1) = APGAD(APSAVE(1))
ADPSV(2) = APGAD(APSAVE(1025))
ADAV = APGAD(CSAVE(1))
ADCV = APGAD(CSAVE(1))
IB = 1
DO 15 I = 1, 6
ADXST(I) = APGAD(XSTOR(IB))
15 CONTINUE
ADXDIS(1) = ADGET(XSTOR(IB))
!if A
8836
IB=IB+1

CONTINUE
8837

C SET UP CONSTANTS
C
8838
NCHANW=0
8839
20 NCHANW=NCHANW+128
8840
IF(NTRACE.GT.NCHANW)GOTO 20
8841
IF(NCHANW.NCHAMW+128)STOP ' TOO MANY TRACES TO MIGRATE'
8842
ITIM=NCHANW/128
8843
NBST=(NCHANW-NTRACE)/2
8844
NEND=NCHANW-NTRACE-NBST
C SET UP CONSTANTS FOR LOOPS
C
8847
ITOR=DTOR/DT
8848
ITORLM=MSAMP/ITOR
8849
LIMIT=NSAMP
8850
NV=NV-1
8851
ACOF=(0.279*DT*DT)/(4.0*DX*DX)
8852
BCOF=(ITOR*DT)/(32.0*DX*DX)
8853
IBLK(1)=1
8854
DO 3 N=2,NSAMP
8855
39 IBLK(L)=IBLK(L-1)+ITIM
8856
40 NVSIZ=0
8857
NVSIZ=NVSIZ+128
8858
IF(ITORLM.GT.NVSIZ)GOTO 40
8859
NVSIZ=NVSIZ/128
8860
NCHNW1=NCHNW-1
8861
NCHNW2=NCHNW*2
8862
ITRLM2=ITORLM*2
8863
IA01=IA0+1
8864
IA01=IA3+1
8865
IA01=IA4+1
C SET UP FILES
C
8866
IF(LOOKUPCIRD,FSPECR.LT.0)STOP 'LOOKUP ERROR'
8867
IFILV=NV*NVSIZ
8868
IF(IENTER<IVRT,FSVEL,IFILV.LT.0)STOP 'ENTER ERROR'
C SET UP I AND I/12
C
8869
CALL VCLR(A4,1,NCHANW)
8870
CALL VTSMUL(A4,1,2331,A4,1,NCHANW)
8871
CALL APGETA(A4,NCHANW,2,ADAPSV(1))
8872
CALL APGETA(A5,NCHANW,2,ADAPSV(2))
C SET UP INT VELOCITIES AT EACH
C DTOR VALUE BY INTERPOLATING
C THEN CONVERTING THE RMS VALUES
C AND THEN WRITE OUT TO A TEMP FILE

C
IBLK=0
DO 50 LV=1,NV
IV(LV)=IV(LV)+NBST
C
READ IN VELS
DO 60 LL=1,NVELS
READ(1,1003)IT,VRMS(LL)
1003 FORMAT(2F10.0)
IT(LL)=IT/DTOR+1
60 CONTINUE
C
DO LINEAR INTERP ON RMS VELS
N1=1
N2=IT(1)
DO 55 LI=N1,N2
VSAVE(LI)=VRMS(LI)
55 IF(NVELS.EQ.1)GOTO 65
DO 70 LJ=2,NVELS
70 VT=VRMS(LJ-1)
DELV=(VRMS(LJ)-VT)/(N2-N1-1)
DO 75 LT=N1,N2
75 VSAVE(LT)=VT
VT=VT+DELV
75 CONTINUE
65 CONTINUE
N1=NZ+1
N2=ITORLM+1
DO 80 LL=N1,N2
80 VSAVE(LL)=VRMS(NVELS)
C
CHANGE INTO INT VELS
VT1=0.0
VT2=VSAVE(1)*VSAVE(1)
DO 90 LINT=1,ITORLM
90 VT=VT2
VT2=VSAVE(LINT+1)*VSAVE(LINT+1)*(LINT+1)
VSAVE(LINT)=SORT(VT2-VT1)
90 CONTINUE
IF(IWRITA(IVRT,ITORLM,IBLK,ADVEL).LT.0)STOP 'WRITV ERROR'
IBLK=IBLK+NVSTZ
CALL IWAIT(IVRT)
50 CONTINUE
C
START MAIN LOOP
DO 180 ITORCT = 1, ITORM
LIMIT = LIMIT - ITOR
C ZERO X ARRAYS
CALL VCLR(A0,1,NCHANW)
CALL APWR
DO 11 1 = 1, 6
IST(I) = I
FAD = ADXST(I)
CALL APGETA(A0,NCHANW,2,FAD)
110 CONTINUE
C READ IN VELOCITIES
IBLK = 0
DO 120 L = 1, NV
IF (IREAD(AIVRT,ITRLM2,IBLK,ADVEL).LT.0) STOP 'READV ERROR'
IBLK = IBLK + NVSIZ
CALL IWAIT(AIVRT)
V(L) = VSAVE(ITORCT)
120 CONTINUE
C GEN V SLICE
CALL APPUT(V,A0,NV,2)
DO 130 I = 1, NVM
VINT(I) = (V(I+1) - V(I))/(IV(I+1) - IV(I))
CALL APPUT(VINT,A1,NVM,2)
CALL APWD
DO 140 I = 1, NVM
NVD = IV(I+1) - IV(I) + 1
CALL VRAMP(A0+I-1,A1+I-1,A2+IV(I)-1,1,NVD)
140 CONTINUE
CALL VFILL(A2+IVC-1,A2+IVC,NBST)
CALL VFILL(A2+IV(NV)-1,A2+IV(NV),1,NEND)
CALL VSQ(A2,1,A2,1,NCHANW)
C GEN A,B
CALL APPUT(A0F,A0,1,2)
CALL APPUT(B0F,A1,1,2)
CALL APWD
CALL VSMUL(A2,1,A0,F,1,NCHANW)
CALL VSMUL(A2,1,A1,F,1,NCHANW)
C SET UP NECESSARY COEFFS FROM SAVE
CALL APWR
CALL APPUT(A2,NCHANW,2,ADAPSV(1))
CALL APPUT(A3,NCHANW,2,ADAPSV(2))
CALL APWD
C GEN COEFFS
C
C (Z+(2B+ACOF)*T)
C
0159 CALL VSIMUL(A2,1,2B50,A4,1,NCHANW)
0160 CALL VADD(A5,1,A4,1,A4,1,NCHANW)
0161 CALL VNEG(A4,1,A5,1,NCHANW)
0162 CALL VADD(A3,1,A5,1,A5,1,NCHANW)
0163 CALL VSIMUL(A5,1,2B50,A5,1,NCHANW)
C
C (I+(B+BCOF)*T)
C
0164 CALL VADD(A1,1,A2,1,A6,1,NCHANW)
0165 CALL VNEG(A6,1,A7,1,NCHANW)
0166 CALL VSIMUL(A7,1,2B50,A7,1,NCHANW)
0167 CALL VADD(A3,1,A7,1,A7,1,NCHANW)
0168 CALL APWR
C
C SAVE CALCD COEFFS
C
0169 CALL APGETA(A4,NCHANW,2,ADRHS(7))
0170 CALL APGETA(A5,NCHANW,2,ADRHS(8))
0171 CALL APGETA(A6,NCHANW,2,ADRHS(9))
0172 CALL APGETA(A7,NCHANW,2,ADRHS(10))
0173 CALL APWD
C
C GET -VE OF ABOVE COEFFS AND SAVE
C
0174 CALL VNEG(A4,1,A4,1,NCHANW)
0175 CALL VNEG(A5,1,A5,1,NCHANW)
0176 CALL VNEG(A6,1,A6,1,NCHANW)
0177 CALL VNEG(A7,1,A7,1,NCHANW)
0178 CALL APWR
0179 CALL APGETA(A4,NCHANW,2,ADRHS(5))
0180 CALL APGETA(A5,NCHANW,2,ADRHS(6))
0181 CALL APGETA(A6,NCHANW,2,ADRHS(3))
0182 CALL APGETA(A7,NCHANW,2,ADRHS(4))
0183 CALL APWD
C
C (I+(B-BCOF)*T)
C
0184 CALL VSUB(A1,1,A2,1,A4,1,NCHANW)
0185 CALL VNEG(A4,1,A5,1,NCHANW)
0186 CALL VSIMUL(A5,1,2B50,A5,1,NCHANW)
0187 CALL VADD(A3,1,A5,1,A5,1,NCHANW)
0188 CALL APWR
0189 CALL APGETA(A4,NCHANW,2,ADRHS(1))
0190 CALL APGETA(A5,NCHANW,2,ADRHS(2))
0191 CALL APGETA(A4,NCHANW,2,ADCSV)
0192 CALL APWD
C
C PARTIALLY SOLVE AND SAVE RES
C
CALL VMOV(A4,1,A6,1,NCHANW)
CALL FACTOR(A4,1,A5,1,A6,1,A3,1,A7,1,NCHANW)
CALL APWR
CALL APGSP(IER)
IF(IER.NE.0)STOP 'FACTOR ERROR'
CALL APGETA(A3,NCHANW,2,ADASV)
CALL APGETA(A7,NCHANW,2,ADBSV)
CALL APWD

C START ON RHS

C LOOP ON SAMPLES

C ISTB=NSAMP
DO 280 ITR=1,LIMIT
IBLKR=IBLK(ISTB)
FAD=ADXDIS(IST(5))
CALL IWAIT(IDR)
IF(IREAD(IDR,NCHWR,IBLKR,FAD).LT.0)STOP 'READ ERROR'
CALL VCLR(A7,1,NCHANW)
IA1=-1
IA2=0
CALL IWAIT(IDR)
DO 210 I=1,5
IA1=IA2+1
IA2=IA1+1
CALL APWR
CALL APPUTA(A8,NCHANW,2,ADRHS(IA1))
CALL APPUTA(A1,NCHANW,2,ADRHS(IA2))
FST=ADXS(IST(I))
CALL APPUTA(A3,NCHANW,2,FST)
CALL APWD
CALL VMUL(A1,1,A3,1,A4,1,NCHANW)
CALL VMUL(A8,1,IA31,1,A4,1,IA41,1,NCHWM1)
CALL VMA(A8,1,IA31,1,A4,1,IA41,1,NCHWM1)
CALL VADD(A4,1,A7,1,A7,1,NCHANW)
210 CONTINUE

C DO SOLUTION TO EQN

C CALL VCLR(A3,1,NCHANW)
CALL VCLR(A4,1,NCHANW)
CALL APWR
CALL APGSP(IER,15)
IF(IER.NE.0)STOP 'SOLVE FAILURE'
FST=ADXS(IST(6))
CALL APGETA(A4,NCHANW,2,FST)
FAD=ADXD(S(IST(6)))
CALL APWD
IF(IWRITA(IRD,NCHWR2,IBLKR,FAD).LT.0)STOP 'WRITE ERROR'
C TURN AROUND VECTORS
IST(7)=IST(1)
IST(8)=IST(2)
DO 2211 I=1,6
IST(I)=IST(I+2)
ISTB=ISTB-1
C END OF LOOP
C
2211 CONTINUE
CALL IWAIT(IRD)
100 CONTINUE
CALL CLOSEC(IRD)
STOP 'NORMAL TERMINATION'
END
Kirchhoff Migration Operator Design :- MPOGEN

Input file.....DK2:MPOGEN.DAT

Input Parameters

READ(1,1000)TSTEP,XSTEP

1000 FORMAT(2F10.0)

TSTEP....Sample interval in seconds
XSTEP....Trace spacing in kms

READ(1,1001)NHLFWD,NINC,NSAMP,NSTEP,IFRNGE,NVEL

1001 FORMAT(6I5)

NHLFWD....Half-width of migration operator
NINC.....Step between operator traces
NSAMP....Number of samples per trace
NSTEP....Operator update step
IFRNGE....Update flag
<0 calculate update using allowed percentage error
>0 use NSTEP to update operator
NVEL.....Number of velocity layers

READ(1,10G2)FSPECO

1002 FORMAT(3A4)
FSPECO...Operator output file

`READ(1,1000)TMIN,TMAX`

TMIN.....Start time for migration, seconds
TMAX.....End time for migration, seconds

`READ(1,1000)(TLYR(I),VLYR(I),I=1,NVEL)`

TLYR.....Two-way travel time
VLYR.....RMS velocity at this time

`READ(1,1000)D`

D........Percentage error allowed in calculating operator update positions.
THIS PROGRAM CALCULATES A SET OF MIGRATION OPERATORS FOR A GIVEN EARTH MODEL.

DATA ARRAYS:

- REAL*8 FSPEC
- REAL*4 OPVAL(256), FBUF(3), VEL(20), DZ(20)
- INTEGER*2 A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, #11, A12, A13, A14, A15, AIOP, A(5), NRANGE(512), NLEAD(512), #17(20), IOFF(20), AC(5), AS, ACON

READ IN DATA:

- IF(FETCHCDEV*.NE.*.)STOP 'FETCH ERROR'
- IRD=IGETC FBUF

SET UP DATA:

- NBUF=NHLFW(0)/S
- NBUF2=2*NBUF

SET UP CONSTANTS:

- PI=3.141592
- CONST=-1.0/(PI*TSTEP)
- NBUF=NHLFW*5
- NBUF2=2*NBUF

SET UP RANGE OF EACH OPERATOR:

- DO 30 J=1,NVEL

SET UP SYSTEM I/O:

- CALL ASSIGN(1, 'DK2:MPOGEN.DAT', 14)

READ IN DATA:

- READ(1, 1000) TSTEP, XSTEP
- READ(1, 1001) NHLFWD, NINC, NSAMP, NSTEP, IFRNGE, NVEL
- READ(1, 1002) FBUF
- READ(1, 1003) FORMAT(3A4)
- CALL IRADF(12, FBUF, FSPEC)
- READ(1, 1004) TMIN, TMAX
- ITMIN=TMIN/TSTEP
- ITMAX=TMAX/TSTEP
- DO 20 J=1,NVEL

CONTINUE:

- IT(J)=TPOS/TSTEP
- VEL(J)=VEL(J)/2.0
IST=0
5 IST=IST+1
IF (ITMIN.GT.IST) GOTO 5
IA=IST
ITMA=ITMIN
NOPTOT=0
IPNOP=1
IVS=0
15 ITM=ITMA
ITMA=ITM(IA)
CALL OPCALC(ITMI,ITMA,NRANGE,IPNOP,NSTEP,IFRNGE,NOP)
NOPTOT=NOPTOT+NOP
IVS=IVS+1
IP(IVS)=NOP
IA=IA+1
ITMA=ITM(IA)
IF (ITMA.LT.ITMAX) GOTO 15
NRANGE(IPNOP-1)=ITMAX
NOUT=(NOPTOT+1)

C SET UP I/O CONSTANTS
IBNOP=0
NBRNG=(NOUT+255)/256
NINLD=(NHLFWD+255)/256
IOPINC=(NBUF+127)/128
NBLD=NOPTOT*NINLD
NBOPS=NOPTOT*IOPINC
NFILS=NBOPS+NBLD+NBRNG+1
INRNGS=1
INLDST=INLDST+NBRNG
INOPST=INOPST+NBRG

C OPEN OUTPUT FILE
IF (IENTER(IRD,FSPEC,NFILS).LT.0) STOP 'ENTER ERROR'
C WRITE OUT NOPTOT
IF (IWRITW(1,NOPTOT,IRD).LT.0) STOP 'NOPWRITW ERROR'
C WRITE OUT NRANGE
IF (IWRITW(NOUT,NRANGE,1,IRD).LT.0) STOP 'WRITW ERR'
C CALC XSQ
CALL APINIT
CALL VCLR(A0,1,NHLFWD)
CALL APWR
CALL APPUT(XSTEP,A0+1,1,2)
CALL APWD
CALL VRAMP(A8,A8+1,A8,1,NHLFWD)
CALL VSQ(A8,1,A8,1,NHLFWD)
CALL APWR

C GET IT VECTOR

CALL APPUT(NRANGE,A8,NOPTOT+1,1)
CALL APWD
CALL VFLT(A8,1,A8,1,NOPTOT+1)
CALL VADD(A8,1,A8+1,1,A7,1,NOPTOT)
CALL VTSMUL(A7,1,2327,A6,1,NOPTOT)
CALL APWR

C LOOP ON DIFFERENT VELOCITIES

IADD=-1
IBLKNL=INLDST
IBLKP=INOPST
DO 2 IV=1,IVS
IAOP=-1
NOP=IOP(IV)
DZINV=1.0/DZ(IV)

C GET (IT*DZ)**2

CALL APPUT(DZ(IV),A7,1,2)
CALL APWD
CALL VSMUL(A8,1,A7,1,NOPTOT)
CALL VSQ(A7,1,A7,1,NOPTOT)

C LOOP ON OPERATORS

DO 3 IL=1,NOP
IADD=IADD+1
IAOP=IAOP+1

C CALC OTHER INTERMEDIATE FACTORS
C USED IN OPERATOR CALCULATION

CALL VSADD(A8,1,A7+IAOP,A1,1,NHLFWD)
CALL VSORT(A1,1,A1,1,NHLFWD)
CALL APWR
CALL APPUT(CONST,A15,1,2)
CALL APWD
CALL VTSMUL(A6+IADD,1,A15,A3,1,1)

C FAC1

CALL VFILL(A3,A3+1,1,NHLFWD-1)
CALL VDIV(A1,1,A3,1,A2,1,NHLFWD)

C FAC2
CALL APWR
CALL APPUT(DZINV,A15,1,2)
CALL APWD
CALL VSMUL(A1,A15,A3,1,NHLFWD)

c K
CALL VINT(A3,1,A5,1,NHLFWD)
DELTA
CALL VSMUL(A1,1,A15,A3,1,NHLFWD)

CALL VSADD(A5,1,A8+IADD,A5,1,NHLFWD)
CALL VSUB(A6+IADD,A5,1,A5,1,NHLFWD)
CALL VTSADD(A5,1,2*49,A5,1,NHLFWD)
CALL VFIX(A5,1,A5,1,NHLFWD)
CALL APWR
CALL APGET(NLEAD,A5,NHLFWD,1)
CALL APWD
IF(IWR ITW<NHLFWD,NLEAD,IBLKNL,IRD).LT.9 STOP 'NLWR ERR'
IBLKNL=IBLKNL+NINLD

GEN OPERATOR FROM INTERMEDIATE VALUES
DO 40 I=1,5
AS=A(I)
ACON=AC(I)
CALL VTSADD(A4,1,ACON,AS,1,NHLFWD)
CALL VDIV(A3,1,AS,1,AS,1,NHLFWD)
CALL VTSADD(A5,1,2*58,A9,1,NHLFWD)
CALL VMUL(A5,1,A9,1,A5,1,NHLFWD)
CALL VSQRT(A5,1,A5,1,NHLFWD)
CALL VMUL(A5,1,A2,1,AS,1,NHLFWD)
40 CONTINUE

CALL VNEG(A11,1,A2,1,NHLFWD)

CALL VTSMUL(A11,1,2*58,A3,1,NHLFWD)
CALL VSUB(A12,1,A3,1,A3,1,NHLFWD)

CALL VTSMUL(A12,1,2*58,A9,1,NHLFWD)
CALL VSUB(A11,1,A9,1,A9,1,NHLFWD)
CALL VSUB(A13,1,A9,1,A9,1,NHLFWD)
FORTRAN IV  
V02.04  
THU 08-JAN-81 00:50:06  

C  
0139  CALL VTSMUL(A13,1,2058,3,A10,1,NHLFWD)  
0140  CALL VSUB(A12,1,A10,1,A10,1,NHLFWD)  
0141  CALL VSUB(A14,1,A10,1,A10,1,NHLFWD)  

C OP5  
0142  CALL VTSMUL(A14,1,2058,11,1,NHLFWD)  
0143  CALL VSUB(A13,1,A11,1,A11,1,NHLFWD)  
0144  CALL VSUB(A15,1,A11,1,A11,1,NHLFWD)  

C WRITE OUT OPERATOR  
0145  CALL APWR  
0146  IAD=I  
0147  CALL APGET(OPVAL(IAD),A2,NHLFWD,2)  
0148  IAD=IAD+NHLFWD  
0149  CALL APGET(OPVAL(IAD),A3,NHLFWD,2)  
0150  IAD=IAD+NHLFWD  
0151  CALL APGET(OPVAL(IAD),A9,NHLFWD,2)  
0152  IAD=IAD+NHLFWD  
0153  CALL APGET(OPVAL(IAD),A10,NHLFWD,2)  
0154  IAD=IAD+NHLFWD  
0155  CALL APGET(OPVAL(IAD),A11,NHLFWD,2)  
0156  CALL APWR  
0157  IF(IWRITCNBUF2.OPVAL.IBLKOP.IRD).LT.01STOP'OPWRIT ERR'  
0158  IBLKOP=IBLKOP+10PINC  
0159  30 CONTINUE  
0160  20 CONTINUE  
0161  CALL CLOSEC(IRD)  
0162  STOP 'NORMAL TERMINATION'  
0164  END  

FORTRAN IV  
V02.04  
THU 08-JAN-81 00:51:10  

0001  SUBROUTINE OPCALC(ITMIN,ITMAX,NRANGE,IPOS,NST,IFLG,NOP)  
C  
C THIS SUBROUTINE USES THE FORMULA  
C STEP SIZE=K*D/50  
C K=CURRENT SAMPLE NO  
C D=X ERROR  
C  
C INTEGER*2 NRANGE(512)  
0033  DATA IN/0/  
0034  IN=IN+1  
0035  IF(IFLG.LE.0.AND.IN.EQ.1)READ(1,10000)D  
0037  FORMAT(F10.0)  
0038  NOP=0  
0039  NRANGE(IPOS)=ITMIN  
0040  IPOS=IPOS+1  
0041  I=ITMIN-1  
0042  IF(IFLG.LE.0)N=I*(D/50.0)  
0044  10 NOP=NOP+1  
0045  IF(IFLG.GT.0)GOTO 20  
0047  N=I+2*N-1  
0048  20 I=I+2*N-1  
0049  GOTO 30  
0050  30 I=1-NST  
0051  NRANGE(IPOS)=I  
0052  IPOS=IPOS+1  
0053  IF(IPOS.GT.512)STOP'TOO MANY OPERATORS'  
0057  IF(I.LT.ITMAX)GOTO 10  
0059  RETURN  
0060  END
Kirchhoff Migration :- MPKMIG

Input File......DK2:MPKMIG.DAT

Input Parameters

READ(1,1000)NTRACE,NO,NSAMP,NHLFWD

1000 FORMAT(4I5)

NTRACE....Number of traces in data file
NO.......Sample number of first sample in each trace
NSAMP....Number of samples per trace
NHLFWD....Half-width of operator

READ(1,1000)ISTART,NSTART,NSTOP,NINC

ISTART....First sample to migrate
NSTART....First trace to migrate
NSTOP....Last trace to migrate
NINC.....Trace increment in migration

READ(1,1001)XSTEP,TSTEP

1001 FORMAT(2F10.0)

XSTEP....Trace spacing in kms
TSTEP....Sample interval in seconds

READ(1,1002)FSPECO
1002 FORMAT(3A4)

FSPECO...Operator input file

READ(1,1002)FSPECR

FSPECR...Data input file

READ(1,1001)FSPECW

FSPECW...Migrated data output file
This is a Kirchoff migration program which uses the operator designed in MPOGEN to perform convolutional migration.

REAL*8 FSPECR, FSPECW, FSPEC0
INTEGER*2 NLEAD(512), N RANGE(512), A0, A1, A2,
A0ST, ATRI, AOP, ASUM, NLEADI(512)
REAL*4 FBUF(3), OPBUF(2560), TRACE(2560)
DATA DEV/3RRK /
DATA ASUM, A1, A2, ASUM/6, 2048, 4096, 5764/

IF (IFETCH (DEV) .NE. 0) STOP 'FETCH ERROR'
IRD = IGETC()
IWRT = IGETC()
IROP = IGETC()
CALL APINIT
READ (1, 1000) NTRACE, NB, NSAMP, NHLFWD
READ (1, 1001) XSTEP, TSTEP
READ (1, 1002) FBUF
NBLKR = NSAMP / 128
NFLO = NBLKR * NTRACE + 1
NW = NHLFWD - 1
NW2 = NW + 2
NA = NTRACE + NHLFWD
NWIDTH = 2 * NHLFWD - 1

OPEN UP I/O FILES
IF (LOOKUP (IROP, FSPEC0), LT. 0) STOP 'LOOKUP ERROR'
IF (LOOKUP (IRD, FSPEC0), LT. 0) STOP 'LOOKUP ERROR'
IF (ENTER (IWRT, FSPECW, NFLO), LT. 0) STOP 'ENTER ERROR'
C GET OPERATOR CONSTANTS

C IF (IREADW(NOP, J, IROP, LT, J) STOP 'READ ERROR'
C IF (IREADW(NOP+1, NRANGE, 1, IROP, LT, J) STOP 'READ ERR'
C NBRNG=(NOP+255)/256
C NBUF=NHFLFWD*5
C NBUF2=2*NBUF
C NBUFAP=5*NOP
C IOIPINC=(NBUF+127)/128
C NINLD=(NHFLFWD+255)/256
C NBLD=NOP*NINLD
C INLSTD=1+NBRNG
C INOPST=INLSTD+NBLD
C IBLKO=1
C ISTOP=NRANGE(NOP+1)

C START MAIN LOOP

C DO 1 ITR=NSTART, NSTOP, NINC
C IBLK=((ITR-1)*NBLKR)+1

C READ IN OUTPUT TRACE

C IF (IREADW(2*NSAMP, TRACE, IBLK, IRD) LT, J) STOP 'READ ERR2'
C CALL APPUT(TRACE, A0, NSAMP, 2)
C CALL APWD
C CALL VCLR(A0+ISTART, 1, ISTOP-ISTART)
C IMIN=MAX(NW2-ITR, 1)
C IMAX=MIN(NA-ITR, NWIDTH)
C IBLKR=((ITR-NW+IMIN-2)*NBLKR)+1

C LOOP ON OPERATOR APPLICATION

C DO 20 IT=IMIN, IMAX
C IOP=IT-NW
C IF (IOP LT, 1) IOP=2-IOP
C A0ST=A0+ISTART
C IF (IREADW(2*NSAMP, TRACE, IBLKR, IRD) LT, J) STOP 'READ ERR3'
C IBLKR=IBLKRD-NBLKR
C CALL APPUT(TRACE, A1, NSAMP, 2)
C LSTOR=1
C NSTOR=1

C GET OPERATOR AND LEADING VALUES FROM FILE

C IBLKLD=INLSTD
C IBLKOP=INOPST
C DO 40 J=1, NOP
C IF (IREADW(NHFLFWD, NLEADI, IBLKLD, IROP, LT, J) STOP 'READ ERR4'
C IF (IREADW(NBUF2, TRACE, IBLKOP, IROP, LT, J) STOP 'READ ERR4'
C IBLKLD=IBLKLD-NINLD
C IBLKOP=IBLKOP+IOIPINC
C IOFF=IOP
C DO CONVOLUTIONAL MIGRATION

C

C DO CONVOLVE

C

C GET MIGRATED TRACE OUT OF AP

C

C WRITE MIGRATED TRACE TO DISK

C

C
Stand Alone Interpolation : - ANINT

Input file......DK2:ANINV.DAT

Input Parameters

READ(1,1000)L,NCHAN,M,NTAP,INDEN

1000 FORMAT(6I5)

L.......Number of samples per channel
NCHAN....Number of channels
M.......Level of interpolation
NTAP.....Number of samples in cosine taper
INDEN....Normalisation flag
         <0 no scaling
         =0 scale to RMS energy
         >0 inverse energy scaling

READ(1,1100)FBUFR,FBUFW

1100 FORMAT(2(3A4))

FBUFR....Input file
FBUFW....Output file
VIRTUAL S@2(23552), S1(23552)
 DIMENSION DUM(2048)
 DIMENSION FBUFR(3), FBUFW(3), FSTAP(23)
 REAL *8 FSPECR, FSPECW
 DATA K1M, K1, K2, KTOP, IA, IB, IB1, IB2, IC, IC1, ID1
 \{/1, 1, 2, 8191, 0, 2O48, 2O49, 2O50, 4O96, 4O97, 6145/\n DATA DEV/3RDK /
 CALL ASSIGN(1, 'DK2:ANINV.DAT', 13)
 IF (IFETCH(DEV).NE.0) STOP 'FETCH ERROR'
 IDCHR=IGETC()
 IDCWH=IGETC()

C READ IN REQUIRED INPUT PARAMETERS
 READ(1, 1000) L, NCHAN, M, NTAP, INDEN
 FORMAT(2I5, 5X, 3I5)
 READ(1, 1100) FBUFR, FBUFW
 FORMAT(2(3A4))

C VALIDATE DATA
 C NREAD IS THE NUMBER OF BLOCKS READ IN/TRACE
 NREAD=L/128
 C STOP IF L IS NOT AN INTEGER NUMBER OF BLOCKS
 IF (L.NE.128*NREAD) STOP 'L NOT INTEGER NO. OF BLOCKS'
 C L2 IS NUMBER OF WORDS READ/TRACE
 L2=2*L
 C IAL IS POINTER NEEDED FOR COSINE TAPERING
 IAL=IA+L-1
 C FIND L21, INTEGER POWER OF TWO G.E. THAN L.
 L21=2
 DO 5 I=1, L
 IF(L2I.GE.L) GOTO 10
 L2I=2*L2I
 5 CONTINUE
 C STOP IF L21 IS GREATER THAN 2048
 IF(L21.GT.2048) STOP 'L2I.GT.2048'
 L21=L2I/2-1
 C STOP IF M.GT.23
 IF(M.GT.23) STOP 'M.GT.23'
 C LM IS NUMBER OF INTERPOLATED SAMPLES/TRACE
 LM=L*M
 C STOP IF LM.GT.23552
 IF(LM.GT.23552) STOP 'LM.GT.23552'
 C LM2 IS THE TOTAL NUMBER OF WORDS WRITTEN OUT/INTERPOLATED TRACE
 LM2=2*M
 C NWRITE IS THE NUMBER OF BLOCKS WRITTEN/TRACE
 NWRITE=LM/128
 C FST0 IS ADDRESS NEEDED FOR DISK READ INTO S@0(1)
 FST0=ADGET(S@0(1))
 C FST1 IS ADDRESS NEEDED FOR DISK WRITE FROM S1
 FST1=ADGET(S1(1))
 C FSTAP IS ARRAY OF ADDRESSES NEEDED FOR AP TRANSFERS
 K1=1
 DO 40 K=1, M
 40 FSTAP(K)=APGAD(S@0(KL))
C THIS PART OF THE PROGRAM SETS UP THE AP FOR INTERPOLATING THE TRACES. 
C A COMPLEX EXPONENTIAL VECTOR IS FORMED IN REGION C, TO BE USED IN 
C THE NEXT PART OF THE PROGRAM TO PRODUCE TIME ShiftS OF TSAMP/M BY 
C MULTIPLICATION IN THE FREQUENCY DOMAIN. 
C-----------------------------------------------
C INITIALISE AP 
C-----------------------------------------------
C PUT 1.5/FLOAT(M*L2I) IN IC AND 1.5/FLOAT(NTAP) IN KTOP 
C-----------------------------------------------
C FORM VECTOR RAMP IN IA USING STARTING VALUE AND RAMP INCREMENT 
C BOTH EQUAL TO VALUE IN IC. TAKE SIN AND COS OF RAMP AND 
C PLACE IN C. 
C-----------------------------------------------
C FORM COSINE TAPER NTAP LONG STARTING AT ID1 
C-----------------------------------------------
C IN THIS PART THE SEISMIC TRACES ARE READ IN ONE AT A TIME OFF DISK. 
C EACH TRACE HAS ITS MEAN REMOVED, IS SCALeD IF REQUIRED), IS PADDED OUT 
C WITH ZEROES FROM L TO L2I SAMPLES AND THEN IS INTERPOLATED SO THAT THE 
C NO. OF SAMPLES BECOMES L'M. THE INTERPOLATED TRACES ARE STORED ON DISK. 
C-----------------------------------------------
C JBLKR IS THE DISK BLOCK ABOUT TO BE READ 
C JBLKW IS THE DISK BLOCK ABOUT TO BE WRITTEN 
C-----------------------------------------------
& NOW INTERATE THROUGH CHANNELS INTERPOLATING THE TRACES.
DO 8,0=1,NCHAN
C READ IN TRACE FROM UNIT 2
IF(IREAD(IDCHR),L2,JBK0,FST0).LT.0)STOP 'READA ERROR'
JSLK0=JBLK0+IREAD
C CLEAR A AND TRANSFER TRACE INTO IT
CALL VCLR(IA,K1,L21)
CALL APWR
CALL IWAIT(IDCHR)
CALL APPUTA(IA,L,K2,FSTAP(1))
CALL APWD
C FIND MEAN VALUE OF TRACE AND PLACE IN AP(KTOP)
CALL MEANV(IA,K1,KTOP,L)
C SUBTRACT MEAN FROM TRACE
CALL VNEG(KTOP,K1,KTOP,K1,K1)
CALL VSADD(IA,K1,KTOP,IA,K1,K1)
CALL VMUL(IA,K1,1D1,K1,IA,K1,NTAP)
CALL VMUL(IAL,K1M,IAL,K1M,IAL,K1M,NTAP)
C IF INDEN IS NEGATIVE, NO SCALING OF TRACE
C IF INDEN IS ZERO SCALE TO UNIT R.M.S VALUE
C IF INDEN IS POSITIVE, USE INVERSE ENERGY SCALING (DIVERSITY STACK)
IF(INDEN.LT.0) GOTO 35
IF(INDEN.GT.0) CALL VSC(KTOP,K1,KTOP,K1,K1)
C PLACE 1.0 FROM TM IN IC
CALL VCLR(IC,K1,K1)
CALL VTSADD(IC,K1,2.0,IC,K1,K1)
CALL VDIV(KTOP,K1,IC,K1,KTOP,K1,K1)
CALL VSMUL(IA,K1,KTOP,IA,K1,L)
CALL VMOV(IB,K1,K1)
CALL RFFTBC(IB,IB,LZ1,K1)
CALL RFFTSX(IC,IC,LZ1,K1)
C TRANSFER MODIFIED TRACE BACK TO S0
CALL APWR
CALL APGD
IF(M.EQ.1) GOTO 50
C TAKE TRANSFORM OF TRACE AND PUT IN B
CALL RFFTB(IA,IB,LZ1,K1)
CALL RFFTSX(IB,IB,LZ1,K1)
CALL VTSADD(INV,IN,IN)
CALL VDIV(IN,IB,IN)
CALL VSMUL(IN,IB,IN)
CALL VMOV(IN,IN)
CALL APWR
CALL APGD
IF(M.EQ.1) GOTO 50
C ITERATE FROM 2 TO M
DO 400 K2=2,M
C MULTIPLY TRACE BY COMPLEX EXPONENTIAL VECTOR
CALL CVMUL(IB2,K2,IC1,K1,IB2,K2,LZ121)
C MOVE B TO A AND TAKE IN PLACE IFFT
CALL VMOV(IB,K1,IA,K1,LZ1)
CALL RFFTSX(IB,IB,K1)
C TRANSFER SHIFTED TRACE BACK TO S0
CALL APWR
CALL APGD
IF(M.EQ.1) GOTO 50
C INTERPOLATED TRACE IS NOW IN S0 IN SCRAMBLED ORDER.
C UNSCRAMBLE INTO S1 SO THAT RECORD IS IN CORRECT TIME SEQUENCE.
Internal Header Interrogation :- MPHIST

This is a fully interactive program which expects the input data to be on disc. The output listing is put onto the screen or the printer. The only input required is the name of the data file. EG.

enter name of file to be examined: DK3:MPDATA.DAT
THIS IS A PROGRAM WHICH INTERROGATES A DATA FILE TO GIVE ALL OF ITS PROGRAMMING PARAMETERS TO THE USER

```
REAL*8 FSPECR, RUNITS(2), SOURCE(4)
REAL*4 FNAMR(3)
INTEGER*2 HBLK(256)
LOGICAL*1 HBLK(512), NOCHAN, IGCODE, IUNITS, ISAMP
EQUIVALENCE (STOFF, HBLK(21)), (SLSPAC, HBLK(29)),
(SDEPT, HBLK(37)), (RDEPT, HBLK(41)),
(NOCHAN, HBLK(25)), (IGCODE, HBLK(19)),
(IUNITS, HBLK(12)), (ISAMP, HBLK(13)), (IBFREE, HBLK(47)),
(NBEG, HBLK(15)), (NFIN, HBLK(17)),
(HBLK(1)), (HBLK(9)), (HBLK(11)),
DATA DEV/3RRK/, RUNITS(1)/'METRES'/, RUNITS(2)/'FEET'/
DATA SOURCE(1)/'AIRGUNS'/, SOURCE(2)/'EXPLOSIVES'/,
SOURCE(3)/'VIBROSEIS'/, SOURCE(4)/'WEIGHTS'/
GET NAME OF INPUT FILE
WRITE(6), 5 FORMAT(7,5) READ(S,1Z1) FNAMR
READ(5,10)FNAMR
CALL IRAD5B(12, FNAMR, FSPECR)
SET UP I/O PARAMETERS
ICH=IGETC()
IF(IGETC(DEV).NE.0) STOP 'FETCH ERR'
ILEN=LOOKUP(ICH, FSPECR)
IF(ILEN.LT.0) STOP 'LOOKUP ERR'
IF(IREADW(256, HBLK(0), ICHR).LT.0) STOP 'READ ERROR'
DECODE HEADER AND PRINT OUT
WRITE(7,20) FNAMR, ILEN
WRITE(*, 20) 'FILE TO BE ANALYZED, LENGTH= ', I5, ' BLOCKS')
WRITE(7, 30) HBLK(1), I=1, 9)
WRITE(*, 30) 'TAPE NO= ', I5)
WRITE(7, 40) HBLK(9), HBLK(11), HBLK(12)
40 FORMAT(' SAMPLING INTERVAL SET AT ', I1, ' MSEC', ')
WRITE(7, 50) HBLK(13)
GOTO (50, 60, 65,66) IGCODE
50 WRITE(7, 110)
GOTO 100
60 WRITE(7, 120)
70 WRITE(7, 130)
GOTO 100
80 CONTINUE
```

---

This is a FORTRAN IV program that interacts with a data file to provide all of its programming parameters to the user. The program includes various declarations and statements for handling real numbers, integers, and logical values. It also includes a section for getting the name of the input file, setting up I/O parameters, decoding the header, and printing out relevant information. The program uses a mix of formatted I/O statements to handle input and output, along with conditional statements to manage different scenarios for file handling and parameter interpretation.
C FORMAT(' THE FILE CONTAINS A SHOT POINT GATHER')
C FORMAT(' THE FILE CONTAINS A COMMON MIDPOINT GATHER')
C FORMAT(' THE FILE CONTAINS A STACKED TRACE')
C FORMAT(' THE FILE CONTAINS AN UNSTACKED TRACE')
C WRITE(1,900)NCHAN,NBEG,NFIN
C 90 FORMAT(' CONSISTING ',I2, ' CHANNELS, STARTING AT SAMPLE NO: ',
C ' AND ENDING WITH SAMPLE NO: ',I5)
C WRITE(7,150)SOURCE(ISCODE),RUNITS(IUNITS)
C 150 FORMAT(' TYPE OF SOURCE = ',A8, '/ ,
C ' AND THE UNITS OF LENGTH USED = ',A8)
C WRITE(7,160)RDEPT,SDEPT,SROFF,RSPAC,SLSPAC,STSPAC
C 160 FORMAT(' RECIEVER DEPTH = ',F10.2, '/ ,
C ' SOURCE DEPTH = ',F10.2, ' ,
C ' SOURCE-RECIEVER OFFSET = ',F10.2, ' ,
C ' RECIEVER SPACING = ',F10.2, ' ,
C ' SHOT SPACING = ',F10.2, ' ,
C ' SHOT REPITITION RATE = ',F10.2, ' SECS')
C WRITE(7,170)(HBLK(I),I=51,254)
C 170 FORMAT(' USER INFO PUT INTO HEADER BLOCK , ,
C ' IS GIVEN BELOW' , ,3(1X,BH1), /))
C IF(IBFREE.EQ.0)GOTO 999
C PROCESS DECODE AREA
C WRITE(7,180)
C 180 FORMAT(' THE FOLLOWING PROCESSES HAVE BEEN 
C PERFORMED ON THE DATA', /)
C IPOS=0
C 190 CONTINUE
C GOTO(205,210,220,230,240)HBLK(129+IPOS)
C PRE STACK DECODE
C IPOS=IPOS+1
C ICNT=HBLK(129+IPOS)
C IPOS=IPOS+1
C WRITE(7,300)ICNT
C 300 FORMAT(' PRE-STACK PROCESSING CONSISTING OF , ,
C ' OPERATIONS IN THE FOLLOWING ORDER')
C DO 310 J=1,ICNT
C 310 IC=HBLK(129+IPOS)
C IPOS=IPOS+1
C GOTO(311,312,313,314,315,316,317,318,319,320,321)ICD
C 311 WRITE(7,400)
C 400 FORMAT(' TRACE EDITING')
C GOTO 310
C 312 WRITE(7,401)
C 401 FORMAT(' POLARITY REVERSAL')
C GOTO 310
C 313 WRITE(7,402)
C 402 FORMAT(' EXP(0.2T) AMP RECOVERY')
C GOTO 310
C 314 WRITE(7,403)
C 403 FORMAT(' T*EXP(0.2T) AMP RECOVERY')
FORTRAN IV

0075 GOTO 310
0076 315 WRITE(7,404)
0077 404 FORMAT(' T**V**2 AMP RECOVERY')
0078 GOTO 310
0079 316 WRITE(7,405)
0080 405 FORMAT(' MUTING')
0081 GOTO 310
0082 317 WRITE(7,406)
0083 406 FORMAT(' SPIKE DECONVOLUTION')
0084 GOTO 310
0085 318 WRITE(7,407)
0086 407 FORMAT( 'BANDPASS FILTERING')
0087 GOTO 310
0088 319 WRITE(7,408)
0089 408 FORMAT(' BANDREJECT FILTERING')
0090 GOTO 310
0091 320 WRITE(7,409)
0092 409 FORMAT(' PREDICTION ERROR DECONVOLUTION')
0093 GOTO 310
0094 321 WRITE(7,410)
0095 410 FORMAT( 'AMPLITUDE/ENERGY NORMALISATION')
0096 GOTO 310
0097 CONTINUE
0098 IF(IPOS.LT.IBFREE) GOTO 190
0099 GOTO 999

C C C C C C C C
C C C C C C C
C C C C C C C
C C C C C C C
C C C C
C

STACKING DECODE

3102 219 WRITE(7,420)
3103 420 FORMAT( ' NMO CORRECTION AND CDP STACK')
3104 IPOS=IPOS+1
3105 NCHST=HBLK(129+IPOS)
3106 IPOS=IPOS+1
3107 NVST=HBLK(129+IPOS)
3108 IPOS=IPOS+1
3109 NLYRST=HBLK(129+IPOS)
3110 MST=HBLK(129+IPOS)
3111 IPOS=IPOS+1
3112 INTSWS=HBLK(129+IPOS)
3113 IPOS=IPOS+1
3114 WRITE(7,421)NCHST,MST
3115 421 FORMAT( 'CHANNELS AT A LEVEL OF',/,' #14', ' TIMES INTERPOLATION')
3116 WRITE(7,422)NVST,NLYRST,INTSWS
3117 422 FORMAT( ' VELOCITY ANALYSES WERE USED EACH WITH',/,' #13', ' LAYERS AND THE INTERPOLATION SWITCH=',/,')
3118 IF(IPOS.LT.IBFREE) GOTO 190
3119 GOTO 999

C C C C C C C C
C C C C C C C
C C C C C C C
C C C C C C C
C C C C
C

POST STACK DECODE

0120 IPOS=IPOS+1
0121 ICNT=HBLK(129+IPOS)
0122 IPOS=IPOS+1

C C C C C C C C
WRITE(7,430) ICNT
430 FORMAT( 'POST STACK PROCESSING CONSISTING OF', /,&
' OPERATIONS IN THE FOLLOWING ORDER')
DO 330 J=1, ICNT
ICD=HBLK(129+IPOS)
IPOS=IPOS+1
GOTO(331,332,333,334,335,336,337,338,339,340)ICD
331 WRITE(7,400)
332 GOTO 330
333 WRITE(7,402)
334 GOTO 330
335 WRITE(7,404)
336 GOTO 330
337 WRITE(7,405)
338 GOTO 330
339 WRITE(7,406)
340 GOTO 330
341 WRITE(7,407)
342 GOTO 330
343 WRITE(7,408)
344 GOTO 330
345 WRITE(7,409)
346 GOTO 330
347 WRITE(7,410)
CONTINUE
IF(IPOS.LT.IBFRE) GOTO 190
348 WRITE(7,450)
450 FORMAT( 'MIGRATION')
IPOS=IPOS+1
IF(IPOS.LT.IBFRE) GOTO 190
GOTO 999
349 WRITE(7,460)
460 FORMAT( 'THREE TRACE MIX')
IPOS=IPOS+1
IF(IPOS.LT.IBFRE) GOTO 190
STOP 'NORMAL TERMINATION'
999 END
Synthetic CMP Gathers: ANSEI

Input file: DK1:ANINS.DAT

Output file: DK2:ANOUTS.DAT

Input Parameters

READ(1,1000)L,NCHAN,NSTART,NLYR

1000 FORMAT(4I5)

L ........ Number of samples per channel
NCHAN .... Number of channels per gather
NSTART ... Number of starting sample from time zero
NLYR ..... Number of events

READ(1,1100)XSTART,XSTEP,FSAMP,FRICK,FNOISE,AMP

1100 FORMAT(6F10.0)

XSTART ... Offset of first receiver from source
XSTEP .... Receiver spacing
FSAMP .... Sampling frequency of data samples/millisecond
FRICK .... Frequency of Source wavelet(Ricker)
FNOISE ... Upper of Frequency of Background noise
AMP ...... Amplitude of Background noise

READ(1,1200)(TOLYR(I),VLYR(I),I=1,NLYR)
TOLYR....Two-way travel time of reflection on zero offset trace

VLYR....RMS velocity down to the event
REAL *8 FSPEC
DIMENSION SEISM(2048), TOLYR(99), VLYR(99), WAVE(500), CONST(4)
DATA KIN, K, K1, K2, K4, KAMP, KSN, KTWOP1, KSEED, KU/-1, 0, 1, 2, 4, 8, 1817,
 1818, 8189, 8190, 8191/
DATA NWDBLK, DEV, FSPEC/256, 3RDK, 12RDK2ANOUTSDAT/
CALL ASSIGN(1, 'DK1:ANINS.DAT', 13)
IF (IFETCH(DEV).NE.0) STOP 'FETCH ERROR'
IDCH = IGETC() 
READ(1, 1000) L, NCHAN, NSTART, NLYR
READ(1, 1000) XSTART, XSTEP, FSAMP, FRICK, FNOISE, SN
READ(1, 1000) (TOLYR(I), VLYR(I), I=1, NLYR)
READ(1, 1200) (TOLYR(I), VLYR(I), I=1, NLYR)
NBLTR = IFIX(2.0*FLOAT(L)/FLOAT(NWDBLK)+0.5)
L = NWDBLK*NBLTR/2
NBLTOT = NBLTR*NCHAN
IF (IENTER<IDCH, FSPEC, NBLTOT+1>. LT 0 STOP 'ENTER ERROR'
JBLOCK = 1
CALL RICKER(FSAMP, FRICK, NBEGIN, NWAVE, WAVE)
L2INT = 2
DO 5 K = 1, 11
DATA NDBLK, DEV, FSPEC/256, 3RDK, 12RDK2ANOUTSDAT/
IF (L2INT.GE.L) GOTO 10
L2INT = 2*L2INT
CONTINUE
CONST<1,1> = 1.0/FSAMP
CONST<2,2> = 8.0*ATAN<1,1,0>
CONST<3,3> = 0.251.0,638
CONST<4,4> = 1.0/FSAMP
CALL APINIT<IDUM, IDUM, IDUM>
CALL APPUT(CONST, KSN, K4, K2)
CALL APWD
ISEISM = 1
IWAVE = IWAVE+L2INT+NWAVE-1
IRAND = IWAVE+NWAVE
NRAND = L2INT*NNOISE/FSAMP
IOISE = IRAND+NRAND
IOISE2 = IOISE+2
IOISE3 = IOISE+3
NRAND2 = 2*NRAND
CALL APPUT(WAVE, IWAVE, NWAVE, K2)
CALL APWD
KCHAN X START
OFFS = NSTART-NBEGIN
DO 33 JCHAN = 1, NCHAN
NSEG = KCHAN+2
KCHAN = KCHAN+XSTEP
CALL VCLR(KD, KI, IWAVE)
DO 33 JLYR = 1, NLYR
NT = IFFIX(FSAMP*SORT(TOLYR(JLYR)**2+XSQ
 0.5)-NOFFS
CALL VADD(N, KI, KU, KI, NT, K1, K1)
33 CONTINUE
SUBROUTINE RICKER(FSAMP,FRICK,NBEGIN,NWAVE,WAVE)
IMPLICIT INTEGER*2(I-N)
DIMENSION WAVE(500)
N=IFIX(FSAMP/FRICK)
NWAVE=2*N+1
NBEGIN=N+1
WAVE(NBEGIN)=1.
IF(N.EQ.3) RETURN
IPLUS=NBEGIN
IMINUS=NBEGIN
X=0.
DELX=3.14159265*FRICK/FSAMP
DO 10 I=1,N
10 IPLUS=IPLUS+1
IMINUS=IMINUS-1
X=X+DELX
WAVE(IPLUS)+0.5*(1.+COS(X))*(1.-2.*X**2)*EXP(-X**2)
WAVE(IMINUS)=WAVE(IPLUS)
RETURN
END
Synthetic Sections :

Input file.....DK2:MPSYN.DAT

Input Parameters

READ(1,1000)TSTEP,XSTEP,FRICK,FNOISE,AMP

1000 FORMAT(10F10.0)

TSTEP....Sampling rate of data in seconds
XSTEP....Trace spacing in kms
FRICK....Frequency of source wavelet
FNOISE...Upper limit to background noise
AMP......Scale factor for noise

READ(1,1001)NTRACE,NSTART,NSTOP,NINC,NO,NSAMP,IFLAG

1001 FORMAT(10I5)

NTRACE...Number of output traces
NSTART...First trace to output
NSTOP....Last trace to output
NINC.....Trace output increment
NO.......Number of first sample on the trace relative to time zero
NSAMP....Number of samples per trace
IFLAG....Scaling flag

<0 3 Dimensional scaling
>0 2 Dimensional scaling
READ(1,1001)NPLANE,NPOINT,NLAYER

NPLANE...Number of plane reflectors <6
NPOINT...Number of point reflectors <6
NLAYER...Number of velocity layers <10

The input for each of the above cases then follows:

IF(NLAYER.GT.0)

READ(1,1002)(VEL(I),THICKN(I),I=1,NLAYER)

READ(1,1002)VEL(NLAYER+1)

VEL......Velocity of layer Km/s
THICKN...Thickness of layer Km
VEL(NLAYER+1)..Velocity of remaining half-space beneath last layer

IF(NPLANE.GT.0)

READ(1,1003)(X0(I),X1(I),X2(I),DIP(I),I=1,NPLANE)

1003 FORMAT(4F10.0)

X0.......X coordinate of surface intersection of layer projection. Depth for horizontal layers.
X1.......Beginning of reflector
X2.......End of reflector
DIP.......Reflector dip in degrees
IF(NPOINT.GT.0)

READ(1,1002)(XP(I),ZP(I),I=1,NPOINT)

XP......Lateral position of reflecting point Kms
ZP......Depth of reflecting point Kms

READ(1,1004)FBUF

FBUF.....output file
PROGRAM SYNTH...MODIFIED FROM
A PROGRAM BY C GODBOLD BY M J POULTER

REAL*8 FSPECW
DIMENSION X@5), X1(5), X2(5), DIP(5), SINDIP(5), COSDIP(5), 
$XP(10), ZP(10), FBUF(3)
COMMON/AA/TRACE(2048), TSTEP, NSAMP, IFLAG1
COMMON/BB/AUX(2048)
COMMON/CC/VEL(11), THICKN(10), NLAVER
EQUIVALENCE(DIP(1), COSDIP(1))
DATA KN1, K2, PI/-1, 1, 2, 3.141593/
DATA DEV/3RRK/
CALL ASSIGN(1, 'DK2:MPsynd.dat', 14)
IVRT = IGETC(
IF(IFETCH(DEV).NE.1), STOP 'FETCH ERROR'
IBLK = 1

READ IN PARAMETERS
READ(1, 1000) TSTEP, XSTEP, FRICK, FNOISE, AMP
READ(1, 1001) NTRACE, NSTART, NSTOP, NINC, NSF, NSAMP, IFLAG1
READ(1, 1001) NPLANE, NPOINT, NLAVER
IF(NLAVER.LE.0), GOTO 10
READ(1, 1002) VEL(I), THICKN(I), I=1, NLAVER
READ(1, 1002) VEL(NLAVER+1)
FORMAT(2F10.0)
IF(NPLANE.LE.0), GOTO 20
READ(1, 1003) X0(I), X1(I), X2(I), DIP(I), I=1, NPLANE
FORMAT(4F10.0)
DO 30 J=1, NPLANE
SINDIP(J) = SIN(DIP(I)*PI/180.0)
COSDIP(J) = SQRT(1.0-SINDIP(I)**2)
IF(NPOINT.LE.0), GOTO 40
READ(1, 1002) XP(I), ZP(I), I=1, NPOINT
NVEL=NLAVER+1
READ(1, 1004) FBUF
FORMAT(3A4)
CALL IRAD50(12, FBUF, FSPECW)
NBLKR = NSAMP/128
NBLKF = (NTRACE*NBLKR)+1
IF(IENTER(IWRT, FSPECW, NBLKF).LT.0), STOP 'ENTER ERROR'

CONVERT TSTEP TO MSEC
TSTEP = INT(TSTEP*1000.0+0.5)

EVALUATE RICKER WAVELET
CALL RICKER(TSTEP, FRICK, N)
C SET UP AP AND GET CONSTANTS

CALL APINIT
KNSAMP=NSAMP
KNOISE=2
IF(AMP.LT.0.)GOTO 50
60 KNOISE=KNOISE*2
IF(KNOISE.LT.KNSAMP)GOTO 60
KWAVE2=KNSAMP
KWAVE3=KWAVE+3
KA=N
KA=8192-KWAVE3
KC=KB+KNSAMP
KD=8199
AUX(KWAVE+1)=2.*PI
AUX(KWAVE+2)=2.510638
AUX(KWAVE3)=AMP
CALL APPUT(AUX,KA,KWAVE3,2)

C START TRACE GENERATION LOOP

DO 10 L=1,NTRACE
X=(L-1)*XSTEP
CALL VCLR(0,1,KNSAMP)
CALL APWR
CALL APGETCTRACE,16,KNSAMP,2
CALL APWD

C EVALUATE ARRIVAL FOR EACH REFLECTOR

IF(NPLANE.LE.0)GOTO 110
DO 200 J=1,NPLANE
R=X.GT.X2(J)GOTO 200
XA=ABS(X-X2(J))*SINDIP(J)**2
ZA=ABS(X-X2(J))*SINDIP(J)*COSDIP(J)
IF(ABS(SINDIP(J)).LT.0.01)ZA=X2(J)
CALL NEWTON(XA,ZA,TJ6)
CALL IMPULSCTJ6)
CONTINUE

C EVALUATE FOR EACH POINT REFLECTOR

110 IF(NPOINT.LE.0)GOTO 120
DO 210 J=1,NPOINT
CALL NEWTON(ABS(X-XP(J)),ZP(J),T0)
CALL IMPULSCT0)
CONTINUE

C CONVOLVE SERIES WITH WAVEFORM

120 CALL VCLR(0,1,KB)
CALL VCLR(KC,1,KB)
CALL APPUT(TRACE,KB,KNSAMP,2)
SUBROUTINE RICKER(TSTEP,FRICK,N)
C
C SUBROUTINE TO EVALUATE A RICKER WAVELET
DUE TO A G NUNNS 1979--- WITH MODS
C
COMMON/BB/AUX(2*48)
N=IFIX(1002.8/(FRICK*TSTEP))
IPLUS=N+1
AUX(IPLUS)=1.0
IF(N.EQ.0)RETURN
IMINUS=IPLUS
X=0.0
DELX=3.14159265*FRICK*TSTEP/1002.8
DO 100 I=1,N
IPLUS=IPLUS+1
IMINUS=IMINUS-1
X=X+DELX
AUX(IPLUS)=1.0+Cos(X)*(0.5-X**2)*Exp(-X**2)
AUX(IMINUS)=AUX(IPLUS)
100 CONTINUE
RETURN
END
SUBROUTINE IMPULS(TB)

C SUB TO CALC BANDLIMTED IMPULSE RESPONSE

C

COMMON/AA/TRACE(2048),TSTEP,NSAMP,IFLAG

DATA FRAC1/1.0/,FRAC2/10.0/

TP=FRAC1*TSTEP

I=(TB-TP)/TSTEP

T=I*TSTEP

I=I+1

IF(IFLAG.GE.0)GOTO 120

I=I+1

IF(I.GT.NSAMP)RETURN

T=T+TSTEP

IF(I.LT.1)GOTO 100

IF(T-TB-TP.GE.0.0)RETURN

IF(T-TB+TP.LE.0.0)STOP 'ERROR 1'

TRACE(I)=TRACE(I)+1000.0/TB

GOTO 100

VALMIN=FRAC2/SQRT(TB*TP)

C1=SQR(TB/2.0)

C2=500.0/(C1*T)

I=I+1

IF(I.GT.NSAMP)RETURN

T=T+TSTEP

IF(I.LT.1)GOTO 140

IF(T-TB-TP.GT.0.0)GOTO 180

IF(T-TB+TP.LT.0.0)STOP 'ERROR 2'

TRACE(I)=TRACE(I)+C2*SQR(T-TB+TP)

GOTO 140

TRACE(I)=TRACE(I)+VAL

I=I+1

IF(I.GT.NSAMP)RETURN

T=T+TSTEP

IF(VL.GT.VALMIN)GOTO 160

RETURN

END
SUBROUTINE NEWTON(X,Z,T)
C SUBROUTINE TO EVALUATE TRAVEL TIMES THROUGH
C A SERIES OF CONSTANT VELOCITY LAYERS
C USES NEWTON - RAPHSON (N-R) TECHNIQUE
COMMON/CC/VEL(11),THICKN(10),NLAYER
COMMON/DD/N,D(11)
IF(NLAYER.LE.0)GOTO 160
C FIND N
TH=0.0
DO 100 I=1,NLAYER
N=I
IF(Z.LE.TH+THICKN(I))GOTO 110
110 N=N+1
DO 1216 I=1,I
TH=TH+THICKN(I)
IF(X.LE.0.0)GOTO 170
C SET VALUES OF D(I)
I1=N-1
DO 120 I=1,11
D< I<1=THICKN< I<1
120 D< I<1=THICKN< I<1
IF(X.LE.0.0)GOTO 170
C FIND N-R STARTING VALUE
C0=3.0*VEL(N)
C=3.0*VEL(N)*0.5
IF(A(C0,1).LE.X)GOTO 130
C FIND C BY N-R
C=1/C0
C0=C1
C1=C0
C0=C1
140 C0=C1
C=ABS((C1-C0)/C1).GT.0.1E-03)GOTO 140
C EVALUATE T
T=X
DO 150 I=1,N
T=T+D(I)*SORT((C1/VEL(I)**2-1.0)
150 T=T*1000.0/C1
RETURN
C NO VELOCITY INTERFACE BETWEEN SOURCE AND RECEIVER
T=SORT(X**2+Z**2)*1000.0/VEL(1)
RETURN
C
FORTRAN IV

FUNCTION A(C,M)

COMMON/CC/VEL(I),THICKN(I),N,LAYER
COMMON/DD/N,D(I)
A=8.8
B=1.8
DO 100 I=1,N
IF(M.EQ.1)GOTO 100
IF(M.LT.1)STOP 'ERROR 3'
B=VEL(I)**(M-1)
100 A=A+D(I)/(B*SQRT((C/VEL(I))**2-1.0)**M)
RETURN
END

C VERTICAL RAYPATH
C
170 T=0.0
DO 180 I=1,N
180 T=T+D(I)/VEL(I)
RETURN
END

FUNCTION A(C,M)

COMMON/CC/VEL(I),THICKN(I),N,LAYER
COMMON/DD/N,D(I)
A=8.8
B=1.8
DO 100 I=1,N
IF(M.EQ.1)GOTO 100
IF(M.LT.1)STOP 'ERROR 3'
B=VEL(I)**(M-1)
100 A=A+D(I)/(B*SQRT((C/VEL(I))**2-1.0)**M)
RETURN
END
Velocity Analysis Display :- MPVCON

This routine is completely self contained. All the information required by the program is contained in the output from the velocity analysis program.

Once the program has been run then it can be put onto the plotter using RASM or rasterised and saved using MPRASM.

Input file..... DK2:ANOUTV.DAT, unformatted fortran data file.
C VELOCITY ANALYSIS CONTOUR PROGRAM
C THIS TAKES UNFORMATTED INPUT
C FROM OUTPUT OF VELOCITY ANALYSIS
C PROGRAM AND CONTOURS AND ANOTATES IT

VIRTUAL Z(120,256)
REAL*4 X(256),Y(120),Z,
INTEGER*2 PTR(20),NMAX(256)
LOGICAL*4 SWCHES(5)
COMMON /PPEPl/ IXl,IVl,IX2,IV2,ISCAN,NSCAN,NBAND,NIPS,NIPG,
1 NIPM1,LYNES,NIBSX,MSGLVL,XDOTS,YDOTS,PREF(2),
2 RORG(2),PORT(2,2),IEND(4),ALMT,FACT,JPEN,XOFF,
3 XFAC,YOFF,YFAC,NBITS,NBITM1,NBYTES,NBYTE1,MSK,LMSK
DATA CZ/1.8,9.95,.8.9,.8.8,.8.75,.8.65,.8.6,.8.5,.8.5/.45,
%0.4,.35,.3,.25,.2,.15,.1,.05/
DATA SWCHES/.TRUE.,.TRUE.,.TRUE.,.TRUE.,.FALSE./
DATA NC/18/

C ZERO CONTOUR ARRAY

DO 1 1=1,256
DO 2 J=1,120
Z(J,1)=0.0
1 CONTINUE
CALL PLOTS(0,0,0)

C READ IN CONTROL DATA

CALL ASSIGN(2,'DK2:ANOUTV.DAT',14)
READ(2) LNUM,NCHAN,NSTART,M
READ(2) XSTART,XSTEPR,FSAMP
READ(2) T81,T82,TGATE
READ(2) VSTEP,V1,V2MIN,V2MAX
READ(2) NT8
XSTEP=10.0/FLOAT(NT8-1)
IXPOS=NT8-1

C READ IN THE SEMBLANCE DATA

DO 18 JT=1,NT8
IXPOS=IXPOS-1
READ(2) NPT,NMAX(JT8),(Z(J,IXPOS),J=1,NPT)
IF(NPT.GT.MVPT)MVPT=NPT
18 CONTINUE
VSTEP=4.0/FLOAT(MVPT-1)

C SET UP COORDINATES OF CONTOUR ARRAY

Y(1)=0.3
DO 29 L=1,MVPT
29 Y(L)=Y(L-1)+VSTEP
X(1)=0.0
DO 30 LL=2,NT8
30 X(LL)=X(LL-1)+XSTEP

C USE CONSYS CONTOUR PACKAGE

CALL CONTUR(Y,MVPT,X,NT8,Z,120,CZ,NC,PTR,SWCHES,
* ...)

C PLOT TIME GRID

T8LEN=T82-T81
TUNIT=1000.0*(X(NT8)-X(1))/T8LEN
TTUNIT=TUNIT/10.0
T0BEG=FLOAT(IFIX(T01+1000.0)/1000)
XOFF=(T0BEG-(T01/1000.0))*TUNIT

C PLOT SECOND GRID

TNUM=T0BEG
XPT=X(NT0)-XOFF
56 CALL PLOT(V(1),XPT,+3)
CALL PLOT(V(MYPT),XPT,+2)
CALL NUMBER(0.0,XPT,0.05,TNUM,0.0,1)
XPT=XPT-TUNIT
TNUM=TNUM+1.0
IF(XPT.GT.X(1))GOTO 56

C PLOT TENTHS OF SECONDS GRID

LMSK="1403
MSK=1
XPT=X(NT0)-XOFF
66 CALL PLOT(V(1),XPT,+3)
CALL PLOT(V(MYPT),XPT,+2)
XPT=XPT-TTUNIT
IF(XPT.GT.X(1))GOTO 66
XPT=X(NT0)-XOFF
76 CALL PLOT(V(1),XPT,+3)
CALL PLOT(V(MYPT),XPT,+2)
XPT=XPT+TTUNIT
IF(XPT.LT.X(NT))GOTO 76

C VELOCITY GRID

VLEN=V2MAX-V1
VUNIT=(Y(MYPT)-V(1))/VLEN
VTUNIT=VUNIT/10.0
VBEG=FLOAT(IFIX(V1+1.0))
YOFF=(VBEG-V1)*VUNIT

C PLOT KM/S GRID

LMSK=-1
MSK=0
YPT=V(1)+YOFF
VNUM=VBEG
86 CALL PLOT(YPT,X(NT0),+3)
CALL PLOT(YPT,X(1),+2)
CALL NUMBER(YPT-0.05,X(NT0),0.05,VNUM,0.0,1)
YPT=YPT+VTUNIT
VNUM=VNUM+1.0
IF(YPT.LT.V(MYPT))GOTO 86

C PLOT TENTHS GRID

LMSK="1403
MSK=-1
YPT=V(1)+YOFF
96 CALL PLOT(YPT,X(NT0),+3)
CALL PLOT(YPT,X(1),+2)
YPT=YPT+VTUNIT
IF(YPT.GT.V(1))GOTO 96
YPT=V(1)+YOFF
106 CALL PLOT(YPT,X(NT0),+3)
CALL PLOT(YPT,X(1),+2)
YPT=YPT+VTUNIT
C PLOT POSITION OF MAX PTS

LMSK=1
MSK=0
IXMP=NTS+1
DO 40 MP=1,NTS
IXMP=IXMP-1
XP=X(IXMP)
YP=Y(MAX(MP))
cALL SYMBOL(YP,XP,S.82,S.8,-1)
40 CONTINUE

C DO ANOTATION

cALL SYMBOL(1.0,1.8,1.0,1.0,'VELOCITY ANALYSIS CONTOURS',S.8,26)
cALL SYMBOL(4.8,10.8,1.0,1.0,'PROCESSING PARAMETERS',S.8,21)
cALL PLOT(4.8,10.3,+3)
cALL PLOT(6.9,10.3,+2)
cALL SYMBOL(5.0,10.8,1.0,1.0,'NO. OF CHANNELS =',S.8,18)
FNUM=NCHAN
cALL NUMBER(6.8,10.8,1.0,1.0,FNUM,S.8,-1)
cALL SYMBOL(5.0,9.8,1.0,1.0,'SAMPLES PER CHANNEL =',S.8,22)
FNUM=LNUM
cALL NUMBER(7.2,9.8,1.0,1.0,FNUM,S.8,-1)
cALL SYMBOL(5.0,9.6,1.0,1.0,'SAMPLE DELAY =',S.8,15)
FNUM=NSTART
cALL NUMBER(6.5,9.6,1.0,1.0,FNUM,S.8,-1)
cALL SYMBOL(5.0,9.4,1.0,1.0,'LEVEL OF INTERPOLATION =',S.8,25)
FNUM=M
cALL NUMBER(7.5,9.4,1.0,1.0,FNUM,S.8,-1)
cALL SYMBOL(5.0,9.2,1.0,1.0,'CHANNEL 1 OFFSET =',S.8,19)
cALL NUMBER(6.9,9.2,1.0,1.0,XSTART,S.8,1)
cALL SYMBOL(5.0,9.0,1.0,1.0,'CHANNEL SPACING =',S.8,18)
cALL NUMBER(6.8,9.0,1.0,1.0,XSTEP,S.8,1)
cALL SYMBOL(5.0,8.8,1.0,1.0,'SAMPLING INTERVAL MS =',S.8,23)
FNUM=1.24/FSAMP
cALL NUMBER(7.3,8.8,1.0,1.0,FNUM,S.8,-1)
cALL SYMBOL(5.0,8.6,1.0,1.0,'START OF ANALYSIS MS =',S.8,23)
cALL NUMBER(7.3,8.6,1.0,1.0,T01,S.8,-1)
cALL SYMBOL(5.0,8.4,1.0,1.0,'END OF ANALYSIS MS =',S.8,21)
cALL NUMBER(7.1,8.4,1.0,1.0,T02,S.8,-1)
cALL SYMBOL(5.0,8.2,1.0,1.0,'TIME STEP MS =',S.8,15)
cALL NUMBER(6.5,8.2,1.0,1.0,TSTEP,S.8,-1)
cALL SYMBOL(5.0,8.0,1.0,1.0,'OPERATOR GATEWIDTH MS =',S.8,23)
cALL NUMBER(7.3,8.0,1.0,1.0,TGATE,S.8,-1)
cALL SYMBOL(5.0,7.8,1.0,1.0,'START VELOCITY KM/S =',S.8,22)
cALL NUMBER(7.2,7.8,1.0,1.0,V1,S.8,2)
cALL SYMBOL(5.0,7.6,1.0,1.0,'END VELOCITY KM/S =',S.8,20)
cALL NUMBER(7.0,7.6,1.0,1.0,V2MAX,S.8,2)
cALL SYMBOL(5.0,7.4,1.0,1.0,'VELOCITY STEP KM/S =',S.8,21)
cALL NUMBER(7.1,7.4,1.0,1.0,VSTEP,S.8,2)
cALL PLOT(S.8,8.999)
STOP
Section Plotting :- MPSPLT

Input file.......DK1:MPPLTD.DAT

Log file.........DK1:MPPLTD.LOG

Input Parameters

READ(1,1000)NTR,NPT,NINT,NDSTEP,ISBEG,ISFIN

1000 FORMAT(6I5)

NTR......Number of traces to plot
NPT......Number of samples per trace
NINT.....Interpolation factor, number of dots per sample.
NDSTEP...Trace spacing in dots, 4,8,10,16,20
ISBEG....First sample to plot
ISFIN.....Last sample to plot

READ(1,1000)TPDRR,TPDRW1,TPDRW2,INFLG,OUTFLG

TPDRR....Input tape drive
TPDRW1...First output tape drive
TPDRW2...Second output tape drive
INFLG....Input flag
  0 - Input from tape
  1 - Input from disc
OUTFLG...Output flag
  0 - Output to tape
1 - Output to disc

READ(1,1001)XSF

1001 FORMAT(F10.0)

XSF.....Plot Scale factor

IF(INFLG.NE.0)READ(1,1002)FSPECR

1002 FORMAT(3A4)

FSPECR....Input data file

IF(OUTFLG.NE.0)READ(1,1002)FSPECW

FSPECW....Raster output file
PLOTTING PROGRAM FOR SEISMIC SECTION DATA

THIS PROGRAM TAKES IN SEISMIC TRACES AND DISPLAYS THEM IN A NIB IMAGE FORM IN AN OUTPUT FILE IN A FORM READY FOR POST PROCESSING.

VIRTUAL IPBUF(256,50)
REAL*8 FSPECR,FSPECW
REAL*4 XBUF(5376),FBUF(257),IDOT(4096),IHBLK(128),
% BUF(5375),JROW(256)

LOGICAL*1 TPDRR,TPDRW1,TPDRW2,ISTAT
EQUIVALENCE (BUF(1),IHBLK(1)),(BUF(257),XBUF(1)),
% BUF(257),IDOT(1))

COMMON /10/NBLKBF,NWOBF,NBVTBF,IWRT
DATA NBLKR=NPT/128
NWTR=NTR+24
NBLKB=NDSTEP/2

NBLKW=(NTR+24)*NBLKB*NTIM
NWDBF=NBLKB*256
NBYTBF=NWDBF/2
Iоф=2*NDSTEP
IBLKR=1
IBLKW=1
IBLKW2=NBLKW/NTIM
IAO=0

DATA READ IN SECTION
NTR,NPT,NINT,NDSTEP,ISBEG,ISFIN
FORMAT(6i5)
READ(1,I000)TPDRR,TPDRW1,TPDRW2,INFLG,OUTFLG
READ(1,1001)XSF
FORMAT(3A4)

SET UP CONSTANTS AND FILE ACCESS
NPTS=ISFIN-ISBEG+1
NPINT=(NPTS*NINT)-1
NTIM=1

IF(NPINT.GT.2048)NTIM=2
IF(NPINT.GT.4096)STOP 'ERROR IN INTERPOLATION SPECS'
NBLKR=NPT/128
NWTR=NPT/2
NBYTR=NWTR/2+512
NBLKB=NDSTEP/2
NBLKW=(NTR+24)*NBLKB*NTIM
NWDBF=NBLKB*256
NBYTBF=NWDBF/2
Iоф=2*NDSTEP
IBLKR=1
IBLKW=1
IBLKW2=NBLKW/NTIM
IAO=0
F O R T R A N IV

SET UP INPUT FILES

IF(INFLG.EQ.0)GOTO 10
READ(1,1002)FBUF
CALL IRAD50(12,FBUF,FSPEC)
IF(LOOKUP(IRD,FSPEC).LT.0)STOP 'LOOKUP ERROR'

SET UP OUTPUT FILES

IF(OUTFLG.EQ.0)GOTO 20
READ(1,1002)FBUF
CALL IRAD50(12,FBUF,FSPECW)
IF(IENTER(IWRT,FSPECW,NBLKW).LT.0)STOP 'ENTER ERROR'

SET UP ROW COUNTER AND CLEAR PLOT BUFFER

DO 15 I=1,SB
ROW(I)=I
15 CONTINUE
CALL APINIT
CALL VCLR(0,1,8192)
CALL APWR

MAIN LOOP FOR PLOTTING DIFFERENT TRACES

READ IN DATA FROM DISC

IF(INFLG.EQ.0)GOTO 110
IF(IREAD(NWDR,XBUF,IBLKR,IRD).LT.0)STOP 'READW ERR'
IBLKR=IBLKR+NBLKR
GOTO 120

TAPE READ ROUTINE

110 ITRY=1
IF(ITRY.GT.3)GOTO 310
IF(IETR.GE.0)GOTO 320
WRITE(7,1100)TPDRR,IFIL
1100 FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I5)
WRITE(7,1101)
1101 FORMAT(‘ ENTER NEW READ DRIVE NUMBER: ’,S)
READ(5,1102)TPDRR
1102 FORMAT(I1)
IEOTR=0
IF(TPDRR.GT.2)STOP’EOT TERMINATE’
DO READ
323 CALL TAPSUB(-1,TPDRR,ISTAT,IFLEN,IFIL,BUF,NBYTR,IEOTR)
1200 FORMAT(’ WARNING READ RETRY FAILED ON FILE NO: ’,I5)
IF(IEOTR .LT.0)GOTO 300
WIND OVER EOF MARKER
CALL TAPSUB(0,TPDRR,ISTAT,IFIL,,IEOTR)
330 ITRY=ITRY+1
IF(IHBLK(1).EQ.”777777”GOTO 310
120 CONTINUE
SET UP PROCESSING CONSTANTS
XBUFI=SF
XBUFC=SF
.XBUFC=FIN+2=FLOAT(IOFF-1)
XBUFC=FIN+3=FLOAT(IOFF)
XBUFC=FIN+4=SF
SECTION WHICH DEALS WITH INTERPOLATION AND SCALING OF DATA BEFORE PLOTTING
CALL APPUT(XBUF,IA1,NPTS+4,2)
CALL APWD
CALL VMOV([IA1,1,IA0,NINT,NPTS)
CALL VSMUL([IA1,1,IC1,IA1,NPTS-1])
IST=IA0
DO 130 J=2,NINT
IST=IST+1
CONTINUE
CALL APWD
CALL APGET(IDOT,0,NPINT,1)
IBIT=16
NP=256
CALL APWD
IDOT(1)=IDOT(1)+IOFF
DO 30 IP=2,NPINT
C SECTION WHERE POSITIVE (SHADED) LOBES ARE PLOTTED

C

3138 IF,IDOT(IP),LT,0,GOTO58
3142 IDOT(IP)=IDOT(IP)+IOFF
3141 IDT=IDOT(IP)
3142 IF,IDOT(IP-1),GE,IOFF,GOTO45
3144 IROW=IDOT(IP-1)
3146 IPBUF(IWORDL,JROW( IROW))=IPBUF(IWORDL,JROW( IROW)).OR.MASK(IBITL)
3147 IROW=IROW+1
3148 IF,IROW,LT,IOFF,GOTO55
3149 IROW=IROW+1
3150 IPBUF(IWORDL,JROW( IROW))=IPBUF(IWORDL,JROW( IROW)).OR.MASK(IBITL)
3152 IROW=IROW+1
3154 GOTO38

C SECTION WHERE NEGATIVE (UNSHADED) LOBES ARE PLOTTED

C

3155 IF,IDOT(IP),LT,0,GOTO58
3156 IF,IDOT(IP),LT,1.OR,IDOT(IP-1),LT,1,GOTO38
3158 IDTB=IDOT(IP)
3161 IDTE=IDOT(IP-1)
3163 MSKB=MASK(IBITL)
3164 IWORDB=IWORD
3165 IWORDE=IWORDL
3166 GOTO38
3167 IROW=IDOT(IP-1)
3168 IDT=IDOT(IP)
3169 MSKB=MASK(IBITL)
3170 MSKE=MASK(IBITL)
3171 IWORDB=IWORDL
3172 IWORDE=IWORD
3173 IDTH=(IDTB+IDTE)/2
3174 IROW=IDTB
3175 IPBUF(IWORDB,JROW( IROW))=IPBUF(IWORDB,JROW( IROW)).OR.MSKB
3177 IROW=IROW+1
3179 IROW=IDTM
3180 IPBUF(IWORDE,JROW( IROW))=IPBUF(IWORDE,JROW( IROW)).OR.MSKE
3181 IROW=IROW+1
3182 IF,IROW,LE,IDTE,GOTO95
3184 CONTINUE
3185 JST=JROW(NDSTEP)
3186 JFIN=JROW(NDSTEP)
FORTAN IV

C FILL OUTPUT BUFFER
C
DO 150 L=1,JFIN
DO 150 J=129,256
IDOT(LIN)=IPBUF(J,L)
LIN=LIN+1
CONTINUE
CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLK)

C DO SECOND SLICE IF NECESSARY
C IF(NTIM.LT.2)GOTO 175
LIN=1
DO 160 L=1,JFIN
DO 160 J=1,128
IDOT(LIN)=IPBUF(J,L)
LIN=LIN+1
CONTINUE
CALL BUFOUT(BUF,OUTFLG,TPDRW2,IBLKW2)

C SORT JROW ARRAY
C CALL APPUT(JROW,500,50,1)
CALL APGET(JROW,NDSTEP,0,NDSTEP,1)
CALL APGET(JROW(1-NDSTEP),500,50,NDSTEP,1)
CALL APWD
CONTINUE

C FLUSH BUFFER AT END OF A STRIP
IRBEG=1
IRFIN=NDSTEP
DO 190 IF=1,3
JST=JROW(IRBEG)
JFIN=JROW(IRFIN)

C DO FIRST BUFFER
C
DO 200 L=1,JFIN
DO 200 J=129,256
IDOT(LIN)=IPBUF(J,L)
LIN=LIN+1
CONTINUE
CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLK)

C DO SECOND BUFFER IF NECESSARY
C IF(NTIM.LT.2)GOTO 195
LIN=1
DO 2118 L=JST,JFIN
     DO 2118 J=1,128
     IDOT(LIN)=IPBUF(J,L)
     LIN=LIN+1
     2118 CONTINUE
     CALL BUFOUT(BUF,OUTFLG,TPDRW,IBLK)
DO 231 195 IRBEG=IRBEG+NDSTEP
       IRFIN=IRFIN+NDSTEP
       CONTINUE
     CALL BUFOUT(BUF,OUTFLG,TPDRW,IBLK)
     IRBEG=IRBEG+NDSTEP
     IRFIN=IRFIN+NDSTEP
     CONTINUE
     CALL BUFOUT(BUF,OUTFLG,TPDRW,IBLK)

C PUT OUT EXTRA LINES TO HELP WITH TONER PROBLEM
C
DO 2318 ICL=1,NWDBF
     IDOT(ICL)=0
C SEND OUT EXTRA LINES
C
DO 2318 ISEN=1,218
     CALL BUFOUT(BUF,OUTFLG,TPDRW,IBLK)
     IF(NTIM.GE.2) CALL BUFOUT(BUF,OUTFLG,TPDRW,IBLK)
     CONTINUE
     CALL CLOSE(IWRT)
     STOP 'NORMAL TERMINATION'
END

SUBROUTINE BUFOUT(BUF,OUTFLG,TPDRW,IBLK)
C
C THIS ROUTINE FOR PUTTING OUT THE BUFFERS FROM THE PLOTTING PROGRAM MPSPLT
C
INTEGER*2 IBUF(1),OUTFLG
LOGICAL*1 TPDRW,ISTAT
COMMON /IO/NBLKBF,NWDBF,NBYTBF,ICHAN
DATA ICOUNT/0/
C
KEEP A COUNT OF NUMBER OF BUFFERS WRITTEN
C
ICOUNT=ICOUNT+1
IF(OUTFLG.EQ.0) GOTO 10
IF(IWRITW(NWDBF,IBUF(257),IBLK,ICHAN).LT.0) STOP 'WRITE ERROR'
IBLK=IBLK+NBLKBF
RETURN
C
DO TAPE OUTPUT
C
10 IF(NBYTBF.GE.512) RETURN
IFLEN=NBLKBF+1
IEOTW=0
CALL TAPSUB(1,TPDRW,ISTAT,IFLEN,ICOUNT,IBUF,NBYT,IEOTW)
C
CHECK FOR ERRORS
C
IF(IEOTW.GE.0) GOTO 20
WRITE(7,1000)TPDRW,ICOUNT
1000 FORMAT(' EOT ON WRITE DRIVE:','I2,' BUFFER NUMBER:'I5')
WRITE(7,1100)
1100 FORMAT(' ENTER NEW WRITE DRIVE NUMBER:','S')
READ (5,1200)TPDRW
1200 FORMAT(I1)
IEOTW=0
IF(TPDRW.GT.2) STOP ' EOT TERMINATE'
IF(ISTAT.GE.0) GOTO 20
WRITE(7,1300)ICOUNT
1300 FORMAT(' FATAL WRITE ERROR ON BUFFER:',I5)
STOP ' WRITE ERROR TERMINATION'
30 RETURN
END
Section Plot Background :- MPPLEK

Input file.....DK2:MPVDAT.DAT

Input Parameters

This program is interactive in its first stage, but input such as velocity functions and annotation comes from the input file.

Interactive input

TSEC.....Trace length in seconds
TDELAY...Time delay to first sample
TSPACE...Trace spacing in plot dots
ISTART...First trace number
IEND.....Last trace number
ISCANS...Interpolation factor used in trace plot
IBKFLG...Background grid flag
  0 - don't plot
  1 - plot background grid
IDSCAN...Documentation flag
  0 - no documentation for plotting
  1 - documentation in input file for plotting
IVSCAN...Velocity Boxes flag
  0 - No velocity information for plotting
  1 - Velocity information in input file
The input to the velocity and documentation parts of the program should be present in the input file, in the format shown below.

```
READ(1,1000)IVCNT

1000 FORMAT(12I5)

READ(1,1000)(IVEL(I),I=1,IVCNT)
```

For each velocity function the format is as below:

```
READ(1,1000)NVEL

READ(1,1001)(T(I),VINT(I),VRMS(I),I=1,NVEL)
```

IVCNT.....Number of velocity functions
IVEL.....Trace positions at which velocity functions are defined
NVEL.....Number of layers in a velocity function
T........Time value to be written in velocity box
VINT.....Interval velocity value to be written in velocity box
VRMS.....RMS velocity values to be written into velocity box

```
READ(1,1002)TITLE

READ(1,1000)ILINES

READ(1,1002)(LINE(I),I=1,ILINES)

1002 FORMAT(80A1)
```
TITLE....Title to be put into documentation box
ILINES...Number of lines of annotation
LINE.....80 characters of annotation per line
C PLOTTING PROGRAM TO GENERATE THE TIME AND POSITION BACKGROUND GRID FOR SECTION PLOTS

C

DIMENSION IVEL(200),ATIME(200),AVINT(200),AVRMS1(200)
COMMON /PPEP1/I1X1,I1Y1,I1X2,I1Y2,ISCN,NISCN,NBAND,NPIS,NPB,
>NPNI,LYNE,NIBSX,MSGVL,XDOTS,YDOTS,PREF(2),
>XYRGE,OFF,MBITS,MBYTE1,MBYTEM,MSK,LMSK

DATA LMASK1/"104210\",LMASK2/"177777/
CALL ASSIGN(1,'DK2:MPVDAT,DAT',14)
CALL PLOTSIS,B,BJ
DATA INPUT
WRITE(7,1000)
1000 FORMAT( 'ENTER TRACE LENGTH IN SECS(F10.0):')
1001 READ(5,1001) TSEC
1002 FORMAT('ENTER TIME DELAY TO START OF TRACES(F10.0):')
1003 READ(5,1003) DELAY
1004 FORMAT('ENTER TRACE INTERVAL IN INCHES(F10.0):')
1005 READ(5,1005) TSPACE
1006 FORMAT('ENTER START TRACE NO(I5):')
1007 READ(5,1007) START
1008 FORMAT('ENTER LAST TRACE NUMBER(I5):')
1009 READ(5,1009) END
1010 FORMAT('ENTER THE SCAN AMPLIFICATION FACTOR(I1):')
1011 READ(5,1011) SCAN
1012 FORMAT('ENTER 1 FOR GRID PLOTTING 0 IF NOT(I):')
1013 READ(5,1013) BKFLG
1014 FORMAT('ENTER 1 IF WANT DOCUMENTATION 0 IF NOT(I):')
1015 READ(5,1015) IDSCAN
1016 FORMAT('ENTER 1 IF VELOCITIES TO BE PLOTTED 0 IF NOT(I):')
1017 READ(5,1017) IVSCAN

C BASIC PROCESSING PARAMETERS

TPAPER=12.56
TMAX=8.192
TLENGTH=10.24
DSHIFT=1.0
TSHIFT=1.6
DSIZE=7.25
**EXPLANATION OF VARIABLES:**

- **TPAPER**: Basic paper width
- **TSPACE**: Trace interval in inches
- **TSEC**: Trace length in seconds
- **TMAX**: Max scan in seconds
- **TLENGTH**: Display width in inches for traces
- **ISTART**: First trace number
- **IEND**: Last trace number
- **ISCAN**: Trace amplification
- **IDSCAN**: Cosmetics flag, 1=on, 0=off
- **DSHIFT**: X shift from origin
- **TSHIFT**: Separation of T slices
- **DSIZE**: Declaration size
- **IVSCAN**: H-velocity display
- **VMAX**: Max velocity
- **VBOX**: Velocity box size
- **ISCAN**: Trace amplification
- **IDSCAN**: Cosmetics flag, 1=on, 0=off
- **DSHIFT**: X shift from origin
- **TSHIFT**: Separation of T slices
- **DSIZE**: Declaration size
- **IVSCAN**: H-velocity display
- **VMAX**: Max velocity
- **VBOX**: Velocity box size

**FORM DISPLAY CONSTANTS AND FIND NO OF TIME SLICES**

```fortran
10 IF(TEQ.LE.TSTART) GOTO 11
11 TSTART=TMAX/FLOAT(ISCAN)
```

**IPAD=NO OF TIME PADS**

```fortran
20 TSTART=NO OF TRUE SECONDS PER PAD
```

**STEPS FOR 1 AND 1/10 SEC INTERVALS**

```fortran
10 ITRACE=IEND-ISTART+1
```

**NO OF TRACES AND DECADE POINTS**

```fortran
10 IT1=ISTART/I0
IT2=IT1*10
IST1=IT2+10-ISTART
IF(I1ST.EQ.1T2)ITR10=ITR10-1
```

**GET TRACE AND DECADE STARTING POSITIONS**
AND REDUCE NO OF DECADES BY 1 IF STARTING ON A DECADE

STARTING POSITIONS FOR TRACES AND DECADES

WIDTH OF DISPLAY AND DECADE TRACE INTERVAL

C STARTING DECADE NUMBER

C UPPER DISPLAY LIMIT

C ANNOTATION PARAMETERS

TOTAL DISPLAY SIZE

BEGIN MAIN DISPLAY LOOP

SWITCHING OUT VELOCITY PLOTS WHEN NOT REQUIRED

DRAW BORDER FOR FIRST FRAME
IF(IPAD.EQ.0)CALL PLOT(X1,8.8,2)

CALL NEWPEN(1)
CALL PLOT(8.8,8.8,3)
CALL PLOT(X1,TLENTH,3)
CALL PLOT(X2,TLENTH,2)
CALL PLOT(TWIDTH,TUPPR,3)
CALL PLOT(TWIDTH,8.8,2)

CALL SYMBOL(X, VA, PSIZE, 3HSEC, 98)
X=X+X2
CALL SYMBOL(X, VA, PSIZE, 3HSEC, 98)

VA=TLENTH+PSIZE*8.8
X=-TSLICE*8.33
CALL SYMBOL(X, VA, PSIZE, 3HSEC, 98)

X=X+XPO1B
V=TLENTH+8.33*(TUPPR-TLENTH)
COUNT=FLOAT(ITNUM)
JLIM=ITRIB-1
DO IPJPP1
I=1,ITRU
CALL NUMBER(X, V, PSIZE, COUNT, 8.8, 1)
X=X+TSPACE
COUNT=COUNT+18.8

IF(IVSCAN.EQ.8)GOTO 31
READ(1,25)IVCNT
25 FORMAT(12I5)
IF(IVCNT.LE.8)GOTO 31

IVCNT=NO OF VELOCITY SCANS
READ(1,25)(IVEL(I), I=1,IVCNT)
READ IN VEL POSITIONS
XBASE=8.5*TSPACE
YBASE=TLENTH
DO 29 I=1,IVCNT
121 ISHIFT=IVEL(I)-ISTART
SHIFT=FLOAT(ISHIFT)*TSPACE+XBASE-8.5*PSIZE
29 CALL SYMBOL(SHIFT, YBASE, PSIZE, 14, 8.8, -1)
GOTO 31

BORDER FOR OTHER DATA PADS THAN FIRST
C INSERT SHOT POSITION LINES

!0136 21 ITR=ITRACE-1
!0139 CALL GRID(XPOPST,H.S,ITR,TSPACE,-1,TLLENGTH,LMASK1)
!0140 ITR=ITR-1
!0141 CALL GRID(XPOP10,H.S,ITR,TSPACEA,-1,TLLENGTH,LMASK2)

C INSERT TIMING LINES

!0142 TBASE=(FLOAT(ISCNT-1)*TSTART*10.0)+(TDELAY*10.0)
!0143 ITI=IFIX(TBASE)
!0144 T1=FLOAT(IT1)
!0145 TERR=TBASE-T1
!0146 TBASE=TBASE*0.1
!0147 IT1=IFIX(TBASE)
!0148 T2=FLOAT(IT1)
!0149 TERR1B=TBASE-T2
!0150 IF(TERR1B.GE.0.0)TERR1B=TERR1B-1.0
!0151 T1ST=TLLENGTH+D1T1M*(1.0-TERR1B)
!0152 TINT=(TSTART+TERR1B)*10.0
!0153 ITR=IFIX(TINT)
!0154 CALL GRID(H.S,T1ST,-1,TWIDTH,ITR1,T1ST1M,LMASK1)
!0155 I1=IFIX(TSTART)
!0156 ITR=ITR-1
!0157 CALL GRID(H.S,T1ST,-1,TWIDTH,ITR1,T1ST1M,LMASK2)

C INSERT ANNOTATION ON SECONDS

!0158 X=-0.66*TSLICE
!0159 X1=TWIDTH+0.33*TSLICE
!0160 Y=T1ST
!0161 TINT=IFIX(0.1*T1+1.0)
!0162 TIM=FLOAT(TINT)
!0163 DO 110 I=1,11
!0164 CALL NUMBER(X,Y,TSIZE,TIM,H.S,-1)
!0165 CALL NUMBER(X1,Y,TSIZE,TIM,H.S,-1)
!0166 TIM=TIM+1.0
!0167 Y=Y+D1T1M

PAGE 155
VELOCITY ANALYSIS PLOT PROGRAM

IF(IVSCAN.EQ.9)GOTO 641
DO 64 1=1,IVCNT
READ(1,26)ATIME1(J),AVINT1(J),AVRMSI(J),J=1,NVEL
618 26 FORMAT(3F15.8)
IVEL1=IVEL(I)
VBSIZE=0.22*VBOX
VBLINE=0.11*VBOX
VBSTX=VPLT1-0.5*VBSIZE*3.0
VBSTY=TLNGTH*(0.5-AMARK*0.25)-23.0*VBLINE
CALL PLOT<VBSTX,VBSTY,3>
CALL GRIO<VBSTX,VBSTY,3,VBSIZE,22,VBLINE,LMASK2>
SIZE=0.04*VBOX
COFFST=0.025*VBOX
Y=TLNGTH*(0.5-AMARK*0.25)-VBSIZE+COFFST
X=VPLT1-2.0*SIZE
CALL NUMBER(X,Y,SIZE,AV,0.0,-1)
Y=Y-VBLINE
X=VBSTX+0.1*VBSIZE
XST=X

DO TITLES
CALL SYMBOL(X,Y,SIZE,4HTIME,0.0,4)
X=X+VBSIZE
CALL SYMBOL(X,Y,SIZE,4HVINT,0.0,4)
X=X+VBSIZE
CALL SYMBOL(X,Y,SIZE,4HVRMS,0.0,4)
Y=Y-VDLINE
X=XST

PUT IN THE NUMBERS
DO 63 JJ=1,NVEL
CALL NUMBER(X,Y,SIZE,ATIME1(JJ),0.0,3)
X=X+VBSIZE
CALL NUMBER(X,Y,SIZE,AVINT1(JJ),0.0,3)
X=X+VBSIZE
CALL NUMBER(X,Y,SIZE,AVRMS1(JJ),0.0,3)
X=XST
Y=Y-VDLINE
63 Y=Y-VDLINE
IF(AMARK.EQ.0)GOTO 632
AMARK=0.0
C PLOT COSMETICS

C IF (IDSCAN.EQ.0) GOTO 73
C XSTART=TWIDTH+TSHIFT
C YSTART=0.0
C CALL PLOT(XSTART,YSTART,-3)

C REORIGIN

C X2=SIZE
C Y2=WIDTH
C CALL NEWPEN(4)
C CALL PLOT(0.0,Y2,2)
C CALL PLOT(X2,Y2,2)
C CALL PLOT(X2.0.0,2)
C CALL PLOT(0.0.0.0,2)
C X1=0.05
C X2=X2-X1
C V1=V
C V2=V2-V1
C CALL NEWPEN(1)
C CALL PLOT(X1,V1,3)
C CALL PLOT(X1,V2,2)
C CALL PLOT(X2,Y2,2)
C CALL PLOT(X2,Y1,2)
C CALL PLOT(X1,Y1,2)
C CALL PLOT(0.0.0.0,3)

C SURROUND DONE

C SIZE1=0.4
C SIZE2=0.3
C SIZE3=0.2
C SIZE4=0.1
C XMID=0.5*DSIZE
C Y=LENGTH
C X=XMID-8.0*SIZE1
C Y=Y-1.5*SIZE1

C DRAW IN THE HEADER

C CALL NEWPEN(3)
C CALL SYMBOL(X,Y,SIZE1,17DURHAM UNIVERSITY,0.0.17)
C Y=XMID-9.0*SIZE2
C Y=Y-1.5*SIZE2
C CALL NEWPEN(1)
C CALL SYMBOL(X,Y,SIZE2,18SEISMIC PROCESSING,0.0.18)
C CALL NEWPEN(1)
C X=0.12
C READ IN TITLE AND PLOT

READ(1,7) (IVEL(I), I=1, 4)

CALL SYMBOL(X, Y, SIZE3, IVEL, 0, 0, 0)

Y = Y - 0.2 * SIZE3

X = 0.0

CALL PLOT(X, Y, 3)

CALL PLOT(DSIZE, V, 2)

Y = Y - 1.5 * SIZE3

X = XMID - 0.0 * SIZE3

CALL SYMBOL(X, Y, SIZE3, 17, SYSTEM PARAMETERS, 0.0, 17)

X = 0.1 * DSIZE

Y = Y - 1.5 * SIZE3

READ(1, 25) I_LINES

C FILL IN PROCESSING PARAMETERS

DO 72 I = 1, I_LINES

READ(1, 7) (IVEL(J), J = 1, 4)

CALL SYMBOL(X, Y, SIZE4, IVEL, 0, 0, 0)

Y = Y - 1.5 * SIZE4

CONTINUE

CALL PLOT(0.0, 0.0, 999)

STOP

END
Section Plotting Merge and Post-Process :– MPMERG

Input file.....DK2:MPMERG.DAT
Log file.......DK1:MPMERG.LOG

Input Parameters

READ(1,1000)LPLT,NSTRIP,NDSTEP,IBKFLG,IPLFLG

1000 FORMAT(5I5)

LPLT....Number of rasters in total plot
NSTRIP...Number of strips to be plotted
NDSTEP...Number of dots between traces
IBKFLG...Background flag
  0 - No background to plot
  1 - Plot a background
IPLFLG...Trace plot flag
  0 - No trace plot
  1 - Plot raster trace plot

READ(1,1000)LSTP,LENP,IOFLGP,ITPDR1,ITPDR2

LSTP....Output raster position of first raster in trace plot
LENP....Output raster position of last raster in trace plot
IOFLGP...Input flag for trace plot
  0 - read from tape
  1 - read from disc
ITPDR1...Input Tape drive number 1
ITPDR2...Input tape drive number 2

IF(IOFLGP.NE.0)READ(1,1002)FBUF

FBUF...Input file for trace plot

READ(1,1000)LSTB,LFINB,IOFLGB,ITPDRB

LSTB.....Output line position for first line of background plot
LFINB.....Output line position of last line in background plot
IOFLGB...Input flag for background plot
    0 - input from tape
    1 - input from disc
ITPDRB...Input tape drive for background plot

IF(IOFLGB.NE.0)READ(1,1002)FBUFB

FBUFB....Input file for background plot
M.J. POULTER OCT 80

POST PROCESS AND MERGE PROG FOR SEISMIC PLOT SYSTEM

INTEGER*2 IPBUF(2112), LBUF(132)
REAL*4 FBUF(3)
REAL*8 FSPECP, FSPECB
LOGICAL*1 ITPDR1, ITPDR2, ITPDRP, ITPDRB
DATA DEV/3RRK /

SET UP AP TO USE FOR ZEROING ARRAYS

CALL VCLR(0,1,2112)

SET UP I/O DEFINITIONS

IF(IFETCH(DEV).NE.0) STOP 'FETCH ERR'
CALL ASSIGN(1,'DK1:MPMERG.DAT',14)
CALL ASSIGN(2,'DK1:MPMERG.LOG',14)

GET INPUT PARAMETERS

READ(1,1000) IPLT,NSTRIP,NDSTEP,IBKFLG,IPFLG

I013 1000 FORMAT(S16)
I014 IF(IPFLG.EQ.0) STOP 'LOOKUP ERR'
I015 READ(1,1000)LSTP,LFNPG,IOFLGP,ITPDR1,ITPDR2
I016 IF(IOFLGP.EQ.0) STOP 'LOOKUP ERR'
I017 IF(IOFLGP.EQ.0) GOTO 10
I018 CALL IRAD5.G'(12,1,FBUF,FSPECB)
I019 ICBB=IGETC()
I020 IF(LOOKUP(ICH,FSPECB).LT.0) STOP 'LOOKUP ERR'
I021 READ(1,1000)LSTB,LFINB,IOFLGB,ITPDRB
I022 IF(IOFLGB.EQ.0) STOP 'LOOKUP ERR'
I023 READ(1,1000) FBUF
I024 CALL IRAD5.G'(12,1,FBUF,FSPECB)
I025 ICBB=IGETC()
I026 IF(LOOKUP(ICH,FSPECB).LT.0) STOP 'LOOKUP ERR'

SET UP M1ATRIX FOR MAIN LOOP

CALL MTXSET
CALL MTX(IPBUF,0.1)
LNUM=9
NTIM=0
IST=1957
NBTP=NDSTEP*256+512
NBYTB=4670
LIMP=4680
NWDP=NDSTEP/2*256
IF(IOFLGP.EQ.0)LIMP=256*NWDP
LIMP=2384
ZERO THE PLOT ARRAY

IST = IST + 2

CALL APGET(IPBUF, IST, 56, 1)
IPSV = IST

RIB 56
RIBS?

CALL APWD

START OF MAIN LOOP

DO 50 I = 1, 8
LNUM = LNUM + 1
EOFFLG = 1

THIS LOOP FILLS ONE LINE OF PLOT BUFFER
WITH INPUT FROM EACH PLOT MASK

DO SEISMIC SECTION FIRST

IF(IPLFLG .EQ. 0) GOTO 60
IF(LNUM .GT. LFINP) GOTO 60
IF(IEOFP .NE. 0) GOTO 60
IF(LNUM .LT. LSTPI) GOTO 60
IPOS = IPSV
ICD = 1

CALL LINFIL(LBUF, ICD, IEOF, ITPDR, IOFLGP, NBVTP, LNUM, LCHP, LIMP)
DO 70 J = 1, 132
IPBUF(IPOS) = IPBUF(IPOS) .OR. LBUF(J)
IPOS = IPOS + 1

DO BACKGROUND

IF(IBKFLG .EQ. 0) GOTO 80
IF(LNUM .GT. LFINB) GOTO 80
IF(IEOFB .NE. 0) GOTO 80
EOFFLG = 0
IF(LNUM .LT. LSTBP) GOTO 80
IPOS = IPSV
ICD = 2

CALL LINFIL(LBUF, ICD, IEOF, ITPDR, IOFLGB, NBYTB, LNUM, LCHB, LIMP)
DO 80 J = 1, 132
IPBUF(IPOS) = IPBUF(IPOS) .OR. LBUF(J)
IPOS = IPOS + 1

80 CONTINUE

CONTINUE
C WHEN A PLOT BUFFER IS FULL COME HERE TO EMPTY IT

CALL MWAIT
CALL MTX(IPBUF(1ST),1856,2)
IF(EOFFLG.NE.0)GOTO 188
IF(LNUM.LT.LPLT)GOTO 48
188 CALL MWAIT
CALL APGET(IPBUF,2112,1)
DO 1118 J=1,2
CALL MWAIT
118 CONTINUE
CALL MTX(IPBUF,2112,2)
CALL MWAIT
118 CONTINUE
CALL MTX(IPBUF,0,1)
CALL MWAIT
NTIM=NTIM+1
ITPDRP=ITPDR2
IF(NTIM.LT.NSTRIP)GOTO 48
STOP 'NORMAL TERMINATION'
END
SUBROUTINE UNFILBUF,
ICD,
IEOF, IDRV, IOFLG, NBYTR, IFIL, ICHAN, NLIM
INTEGER*4 LBUF(1), INBUF(5120), ITPST(2), IST(2), NPTS(2),
%IBLKS(2), IPOSS(2)
LOGICAL*1 IDRV, ISTAT
DATA ITPST(1)/2301/, ITPST(2)/1/, IST(1)/2561/,
%IST(2)/257/, NPTS(1)/128/, NPTS(2)/132/, IPOSS(1)/5120/,
%IPOSS(2)/6120/, IBLKS(1)/1/, IBLKS(2)/1/, IEOTR/0/
C C CLEAR LINE BUFFER C
CALL APGET(LBUF,8,132,1)
NPT=NPTS(ICD)
IPOS=IPOSS(ICD)
IBLK=IBLKS(ICD)
IST=ITPST(ICD)
IBEG=IST(ICD)
CALL APWD
C C FILL BUFFER C
DO 16 J=1,NPT
IPOS=IPOS+1
IF(IPOS.LE.NLIM)GOTO 3
IF(IOFLG.EQ.0)GOTO 4
C DISC INPUT C
IN=IREADV(248, INBUF(IBEG), IBLK, ICHAN)
IBLK=IBLK+8
IPOS=IBEG
IF(IN.EQ.248)GOTO 3
IEOF=1
NLIM=IN+IPOS
IF(IN.LT.-1)STOP 'READ ERROR'
GOTO 16
C TAPE INPUT C
ITRV=1
IF(ITRV.GT.3)GOTO 600
IF(IEOTR.GE.B)GOTO 7
WRITE(7,1F0)IDRV
FORMAT('EOT ON READ DRIVE:',I2,/,%
ENTER NEW DRIVE NUMBER:','$$)
READC5,1!F!1,IDRV
IF(ITRV.LT.1)GOTO 8
CALL TAPSUB(-1, IDRV, ISTAT, IFLEN, IFIL, INBUF(IST), NBYTR, IEOTR)
IF(ITRV.GT.1)RETURN
700 WRITE(2,1002)IFIL
FORMAT(' WARNING RETRIES FAILED ONREAD FILE NO: ''15')
IF(IEOTR.LT.0)WRITE(80)
C FORTRAN IV V82.04 THU 08-JAN-81 08:48:09 PAGE 002
C CALL TAPSUB(9, IDRV, ISTAT, , IFIL, , , IEOTR)
ITRV=ITRV+1
IF(INBUF(IST).EQ.1777777)GOTO 50
IPOS=IBEG
CONTINUE
LB (J)=INBUF(IPOS)
CONTINUE
IBLK(IDC)=IBLK
IPOS(ICD)=IPOS
RETURN
END
Gather Plotting(Small Trace separation) :- MPSPLI

This program is interactive, it expects the data to be on disc and the output rasters are put back to a user specified disc file.

The following parameters have to be input.

- NTR......Number of traces to plot
- NPT......Number of samples on each trace
- NDPT.....Interpolation factor, number of dots per trace
- NDSTEP...Trace separation in dots
- XSF......Plot scale factor
- FSPECR...Input data file
- FSPECW...Output raster file
M J POULTER SEPT 79

PLOTTING PROGRAM FOR SEISMIC SECTION DATA
THIS PROGRAM TAKES IN SEISMIC TRACES AND
DISPLAYS THEM IN A NIB IMAGE FORM IN AN OUTPUT
FILE IN A FORM READY FOR POST PROCESSING.

DIMENSION IPBUF(128,80),JROW(80),FBUF(3),
IDOT(2048),XBUF(2052),MASK(16),NSAVE(20)
REAL*8 FSPEC,FSPECW
EQUIVALENCE (XBUF(1),IDOT(1))
DATA MASK/"200","100","00","00","00","00","00","00","00"

DATA READ IN SECTION

WRITE(7,1005)
WRITE(7,1006)
READ(5,1000) NTR,NPT,NDPT,NDSTEP
FORMAT(415)
WRITE(7,1007)
FORMAT(415)
READ(5,1001) XSF
READ(5,1002)FBUF
CALL IRA50(12,FBUF,FSPEC)
WRITE(7,1008)
READ(5,1003)FBUF
CALL IRA50(12,FBUF,FSPECW)

SET UP CONSTANTS AND FILE ACCESS

ICHR=IGETC()
ICHW=IGETC()
IF(2*FETCH(DEV).NE.0)STOP 'FETCH ERR'
IER=LOOKUP(ICHR,FSPEC)
IF(IER.LT.0)WRITE(7,*)IER
IF(IER.LT.0)STOP 'LOOKUP ERR'
NFIN=NPT*NDPT-1
NBLK=NDSTEP/2
NBLKBF=NBLK/2
NWBFL=(NTR+4)*(NDSTEP/2)
IBUF=NBLKBW*256
IOFF=2*NDSTEP
ISLKR=1
SECTION WHICH DEALS WITH INTERPOLATION AND SCALING OF DATA BEFORE PLOTTING

CALL APPUT(XBUF,2048,NPT+4,2)
CALL APWD
CALL VMOV(2048,1,0,NDPT,NPT)
IF(NSPT.EQ.1)GOTO 25
CALL VSUB(2048,1,2049,1,6144,1,NPT-1)
CALL VSMUL(6144,1,2048+NPT,6144,1,NPT-1)
IBEG=0
IFIN=1
DO 20 J=2,NDPT
CALL VADD(IBEG,NDPT,6144,1,IFIN,NDPT,NPT-1)
IBEG=IBEG+1
IFIN=IFIN+1
20 CONTINUE
CALL VSMUL(0,1,2051+NPT,0,1,NFIN)
CALL VTSADD(0,1,4427,0,1,NFIN)
CALL VINFT(0,1,0,1,NFIN)
CALL VFIX(0,1,0,1,NFIN)
CALL APWR
CALL APGET(I,0,NFIN,1)
IBIT=16
IWORD=128
CALL APWD
IDOIT=IDOIT+1+IOFF
DO 30 IP=2,NFIN
IBITL=IBIT
IWORDL=IWORD
IF(IP.LE.0)GOTO 40
IF(IWORD.GT.0)GOTO 40
IWORD=IWORD-1
30 CONTINUE
IF(IWORD.LT.0)GOTO 60
IBIT=16
CALL APWR
CALL APGET(I,0,NFIN,1)
IBIT=IBIT+1
IF(I,0.LE.0)GOTO 40
IF(I,0.GT.0)GOTO 40
IWORD=IWORD-1
50 CONTINUE
SECTION WHERE POSITIVE (SHADED) LOBES ARE PLOTTED

IF(IDOT(IP).LT.0)GOTO 50
IDOIT(IP)=IDOIT(IP)+IOFF
ID=IDOIT(IP)
IF(IDOT(IP-1).GE.IOFF)GOTO 45
IF(IOFF.LE.IDT)GOTO 45
IBUF(IP,0ROW(IP,0ROW)=IPBUF(IP,0WORDL,0ROW(IP,0ROW)).OR.MASK(IBITL)
IBUF(IP,0ROW(IP,0ROW)=IPBUF(IP,0WORDL,0ROW(IP,0ROW)).OR.MASK(IBIT)
IF(IOFF.LE.IDT)GOTO 60
GOTO 30

SECTION WHERE NEGATIVE (UNSHADDED) LOBES ARE PLOTTED
58 IDOT(IP)=IDOT(IP)+IOFF
59 IF(IIDOT(IP),LT.1.OR.IDOT(IP-1),LT.1)GOTO 38
60 IF(IIDOT(IP),GT.IDOT(IP-1))GOTO 78
61 IDTB=IDOT(IP)
62 IDTE=IDOT(IP-1)
63 MSKB=MASK(IBIT)
64 MSKE=MASK(IBITL)
65 IWORDB=IWORD
66 IWORDE=IWORDL
67 GOTO 80
68 IDTB=IDOT(IP-1)
69 IDTE=IDOT(IP)
70 MSKB=MASK(IBITL)
71 MSKE=MASK(IBIT)
72 IWORDB=IWORD
73 IWORDE=IWORDL
74 GOTO 80
75 IDTM=(IDTB+IDTE)/2
76 IROW=IDTB
77 IF(IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
78 IROW=IROW+1
79 IF(IROW.LE.IDTM)GOTO 90
80 IROW=IDTM
81 IF(IROW.LE.IDTE)GOTO 180
82 CONTINUE
83 JST=JROW(1)
84 WRITE OUT PART OF BUFFER AND INITIALISE
85 FOR A NEW TRACE
86 IF(IWRITE(NWBUF,IPBUF(1,JST),IBLK,W,ICHW),LT.5)STOP 'WRITE ERR'
87 IBLKW=IBLKW+NBLKBF
88 CALL APPUT(JROW,0,B,1)
89 CALL APGET(JROW,NDSTEP,0-NDSTEP,1)
90 CALL APGET(JROW(1-NDSTEP),0,NDSTEP,1)
91 CALL APWD
92 CALL VCLR(0,1,NWBUF)
93 CALL APGET(IPBUF(1,JST),0,NWBUF,1)
94 CONTINUE
95 IROW=1
96 FLUSH BUFFER AT END OF A STRIP
97 DO 110 IF=1,3
98 IF(IWRITE(NWBUF,IPBUF(1,JROW(IROW),IBLK,W,ICHW),LT.0)STOP 'WRT ER'
99 IBLKW=IBLKW-NBLKBF
100 IROW=IROW+NDSTEP
101 CONTINUE
110 CHECK IF MORE STRIPS TO BE DONE
111 IF(NSTRIP.LT.NTIM)GOTO 4
112 CALL CLOSEC(ICHW)
Gather Ploting(Large Separation) :- MPPGLI

This program is interactive, and it expects the seismic data to be in a disc file. The output rasters are written to a user specified disc file. Interactive input consists of the following.

NTR......Number of traces to be plotted
NPT......Number of samples per trace
NDPT.....Interpolation factor, dots per sample
XSF......Plot scale factor
FSPECR....Input data file
FSPECW....Output raster file
C PROGRAM TAKES IN SEISMIC TRACES AND DISPLAYS THEM IN A NIB IMAGE FORM IN AN OUTPUT FILE IN A FORM READY FOR POST PROCESSING.

DIMENSION IPBUF(128,80),JROW(80),FBUF(3),
XDOT(2048),XBUF(2052),MASK(16),NSAVE(20)
REAL*8 FSPEC,R,F,FSPECW
EQUIVALENCE (XBUF(1),IDOT(1))
DATA DEV/3RRK/
DATA MASK/"200","100","40","20","10","4","2","1",
"1","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","

DATA READ IN SECTION

WRITE(7,1005)
1005 FORMAT( 'ENTER NO OF TRACES',/,
'ENTER NO OF POINTS PER TRACE',/,
'ENTER NO OF TIMES TO EXPAND'
)
READ(5,1006) NTR,NPT,NDPT
WRITE(7,1007) NTR,NPT,NDPT
WRITE(7,1006)
1006 FORMAT( 'ENTER SCALE FACTOR:',$)
READ(5,1001) XSF
WRITE(7,1007)
1007 FORMAT( 'ENTER INPUT FILE NAME:',$)
READ(5,1002) FBUF
WRITE(7,1008)
1008 FORMAT( 'ENTER OUTPUT FILE NAME:',$)
READ(5,1007) FBUF
WRITE(7,1008)
1008 FORMAT( 'ENTER OUTPUT FILE NAME:',$)
WRITE(7,1007) FBUF

SET UP CONSTANTS AND FILE ACCESS

ICHR=IGETC()
ICHW=IGETC()
IF(ICHW.EQ.0) STOP 'FETCH ERR'
IER=LOOKUP(ICHR,FSPEC)
IF(IER.LT.0) WRITE(7,*IER)
IF(IER.LT.0) STOP 'LOOKUP ERR'
NDSTEP=2F
NBFSZ=8F
NBSN=NDSTEP/128
NBSZ=NBFSZ*256
NFIN=NPT*NDPT-1
NBLKR=NPT/128
NWD=2*NPT
NBLK=(NTR*NBFSZ/2)+(NTR*10)
NBLKB=NBFSZ/2
SECTION WHICH SETS UP PARAMETERS TO ALLOW STRIPPING OF PLOT IF REQUIRED

IF(NFIN.GT.2548)GOTO 1
1059 IF(IENTER(NCW,FSPECW,NBLKW).LT.0)STOP 'ENTER ERR'
1052 NSTRIP=1
1053 GOTO 2
1054 NS=NFIN
1055 NSAVE(NTIM)=2047
1056 NS=NS-2048
1057 NTIM=NTIM+1
1058 IF(NTIM.GT.25)STOP 'TOO MANY STRIPS'
1059 IF(NS.GT.2048)GOTO 3
1060 NSAVE(NTIM)=NS
1061 NBLKW=NBLKW*NTIM
1062 IF(IENTER(NCW,FSPECW,NBLKW).LT.0)STOP 'ENTER ERR'

START OF LOOP IF STRIPPING NECESSARY

4 NSTRIP=NSTRIP+1
2065 NFIN=NSAVE(NSTRIP)
2066 NPT=(NFIN+1)/NDPT
2067 NWDR=2*NPT
3071 NBLKR=I+IOFF
3072 NBOF=NPT/128
3073 IBOFF=IOFF+NBOF

SET UP ROW COUNTER FOR BUFFER USAGE

2DO 5 I=1,80
274 JROW(I)=I

SET UP AP AND CLEAR PLOT BUFFER

CALL APINIT
276 CALL VCLR(0,1,5120)
277 CALL APWR
278 CALL APGET(IPBUF(1,1),0,5120,2)

MAIN LOOP FOR PLOTTING DIFFERENT TRACES

279 DO 1 I=1,NTR
280 IF(IREADW(NWDR,XBUF,NBLKR,ICHR).LT.0)STOP 'READW ERR'
282 NBUF=NBLKR+NBLKR
283 XBUF(NPT+1)=1.0/FLOAT(NDPT)
284 XBUF(NPT+2)=-FLOAT(IOFF-1)
SECTION WHICH DEALS WITH INTERPOLATION AND SCALING OF DATA BEFORE PLOTTING

CALL APPUT(XBUF, 2048, NPT+4, 2)
CALL APWD
CALL VMOV(2048, 1, 6, NDPT, NPT)
IF(NDPT.EQ.1) GOTO 25
CALL VSMUL(6144, 1, 2048+1, 6144, 1, NPT-1)
IF(N=1) BEG=0
IF(N.EQ.1) GOTO 25
CALL VSUB(2048, 1, NPT+4, 1, NPT-1)
CALL VSMUL(1, 1, NPT+5, 1, NPT-1)
IBEG=0
IF(N=1) GOTO 25
CONTINUE

CALL VSMUL(1, 1, 2051+NPT, 1, NFIN)
CALL VTSADD(1, 1, 4427, 1, NFIN)
CALL VCLIP(1, 1, 2058+NPT, 1, NFIN)
CALL VINT(1, 1, NFIN)
CALL VFIX(1, 1, NFIN)
CALL APWR
IBIT=16
IWORD=128
CALL APWD
IDOT=IDOT()+IOFF
DO 3 IP=2, NFIN
IBIT=IBIT
IWORD=IWORD
IBIT=IBIT-1
IF(IBIT.GT.1) GOTO 4
IWORD=IWORD-1
IF(IWORD.EQ.0) GOTO 12
IBIT=16

SECTION WHERE POSITIVE (SHARED) LOBES ARE PLOTTED

IF(IDOT(IP).LT.0) GOTO 55
IDOT(IP)=IDOT(IP)+IOFF
IF(IDOT(IP).GT.0) GOTO 55
IF(IDOT(IP-1).GE.IOFF) GOTO 45

55 IPBUF(IPWORDL, JROW(IPROW)+1)= IPBUF(IPWORDL, JROW(IPROW)).OR.MASK(IBITL)
IROW=IROW+1
IF(IROW.LT.IOFF) GOTO 55

55 IF(IOFF) GOTO 28
65 IPBUF(IPWORD, JROW(IPROW)+1)= IPBUF(IPWORD, JROW(IPROW)).OR.MASK(IBIT)
IROW=IROW+1
IF(IROW.LT.IDT) GOTO 55
C SECTION WHERE NEGATIVE (UNSHADDED) LOBES ARE PLOTTED

50 IDOT(IP)=IDOT(IP)+IOFF
1. IF(IDOT(IP).LT.1.OR.IDOT(IP-1).LT.1)GOTO 30
2. IF(IDOT(IP).GT.IDOT(IP-1))GOTO 70
3. IDTB=IDOT(IP)
4. IDTE=IDOT(IP-1)
5. MSKB=MASK(IBIT)
6. MSKE=MASK(IBITL)
7. IWORDB=IWORD
8. IWORD=IWORDL
9. GOTO 80
10. IDTB=IDOT(IP-1)
11. IDTE=IDOT(IP)
12. MSKB=MASK(IBITL)
13. MSKE=MASK(IBIT)
14. IWORDB=IWORD
15. IWORD=IWORDL
16. GOTO 80
17. IDTM=(IDTB+IDTE)/2
18. IROW=IDTB
19. IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
20. IROW=IROW+1
21. IF(IROW.LE.IDTM)GOTO 90
22. IWORD=IDTM
23. IPBUF(IWORD,JROW(IROW))=IPBUF(IWORD,JROW(IROW)).OR.MSKE
24. IROW=IROW+1
25. IF(IROW.LE.IDTE)GOTO 110
26. CONTINUE
27. JST=JROW(1)

WRITE OUT PART OF BUFFER AND INITIALISE FOR A NEW TRACE
28. IF(IWRITE(NWBUF,IPBUF(1,JST),IBLKW,ICHW).LT.0)STOP 'WRITE ERR'
29. IBLKW=IBLK+NBKBF
30. CALL APWR
31. CALL VCLR($1,NWBUF)
32. CALL APGET(IPBUF(1,JST),$1,NWBUF,1)
33. CALL APWD
34. IF(IWRITE(NWSPAC,IPBUF(1,JST),IBLKW,ICHW).LT.0)STOP 'WRITE ERR'
35. IBLKW=IBLK+NBSPAC
36. CONTINUE
37. IROW=1

CHECK IF MORE STRIPS TO BE DONE
38. IF(HSTRIP.LT.NTIM)GOTO 4
39. CALL CLOSE(ICHW)
40. STOP 'NORMAL TERMINATION'
41. END
Quick Raster Plot Processor :- MPPROC

This program is interactive and is designed to put out rasterised trace plots onto the electrostatic plotter.

Only input parameter is the disc file containing the rasters to be plotted.
DIMENSION IBUF(132,8), IBUFC(1056), INBUF(1024), FBUF(3)
REAL*8 FSPECR
EQUIVALENCE (IBUF(1), IBUFC(1))
DATA DEV/3RRK WRITE(7,1001)
1001 FORMAT (' ENTER FILE NAME TO BE PROCESSED: ', $)
READ(5,1000) FBUF
1000 FORMAT (AA4)
CALL IRADS(12, FBUF, FSPECR)
IF (IFETCH(DEV).NE.0) STOP 'FETCH ERR'
IDCH = IGETC()
IF (LOOKUP(IDCH, FSPECR).LT.0) STOP 'LOOKUP ERR'
CALL MTXSET
CALL MTX (IBUF, 1)
IBLK = 1
IN = IREAD(1024, INBUF, INBUF, IBLK, IDCH)
IBLK = IBLK + 4
IF (IN.EQ.1024) GOTO 20
IF (IN.LT.-1) STOP 'READ ERR'
IF (IN.LE.0) GOTO 40
20 IPOS = 1
IBUF(IPOS, IROW) = INBUF(I)
IPOS = IPOS + 1
IF(IPOS.LE.128) GOTO 30
30 IPOS = 1
IROW = IROW + 1
CONTINUE
DO 35 J = 1, IN
IBUF(IPOS, IROW) = INBUF(I)
IPOS = IPOS + 1
35 IF (IPOS.LE.128) GOTO 30
IF (IN.EQ.1024) GOTO 10
CALL MWAIT
CALL MTX (IBUF, NWORD, 2)
DO 35 J = 1, IN
40 CALL MWAIT
CALL MTX(IBUF, -1, 1)
CALL MWAIT
STOP 'NORMAL TERMINATE'
END
General Purpose Raster Merge and Output: - MPMERP

Input File........DK1:MPMERD.DAT

Input Parameters

READ(1,10)NPLT,LPLT,NSTRIP

10 FORMAT(3I5)

NPLT........Number of images to be merged
LPLT........Length of output raster image
NSTRIP......Number of strips

For each of the NPLT images the following input is needed

READ(1,10)LST,LFIN

READ(1,10)ICOM,ILOG

READ(1,10)NPLIN

READ(1,30)FBUF

30 FORMAT(3A4)

LST........Output raster line number for first input line
LFIN........Output raster line number for last input line
ICOM........Data complement flag
            0 - Merge in data as it is
            1 - complement data before merging
ILOG.....Logical merge flag
    0 - use an OR to merge the data
    1 - use an AND to merge the data

NPLIN.....Number of words per raster, 128 for seismic, 132 for others

FBUF.....Input raster file
M.J. POULTER SEP 79

POST PROCESS AND MERGE PROG FOR
SEISMIC PLOT SYSTEM

INTEGER*2 IPBUF(2112), INBUF(8192), LBUF(132), IOFF(8),
NLST(8), LFIN(8), IBLK(8), NBUF(8), NPLIN(8), ISPOS(8), ICOM(8),
ILOG(8), ICHAN(8), EOF(8)

REAL*4 FBUC(3)

REAL*8 FSPEC(8)

COMMON /LFIL/ INBUF, IOFF, LBUF, IBLK, NBUF, NPLIN, ISPOS, EOF, ICHAN, ICOM

DATA IOFF/1, 1025, 2049, 3073, 4097, 5721, 6145, 7169/
DATA DEV/3RRK /

CALL APINIT
CALL VCLR(l, 1, 1056)

SET UP I/O DEFINITIONS

IF (IFETCH(DEV).NE.0) STOP 'FETCH ERR'
IF (ICDFN(30).NE.0) STOP 'CHAN DEF ERR'
CALL ASSIGN(1, 'DK1:MPMERT.DAT', 14)

GET INPUT PARAMETERS

READ(1, 10) NPLT, LPLT, NSTRIW
IF (NPLT.GT.8) STOP 'TOO MANY MASKS'
DO 28 I = 1, NPLT
READ(1, 10) LST(I), LFIN(I)
READ(1, 10) ICOM(I), ILOG(I)
READ(1, 10) NPLIN(I)
READ(1, 30) FBUC
FORMAT(3E1)
WRITE(7, *) NPLT, LPLT, LST(I), LFIN(I), ICOM(I), ILOG(I)
WRITE(7, 30) FBUC
CALL IRADSB(12, FBUC, FSPEC(I))
ICHAN(I) = 28 + I
IBLK(I) = 1
EOF(I) = 0
NBUF(I) = 1023 + IFF(I)
ISPOS(I) = NBUF(I)
IF (LOOKUP(ICHAN(I), FSPEC(I)).LT.0) STOP 'LOOKUP ERR'
CONTINUE

SET UP MATRIX FOR MAIN LOOP

CALL MTXSET
CALL MTX(IPBUF, 0, 1)
LNUM = S
NTIM = S
IST = 1057
ZERO THE PLOT ARRAY

49 IST=1056-IST+2
50 CALL APGET(IPBUF(IST),B,1056,1)
51 IPSV=IST
52 CALL APWD

START OF MAIN LOOP

43 DO 50 I = 1,8
44 LNUM=LNUM+1
45 EOFLLG=1

THIS LOOP FILLS ONE LINE OF PLOT BUFFER WITH INPUT FROM EACH PLOT MASK

46 DO 60 J=1,NPLT
47 IF(LNUM.GT.LFIN(J))GOTO 60
48 IF(EOF(J).NE.0)GOTO 60
49 EOFLLG=0
50 IF(LNUM.LT.LST(J))GOTO 60
51 IPOS=IPSV
52 ID=J
53 CALL LMF1(ID)
54 IF(LNUM.LE.LPT(J))GOTO 70
55 DO 60 L=1,132
56 IPBUF(IPOS)=IPBUF(IPOS).OR.LBUF(L)
57 IPOS=IPOS+1
58 GOTO 60
59 DO 70 L=1,132
60 IPBUF(IPOS)=IPBUF(IPOS).AND.LBUF(L)
61 IPOS=IPOS+1
62 60 CONTINUE

WHEN A PLOT BUFFER IS FULL COME HERE TO EMPTY IT

63 IF(NTIM.LE.NSTRIP)GOTO 10
64 STOP 'NORMAL TERMINATION'
65 END
SUBROUTINE LINFIL(J)

INTEGER*2 INBUF(8192), IOFF(8), LBUF(132), IBLK(8), NBUF(8),
%NLIN(E), ISPOS(8), EOF(8), ICHAN(8), ICOM(8)

COMMON /LFIL/ INBUF, IOFF, LBUF, IBLK, NBUF, NLIN, ISPOS, EOF, ICHAN, ICOM

ZERO LINE ARRAY

CALL APGET(LBUF, 0, 132, 1)
CALL APWD

FILL LINE BUFFER FROM INPUT BUFFER

NPT=NPLIN(J)
IPOS=ISPOS(J)
NLIM=NBUF(J)
IF (ICOM(J).NE.0) GOTO 20
DO 10 I=1,NPT
10 IPOS=IPOS+1
IF (IPOS.LE.NLIM) GOTO 10

REFILL BUFFER FROM FILE WHEN EMPTY

IN=IREADW(IO(24), INBUF(IOFF(J)), IBLK(J), ICHAN(J))
IBLK(J)=IBLK(J)+4
IPOS=IOFF(J)
IF (IN.EQ.0) GOTO 20
EOF(J)=1
NLIM=IN+IPOS
IF (IN.LT.-1) STOP 'READ ERR'
IF (IN.LE.0) RETURN
LBUF(I)=INBUF(IPOS)
CONTINUE
GOTO 90
20 DO 40 I=1,NPT
40 IPOS=IPOS+1
IF (IPOS.LE.NLIM) GOTO 20

REFILL BUFFER FROM FILE WHEN EMPTY

IN=IREADW(IO(24), INBUF(IOFF(J)), IBLK(J), ICHAN(J))
IBLK(J)=IBLK(J)+4
IPOS=IOFF(J)
IF (IN.EQ.0) GOTO 20
EOF(J)=1
NLIM=IN+IPOS
IF (IN.LT.-1) STOP 'READ ERR'
IF (IN.LE.0) RETURN
LBUF(I)=.NOT.INBUF(IPOS)
CONTINUE
NBUF(J)=NLIM
IPOS(J)=IPOS
RETURN
END
In this appendix are brief descriptions of the main subroutines, and their arguments, which were produced in the course of this work.

Also presented are some utility programs, which were found to be useful, in handling data on the system.
Tape Subroutine Descriptions

TREAD

CALL TREAD(BUFFER,NBUF,ISTAT,IDRV)

Purpose:- To read from tape into a specified memory buffer

Arguments:

BUFFER...Integer...Buffer to read into
NBUF.....Integer...Number of bytes to read
ISTAT....Byte......Returned tape status
IDRV......Byte......Tape drive to read from

TWRT

CALL TWRT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)

Purpose:- To write data to tape from a memory buffer

Arguments:

BUF......Integer...Buffer to write from
NBUF.....Integer...Number of bytes to write from the buffer
ISTAT....Byte......Returned tape status
IPAD.....Integer...Number of zero bytes to transfer at end of data
IFLEN....Integer...Total transfer length in 4kbyte blocks
IDRV......Byte......Tape drive number to write to
CALL SDS10(COM, ISTAT, ITLEN, ILEN)

Purpose: Utility tape control subroutine, allowing drive manipulation and reads and writes to tape from the disc.

Arguments:

COM......Byte......4 Byte command buffer sent to pdp8/e

COM(1)  Tape command
  0 - Read tape status
  1 - Rewind
  2 - Rewind offline
  3 - Read
  4 - Space foward
  5 - Space reverse
  6 - Write
  7 - Return to OS/8

COM(2)  File length, or number of records
        when spacing

COM(3)  Tape drive number

COM(4)  SDS10 command
  <0 Read from tape into file on
          unit 20
  =0 Tape wind operation
  >0 write to tape from file on
          unit 21

ISTAT....Byte.......Returned tape status
ITLEN......Byte.......Returned file length in seconds for field file
ILEN......Byte.......Returned file length in blocks on a read

TAPSUB

CALL TAPSUB(ICOM, IDRV, ISTAT, ILEN, IFNUM, BUF, NBYT, IEO)

Purpose :- General purpose tape handler for memory/tape transfers

Arguments:

ICOM......Byte.......Command flag
  <0 - read data
  =0 - wind on tape one record
  >0 - write data
IDRV......Byte.......Tape drive number
ISTAT....Byte.......Returned tape status
ILEN......Integer....Number of 4Kbyte blocks in transfer
IFNUM....Integer....File number of transfer(for log)
BUF......Integer....Data buffer
NBYT......Integer....Number of bytes to transfer, on write
IEOT.....Integer....End of tape flag
  0 - tape OK
  -1 - End of tape mark encountered in last operation
TAPRED

CALL TAPRED(ICOM,IDRV,ISTAT,ITLEN,ILEN,IFNUM,IEOT)

Purpose: General purpose tape handler for transfers to and from the disc.

Arguments:

ICOM......Byte........Command flag
  <0 read data from tape to disc file on unit 20
  =0 Wind tape forward one record
  >0 write data to tape from disc file on unit 21

IDRV......Byte........Tape drive number

ISTAT....Byte........Returned status

ITLEN.....Integer......Returned file length in seconds, on read.

ILEN.....Integer......File length of transfer, in blocks

IFNUM.....Integer......File number of operation

IEOT.....Integer......End of tape flag
  0 - no end of tape problem encountered
  -1 - end of tape encountered on read/write
.TITLE MEMTAP
.GLOBL TREAD,TWRIT
DRS$CSR=154000
DRB=154002
DOS$CSR=164010
DOB=164012
:READ FROM TAPE TO PDP11 MEMORY
CALL TREAD(BUFF,NBYTE,STATUS,IDRV)

TREAD:
MOV     2(R5),R1 ;GET MEMORY ADDRESS OF TARGET
MOVB    @4(R5),R0 ;GET NO OF BYTES REQUIRED
MOVB    RDLST,ARGLIST ;SET UP SDS10 COMMAND
MOVB    @8(R5),ARGLT2
JSR PC,MSG ;SEND COMMAND

;MAIN TRANSFER LOOP
1S:
TSTB   @#DR$CSR ;WAIT TILL 8 READY
BPL 1S
MOVB   @#DRB,(R1)+ ;MOVE DATA FROM BUFFER TO TARGET
DEC R0
BEQ 2S ;EXIT LOOP IF COUNT COMPLETE
TSTB   @#DR$CSR ;TEST FOR END OF DATA
BMI 3S ;FROM TAPE
BIS #1,@#DR$CSR ;KEEP ENABLE BIT SET
BR 1S ;LOOP BACK

;EXIT FOR COUNT COMPLETE
2S:
MOV R0,04(R5) ;RETURN ZERO COUNTER
TST @#DR$CSR ;TEST IF 8 IS FINISHED ALSO
BPL 4S ;IF NOT GO TO TAKE REST OF DATA
DEC R1 ;IF YES GET STATUS OUT OF DATA ARRAY
MOVB   (R1),@6(R5)
BR 6S ;GOTO EOF TRANSFER CODE

;EXIT FOR EOF SIGNALLED
3S:
MOV R0,04(R5) ;RETURN NO OF BYTES LEFT
DEC R1
MOVB   (R1),@6(R5) ;RETURN STATUS

;COMMON EXIT
6S:
TSTB   @#DR$CSR ;GET DATA ADDRESS
MOVB   @#DRB,R2 ;TAKE LAST CLEAN UP BYTE
RTS PC ;AND EXIT

;ROUTINE TO GO FROM MEMORY TO TAPE
CALL TWRIT(BUFF,NBYTE,STATUS,IBYTEPAD,NBLK,IDRV)

TWRIT:
MOV     2(R5),R1 ;GET DATA ADDRESS
MOVB    @4(R5),R0 ;GET NUM OF BYTES
MOVB    WRLST,ARGLIST ;SET UP COMMAND
MOVB    @16(R5),ARGLT1
MOVB    @12(R5),ARGLT2
JSR PC,MSG ;SET UP CODE

;EXIT FOR EOF SIGNALLED
8S:
TSTB   @#DOS$CSR ;WAIT FOR SYNCHRONISATION
BPL 8S
BIC #408, @DOSCSR ; CLEAR EOF BIT

; MAIN TRANSFER LOOP
11$: BIS $1, @DOSCSR ; SET ENABLE BIT
MOV (R1)+, @DOB ; TRANSFER BYTE
12$: TSTB @DOSCSR ; WAIT TILL
BPL 12$ ; ACCEPTED
DEC R8 ; DEC COUNTER
BNE 11$ ; GO BACK FOR MORE

; ZERO PASSING LOOP
MOV @B(R5), R2
BEQ EOF ; IF COUNT OF ZEROS=0 EXIT
14$: BIS #1, @DOB ; SET ENABLE BIT
MOV #0, @DOB ; TRANSFER BYTE
15$: TSTB @DOB ; WAIT TILL ACCEPTED
BPL 15$ ; DEC COUNTER
DEC R2
BNE 14$ ; DEC COUNTER

; STATUS REPLY LOOP
EOFT: BIS #408, @DOSCSR ; SET EOF BIT
MOV #8, @DOB
16$: TSTB @DRCSR ; SEE IF REPLY READY
BPL 16$ ; GET STATUS
MOV @DRB, @G(R5)
17$: TSTB @DRCSR ; CLEAR SYNCH ZEROS
BPL 17$ ; CLEAR EOF BIT
MOV @DRB, R2
DEC R8
BNE 17$ ; GO BACK FOR MORE
RTS PC

; MESSAGE SENDING SUBROUTINE
MSG: BIC #408, @DOSCSR ; CLEAR EOF BIT
MOV #ARGLST, R3 ; GET ARGLIST ADDRESS
MOV #3, R4
MLOOP: BIS $1, @DOSCSR ; SET ENABLE BIT
CLR R2 ; TRANSFER BYTE
MOV (R3)+, R2 ; MOV TO BUFFER
9$: TSTB @DOSCSR ; WAIT TILL ACCEPTED
BPL 9$ ; GET COMMAND INTO R2
MOV R2, @DOB
DEC R4
BNE MLOOP
BIS #408, @DOSCSR
MOV #8, @DOB ; CLEAR EOF BIT
RTS PC ; RETURN

; STORAGE AREA
ARGLST: .BYTE 0
ARGLT1: .BYTE 0
ARGLT2: .BYTE 0
WRTLST: .BYTE 6
.EVEN
RDUST: .WORD 403
.EVEN
.END
COMMON ENTRY POINT

MOV 2(R5),R1
MOV ARGLIST,R2
MOV #4,R0
IS: MOVB (R1)+,(R2)+
DEC R0
BNE IS
TSTB FLAG
BMI READ
BEQ WIND
JMP WRITE

ARGLIST: .BYTE 0
.FLAG: .BYTE 0

END OF COMMON ENTRY POINT

NEXT SECTION IS TAPE FAST READ

READ: CLR BLKN
.WRITW #AREA,#BUFF,#256,BLKN
JSR PC,MSG
.RESTRAT: MOV #BUFF,R4
MOV #2048,R3
DRLOOP: TSTB #DRSCSR
BPL DRDONE
MOV #ORB,(R4)+
DEC R3
BEQ DRDONE
TST #DRSCSR
BIS #1,#DRSCSR
BR DRLOOP

DRDONE: .WRITW #AREA,#BUFF,#1024,BLKN
BCS WERR
ADD #4,BLKN
.RESTRAT: GO BACK TO FILL ANOTHER BUFFER

DREOF: TSTB #DRSCSR
BPL DREOF
MOV #ORB,R1
DEC R4
MOVB -(R4),#0(R5)
MOVB -(R4),#0(R5)
ADD #2,R4
MOVB #2,(R4)
SUB #BUFF,R4
ASR R4
BCC 2S
INC R4
ZS: .WRITW #AREA,#BUFF,R4,BLKN
BCS WERR
WIND: JSR PC,MSG ;TELL B/E WHAT IS REQUIRED
REPLY: TSTB @#DRBCSR ;AND GO THROUGH PROC
         BPL REPLY ;FOR RECEIVING A REPLY
         MOVB @#DRB,#4(R5) ;WHICH IS RETURNED TO CALLING PROG
         MOV @#2,R6 ;TAKE LAST TWO SYNCHRO BYTES
5S: TSTB @#DRBCSR
     BPL 5S
     MOVB @#DRB,R1
     DEC R6
     RTS PC

;THIS SECTION IS RESPONSIBLE FOR THE FAST TAPE WRITE
;IN CONJUNCTION WITH THE B/E

WRITE: CLR BLKN ;CLEAR BLOCK COUNTER
        CLR EOFW ;CLEAR EOF FLAG
        JSR PC,MSG ;SEND MESSAGE
9S: TSTB @#DOBCSR ;TEST MESSAGE RECEIVED
     BPL 9S
     BIC #400,#DOBCSR ;CLEAR EOF
     BR DODONE ;START TRANSFER
WRSTRT: MOV #BUFF,R4 ;MOVE BUFFER ADDR->R4
         ASL R6
         JSR PC,MSG ;SEND MESSAGE
         MOVB (R4)+,#DOBCSR ;MOV FROM BUFF->DOB
9S: TSTB @#DOBCSR ;TEST IF READY FOR NEXT BYTE
     BPL 9S
     DEC R6
     BNE DOLOOP ;DEC THE COUNTER
     WRSTRT ;AND WRITE OUT AMOUNT OF BUFFER REQUIRED

DODONE: BMI DOEFLP ;IF SET GOTO EOF AREA
         .READW #AREA,#21..#BUFF,#1024..BLKN
         BCS RERR ;READ IN A FULL BUFFER AND CHECK FOR ERRORS
         ADD #4,BLKN ;BUMP UP BLOCK COUNT
         CMP #1024..R6 ;SEE IF GOT A FULL BUFFER
         BEQ WRSTRT ;YES THEN TRANSFER
         BIS #100000,EOFW ;NO THEN SET EOF FLAG
         MOV #1024..R1 ;SET UP COMPLETION COUNTER
         SUB #2,R1
         ASL R1
         BR WRSTRT ;AND WRITE OUT AMOUNT OF BUFFER REQUIRED

DOEFLP: BIS #1,#DOBCSR
        MOV #0,#DOBCSR ;MOVE ZEROS TO OUTPUT
8S: TSTB @#DOBCSR ;TEST IF OK FOR MORE
     BPL 8S
     DEC R1
     BEQ DOEEOF ;DEC THE COUNTER
     BR DOEFLP ;IF 0 GOTO END
DOEOF: BIS #488,@#DOSCSR ;SET INTERFACE EOF FLAG
MOV @8,@#DOB ;CLEAR INTERFACE BUFFER
BR REPLY ;GET REPLY

;THIS SECTION IS THE MESSAGE SENDING
;SUBROUTINE MSG AND THE BUFFER AND ERROR MESSAGE BLOCKS

MSG: BIC #488,@#DOSCSR ;CLEAR OUTPUT EOF FLAG
MOV #ARGLIST,R1 ;MOV ARGLIST ADDR TO R1
MOV #3,R0 ;MOV COUNTER ->R0
MLOOP: BIS #1,@#DOSCSR ;SET ENABLE BIT
CLR R2 ;CLEAR INTER BUFFER
MOVB (R1)+,R2 ;MOV FROM ARGLIST TO R2
MOV R2,@#DOB ;MOV R2->INTERFACE
7$: TSTB @#DOSCSR ;SEE IF OK FOR NEXT BYTE
BPL 7S
DEC R0 ;DEC THE COUNTER
BNE MLOOP ;AND GO BACK FOR MORE IF NON$ BIS #488,@#DOSCSR ;OR IF $ SET EOF FLAG AND
MOV @8,@#DOB ;FLUSH THE BUFFER
RTS PC ;RETURN
RERR: TSTB @#ERRBYTE ;TEST FOR TYPE OF ERROR
BEQ 8S
.PRINT @RMSG
.EXIT
8$: BIS #488,@#DOSCSR ;SET UP ERROR FINISH
MOV @8,@#DOB
JMP REPLY
RMSG: .ASCIZ /WRITE ERROR/

;STORAGE AREA

AREA: .BLKW 10
BLKN: .WORD 0
BUFF: .BLKW 1024
EOFW: .WORD 0
.END
SUBROUTINE TAPRED(icom, idrv, istat, itlen, ilen, ifnum, ieor)

TAPE HANDLING SUBROUTINE

icom is the command signal

$ is a read, $ is a wind, 1 is a write

idrv is the drive being used

istat is the status on return

itlen is the time length of a file read

ilen is the block length of a file read or written

INTEGER*2 mask(8), estat
LOGICAL*1 istat, com(4), sdcom(8), idrv, itlen, ecom(4),

%ilen, estat, errs(8)

DATA mask/'2', '4', '10', '20', '40', '100', '200'/

DATA sdcom/'1', '2', '3', '4', '5', '6', '7'/

DATA errs/'377', '377', '377', '377', '377', '377', '377'/

%ilen

i try=0

IF (icom) 10, 30, 20

SECTION CONTROLLING A READ

CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED

10 i try=ityr+1

SET UP COMMAND FOR READ

101.

com(1)=sdcom(4)
com(2)=1
com(3)=idrv
com(4)=-1

CALL SDS16(com, istat, itlen, ilen)

IF (istat.eq.0) RETURN

ERROR DETECTED ON READ

1017

istati=istat

GOTO 40

IF SHORT RECORD FOUND REREAD TAPE

40 ITMP=ISTATI.AND.MASK(6)

.21 IF (ITMP.NE.8) GOTO 10

421 ITMP=ISTATI.AND.MASK(2)

.2 IF (ITMP.EQ.0) RETURN

CRC ERROR FOUNDREWIND TAPE AND RETRY

25 WRITE(2,2010) IFNUM

2010 FORMAT(1 ' FILE NO ', I4, ' CRC ERROR REWINDING')

.271 IF (ITRY.GE.2) GOTO 130

30 ECOM(1)=SDCOM(6)

32 ECOM(2)=1
WRITE SECTION

WRITE ERROR DETECTED

REPORT AND RETRY

CLEAR IRRELEVANT BITS FROM ERROR BYTE
IF ISTAT=0 REWIND AND SET UP FOR NEXT READ

This was a data file not a short record

ECOM(1)=SDSCOM(6)
ECOM(2)=1
ECOM(3)=IDRV
ECOM(4)=0

CALL SDS10(ECOM,ESTAT, )
RETURN

IN THIS SECTION THE MAIN TAPE ERRORS ARE HANDLED SUCH AS: TAPE BUSY, TAPE OFFLINE

BOT, EOT

TAPE BUSY SECTION...AFTER CLEARING BOT FLAG

WRITE(2,1010)ISTAT,IFNUM
1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)

ISTAT=ISTATI.AND..NOT.MASK(4)
ITMP=ISTATI.AND.MASK(5)
IF(ITMP.EQ.0)GOTO 80
ECOM(1)=SDSCOM(1)
ECOM(2)=ISTAT
ECOM(3)=IDRV
ECOM(4)=0
CALL SDS10(ECOM,ESTAT, )

HAVING EXAMINED STATUS IF TAPE STILL BUSY, LOOP AGAIN, IF NOT TRY COMMAND AGAIN

ESTATI=ESTAT
ITMP=ESTATI.AND.MASK(5)
IF(ITMP.NE.0)GOTO 90
IF(ICON)10,30,20

TAPE OFFLINE

ITMP=ISTATI.AND.MASK(1)
IF(ITMP.EQ.0)GOTO 100
TYPE 1801, IDRV
1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')

HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED

IF(ICON)18,30,20
ECOM(1)=SDSCOM(1)
ECOM(2)=0
ECOM(3)=IDRV
ECOM(4)=0

CALL SDS10(ECOM,ESTAT, )

ESTATI=ESTAT
ITMP=ESTATI.AND.MASK(1)
IF(ITMP.NE.0)GOTO 110

RETURN
SUBROUTINE TAPSUB(IMCOM,IDRV,ISTAT,ILEN,IFNUM,BUF,NBYT,IEOT)

C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ, 0 IS A WRITE, 1 IS AWRITE
C IDRv IS THE DRIVE BEING USED
C ISTAT IS THE STATUS ON RETURN
C ILEN IS THE BLOCK LENGTH OF A FILE READ OR WRITTEN
C
INTEGER*2 MASK(8), ESTATI,BUF(1)
LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ECOM(4),

%IFLEN,ESTAT,ERRS(8)
DATA MASK/"1.,"2.,"4.,"18.,"48.,"188.,"288/"DATA SDSCOM/"8.,"1.,"2.,"3.,"4.,"5.,"6.,"7/"
DATA ERRS/"377.,"377.,"377.,"377.,"377.,"377.,"377/"
ITYR=8
IF (ICOM) 10,38,28

C SECTION CONTROLLING A READ
C
C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
C
10 ITRY=ITYR+1
C
C SET UP COMMAND FOR READ
C
NBUF=NBYT
CALL TREAD(BUF,NBUF,ISTAT,IDRV)
IF (ISTAT.EQ.8) RETURN
C
C ERROR DETECTED ON READ
C
ISTATI=ISTAT
GOTO 48
C
C IF SHORT RECORD FOUND REREAD TAPE
C
50 ITMP=ISTATI.AND.MASK(6)
IF (ITMP.NE.8) GOTO 10
ITMP=ISTATI.AND.MASK(2)
IF (ITMP.EQ.8) RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
WRITE(2,2810)IFNUM
2810 FORMAT(' FILE NO ',I4, ' CRC ERROR REWINDING')
IF (ITYR.GE.2) GOTO 130
ECOM(1)=SDSCOM(6)
ECOM(2)=IDRV
ECOM(3)=8
CALL SDS18(ECOM,ESTAT,

GOTO 18
C
C WRITE SECTION
C
20 ITRY=ITYR+1
IF (ITYR.GT.3) GOTO 130
NBUF=NBYT
IFLEN=(ILEN+3)/4
IF (IFLEN.LT.2) IFLEN=2
IPAD=(IFLEN*2048)-NBUF
CALL TWRIT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
IF(ISTAT.EQ.9)RETURN
C WRITE ERROR DETECTED
ISTAT=ISTAT
GOTO 40
70 ITMP=ISTATI.AND.MASK(6)
ITMPI=ISTATI.AND.MASK(2)
IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
C REPORT AND RETRY
WRITE(2,2020)IFNUM
2020 FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
ECOM(1)=SDSCOM(6)
ECOM(2)=2
ECOM(3)=IDRV
ECOM(4)=0
CALL SDSIO(ECOM,ESTAT, , )
NBUF=8
IPAD=(IFLEN*256)-NBUF
CALL TWRIT(Errs,NBUF,ESTAT,IPAD,IFLEN,IDRV)
GOTO 20
C WIND FORWARD ONE FILE
30 COM(1)=SDSCOM(5)
COM(2)=1
COM(3)=IDRV
COM(4)=0
CALL SDSIO(COM,ISTAT, , )
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
ISTAT=ISTAT.IAND..NOT.MASK(6)
IF(ISTAT.EQ.2)RETURN
ISTATI=ISTAT
IF(ISTAT.NE.8)GOTO 40
C IF ISTAT=8 REWIND AND SET UP FOR NEXT READ
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
ECOM(1)=SDSCOM(6)
ECOM(2)=1
ECOM(3)=IDRV
ECOM(4)=0
CALL SDSIO(ECOM,ESTAT, , )
35 RETURN
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS:= TAPE BUSY, TAPE OFFLINE
C BOT,EOT
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
C
40 WRITE(2,1010)ISTATI,IFNUM
1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
ISTATI=ISTATI.IAND..NOT.MASK(4)
ITMP=ISTATI.AND.MASK(5)
IF(ITMP.EQ.0)GOTO 80
90 ECOM(1)=SDSCOM(1)
ECOM(2)=0
Having examined status if tape still busy, loop again, if not try command again.

ESTAT1 = ESTAT
ITMP = ESTAT1 .AND. MASK(5)
IF (ITMP.NE.0) GOTO 90
IF (ICOM) 10, 30, 20

C TAPE OFFLINE

80 ITMP = ISTAT1 .AND. MASK(1)
IF (ITMP.EQ.0) GOTO 80
TYPE 1801, IDRV
801 FORMAT(' TAPE DRIVE ', I1, ' OFFLINE')

C Having announced error skip until corrected

110 ECOM(1) = SDS1OM(1)
ECOM(2) = 0
ECOM(3) = IDRV
ECOM(4) = 0
CALL SDS10(ECOM, ESTAT, )
ESTAT1 = ESTAT
ITMP = ESTAT1 .AND. MASK(1)
IF (ITMP.NE.0) GOTO 110
IF (ICOM) 10, 30, 20

C EOT

100 ITMP = ISTAT1 .AND. MASK(3)
IF (ITMP.EQ.0) GOTO 120
TYPE 1002, IDRV
1002 FORMAT(' EOT ON DRIVE ', I1)
IEOT = -1
RETURN
120 IF (ICOM) 50, 35, 70

C ERROR EXIT RETURN

130 ISTAT1 = -1
RETURN
END
Floating Point Transfers

GETNO

CALL GETNO(RNUM)

Purpose: - to find the number of floating point numbers to be transferred from the pdp8/e to the pdp11.

RNUM.....Floating point.....Number of values to follow

GETDAT

CALL GETDAT(NUM,BUFFER)

Purpose: - To get a set of floating point values from the pdp8/e

NUM......Integer.........Number of values to expect(IFIX(RNUM))

BUFFER...Floating Point....Buffer to put values into

SENDAT

CALL SENDAT(NUM,RNUM,BUFFER)

Purpose: - To send a set of floating point numbers to the pdp8/e

NUM......Integer.........Number of values to transfer

RNUM.....Floating point....Floating point equivalent of the
above

BUFFER...Floating Point....Buffer containing the values to transfer
.TITLE FPTR
.GLOBL GETNO, GETDAT, SENDAT

DRBCSR=164000
DRB=164002
DOB=164010

GETNO: MOV (R5)+, R0
        MOV (R5)+, R1
        MOV R1, R2
        MOV #4, R0
        1S: TSTB @DRBCSR
            BPL 1S
            MOVB @DRB, (R1)+
            BIS #1, @DRBCSR
            DEC R0
            BNE 1S
            SWAB (R2)+
            RTS PC

GETDAT: MOV (R5)+, R0
        MOV (R5)+, R0
        MOV (R5)+, R1
        MOV R1, R2
        ASL @R0
        2S: TSTB @DRBCSR
            BPL 2S
            MOVB @DRB, (R1)+
            BIS #1, @DRBCSR
        3S: TSTB @DRBCSR
            BPL 3S
            MOVB @DRB, (R1)+
            BIS #1, @DRBCSR
            SWAB (R2)+
            DEC OR0
            BNE 2S
            RTS PC

SENDAT: MOV (R5)+, R0
        MOV (R5)+, R0
        MOV (R5)+, R1
        MOV (R5)+, R2
        MOV #2, R3
        CLR R4
        4S: SWAB (R1)
            BIS #1, @DOBCSR
            MOVB (R1)+, R4
            MOV R4, @DOB
        5S: TSTB @DOSCSR
            BPL 5S
            BIS #1, @DOSCSR
            MOVB (R1)+, R4
            MOV R4, @DOB
        6S: TSTB @DOSCSR
            BPL 6S
            DEC R3
            BNE 4S
        7S: SWAB (R2)
            BIS #1, @DOSCSR
            MOVB (R2)+, R4
            MOV R4, @DOB
        8S: TSTB @DOSCSR
BPL B5
BIS @1,@DOSCSR
MOVB (R2)+,R4
MOV R4,@DOB
TSTB @4@DOSCSR
BPL 9$  
DEC @R0
BNE 7$  
RTS  PC
.END  PC
Extended Memory Input/Output

AP to Memory

FAD = APGAD(VM(I))

Purpose:- To get a full 18 bit address for a virtual memory element

FAD......Floating point....Returned 18 bit address
VM(I)....Any............Virtual memory element

CALL APPUTX(APAD,WCNT,FORMAT)

CALL APGETX(APAD,WCNT,FORMAT)

Purpose:- To transfer data to(PUT) and from(GET) the AP using the 18 bit address stored internally by an immediately preceeding call to APAD.

CALL APPUTA(APAD,WCNT,FORMAT,FAD)

CALL APGETA(APAD,WCNT,FORMAT,FAD)

Purpose:- To exchange data with the AP as above except the data is provided by the value FAD which has been stored previously.

APAD......Integer......AP memory address
WCNT......Integer......Number of elements to transfer
FORMAT....Integer......AP data transfer format
Disc to Memory

FAD=ADGET(VM(I))

Purpose:- To get a full 18 bit address for the virtual memory element in the format for a disc transfer.

IWRITX(CHAN,WCNT,BLK)

IREADX(CHAN,WCNT,BLK)

Purpose :- To transfer between disc and virtual memory using the 18 bit address calculated in an immediately preceeding call to ADGET.

IWRITA(CHAN,WCNT,BLK,FAD)

IREADA(CHAN,WCNT,BLK,FAD)

Purpose:- to transfer data between disc and virtual memory as above but with the 18 bit address being provided by FAD.

FAD......Floating point......18 bit address of virtual memory element

VM(I).......Any..............Virtual memory element
CHAN.......Integer..............I/O channel to be used in transfer
WCNT.......Integer..............Number of words to transfer
BLK.......Integer..............Starting block in file for transfer
C

FOR PDP-11 RT-11 OR DOS

---------------------------------------------------------·----------------

C

PDP-11 DEFINITIONS

RB % 8
R1 % A
R2 % B
R3 % C
R4 % D
R5 % E
R6 % F
SP % 0
PC % 7

AP-DEVICE ADDRESSES

FMTH = FPS
FMTL = FPS + 2
WC = FPS + 100
HMA = FPS + 102
CTRL = FPS + 104
APMA = FPS + 106
SWR = FPS + 110
FN = FPS + 112
LITES = FPS + 114
ABRT = FPS + 116

.MACRO CALL X
MOV R5,-(SP) ;SAVE R5
.IF EQ,<DOS-1>
.IFT
JSR R5,X
BR +2
.IFF
MOV #ZERO,R5
JSR PC,X
.IFFT

MOV (SP)+,R5 ;RETRIEVE R5
.ENDC
.ENDM

.MACRO RETURN
.IF EQ,<DOS-1>
.IFT
RTS R5
.IFF
RTS PC
.ENDC
.ENDM

;RETRIEVE
SPLDGO = S-PAD LOAD AND GO = REL 2.Ø , NOV 77 ...............................  

SUBROUTINE SPLDGO(SLIST,NSPADS,STRT,BRKLOC) INTEGER SLIST(16),NSPADS,STRT,BRKLOC

FIRST WAIT FOR THE LAST PROGRAM STARTED BV 'RUNAP' TO BE COMPLETED, THEN:
LOAD 'N' VALUES INTO S-PAD FROM 'SLIST' AND START THE AP AT LOCATION 'STRT' WITH A BREAKPOINT
SET AT 'BRKLOC'

ROUTINES USED: APWR, APOUT

C-------LOCAL STORAGE
INTEGER I
1. WAIT FOR RUNNING DONE (APWR)
2. FOR EACH S-PAD PARAMETER:
   A. PUT S-PAD PARAMETER ADDRESS INTO SPD (CALL WREG(I-1,513))
   B. PUT PARAMETER VALUE INTO S-PAD (CALL WREG(SLIST(I),517))
3. PUT PROGRAM STARTING LOCATION INTO TMA (CALL WREG(STRT,516))
4. CALL RUNAP TO START AT LOCATION 8 OF THE BOOTSTRAP, WITH
   THE SPECIFIED P.S. BREAKPOINT SET

CALL APWR
SPLDGO: CALL APWR

LOAD PARAMETERS INTO S-PAD (IF ANY)
IF (NSPADS.EQ.Ø) GOTO 2Ø
DO 1Ø I = 1, NSPADS
CALL APOUT(I-1,1)
CALL APOUT(S13,2)
CALL APOUT(SLIST(I),1)
1Ø CALL APOUT(S17,2)

LDSP:
GET S-PAD VALUE POINTER
GET NSPADS
SET S-PAD ADDRESS
SET NSPAD ADDRESS
INTO SPD
SET PARAMETER VALUE
INTO S-PAD
BUMP PARAMETER VALUE POINTER
AND S-PAD ADDRESS
DEC R6 ;SEE IF DONE??
BNE LDSP

; PUT THE STARTING ADDRESS INTO TMA, START BOOTSTRAP AT 4,
; WITH BREAK ON PSA ENABLED AND BREAK IN SWR
; CALL APOUT(STRT,1)
; CALL APOUT(515,2)
; CALL RUNAP(4,8,SRKLOC,8448)
; RETURN

SBRG0: MOV @(R5)+,@#SWR ;SET STARTING ADDRESS INTO TMA
    MOV #515.,@#FN
    MOV #8.,@#SWR ;PUT STARTING ADDRESS OF BOOT-STRAP STARTER
    MOV #512.,@#FN ;INTO PSA
    CLR @#S12 ;ZERO APSTAT, CLEAR PARITY ENABLE
    MOV #518.,@#FN ;DEP TO APSTAT
    MOV @(R5)+,@#SWR ;SET PSA BREAKPOINT
    MOV #8448.,@#FN ;AND GO
    RETURN

ZERO:  S
    END

***** ABORT = ABORT AP-EXECUTION = REL 2.9, NOV 77 ***********************
***** APRSET = RESET THE AP = REL 2.9, NOV 77 ***********************

SUBROUTINE ABORT

STOPS ANY TRANSFER, AND/OR RUN IN PROGRESS, RESETS INTERFACE AND
Cleans up any software state indicators

Routines used: APOUT

1. DO AN INTERFACE RESET (ORESET)
2. CLEAR THE CONTROL REGISTER (OCTRL(9))
3. DO AN INTERNAL RESET (OFN(2048))

CALL APOUT(9,19)
CALL APOUT(9,7)
CALL APOUT(2048,2)
RETURN

APRSET:
CLR @#ABRT
CLR @#CTRL
MOV #B003,#@#FN
CLR @#LITES
RETURN

END

***** RUNDMA = START A DMA TRANSFER = REL 2.9, NOV 77 ***********************
SUBROUTINE RUNDMA(HOST,APMA,N,CTRL)
INTEGER HOST,APMA,N,CTRL

WAIT FOR ANY PREVIOUS DMA TRANSFER STARTED BY 'APPUT', 'APGET', OR 'RUNDMA' TO COMPLETE, THEN:

START A DMA TRANSFER WITH THE ADDRESS OF 'HOST' AS THE INITIAL HOST MEMORY ADDRESS, 'APMA' AS THE INITIAL AP-120B MAIN DATA MEMORY ADDRESS, 'N' AS THE NUMBER OF DATA ITEMS TO BE TRANSFERRED, AND 'CTRL' AS THE CONTROL REGISTER SETTING (WITH INTERRUPT CONTROL BITS MASKED OUT) TO USE.

ROUTINES USED: APWD, APOUT, ILOC, IAND16, IRSH16

-------NOTE: THE DETERMINATION OF 'WC' FROM 'N' BELOW DEPENDS ON THE HOST WORD LENGTH AS IF AFFECTS THE NUMBER OF HOST WORDS PER AP-120B MEMORY WORD. THIS CODE IS APPROPRIATE FOR A 16-BIT COMPUTER.

-------LOCAL STORAGE
INTEGER WC

1. WAIT FOR DMA DONE
2. SET HOST ADDRESS <OHMA>
3. SET AP ADDRESS <OAPMA>
4. SET WORD COUNT <OWC>
5. SET CONTROL REGISTER <OCTRL>

CALL APWD

RUNDMA: CALL APWD

CALL APOUT(APMA,4)
CALL APOUT(ILOC(HOST),5)
WC = N

TST (R5)+
MOV (R5)+,@#HMA ;SET PDP-11 ADDRESS
MOV #0,@#LITES ;CLEAR EXTENDED PDP11 ADDRESS
MOV @R5)+,#APMA ;SET AP MEMORY ADDRESS
MOV @R5)+,R1 ;GET DATA COUNT

ISOLATE FMT FIELD AND ADJUST WC ACCORDINGLY
IF(IAND16(IRSH16(CTRL,1),3).NE.1) WC=2*N
CALL APOUT(WC,6)

MOV @R5)+,R8 ;GET CONTROL WORD
BIT R8,#4 ;TEST 'FMT' FIELD FOR A 2
BNE 1S
BIT R8,#2
BNE 2S
ASL R1 ;DOUBLE COUNT UNLESS FORMAT #1

1S: MOV R1,@#WC ;SET WORD COUNT
2S:
CLEAR OFF HOST INTERRUPT ENABLE BITS
CALL APOUT(IAND16(CTRL,1623), 7)
RETURN

BIC #174000,R0 ; CLEAR INTERRUPT ENABLES
MOV R0,R#CTRL
RETURN

END

***** RUNAP = START AN AP-PROGRAM = REL 2.0, NOV 77 ****************************

SUBROUTINE RUNAP(PSA,NOLOAD,SWR,FN)
INTEGER PSA,NOLOAD,SWR,FN

WAIT FOR ANY PREVIOUS PROGRAM STARTED BY 'SPLDGO' OR 'RUNAP'
TO COMPLETE, THEN:
1. IF 'NOLOAD' IS ZERO, PUT 'PSA' INTO PSA
2. PUT 'SWR' INTO THE SWITCH REGISTER
3. PUT 'FN' (WITH 'START BIT' CLEARED AND 'CONTINUE BIT' SET)
   INTO THE FUNCTION REGISTER

ROUTINES USED: APWR, APOUT, IOR16, NAND16

1. WAIT FOR RUNNING DONE
2. IF NO-LOAD NOT SPECIFIED, PUT 'PSA' INTO PSA
   (CALL WREG(PSA,512))
3. PUT 'SWR' INTO SWR (OSWR)
3. CLEAR POSSIBLE START BIT, OR IN CONTINUE BIT,
   AND PUT INTO FUNCTION (OFN)

CALL APWR

RUNAP:
   CALL APWR
   CLEAR PARITY ENABLE IN STATUS REGISTER
   CLR @#SWR
   MOV #518.,@#FN

   IF (NOLOAD.NE.0) GO TO 100
   CALL APOUT(PSA,1)
   CALL APOUT(512,2)

   TST (R5)+
   MOV @(R5)+,@#SWR ; PUT 'PSA' INTO THE SWITCHES
   TST @(R5)+ ; SEE IF LOAD PSA??
   BNE NOLOAD
   MOV #512.,@#FN ; PUT 'PSA' INTO PSA IF 'NOLOAD' IS ZERO

   100 CALL APOUT(SWR,1)
   CLEAR POSSIBLE SET START BIT & OR IN CONTINUE BIT TO FN REG
   CALL APOUT(IOR16(AND16(FN,271),8192), 2)
NOLCAD: CLR @#SWR ;CLEAR SWR
MOV @#0382+,@#FN ;CLEAR PARITY ERROR ENABLE
MOV @(R5)+,@#SWR ;PUT 'SWR' INTO THE SWITCHES
MOV @#RS+,R5 ;CLEAR ALL BUT POSSIBLE BREAKPOINT
BIC #17736F,R0 ;SET CONTINUE BIT
BIS #8192.,R0 ;AND GO
RETURN

RETURN

;/**** TSTDMA = TEST DMA TRANSFER COMPLETE = REL 2.0 , NOV 77 ***************/
;
SUBROUTINE TSTDMA(I)
INTEGER I
;
SET 'I' TO ONE IF THE LAST DMA TRANSFER STARTED BY 'APPUT'
'APGET' OR 'RUNDMA' IS DONE; SET 'I' TO ZERO IF THE TRANSFER IS
STILL IN PROGRESS
;
ROUTINES USED: APIN, NAND16
;
READ CTRL AND MASK TO .NOT. LOW BIT
CALL APIN(I,7)
I=NAND16(I,1)
RETURN
TSTDMA:
MOV #1,R1
BIC @#CTRL,R1 ;DO NOT.CTRL.AND.1
MOV R1,02(R5)
RETURN

END

;/**** TSTRUN = TEST RUN COMPLETE = REL 2.0 , NOV 77 ***************/
;
SUBROUTINE TSTRUN(I)
INTEGER I
;
SET 'I' TO ONE IF THE AP-120B IS STOPPED AFTER THE LAST RUN
STARTED BY 'SPLDGO' OR 'RUNAP'; ELSE SET 'I' TO ZERO IF THE AP-120B
IS STILL RUNNING.
;
ROUTINES USED: APIN, NEGCHK
;
READ FN AND SHIFT DOWN TO LOW BIT
CALL APIN(I,2)
I=NEGCHK(I)
RETURN
;
TSTRUN:
CLR R1
TST @#FN
BGE IS
INC R1 ;RETURN A 1 IF THE HIGH BIT OF 'FN' WAS ON
IS:
MOV R1,02(R5)
RETURN
SUBROUTINE WTDMA(IERR)
  INTEGER IERR

  WAITS FOR DATA TRANSFER COMPLETE, 'IERR' SET TO ONE IF A DATA LATE
  ERROR WAS DETECTED BY THE HARDWARE, ELSE SET TO ZERO.

  ROUTINES USED: APIN, IAND16, IRSH16

  SPIN WHILE THE LOW BIT OF 'CTRL' IS ON
  CALL APIN(IERR,7)
  IF (IAND16(IERR,1).EQ.1) GO TO 100

  SHIFT THE 'DATA LATE' ERROR BIT DOWN TO THE LOW END
  IERR=IRSH16(IAND16(IERR,256),8)
  RETURN

END

SUBROUTINE WTRUN(IERR)
  INTEGER IERR

  WAIT FOR AP RUN TO FINISH (HALT), SET IERR TO ONE IF AN SRAO ERROR,
  TWO IF PARITY, ELSE ZERO.

  TST @#FN
  BGE WTRUN
  CLR R1
  MOV @#R1, @#FN
  BIT @#LITES,#248
  BEQ 1S
  MOV #2,R1
  DEC R1
  BNE 1S
  DEC R1
  MOV R1, @2(R5)
  RETURN

END

SUBROUTINE APASGN

END
SUBROUTINE APASGN(APNUM, ACTION, STATUS)
INTEGER APNUM, ACTION, STATUS

APASGN IS A 'NOP' ON RT-11
RETURN
A 1 IN STATUS TO INDICATE THE AP IS ASSIGNED;
APASGN: MOV #1, @6(R5) ; SET THIRD PAPARAMETER TO 1
RETURN
APRLSE = RELEASE THE AP = REL 2.Ø , NOV 77 ***********************
APRLSE IS A 'NOP' UNDER RT-11
APRLSE: RETURN

APIENA = INABLE INERRUPT = REL 2.Ø , NOV 77 ***********************
APIENA: RETURN
APIDIS = DISABLE INTERRUPT = REL 2.Ø , NOV 77 ***********************
APIDIS: RETURN
TSTINT = TEST FOR INTERRUPT = REL 2.Ø , NOV 77 ***********************
TSTINT: RETURN
APWI = WAIT FOR INTERRUPT = REL 2.Ø , NOV 77 ***********************
APWI: RETURN

APIN = INPUT AN AP-12ØB INTERFACE REGISTER = REL 2.Ø , NOV 77 ********
SUBROUTINE APIN(DATA, NUM)
INTEGER DATA, NUM
READ THE CONTENTS OF INTERFACE REGISTER NUMBER 'NUM' INTO 'DATA'
PARAMETERS:
DATA - RECEIVES THE CURRENT CONTENTS OF REGISTER 'NUM'
NUM - SPECIFIES WHICH AP-12ØB INTERFACE REGISTER IS TO BE READ:
1. SWR SWITCH REGISTER
2. FN FUNCTION REGISTER
3. LITES LITES REGISTER
4. APMA AP DMA MEMORY ADDRESS REGISTER
5. HMA HOST DMA MEMORY ADDRESS REGISTER
6. WC DMA WORD COUNT REGISTER
7. CTRL DMA CONTROL REGISTER
8. FMTH FORMAT HIGH REGISTER
9. FMTL FORMAT LOW REGISTER
10. REMR DO AN EXTERNAL RESET (NO-OP FOR APIN)
11. IFSTAT INTERFACE STATUS REGISTER (APIN READS, APOUT NO-OP)
12. MASK MEMORY PROTECTION AND I/O BITS
ROUTINES USED: NONE

NOTE: THIS ROUTINE WILL TYPICALLY BE IN ASSEMBLY LANGUAGE.

SINCE FORTRAN CANNOT OUTPUT DIRECTLY TO AN I/O DEVICE

APIN:

13. APMAE AP PAGE SELECT
14. MAE DMA PAGE SELECT

RETURN

APIN:

MOV @4(R5),R1 ;GET REGISTER NUMBER
MOV R1,R2 ;CHECK FOR EXTENDED MEMORY REGISTER READ
SUB #11,R2 ;LOOK FOR 12,13 OR 14
BLE APIN1 ;IF LESS THAN 12
SUB #4,R2
BLT IMASK ;IF EXTENDED MEMORY REGISTER READ

APIN1:

ASL R1 ;CONVERT TO BYTES
MOV @TABLE(R1),@2(R5) ;GET FROM APPROPRIATE DEVICE ADDRESS
RETURN

READ MASK OR APMAE OR MAE


UPON ENTRY OR EXIT THE VALUE OF MASK OR APMAE OR MAE ARE RIGHT JUSTIFIED, ZERO FILLED.

IMASK:

MOV @#ABRT,R2 ;READ MASK,APMAE,MAE
CMP #12,R1 ;CHECK FOR MASK
BNE IAPMAE ;IF NOT MASK
SWAB R2 ;RIGHT JUSTIFY MASK
BIC #177700,R2 ;CLEAR ALL BUT MASK
MOV R2,@2(R5) ;RETURN VALUE

IAPMAE:

CMP #13,R1 ;CHECK FOR APMAE
BNE IMAE ;IF MAE
ASR R2 ;RIGHT JUSTIFY APMAE
ASR R2
ASR R2

IMAE:

BIC #177760,R2 ;CLEAR ALL BUT MAE OR APMAE
MOV R2,@2(R5) ;RETURN REGISTER VALUE

END

TABLE:

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPS+110</td>
<td>SWR</td>
</tr>
<tr>
<td>FPS+112</td>
<td>FN</td>
</tr>
<tr>
<td>FPS+114</td>
<td>LITES</td>
</tr>
<tr>
<td>FPS+106</td>
<td>APMA</td>
</tr>
</tbody>
</table>
SUBROUTINE APOUT (DATA, NUM)

INTEGER DATA, NUM

PUT THE CONTENTS OF 'DATA' INTO INTERFACE REGISTER NUMBER 'NUM'.

PARAMETERS:
DATA - DATA TO BE PUT INTO AN INTERFACE REGISTER
NUM - NUMBER OF THE DESTINATION INTERFACE REGISTER, SEE 'APIN' FOR THE NUMBERING

ROUTES USED: NONE

--- NOTE: THIS ROUTINE WILL TYPICALLY BE IN ASSEMBLY CODE.

RETURN

APOUT:

MOV #4(R5), R1
MOV R1, R2
SUB #11, R2
BLE APOUT1
SUB #4, R2
BLT OMASK

APOUT1:

MOV #4(R5), @TABLE(R1)

WRITE MASK OR APMAE OR MAE

SEE COMMENTS IN APIN FOR EXTENDED MEMORY REGISTERS (MASK, APMAE, MAE)

OMASK:

MOV @#ABRT, R3
CMP #12, R1
BNE APOAMAE
BIC #37400, R3
SWAB R2
BIS R2, R3
MOV R3, @#LITES

READ MASK, APMAE, MAE
FETCH REGISTER VALUE
CHECK FOR MASK
IF NOT MASK
CLEAR MASK, KEEP APMAE AND MAE
POSITION MASK TO BITS 8-13
ADD NEW MASK TO OLD APMAE AND OLD MAE
WRITE TO AP

OAPMAE:

CMP #13, R1
BNE OMAE
BIC #36000, R3
ASL R2
ASL R2
ASL R2
BIS R2, R3
MOV R3, @#LITES

CHECK FOR APMAE
IF MAE
CLEAR APMAE, KEEP MASK AND MAE
POSITION APMAE TO BITS 4-7
ADD NEW APMAE TO OLD MASK AND OLD MAE
WRITE TO AP

RETURN
OMAE:     BIC #17, R3 ; CLEAR MAE KEEP APMAE AND MASK
          BIS R2, R3 ; ADD NEW MAE TO OLD APMAE AND OLD MASK
          MOV R3, @#LITES ; WRITE TO AP
          RETURN

.TITLE RK05 V03-01
.IDENT /V03.01/

RT-11 DISK (RK11) HANDLER
DEC-11-ORTSB-A
EF/ABC/RGB/DV/JD
COPYRIGHT (C) 1977
DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
SOFTWARE SHALL AT ALL TIMES REMAIN IN DEC.

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO
CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

DEC ASSUMES NO RESPONSIBILITY FOR THE USE
OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
WHICH IS NOT SUPPLIED BY DEC.
.MCALL .V2..REGDEF,.DRBEG,.DREND,.FORK
.MCALL .DRAST,.DFIN,.DELFF
..V2..
..REGDEF
.IIF NDF TIMSIT,TIMSIT=0
.IIF NDF MMGST, MMGST=0
.IIF NDF ERLSG, ERLSG=0
.GLOBL DPSYS, DSSYS, DXSYS, DMSYS
.GLOBL RKSYS, RFSYS, DTSYS
DTSYS = @
DSSYS = @
DXSYS = @
DPSYS = @
RFSYS = @
DMSYS = @
RKDS = 177480
RKER = 177482
RKCS = 177484
RKMC = 177486
RKBA = 177410
RKDA = 177412
RKCNT = 10
RKSTS = 160000
RKSIZ = 11388
RKIDEN = @
RKIDS = 377
RKRCNT = 4000
RKREG = 7
RKREGA = 177488
.DRBEG RK,22@,RKSIZ,RKSTS
.IF EQ MMGST
.IFF
RETRY:
MOV #RKCNT,(PC)+
MOV @R5,R2
MOV 2(R5),R4
ASR R4
ASR R4
ASR R4
SWAB R4
BIC #<160000>,R4
BR 2S
1S:
ADD R2,R4
ASR R2
ASR R2
ADD R3,R2
2S:
MOV R2,R3
BIC #177760,R3
BIC R3,R2
BNE 1S
CMP #12..R3
BGT 3S
ADD #4,R3
3S:
ADD R3,R4
MOV R4,DISKAD
AGAIN:
MOV RKCOE,R5
MOV #183,R3
MOV #RKDA,R4
MOV (PC)+,OR4
DISKAD: 0
CMP (R5)+(R5)+
.IFT
MOV (R5)+,-(R4)
.IFF
JSR PC, @SMPPTR
MOV (SP)+,-(R4)
.IFFT
MOV (R5)+,-(R4)
BEQ 7S
BMI 5S
NEG @R4
ADD #2,R3
5S:
MOV @#RKCS,-(SP)
BIC #177717,(SP)
BIS (SP)+,R3
.IFF
BIS (SP)+,R3
.IFFT
MOV R3,-(R4)
RTS PC
7S:
MOV #111,R3
BR 5S
.DRAST RK,R5
MOV #RKER,R5
MOV (R5)+,R4
TST RETRY
BPL NORMAL
TST @R5
BMI NORMAL
BIT #200000,R5
BEQ RTSPC
.RKRETR: CLR B RETRY+1
BR AGAIN
.NORMAL: CMP @R5,#310
BEQ RTSPC
TST @R5
BPL DONE
.FORK RKFBLK
.RKERR: BNE RKERR
MOV PC,R5
ADD #RKRBUF-.,R5
MOV R5,R2
MOV #RKREGA,R3
MOV #RKNREG,R4
RKRREG:
MOV (R3)+,(R5)+
DEC R4
BNE RKREG
MOV #RKNREG,R3
ADD #RKRNTC,R3
MOV RKQE,R5
MOVB RETRY,R4
DEC R4
JSR PC, @SELPPTR
MOV #RKER,R5
MOV (R5)+,R4
. ENDC
.RKERR:
MOV #1,0R5
3S:
TSTB 0R5
BPL 3S
DECB RETRY
BEQ HERROR
BIT #111111, R4
BEQ RKRETR
MOV DISKAD, @RKDA
MOV #115, R5
BIS #100000, RETRY
RTSPC: RTS PC
HERROR: MOV RCQCE, R5
BIS #1.0-(R5)
IFDEF ERRLSG
BR RKEXIT
DONE: .FORK RKFBLK
MOV #RKIDS, R4
MOV RCQCE, R5
JSR PC, @SELPTR
ENDIF
DONE:
ENDIF
DONE: .FORK RKEXIT:
CLR RETRY
.DRFIN RK
.ENDC
RKEXIT: .WORD B.B.B.B
IFDEF ERRLSG
RKRBUF: .BLKW RKNREG
.ENDC
.DEND RK
.END
.TITLE MPAPLEX
.ENDBL
.GLOBL APGAD,APPUTX,APGETX,APGETA,APPUTA

USP0=177648
AP=176008
WC=AP+188
HMA=AP+182
CTRL=AP+184
APMA=AP+186
LITES=AP+188

APGAD:     MOV @#USP0,R0       ;GET PAR # BLOCK OFFSET
            ASL R0
            ASL R0
            ASL R0
            MOV R0,HIGHBT  ;GET HIGBIT INTO PLACE
            BIC #37777,HIGHBT
            ASL R0
            ASL R0
            BIC #17777,R1
            BIC #77,R0
            BIS R1,R0
            MOV R0,LOWBIT
            MOV HIGHBT,R0  ;SAVE INTO HIGBIT
            MOV LOWBIT,R1
            RTS

APPUTA:    MOV 8.(R5),R0       ;GET ADDRESS OF STORE
            MOV (R0)+,HIGHBT
            MOV (R0),LOWBIT

APPUTX:    MOV R5,-(SP)       ;GET HIGBIT FROM STORE
            JSR PC,APWD
            MOV (SP)+,R5
            BR COMMON

APGETA:    MOV 8.(R5),R0       ;SET STORE ADDRESS
            MOV (R0)+,HIGHBT
            MOV (R0),LOWBIT

APGETX:    MOV R5,-(SP)       ;AND LOWBITS
            JSR PC,APWD
            MOV (SP)+,R5

COMMON:    MOV @2(R5),@#APMA
            MOV @4(R5),R1
            ADD @6(R5),R0
            ADD @6(R5),R0
            BIT R0,0
            BNE 1S
            BIT R0,#2
            BNE 2S
            ASL R1
            STD R0
            BIC #174000,R0
            MOV RVBT,#HMA
            MOV #174000,R0

1S:        MOV R1,#+WC
            MOV HIGBT,#LITES
            MOV HIGHBT, @#APMA
            MOV LOWBIT,#HMA
            BIC #174000,R0
            MOV R0, @#CTRL
            MOV RVBT,#HMA
            MOV #174000,R0
            RTM PC

LOWBIT:    .WORD 0
HIGHBT:    .WORD 0
.END
.MCALL .READC,.WRITC,.EXIT,.PRINT

.GLOBL ADGET,IREADX,IWRITX,IREADA,IWRITA

USP$=17764$  
RKCS=1774.0'

ADGET: MOV @#USP$,R0   ;GET EXTENSION IN R0
        ASL R0     ;GET HIGHBITS INTO PLACE
        ASL R0     ;SAVE IN R2
        MOV R0,R2  ;GET LOWBITS INTO PLACE
        ASL R0     ;SAVE IN R2
        ASL R0     ;GET LOWBITS INTO PLACE
        ASL R0     ;CLEAR UNWANTED BITS
        BIC #1777000$,R1   ;IN R1 + R0
        BIC #7700,R0
        MOV R1,R0

        MOV R0,LOWBIT
        MOV R1,LOWBIT
        BIC #177717,R2   ;GET HIGHBITS INTO
        MOV R2,HIGHBT    ;CORRECT PLACE
        MOV R0,LOWBIT    ;SAVE IN HIGHTBS
        MOV R2,HIGHBT    ;PUT HIGHTBS IN R0
        MOV R2,LOWBIT    ;LOWBITS IN R1 TO RETURN FUNCTION
        RTS PC

IREADA: MOV @(R5),R0   ;GET ADDRESS OF STORE
        MOV (R0)+,HIGHBT  
        MOV (R0),LOWBIT   ;AND GET ADDRESS BITS
        IREADX: MOV @2(R5),R1   ;GET ARGUMENTS INTO
        MOV @4(R5),R3    ;INTO REGISTERS
        MOV @6(R5),R4
        MOV LOWBIT,R2  
        BIS HIGHBT,@#RKCS    ;SET EXTENDED BITS
        .READC #AREA,R1,R2,R3,#XMMCMP,R4 ;INITIATE READ
        RTS PC

IWRITX: MOV @2(R5),R1  
        MOV @4(R5),R3
        MOV @6(R5),R4
        MOV LOWBIT,R2  
        BIS HIGHBT,@#RKCS    ;SET EXTENDED BITS
        .WRITC #AREA,R1,R2,R3,#XMMCMP,R4

        RTS PC

XMMCMP: BIC #6000,RKCS
        RTS PC

ERROR: NEG R0   ;GET ERROR INTO STANDARD FORTRAN TYPE
        SUB #1,R0
        RTS PC

.XMEN

AREA: .BLKW 10
LOWBIT: .WORD 0
HIGHBT: .WORD 0

.END
Microcode Routines

Demultiplex Microcode

CALL VBINSC(A1,I1,A2,I2,G1,IG1,N)

Purpose:- To binary scale an input vector using a vector of binary gain values

A1.......AP address of input vector
I1.......AP address increment for each element
A2.......AP address of output vector
A2.......AP address increment for each output element
G1.......AP address of gain vector
IG1......AP address increment for gain values
N.......Number of vector elements to apply the gains to

CALL DMXA(A1,A2,I2,A3,A4,N)

Purpose:- To demultiplex and reformat a frame of SEG-A field data

A1.......AP address of running gains vector
A2.......AP address of Field data
I2.......AP address increment per data element
A3.......AP address of submultiplexed gain check
A4.......AP address of Gain switch direction to use next
N.......Number of points to demultiplex form the frame

Tri-Diagonal Matrix equation solver

CALL FACTOR(A1,I1,A2,I2,A3,I3,A4,I4,A5,I5,N)
Purpose: To factorise a tri-diagonal matrix

A1........AP address of the 1st diagonal, vector A
I1........AP increment of the above
A2........AP address of the major diagonal, vector B
I2........AP increment of the above
A3........AP address of the 3rd diagonal, vector C
I3........AP increment of the above
A4........AP address of the output L vector
I4........AP increment of the above
A5........AP address of the output U vector
I5........AP increment of the above
N........Number of elements in the major diagonal

CALL SOLVE(A1,I1,A2,I2,A3,I3,A4,I4,A5,I5,A6,I6,N)

Purpose: to solve the Tri-diagonal matrix equation given the factorised input

A1........AP address of the Factorised L vector
I1........AP increment of the above
A2........AP address of the Factorised U vector
I2........AP increment of the above
A3........AP address of the RHS vector
I3........AP increment of the above
A4........AP address of the C vector
I4........AP increment of the above
A5........AP address of workspace vector
I5........AP increment of the above
A6.......AP address of X result vector
I6.......AP increment of the above
N.......Number of elements in the major diagonal
STITLE DMXA
SENTRY DMXA, 6
" THIS PROGRAM DOES A FAST DEMUX
" OF SEG A FORMAT DATA PRESENTED AS A 4K
" BLOCK STARTING AT ADDRESS ZERO WITH THE
" PREVIOUS GAINS ELSEWHERE IN MEMORY
" THE GAIN ADDRESS THE DEMUXED OUTPUT ADDRESS AND INCREMENT ARE INPUT
" TO THE ROUTINE ALONG WITH THE NO OF THE GAIN CHECK AND THE
" GAIN CHANGE DIRECTION
GAIN SEQU 8
DBASE SEQU 1
DINC SEQU 2
GCNT SEQU 3
D1R SEQU 4
N1 SEQU 5
DADR SEQU 6
DIRCK SEQU 7
DATA SEQU 7
IADR SEQU 10
TEMP SEQU 11
N SEQU 12
N27 SEQU 13
SYNC SEQU 14
BIAS SEQU 15
TWO SEQU 16
MASK SEQU 17
ERR SEQU 17

"END OF REGISTER ASSIGNMENTS START OF CODE
DMXA: LDSPI BIAS;DB=15.;FADD ZERO,ZERO
       LDSPI TWO ;DB=2. ;FADD
       LDSPI N27 ;DB=27.
       LDSPI SYNC;DB=1.
       LDSPI MASK;DB=200
       MOV GAIN,GAIN;SETMA
       SUB DINC,DBASE;LDDPA;DB=1.
       LDSPI N;DB=36.
       MOV DBASE,DADR;INCMA
       PUSH: DEC N;INCOPA;DPY<MD
              INCMA;BGT PUSH
              MOV GAIN,GAIN;DPY<SPFN
              CLR IADR;SETMA
       OUTLP: MOV MASK,TEMP
              INC IADR;SETMA
       LDSPI DATA;DB=MD
       SUB SYNC,DATA
       LDSPI GAIN;DB=MD;BNE ERR1
       AND GAIN,TEMP
       LDSPI DIRCK;DB=1.;BNE SET
       MOV SYNC,DIRCK
       SET: LDSPI TEMP;DB=31.
       AND TEMP,GAIN
       ADD BIAS,GAIN;BEQ NEW
       INC GCNT;SETDPA
       LDSPI TEMP;DB=DPY
       SUB GAIN,TEMP
       SUB DIR,DIRCK;BNE ERR2
       LDDPA;DB=1.;BNE ERR3
       BR SKIP
       ERR1: LDSPI ERR;DB=1.
              RETURN
       ERR2: MOV GAIN,GAIN;DPY<SPFN
              JMP SKIP
       ERR3: COM DIR

"SET UP CONSTANTS
"SET UP SAVE ON DPY
"SET UP BASE ADDRESSES
"MD-DPY SAVE LOOP
"SAVE GAIN ADDRESS
"SET UP MEMORY ACCESS
"PUT MASK IN TEMP
"CHECK SYNC BITS
"GOTO ERROR IF NE
"CHECK DIRECTION BIT
"CLEAR UNWANTED SYNC BITS
"ADD BIAS TO GAIN
"GET GAIN TO CHECK
"CHECK GAIN AND SUBMUX GAIN
"CHECK DIRECTION BIT
"SET UP FOR NEXT LOOP
"SET UP ERROR
INC DIR
JMP SKIP

"HERE EVERY 30 TIMES"

NEW:
CLR GCNT;LDLPA;DB=1.

"START OF MAIN LOOP"

SKIP:
INC IADR;SETMA
LDSPI N;DB=3B.

"SHIFT DATA WORD R"

CONT:
LDSP GAIN;DB=DPY
LDSP DATA;DB=MD
MOVR DATA,DATA
BZC SAME
ADD DIR,GAIN

SAME:
MOVL DATA,DATA;DPX<DB;DB=SPFN
BGE NMI

"TURN -1 TO +1 AND VV"

ADD TWO,DATA;DPX<DB;DB=SPFN

MVI:
MOV N27,N27;FADD ZERO,MDPX
MOV GAIN,GAIN;DPY<DB;DB=SPFN;FADD

DPX<FA
LDSPE TEMP;DB=DPX

SUB GAIN,TEMP;FADD ZERO,MDPX
INC IADR;SETMA;FADD

DEC N;INCDPA
ADD DIR,DATA;DATA;DPX<DB;OB=SPFN

BGE NMI

MOV DBASE,DADR;BEQ FIN

JMP OUTLP

FIN:
LDLPA;DB=31.
LDSPI TEMP;DB=DPV
CLR ERR;LDLPA;DB=1.

"GET GAIN AND INIT FADDER"

MOV TEMP,TEMP;SETMA;MI<DB;DB=DPV;INCDPA
LDSPI N;DB=29.

"GET A<.0'>"

POP:
INCMA;MICDB;DB=DPY;INCDPA;DEC N

"SET UP RESULT ADDRESS FOR LOOP"

BGT POP

RETURN

SEND

STITLE VBINSC
SENTRY VBINSC,7

"THIS IS A PROGRAM WHICH"

"REDUCES THE EXPONENT OF A"

"FLOATING POINT NUMBER BY A"

"SPECIFIED AMOUNT"

" S-PAD DEFINITIONS
A SEQU $EQU .0' "VECTOR BASE ADDRESS"
I SEQU 1 "INC OF VECTOR A"
C SEQU 2 "BASE ADDRESS OF RESULT"
K SEQU 3 "INC OF VECTOR C"
G SEQU 4 "GAIN ADDRESS"
J SEQU 5 "GAIN INCREMENT"
N SEQU 6 "NO OF VECTOR ELEMENTS"
FACT SEQU 7 "GAIN VALUE"
RES SEQU 8 "NEW EXPONENT"

VBINSC: MOV G,G;SETMA;FADD ZERO,ZERO

"GET GAIN AND INIT FADDER"

MOV A,A;SETMA;FADD

"GET A<.0'>"

SUB K,C

"SET UP RESULT ADDRESS FOR LOOP"

LDSPI FACT;DB=MD

"GET GAIN ON S-PAD"

LOOP:
LDSPE RES;DPX<DB;DB=MD

"GET VECT ELEMENT EXPONENT"

ADD J,G;SETMA

"INIT ACCESS TO NEXT G"

SUB FACT,RES;FADD ZERO,MDPX

"PUT NEW EXP ON THE NO."

ADD J,A;SETMA

"INIT ACCESS TO NEXT A"

LDSPI FACT;DB=MD

"GET NEXT GAIN ONTO S-PAD"

DEC N;FADD

"DEC COUNTER AND PUSH FADDER"

ADD K,C;SETMA;MI<FA;BGT LOOP

RETURN

SEND
TITLE FACTOR
SENTRY FACTOR,13
SEXT DIV

* MPFACT.APS
* THIS IS A ROUTINE TO DO FACTORISATION
* OF A TRIDIAGONAL MATRIX
* CALL FACTOR(A,AINC,B,BINC,C,CINC,L,LINC,U,UINC,N)
* WHERE A,B,C ARE THE THREE DIAGONALS OF THE
* TRIDIAGONAL MATRIX AND L AND U ARE THE FACTORED
* RESULTS

A  SEQU  1
AIRC  SEQU  2
BINC  SEQU  3
CINC  SEQU  4
LINC  SEQU  5
UINC  SEQU  6
UINC  SEQU  7
UINC  SEQU  8
UINC  SEQU  9
UINC  SEQU  10
UINC  SEQU  11
UINC  SEQU  12

* END OF ASSIGNMENTS
* BEGINNING OF MAIN CODE

FACTOR: MOV B,B;SETMA;FADD ZERO,ZERO  "GET B(1) AND CLEAR ADDER
SUB CINC,C;FADD  "SET UP C ADDRESS
ADD AINC,A;SETMA  "GET A(2)
MOV U,U;SETMA;DB=MD;MI<DB;DPX<DB  "GET U(1) AND SAVE ON DPX
DEC N  "DEC COUNTER

* START OF MAIN CALCULATION LOOP

LOOP:  ADD CINC,C;SETMA;DPY<MD  "INIT C GET DO A/U
JSR DIV
FMUL DPX,MD;ADD BINC,B;SETMA  "L*C GET NEXT B
FMUL  "PUSH MULTIPLIER
FMUL ADD LINC,L;SETMA;MI<DPX  "B-L*C GET NEXT A
FSUBR FM,MD;ADD AINC,A;SETMA  "PUSH ADDER AND DEC COUNTER
FAA;DEC N
ADD UINC,U;SETMA;MI<FA;DPX<FA;BGT LOOP  "SAVE U IN MEM AND DPX

* END OF MAIN LOOP CHECK FOR ERRORS

CLR 17;BFPE ERR
RETURN
ERR:  INC 17
RETURN

END
SOLVE
ENTRY
SOLVE, 15
SEXT DIV
SEXT SPUFLT
" MPSOLV.APS
" THIS IS A ROUTINE TO SOLVE A TRIDIAGONAL
" MATRIX SET OF EQUATIONS ONCE THEY HAVE BEEN FACTORISED
" BY MPFACT.APS
" CALL SOLVE(L, LINC, U, UINC, RHS, RHSINC, C, CINC, Y, YINC, X, XINC, N)
" L AND U ARE THE FACTORED COEFFICIENTS
" RHS IS THE RIGHT HAND SIDE
" C IS THE TOP DIAGONAL OF ORIg. MATRIX
" Y IS TEMPORARY STORAGE
" AND X ARE THE RESULTS

L  S Equ  0
LINC S Equ  1
U  S Equ  2
UINC S Equ  3
RHS S Equ  4
RHSINC S Equ  5
C  S Equ  6
CINC S Equ  7
Y  S Equ  10
YINC S Equ  11
X  S Equ  12
XINC S Equ  13
N  S Equ  14

" END OF ASSIGNMENTS BEGINNING OF MAIN INTRO

SOLVE: MOV N, 17
JSR SPUFLT
" GET N ONTO SPAD 15
MOV RHS, RHS; SETMA
" FLOAT IT
ADD LINC, L; SETMA; FSUBR TM, DPX(1)
" MANIPULATE COUNTER
ADD RHSINC, RHS; SETMA
MOV Y, Y; SETMA; MI<DB; DPX<DB; DB=MD
" GET RHS(1)

" START FIRST MAJOR LOOP

LOOFA: FMUL DPX, MD; ADD CINC, C; FADD
" INC C ADDR MANIP COUNTER
FMUL: DPX<MD; ADD UINC, U; FSUBR TM, DPX(1)
" PUSH MULT GET RHS
FSUBR FM, DPX; ADD XINC, X
" RHS−L*Y
FADD ZERO, FA; ADD RHSINC, RHS; SETMA
" PUSH ADDER GET NEXT RHS
ADD YINC, Y; SETMA; M<FA; DPX<FA; BF GT LOOPA
" GET NEXT Y CHECK FOR LOOP END

" END OF LOOP ONE NEXT SET UP FOR CALC
" WHICH FINALLY GET US X

MOV U, U; SETMA; FADD ZERO, ZERO
" GET U(N) ZERO ADDER
SUB CINC, C; SETMA; FADD
" GET C(N−1)
DEC N; DPX<DPX
" DEC COUNTER GET Y(N)
JSR DIV
FMUL DPX, MD; SUB YINC, Y; SETMA
" X(N)*C(N−1) AND GET NEXT Y

" START OF LAST LOOP

LOOPB: FMUL
" PUSH MULTIPLIER
FMUL: SUB UINC, U; SETMA
" GET NEXT U
FSUBR FM, MD; SUB CINC, C; SETMA
" Y−REST NEXT C
FADD: SUB XINC, X; SETMA; MI<DPX
" SAVE X(N)
DPX<FA; DPX<XMD; JSR DIV
" GET X(N−1)/U(N−2)

DEC N
BGT LOOPB; FMUL DPX, MD; SUB YINC, Y; SETMA
" DO MULT X*C/U

" END OF LOOP TIDY UP HERE

SUB XINC, X; SETMA; MI<DPX
" SAVE X(1)

" CHECK FOR ERRORS
CLR 17; BFPE ERR
RETURN
ERR: INC 17
RETURN
"
Plotting subroutines

CONSYS

Consys was implemented from the version resident on MTS at Newcastle, and full documentation is available from this source. Basically it is a general contouring subroutine.

CALL CONTUR(X,IX,Y,IY,Z,IDX,CZ,NC,PTR,SWCHES,MINDIS,

NXL,XLOC,NYL,YLOC)

Arguments:-

X.....Floating point...Grid positions in X direction
IX....Integer..........Number of X grid points
Y.....Floating point...Grid positions in Y direction
IY....Integer..........Number of Y grid points
Z.....Floating Point...Virtual array containing values to be contoured
      Z(I,J)= Function of (X(I),Y(J))
IDX...Integer..........Declaration of column size for Z array
      Z(IDX,IDY)
CZ....Floating point...Values at which to have contour lines
NC....Integer..........Number of contour values
PTR...Floating point...Work array of size NC
SWCHES.Logical........5 element array of logical switches
                      1 - draw XB,YB XT,YB border
                      2 - draw XB,YB XB,YT border
                      3 - draw XB,YT XT,YT border
4 - draw XT,YT XT,YB border
5 - label contours

MINDIS. Floating point...Minimum distance between contour labels
NXL...Integer.........Number of constant X points for labels
XLOC...Floating point...Constant X positions for labels
NYL...Integer.........Number of constant Y values for labels
YLOC...Floating point...Constant Y positions for labels

**Rasterising Interception**

The rasterising interception program MPRASM picks up the active vector plot file from the system disc and then is fully interactive for the remaining options. The user is asked if output is to disk or tape. If it is to disc he is then asked for an output file name, or if it is to tape the drive number. The plot is then rasterised and saved to the chosen medium. At the end of the program the total number of raster lines generated is written out for later use with the merge programs.
SUBROUTINE CONCUR(X, IX, Y, IY, Z, IDX, CZ, NC, PTR, SWCHES,
+ MINDIS, NXL, NYL, YLOC)
C CONCUR PRODUCES COORDINATE PAIRS FOR DRAWING A PICTURE
C WHICH IS A CONTOUR MAP OF THE DATA IN THE ARRAY Z. Z IS A
C DATA SURFACE, I.E., Z(I, J) = F(X(I), Y(J)).
C THE BASIC ALGORITHM FOR THIS ROUTINE WAS SUGGESTED BY:
C G. W. HARTWIG, "CONCUR - A FORTRAN IV SUBROUTINE FOR PLOT-
C TING CONTOUR LINES," BALLISTIC RESEARCH LABORATORIES MEMO-
C RANDUM REPORT # 2282, ABERDEEN PROVING GROUND, MARYLAND,
C THIS ROUTINE COMPRIS ES THE FIRST HALF OF HARTWIG'S ALGORITHM;
C IT HAS BEEN MODIFIED TO REFLECT THE FACT THAT THE MOST COM-
C PUTATIONALLY EFFICIENT PROCEDURE IS TO QUICKLY REJECT SURFACE
C CELLS WHICH CONTAIN NO CONTOURS. IF A CELL DOES CONTAIN ONE
C OR MORE CONTOURS, SUBROUTINE CTQQ IS CALLED TO COMPLETE
C HARTWIG'S PROCEDURE, I.E., ACTUALLY FORM THE COORDINATE PAIRS
C FOR THE CONTOUR LINES.
C THE FOLLOWING CODE IS FOR VERSION 1.2 OF CONSYS, PRODUCED
C IS MAY, 1976. GGC

VIRTUAL Z(IDX, IY)
REAL Z, X(IDX), Y(IY), CZ(NC)
REAL MINDIS, XLOC(NXL), YLOC(NYL)
LOGICAL*1 SORTED, SWCHES(S)
LOGICAL*1 LBLLOC, DISTOK
INTEGER*2 PARERR, CONCNT, CCP1, NTEMP
INTEGER*2 PL, PH
COMMON /CONCOM/ LOWER, UPPER, XL, XR, XC, YL, YU, YC,
+ ZLL, ZUL, ZLR, ZUR
INTEGER*2 LOWER, UPPER, ERRUNT
DATA IOMAX/16/, ERRUNT/7/

CHECK FOR OBVIOUS ERRORS IN THE PARAMETERS.

PARERR = 0
IF (IX .GT. IDX ) PARERR = PARERR + 1
IF (IX .LT. 2 ) PARERR = PARERR + 1
IF (IY .LT. 2 ) PARERR = PARERR + 1
IF (IDX .LT. 2 ) PARERR = PARERR + 1
IF (NC .LT. 1 ) PARERR = PARERR + 1
IF (PARERR .NE. 0 ) GO TO 986
IF (.NOT. SWCHES(B)) GOTO 1994
IF (MINDIS.LT.0) PARERR = PARERR+1
IF (NXL .LT. 0) PARERR=PARERR+1
IF (NYL .LT. 0) PARERR=PARERR+1
IF (NXL+NYL .LT .1) PARERR=PARERR+1
IF (PARERR .NE. 0 ) GOTO 1993
C
C CHECK X AND Y TO ENSURE THEY ARE IN STRICTLY ASCENDING
C ORDER.  NOTE THAT THE ERROR MESSAGE WHICH IS WRITTEN IF THEY
C ARE NOT SCARES THE USER INTO CHECKING BOTH ARRAYS, EVEN
C THOUGH THE CODE DOESN'T CHECK Y IF X IS BAD.
C
DO 889 I = 2, IX
  IF( X(I) .LE. X(I-1) ) GO TO 988
CONTINUE
DO 889 I = 2, IY
  IF( Y(I) .LE. Y(I-1) ) GO TO 988
CONTINUE
C
C SORT THE ARRAY OF CONTOUR VALUES.
C
IF( NC .EQ. 1 ) GO TO 5
DO 1 M = 1, NC
  PTR(M) = M
1 CONTINUE
M = NC-1
2 CONTINUE
SORTED = .TRUE.
DO 4 K = 1, M
  PL = PTR(K)
  PH = PTR(I+K)
  IF( CZ(PH) .GT. CZ(PL) ) GO TO 3
  PTR(K) = PH
  PTR(I+K) = PL
3 CONTINUE
SORTED = .FALSE.
4 CONTINUE
IF( SORTED ) GO TO 6
5 M = M - 1
IF( M .GE. 1 ) GO TO 2
GO TO 6
6 CONTINUE
7 OFS = 8
8 CONTINUE
97 CONTINUE
CZMAX = CZ(PTR(NC))
CZMIN = CZ(PTR(1))
C
BEGIN THE CONTOURING PROCESS BY LOOKING AT EACH CELL IN THE
SURFACE IN TURN.  FOR EACH CELL, WE ASK THE QUESTION, "DOES
THIS CELL CONTAIN ANY CONTOUR LINES AT THE USER-SPECIFIED
VALUES IN THE CZ ARRAY?"  IF THE ANSWER IS NO, WE IMMEDIATELY
PROCEED TO THE NEXT CELL.  IF THE ANSWER IS YES, WE FIND
THE LOWER AND UPPER LIMITS IN THE SORTED CZ ARRAY OF CONTOUR
VALUES WHICH INTERSECT THIS CELL, AND PASS THIS INFORMATION
AND THE CELL COORDINATES TO SUBROUTINES CTDO (VIA CONCOM)
WHERE THE COORDINATES FOR THE CONTOURS ARE PRODUCED.
C
IYM1 = IY - 1
YU = Y(1)
DO 30 J = 1, IY1
YL = Y(1) + Y(J+1)
YC = .5*(YL + YU)
XR = X(1)
ZLR = Z(1, J)
ZUR = Z(1, J+1)
DO 37 I = 2, IX
XL = XR
XR = X(I)
ZLL = ZLR
ZUL = ZUR
ZLR = Z(I, J)
ZUR = Z(I, J+1)
ZMIN = AMIN1(ZLL, ZUL, ZLR, ZUR)
IF( ZMIN .GT. CZMAX ) GO TO 37
ZMAX = AMAX1(ZLL, ZUL, ZLR, ZUR)
IF( ZMAX .LT. CZMIN ) GO TO 37
IF( ZMAX .EQ. ZMIN ) GO TO 37
DO 12 CONCNT = 1, NC
ZB = CZ(PTR(CONCNT))
IF( ZB .LT. ZMIN ) GO TO 12
IF( ZB .GT. ZMAX ) GO TO 37
LOWER = CONCNT
UPPER = LOWER
IF( UPPER .EQ. NC ) GO TO 14
CCP1 = CONCNT + 1
DO 11 II = CCP1, NC
IF( ZMAX .LT. CZ(PTR(II)) ) GO TO 14
UPPER = II
CONTINUE
CONTINUE
GO TO 14
CALL CTQQ(CZ,PTR,NC)
C IF THE USER SPECIFIED VIA SWCHES(5) THAT LABELS ARE TO BE DRAWN, FIND OUT HERE IF THIS CELL IS A CANDIDATE FOR LABELING, AND IF IT IS, CALL LABELR TO DRAW THE LABEL.
IF( .NOT. SWCHES(5) ) GO TO 29
LBLLOC = .FALSE.
IF( NXL .EQ. 0 ) GO TO 23
DO 22 M = 1, NXL
IF( .NOT.(XL .LE. XLOC(M) .AND. XLOC(M) .LT. XR ) ) GO TO 22
LBLLOC = .TRUE.
GO TO 27
CONTINUE
GO TO 14
CALL CTQQ(CZ,PTR,NC)
C IF THE USER SPECIFIED VIA SWCHES(5) THAT LABELS ARE TO BE DRAWN, FIND OUT HERE IF THIS CELL IS A CANDIDATE FOR LABELING, AND IF IT IS, CALL LABELR TO DRAW THE LABEL.
IF( .NOT. SWCHES(5) ) GO TO 29
LBLLOC = .FALSE.
IF( NXL .EQ. 0 ) GO TO 23
DO 22 M = 1, NXL
IF( .NOT.(XL .LE. XLOC(M) .AND. XLOC(M) .LT. XR ) ) GO TO 22
LBLLOC = .TRUE.
GO TO 27
CONTINUE
GO TO 14
CALL CTQQ(CZ,PTR,NC)
C IF THE USER SPECIFIED VIA SWCHES(5) THAT LABELS ARE TO BE DRAWN, FIND OUT HERE IF THIS CELL IS A CANDIDATE FOR LABELING, AND IF IT IS, CALL LABELR TO DRAW THE LABEL.
3157      LBLLOC = .TRUE.
3158      GO TO 27
3159 CONTINUE
3160 27      CONTINUE
3161 28      IF( .NOT. LBLLOC ) GO TO 28
3162     IF( DISTOK(XC, YC, MINDIS) )
3163 CALL NUMBER(XC, YC, S1, ZS, S.4)
3164 CONTINUE
3165 CONTINUE
3166 CONTINUE
3167 CONTINUE
3168 CONTINUE
3169 CONTINUE
3170 CONTINUE
3171 CONTINUE
3172 CONTINUE
3173 CONTINUE
3174 CONTINUE
3175 CONTINUE
3176 CONTINUE
3177 CONTINUE
3178 STOP
3179 CONTINUE
3180 CONTINUE
3181 STOP
3182 CONTINUE
3183 STOP
3184 CONTINUE
3185 FORMAT STATEMENTS FOR CONTUR ERROR COMMENTS.

DRAW BORDER LINES, IF THE USER INDICATED VIA THE SWCHES VECTOR THAT THEY ARE WANTED.

XL = X(1)
XR = X(I)
YL = Y(1)
YU = Y(I)
IF( .NOT. SWCHES(1) ) GO TO 42
CALL PLOT(XR, YL,+3)
CALL PLOT(XL, YL,+2)
42 CONTINUE
43 CONTINUE
IF( .NOT. SWCHES(2) ) GO TO 44
CALL PLOT(XL, YL,+3)
CALL PLOT(XL, YU,+2)
44 CONTINUE
45 CONTINUE
IF( .NOT. SWCHES(3) ) GO TO 46
CALL PLOT(XL, YU,+3)
CALL PLOT(XR, YU,+2)
46 CONTINUE
47 CONTINUE
IF( .NOT. SWCHES(4) ) GO TO 48
CALL PLOT(XR, YU,+3)
CALL PLOT(XR, YL,+2)
48 CONTINUE
RETURN
HANDLE BAD PARAMETERS IN THE CONTUR CALL HERE.

WRITE(ERRUNT, 997) PARERR, IX, IV, IDX, NC
WRITE(ERRUNT, 999)
CONTINUE
STOP
CONTINUE
WRITE(ERRUNT, 994)
CONTINUE
STOP
FORMAT STATEMENTS FOR CONTUR ERROR COMMENTS.
**FORTRAN IV**

**V02.B4**

**THU 88-JAN-81 00:03:57**

**PAGE 895**

```
J.64  594 FORMAT(' **** ERROR: SUBROUTINE "CONL' !  
  + "TUR" HAS BEEN CALLED', 'WITH EITHER THE X OR', 'THE Y VECTOR (OR', 'BOTH) NOT IN STRICTLY ASCENDING ORDER.')
J.65  597 FORMAT(' **** THERE ARE ', I1, ' ERROR(S) IN THE PARA', 'METERS IN A CALL', 'TO SUBROUTINE "CONTUR".', 'DIAGNOSTIC INFORMATION FOLLOWS:', 'IX = ', 'IY = ', 'IDX = ', 'IDY = ', 'NC = ', 'NXL = ', 'NYL = ')
J.66  599 FORMAT(' **** DUE TO THE ABOVE ERROR, CONTUR WILL NOT', 'DRAW A CONTOUR MAP', 'BUT WILL INSTEAD', 'IGNORE THE CALL.')

HANDLE ERRORS IN PARAMETERS TO CONLBL HERE.

9197 CONTINUE
9198 WRITE(ERRUNT, 1998) PARERR, MINDIS, NXL, NYL
9199 CONTINUE
9200 FORMAT(' **** ERROR: SUBROUTINE "CONL" HAS BEEN CALLED', 'WITH NO PRECEDING INITIALIZATION CALL', 'TO SUBROUTINE "CONSET".')
9201 1998 FORMAT(' **** DUE TO THE ABOVE ERROR, CONL WILL NOT INITIALIZE THE CONTOUR LABELING ROUTINES, BUT WILL INSTEAD IGNORE THE CALL.')
9202 END
```
LOGICAL FUNCTION DISTOK(X, Y, MINDIS)

COUNTUR CALLS DISTOK TO SEE WHETHER A CONTOUR LABEL CAN
BE PLACED AT (X, Y) AND BE MORE THAN MINDIS UNITS AWAY
FROM ANY OTHER LABEL PREVIOUSLY PLACED ON THE CONTOUR MAP.
IF A LABEL CAN BE SAFELY PLACED ON THE MAP, DISTOK RETURNS
THE VALUE .TRUE. AFTER SAVING THE VALUES OF X AND Y IN A
DYNAMICALLY ALLOCATED LOCAL ARRAY. IF THE LABEL WOULD
FALL WITHIN MINDIS UNITS FROM A PREVIOUS LABEL, DISTOK
SIMPLY RETURNS THE VALUE .FALSE.

THE FOLLOWING CODE IS FOR VERSION 1.2 OF CONSYS, PRODUCED
15 MAY, 1976. GNC

REAL X, Y, MINDIS
REAL COORD(100)
INTEGER CURLEN
DATA CURLEN/0/, NCOORD/100/

DISTOK = .FALSE.
IF( CURLEN .LT. 1 ) GO TO 9
IF( CURLEN .GE. NCOORD ) RETURN
DO 8 I = 1, CURLEN, 2
XI = COORD(I)
IF( ABS(X-XI) .GT. MINDIS ) GO TO 5
YI = COORD(I+I)
IF( SQRT((X-XI)**2+(Y-YI)**2) .LE. MINDIS ) RETURN
CONTINUE
CONTINUE
CONTINUE
DISTOK = .TRUE.
COORD(1+CURLEN) = X
COORD(2+CURLEN) = Y
CURLEN = CURLEN + 2
RETURN
END
SUBROUTINE CTQQICZ(PTR, NC
CTQ IS CALLED FROM CONTUR TO PRODUCE CONTOUR COORDINATE
VALUES FOR THE GRID CELL BOUNDED BY XL AND XR, VL AND VU, AND
ZLL, ZUL, ZUR, AND ZLR. COORDINATE PAIRS THUS PRODUCED ARE
DISPOSED OF VIA CALLS TO THE USER-SPECIFIED ROUTINES PLOT
AND CVKA. THE Z VALUES TO BE CONTOURED ARE STORED IN
CZP(TRICLOWER), .... CZP(TRICUPPER)).

THE ALGORITHM FOR THIS ROUTINE WAS TAKEN FROM:
G. W. HARTWIG, "CONTUR - A FORTRAN IV SUBROUTINE FOR
PLOTTING CONTOUR LINES," BALLISTIC RESEARCH LABORATORIES
MEMORANDUM REPORT # 2282, ABERDEEN PROVING GROUND,

THE FOLLOWING CODE IS FOR VERSION 1.2 OF CONVS, PRODUCED
15 MAY, 1976. GNC

REAL CZINC
LOGICAL KCKI(8), CENTER
REAL PX(8), PY(8), PTEMP
EQUIVALENCE (PX(1), PX1), (PX(2), PX2), (PX(3), PX3),
+ (PX(4), PX4), (PX(5), PX5), (PX(6), PX6),
+ (PX(7), PX7), (PX(8), PX8)
EQUIVALENCE (PY(1), PY1), (PY(2), PY2), (PY(3), PY3),
+ (PY(4), PY4), (PY(5), PY5), (PY(6), PY6),
+ (PY(7), PY7), (PY(8), PY8)
INTEGER*2 PTRINC
COMMON /CONCOM/ LOWER, UPPER, XL, XR, XC, VL, VU, YC,
+ ZLL, ZUL, ZLR, ZUR
INTEGER*2 LOWER, UPPER

XC = .5*(XL + XR)
XLMXR = XL - XR
XLMXC = XL - XC
XRMXC = XR - XC
YLMYC = YL - YC
YLMYU = YL - YU
YUMYC = YU - YC
ZC = .25*(ZLL + ZUL + ZLR + ZUR)
DO 12 LEVEL = LOWER, UPPER
12 20 = CZ(PTR(LEVEL))
TL = ZLL - 20
TUL = ZUL - Z0
TLR = ZLR - 20
TUR = ZUR - 20
TC = ZC - 20
IC = 0
CENTER = .FALSE.
DO 11 M = 1, 8
KCHK(M) = .FALSE.
CONTINUE

SEGMENT 1:

IF ( TLL*TLR .GT. & ) GO TO 19
KCHK(1) = .TRUE.

IF ( TLL*TLR .EQ. & ) GO TO 12
IC = IC + 1
PX(IC) = TLL * XLMXR/(ZLR-ZLL) + XL
PY(IC) = YL
GO TO 18
CONTINUE

IF ( TLL .GT. & ) GO TO 16
IC = IC + 1
PX(IC) = XR
PV(IC) = VL
CONTINUE

GO TO 17
CONTINUE
GO TO 18
CONTINUE

CONTINUE

SEGMENT 2:

IF ( TLL*TC .GT. & ) GO TO 29
KCHK(2) = .TRUE.

IF ( TLL*TC .EQ. & ) GO TO 22
IC = IC + 1
FAC = TLL/(ZC-ZLL)
PX(IC) = XLMXC*FAC + XL
PY(IC) = YLMYC*FAC + YL
GO TO 28
CONTINUE

IF ( TC .NE. & ) GO TO 25
CENTER = .TRUE.
IC = IC + 1
PX(IC) = XC
PY(IC) = YC
CONTINUE
GO TO 28
CONTINUE
CONTINUE
SEGMENT 3:

IF( TUL*TUR .GE. S. ) GO TO 39  
KCHK(4) = .TRUE.
IC = IC + 1  
PX(IC) = TUL  
PY(IC) = YU  
CONTINUE

IF( TUL .NE. S. ) GO TO 35
IC = IC + 1
PX(IC) = XL
PY(IC) = YL
CONTINUE

CONTINUE

SEGMENT 4:

IF( TUL*TC .GE. S. ) GO TO 49
KCHK(4) = .TRUE.
IC = IC + 1
FAC = TUL/(ZC - ZUL)
PX(IC) = XL
PY(IC) = YU
CONTINUE

SEGMENT 5:

IF( TUL*TUR .GT. S. ) GO TO 59
KCHK(5) = .TRUE.
IC = IC + 1
PX(IC) = TUL * XLMXR/(ZUR-ZUL) + XL
PY(IC) = YU
CONTINUE

IF( TUR .NE. S. ) GO TO 55
IC = IC + 1
PX(IC) = XR
PY(IC) = YU
CONTINUE

CONTINUE

SEGMENT 6:

IF( TUR*TC .GE. S. ) GO TO 69
KCHK(6) = .TRUE.
IC = IC + 1
FAC = TUR/(ZC - ZUR)
CONTINUE

SEGMENT 7:

IF( TLR * TUR .GE. 8. ) GO TO 79
KCHK(7) = .TRUE.
IC = IC + 1
PX(IC) = XR
PY(IC) = TUR * YLMYC / (ZUR - ZLR) + YU
CONTINUE

SEGMENT 8:

IF( TLR * TC .GE. 8. ) GO TO 89
KCHK(8) = .TRUE.
IC = IC + 1
FAC = TC / (ZC - ZLR)
PX(IC) = XRMXC * FAC + XC
PY(IC) = YLMYC * FAC + YC
CONTINUE

NOW DERIVE THE LINE SEGMENTS TO BE DRAWN FROM THE CONTENTS
OF THE PX AND PY ARRAYS.

IF( IC .LE. 1 ) GO TO 117
IF( IC .GE. 6 .OR.
   ( IC .EQ. 5 .AND. CENTER ) ) GO TO 188
IF( .NOT. KCHK(8) ) GO TO 95
DO 94 L = 1, 7
LS=L
IF( .NOT. KCHK(L) ) GO TO 93
PX(IC+1) = PX(1)
PY(IC+1) = PY(1)
DO 92 M = 1, IC
   PX(M) = PX(M+1)
   PY(M) = PY(M+1)
CONTINUE
CONTINUE
GO TO 97
IF( .NOT. CENTER .OR. KCHK(1) ) GO TO 97
PTemp = PX1
PX1 = PX2
PX2 = PTemp
PTemp = PY1
PY1 = PY2
PY2 = PTemp
GO TO 97
CONTINUE
CALL PLOT( PX1, PY1 + 3)
DO 98 M = 2, IC
0184 PXM = PX(M)
0185 PYM = PY(M)
0186 CALL PLOT(PXM, PYM,+2)
0187 CONTINUE
0188 GO TO 109
0189 CONTINUE
0190 IF( IC .GE. 6 ) GO TO 104
0191 PX6 = PX5
0192 PY6 = PY5
0193 PX5 = PX2
0194 PY5 = PY2
0195 CONTINUE
0196 IF( KCHK(2) ) GO TO 105
0197 CALL PLOT(PX5, PY5,+3)
0198 CALL PLOT(PX6, PY6,+2)
0199 CALL PLOT(PX1, PY1,+2)
0200 CALL PLOT(PX2, PY2,+3)
0201 CALL PLOT(PX3, PY3,+2)
0202 CALL PLOT(PX4, PY4,+2)
0203 GO TO 108
0204 CONTINUE
0205 CALL PLOT(PX1, PY1,+3)
0206 CALL PLOT(PX2, PY2,+2)
0207 CALL PLOT(PX3, PY3,+2)
0208 CALL PLOT(PX4, PY4,+3)
0209 CALL PLOT(PX5, PY5,+2)
0210 CALL PLOT(PX6, PY6,+2)
0211 GO TO 108
0212 CONTINUE
0213 GO TO 109
0214 CONTINUE
0215 CONTINUE
0216 CONTINUE
0217 CONTINUE
0218 CONTINUE
0219 RETURN
0220 END
C-T PROGRAM RASM

C-F MAIN PROGRAM FOR VECTOR TO RASTER CONVERSION - MAPPED ALGORITHM

COPYRIGHT 1976, VERSATEC INC., SANTA CLARA, CALIFORNIA 95051

THE CONTENTS OF THIS DOCUMENT ARE PROPRIETARY TO VERSATEC, INC., AND ARE NOT TO BE DISCLOSED TO OTHERS OR USED FOR PURPOSES OTHER THAN INTENDED WITHOUT THE WRITTEN APPROVAL OF VERSATEC.

CALLS: DOPEN, DREAD, DWAIT, MWRITE, PREAD, INVECT, CYCLER, IRZERO
CALLED BY: -NONE-

COMMON VARIABLES USED: /PPEP2/ ISCAN, NWORD, LYNEND, IQ, MAXQ, NEPL,
MSGLVL, LOST, IX0, IY1, NDLTX, IV2,
IBUFG, I2FLG, IR2, MXSTEP, IM
/IOCOM/ LBLK, NBLK, LREC, JUNIT, LUNIT, IPARM, MUNIT

ASSUMPTIONS: WHEN THE 'IM' OR 'INBUF' ARRAYS ARE DIMENSIONED OTHER CHANGES ARE AS FOLLOWS: IF IM('I') THEN SET 'IMD' = 'I'; IF INBUF('J') SET 'INBUFD' = 'J'. 'J' MUST BE AN INTEGER MULTIPLE OF AND GREATER THAN OR EQUAL TO 2*(NBLK*LBLK).

C-P 58538-28583 REV. A - PART NUMBER
C-S RT-11 - OPERATING SYSTEM

AUTHOR: M.D. DOBERVICH 87/87/76

PROGRAM RASTER

COMMON /PPEP2/ ISCAN, NWORD, LYNEND, IQ, MAXQ, NEPL,
1 NBIT, IBI(16), KB(16), JBT(16), MSGVL, LOST,
2 IX0, IY1, NDLTX, IV2,
3 NW, NDB, NRUN, ISUM, NDLTY,
4 IBUFG, I2FLG, IR2, MXSTEP,
5 IMD/3500/, INBUFD/512/, MAPEND/650/

COMMON /IOCOM/ LBLK, NBLK, LREC, LVEC, JUNIT, JUNIT, KUNIT, LUNIT, MUNIT,
1 IPARM, IPCTR, IPREC, IPBUF(256)

DIMENSION INBUF(512)

DATA IMD/3500/, INBUFD/512/, MAPEND/650/

DATA IREC/0/, IOLD/1/, MAPKEY/102/

1 FORMAT (25H FILE/ALGORITHM MISMATCH , I6)
2 FORMAT (21H IX0, IY1, NDLTX, IV2 = 4(1X16))
3 FORMAT (44H IM OR INBUF ARRAY NOT PROPERLY DIMENSIONED )
4 FORMAT (21H MAP BUFFERS EXCEEDED)
6 FORMAT (1X16,13H VECTORS LOST / 1 1X16,18H ACTIVE LINES USED /)

C... ATTACH THE MATRIX TO THIS JOB.
C... FORM FEED AT PLOT/FRAME START.
C... OPEN MAP/PARAMETER FILE.
   CALL DOPEN (IPARM,-1,1)
   CALL MTXSET
   CALL MTX(IM(IJBFG),0,2)
C... CHECK ALGORITHM KEY.
CALL PREAD (KEY, 1)
  IF (KEY.EQ.MAPKEY) GO TO 7018
WRITE (MUNIT, 1) KEY
STOP

C

7018 CALL PREAD (NSCAN, 1)
CALL PREAD (IR1, 1)
CALL PREAD (IR2, 1)
CALL PREAD (ISCAN, 1)
CALL PREAD (INWORD, 1)
CALL PREAD (IWORD, 1)
CALL PREAD (I2FLG, 1)
CALL PREAD (IOUT1, 1)
CALL PREAD (MXSTP, 1)
CALL PREAD (MSGLVL, 1)
CALL PREAD (LINES, 1)
CALL PREAD (LYNES, 1)
CALL PREAD (NBLK, 1)

C

LREC = LBLK * NBLK
IBLKSZ = 2*LHK*NBLK

C... INPUT AND OUTPUT BUFFERS DIMENSIONED PROPERLY?
  IF (MOD(IBLKSZ,INBUF).EQ.0.AND.IBLKSZ.LE.INBUF.AND.IWORD.LE.IMD)
  1 GO TO 7011
WRITE (MUNIT,3)
STOP

C

7011 INEW = LREC + 1

C... OPEN MAPPED VECTOR FILE
  CALL DOPEN (JUNIT,-1,NBLK)

C... INPUT MAP ENTRIES FOR CURRENT PLOT AT START OF THE IM ARRAY.

C... ALLOW DYNAMIC MAP SIZE ALLOCATION ONLY IF VECTOR QUEUING IS USED.
  IF (LYNES.NE.0) MAPEND = (IWOrd-(NEPL*LYNES) - (I2FLG*ISCAN))
  7020 MAPSZ = 1
  7038 CALL PREAD (IM(MAPSZ) ,2)

C... END OF MAP FOR THIS PLOT?
  IF (IM(MAPSZ).LT.0) GO TO 7040
  MAPSZ = MAPSZ + 2

C... DOES MAP SIZE FIT WITHIN THE MAXIMUM MAP ALLOCATION AREA?
  IF (MAPSZ.LT.MAPEND) GO TO 7030

C... MAP BUFFERS EXCEEDED.
  WRITE (MUNIT,4)
STOP

C... CHECK FOR END OF ALL PLOTING.
  7040 IF (MAPSZ.EQ.1) GO TO 7700
  MMAXX = IM(MAPSZ+1)
  IINDX = MAPEND

C... IF VECTOR QUEUING IS USED THEN MAP ALLOCATION BECOMES DYNAMIC.
  IF (LYNES.NE.0) IINDX = MAPSZ

C... CALCULATE NSCAN, NWORD AND ALLOCATE BUFFERS IN IM ARRAY.
  NSCAN = (IWOrd - (NEPL*LYNES) - IINDX)/(I2FLG*ISCAN)
  NWORD = ISCANN*NSCAN
  LYNEND = IWOrd-IINDX-(I2FLG*NWORD) - 1
  IBUFG = LYNEND + IINDX
  IR2 = IBUFG+NWORD+1
C... INITIALIZE ASSEMBLY LANGUAGE ROUTINES
C CALL INIT (IM(IBUFG), IM(IR2), IM(INDX))
C...
C... CONVERT IOUT1 AND MXSTEP VALUES TO IOUT AND MXSTEP BAND VALUES.
MXSTEP = MXSTEP/MSCAN
IOUT = IOUT1/MSCAN
C...
C... INITIALIZE JREC AND MAP SEARCH POINTERS.
NREC = ((MAPSZ+1)/2) - 2
JREC = NREC + IREC + 1
NDXFTR = MAPSZ - 4
NDXPS = MAPSZ - 2
C...
C... RESET INITIAL BAND LIMITS AND COUNTERS.
IXSTR = 0
IXEND = NSCAN-1
IXSTRP = IXSTR
NBANDS = 0
C...
C... READ INITIAL BUFFER:
CALL DREAD (JUNIT, IM(IBUFG), JREC)
CALL IRZERO (IM(IBUFG))
C...
C... WAIT FOR LAST READ OPERATION COMPLETE
CALL DWAIT (JUNIT, IERR)
C...
C... SEARCH MAP FOR VALID VECTOR BLOCK TO READ.
Ijmp = 1
GO TO 7120
7110 NDXFTR = NDXFTR - 2
NREC = NREC - 1
C...
C... HAS ENTIRE MAP BEEN SEARCHED?
7130 IF (NDXFTR.LT.0) GO TO 7130
C...
C... DOES MAXIMUMVECTOR START BEFORE CURRENT BAND?
IF (IM(NDXFTR).LT.IXSTR) GO TO 7110
C...
C... DOES MINIMUM VECTOR START AFTER CURRENT BAND?
IF (IM(NDXFTR+1.GT.IXEND) GO TO 7110
C...
C... VALID VECTOR DATA FOUND. (INCREMENT MAP AND BLOCK INDEX)
NDXPRS = NDXFTR
NDXFTR = NDXFTR - 2
JREC = NREC + IREC
NREC = NREC - 1
GO TO 7150
C...
C... END OF BAND. (RESET MAP AND BLOCK INDEX)
7150 NDXFTR = MAPSZ - 2
NREC = ((MAPSZ+1)/2) - 1
NBANDS = NBANDS + 1
Ijmp = 0
C...
C... INCREMENT CURRENT BAND LIMITS.
IXSTR = IXSTR + NSCAN
IXEND = IXEND + NSCAN
C...
C... MORE BANDS IN THIS PLOT?
IF (IXSTR.LE.MMXX) GO TO 7120
IJMP = -1
GO TO 7280

C... START INPUT OF NEXT BUFFER
7150 CALL DREAD (JUNIT, INBUF(INEW), JRECV
C... RESET END OF 'OLD' BUFFER, BUFFER INDEX, AND CURRENT MAP MINIMUM.
C... SEARCH FOR VALID VECTORS IN THE CURRENT BUFFER
GO TO 7250
C... UPDATE 'NEW' MAP MINIMUM IF VECTOR IS LESS THAN CURRENT MINIMUM.
7220 IF (INBUF(JDX).LT.IM(NDXST+1)) IM(NDXST+1) = INBUF(JDX)
7250 JDX = JDX + 4
C... END OF 'OLD' BUFFER?
    IF (JDX.GT.IBEND) GO TO 7270
    IXN = INBUF(JDX) - IXSTRP
C... DOES 'RELATIVE' VECTOR START BEFORE BAND?
    IF (IXN.LT.IX8) GO TO 7260
C... DOES 'RELATIVE' VECTOR START AFTER BAND?
    IF (IXN.GE.NSCAN) GO TO 7220
C... VALID VECTOR LOCATED WITHIN BAND FOR VECTOR/RASTER CONVERSION.
    IY1 = INBUF(JDX+1)
    IX1 = INBUF(JDX+2)
    IY2 = INBUF(JDX+3)
    IF (MSGLVL.GE.6) WRITE(LUNIT,2) IX0, IY1, IX1, IY2
    IF (MSGLVL.GE.6) WRITE(LUNIT,2) IX0, IY1, IX1, IY2
C... TEST FOR POSSIBLE END OF BUFFER (DATA)
7260 IF (INBUF(JDX).GE.8) GO TO 7250
7270 NDXST = NDXPRS
C... END OF CURRENT VECTOR INPUT BLOCK.
C... IS STATUS - END OF PLOT, END OF BAND, BAND NOT COMPLETE?
C... EOP, EOB, BNC
    IF (IJMP) 7350, 7450, 7500
C 7350 CALL CYCLER (NBANDS, IM(IQNDX))
C... LOOP UNTIL ACTIVE LINE TABLE IS EMPTY
    NBANDS = 1
    IF (IQ.GT.1) GO TO 7350
C... OUTPUT LAST BAND OF CURRENT PLOT.
    CALL CYCLER (1, IM(IQNDX))
C... ACTION CHECK: FORM FEED ONLY, NO ACTION, IOUT AND FORM FEED?
C    FFO, NA, IFF
    IF (IOUT) 7370, 7380, 7360
C... CYCLE IOUT TIMES TO MOVE PLOT/FRAME PAST TONER.
7360 CALL CYCLER (IOUT, IM(IQNDX))
C
C... FORM FEED AT END OF PLOT/FRAME:
7370 CALL MTX (IM(IBUFG), 0, 2)
7380 MAXQ = MAXQ/NEPL
IF (MSGLVL.GE.1) WRITE (MUNIT,6) LOST, MAXQ
LOST = 0
MAXQ = 0
C
C... ADJUST CUMULATIVE PLOT (RECORD) POINTER.
IREC = IREC + ((MAPSZ+1)/2) - 1
CALL MWAIT
C
C... CHECK FOR NEXT PLOT
GO TO 7820
C
C... END OF BAND
7400 CALL CYCLER (NBANDS, IM(IQNDX))
NBANDS = 0
IXSTRP = IXSTR
C
C... SWAP INPUT BUFFERS
7500 IS = INEW
INew = IOLD
IOLD = IS
GO TO 7188
C
C... END OF ALL PLOTTING
7700 CALL MTX (IM(IBUFG), -1, 2)
STOP
END
THIS IS A SUBROUTINE PACKAGE WHICH IN CONJUNCTION WITH MPRASM INTERCEPTS PLOT DATA INTENDED FOR THE PLOTTER AND PUTS IT TO DISC OR TAPE

SUBROUTINE MWAIT

C DUMMY ROUTINE

RETURN
END

SUBROUTINE MTXSET

C SET UP ROUTINE

REAL*8 FSPEC;
REAL*4 FBUF(3)
LOGICAL*1 ITPDRW
COMMON /MPMTX/ICH,IOFLG,IBLK,IWD,ITPDRW

GET DATA PARAMETERS

IBLK=1
IWD=1
WRITE(7,1000)
1000 FORMAT(' ENTER A 0 FOR TAPE OUTPUT,'/,'% OR A 1 FOR DISC OUTPUT:','$)
READ(5,1001)IOFLG
1001 FORMAT(11)
GET FILE NAME IF REQD

IF(IOFLG.EQ.0)GOTO 10
WRITE(7,1002)
1002 FORMAT(' ENTER FILE NAME FOR OUTPUT:','$)
READ(5,1003)FBUF
CALL IRA050(12,FBUF,FSPEC)
ICH=IGETC()
IF(ENTER(ICH,FSPEC,-1).LT.0)STOP 'ENTER ERROR'
RETURN
GET TAPE NO

WRITE(7,1004)
1004 FORMAT(' ENTER TAPE DRIVE NUMBER:','$)
READ(5,1005)ITPDRW
RETURN
END
SUBROUTINE MTX(IPBUF,NWDS,IFLAG)

MAIN TRANSFER ROUTINE

INTEGER IBUF(257),IPBUF(NWDS),BUF(NBVT)
LOGICAL ISTAT,ITPDRW
COMMON /MPMTX/ICH,IOFLG,IBLK,IWD,ITPDRW
EQUIVALENCE (BUF(257),IBUF(1))

DATA NBVTBF/460/4/,NBLKBF/9/,ICOUNT/0/

DO 5 J=1,NWDS
% BUF(J)=B

CHECK FOR OPERATION TYPE

IF(NWDS.EQ.0)RETURN
ICOUNT=ICOUNT+1
IF(NWDS.LT.0)GOTO 50

SWAP DATA TO OUTPUT BUFFER

DO 10 J=1,NWDS
IBUF(IWD)=IPBUF(J)
IWD=IWD+1
IF(IWD.LE.20)GOTO 10
IF(ISTAT.EQ.0)GOTO 20
C OUTPUT

IF(IOFLG.EQ.20)STOP 'WRITE ERROR'
IBLK=IBLK+S
IWD=1
GOTO 10

TAPE OUTPUT

IFLEN=NBVTBF
NEVT=NBVTBF
IF(ITPDRW.EQ.0)GOTO 80
CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBVT,IEOTW)
IWD=1

CHECK FOR ERRORS

IF(IEOTW.GE.0)GOTO 30
WRITE(7,1000)ITPDRW,ICOUNT
1000 FORMAT(11,E10.0)
WRITE(7,1001)
1001 FORMAT('ENTER NEW WRITE DRIVE NUMBER:',S)
READ(5,1002)ITPDRW
1002 FORMAT(11)
IF(ISTAT.GE.0)GOTO 10
WRITE(7,1003)ICOUNT

FORMAT(' FATAL ERROR ON WRITING BUFFER NUMBER: ', I5)
STOP ' FATAL WRITE ERROR'
CONTINUE
RETURN

COME HERE TO FLUSH BUFFERS

DO 40 J=NWD, 2048
IF(IOFLG.EQ.0) GOTO 50
IF(IWRIT(2048,IBUF,IBLK,ICH).LT.0) STOP ' WRITE ERROR'
CALL CLOSE(ICH)
WRITE(7,1010) ICOUNT
RETURN

CONTINUE

IFLEN=NBLKBF
NBYT=NBYTBF
IEOTW=0
CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
IF(IEOTW.LT.0) GOTO 60
AS ONLY FLUSHING BUFFERS IGNORE ERRORS
BUF(1)='E'
BUF(2)='O'
BUF(3)='D'
CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
WRITE(7,1010) ICOUNT
FORMAT(' BUFFERS WRITTEN=', I5)
RETURN
END
Utilities

File Save/Restore Utility: MPTPSV

This program is completely interactive and the user is prompted for most of the required input. When in command mode the program puts a "?" on the screen and awaits one of the 4 commands shown below. Any other input required is then prompted.

**SAVE**

This causes a file to be written to tape. The program prompts the user for the file name and version number.

**REST**

This command causes a file to be brought and put back onto disc. The user is prompted for the file name and if necessary the version number.

**TDIR**

This command causes the program to compile a directory of the files on tape and put the output to either the printer or terminal.

**STOP**

This command causes the program to execute any queued restore jobs and then terminate the execution.
C TAPE/DISC SAVE RESTORE PROGRAM

VERSION 1

REAL*8 FSPEC,FSPECW,FSPEC(1000),DATE(1000),NBUF,DBUF,FSBUF,
  %EOTCOD,OSPEC(20),TPNUM,TODATE
REAL*4 FBUF(3),COM(4),CBUF
INTEGER*2 VNUM(1000),SIZE(1000),VBUF,SBUF,
  %WINUM(10),IFNUM(10),QNO(20),BUF(1024)
LOGICAL*1 EOT,EOR,HBLK(20),ANS,Yes,No,DRV,ISTAT
EQUIVALENCE (HBLK(1),NBUF),(HBLK(5),DBUF),(HBLK(17),VBUF),
  % (HBLK(19),SBUF)
COMMON/SERCH/FSPEC,DATE,VNUM,SIZE,HBLK,IFILE
DATA DEV/3RRK/,FSBUF/12RDKMPFVSDAT/,YES/'Y'/,NO/'N'/
DATA COM(1)/'SAVE'/,COM(2)/'REST'/,COM(3)/'STOP'/,COM(4)/'TDIR'/
DATA EOT'COD/12ROETOTEOT/

SET UP I/O CHANNELS

IF(IFCN(25).NE.0)STOP 'CHAN OVERFLOW'
IF(IFETCH(DEV).NE.0)STOP 'FETCH ERR'
ITR=IGETC(1)
IRD=21
IFILE=.0'
IQNUM=21
EOR=.FALSE.
EOT=.FALSE.

GET TODAYS DATE

CALL GDATE(TODATE)

CALL TDATE(TODATE)

10 FORMAT( 'TAPE SAVE/RESTORE PROGRAM VERSION 1' )
15 FORMAT( 'ENTER TAPE NAME AND DRIVE NUMBER' )
16 ACCEPT 16,TPNUM,DRV
17 FORMAT(A8,12)
18 ACCEPT 18,ANS
19 IF(ANS.EQ.NO)GOTO 19

TAPE INITIALISATION CODE

NBUF=TPNUM
DBUF=TODATE
ILEN=1
CALL TPHAND(4,DRV,ISTAT,,ILEN,IFILE,HBLK,20)

WRITE EOT BLOCK ANDREWIND TO BEGINNING OF BLOCK
NORMAL INTRO TO A SESSION

START OF MAIN SOFTWARE LOOP

AFTER COMMAND RECEIVED DECODE IT

CODE FOR SAVING A FILE

OPEN FILE TO BE SAVED

SET UP INTERNAL DIRECTORY
DATE(IFILE)=TODATE
VNUM(IFILE)=NVNO
SIZE(IFILE)=ISIZE

C SET UP HEADER BLOCK
NBUF=FSPECW
DBUF=TODATE
VBUF=NVNO
SBUF=ISIZE

C WRITE HEADER BLOCK
ILEN=1
CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,2)

C WRITE OUT THE FILE
ILEN=ISIZE
IF(ILEN.GT.4)CALL TPHAND(3,IDRV,ISTAT, ,ILEN,IFILE, , )
IF(ILEN.LE.4)CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,BUF,NBYTES)

SEE IF MORE SAVES RTO BE DONE

CALL CLOSEC(IWRT)

1 FORMAT(' MORE FILES TO BE SAVED Y/N ?',$)
ACCEPT I,ANS

WRITE EOT FILE IF NO MORE
NBUF=EOTCOD
ILEN=1
CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,2)

REWIND TO BEGINNING OF EOT FILE

CALL TPHAND(6,IDRV,ISTAT,2, ,IFILE, , )
GOTO 1

C CODE FOR A RESTORE

2 FORMAT(' ENTER FILE TO BE RESTORED' )
ACCEPT 2,FBUF
2 FORMAT(3AUD)
CALL IRAD50(12,FBUF,FSPECW)
IVER=8

SEARCH TO SEE IF FILE ALREADY PASSED

DO 2030 I=1,IFILE
IF (FSPEC(I).NE.FSPEC(I)) GOTO 2030

IVER = IVER + 1
IFNUM(IVER) = 1
IVNUM(IVER) = VNUM(I)
2033 CONTINUE
IF (IVER.LE.1) GOTO 2040

C IF MORE THAN ONE VERSION FIND WHICH ONE REQUIRED
C
2059 TYPE 2059
2069 FORMAT(' ENTER VERSION NUMBER REQUIRED: ', $)
2079 ACCEPT 2069,NVNO
2089 FORMAT(12)
2099 DO 2095 I = 1, IVER
2109 IF (NVNO.EQ.IVNUM(I)) IFNUM(I) = IFNUM(I)
2119 CONTINUE
C FIND OUT IF NEED TO QUEUE OR CONTINUE
C
2129 IF (IVER.NE.0) GOTO 2079
2139 TYPE 2059
2149 ACCEPT 2069,NVNO
2159 CALL FSERCH(FSPECW,IORV,EOT)
2169 IF (EOT) GOTO 2999
2179 CALL TPHAND(2, IDRV, ISTAT, , ILEN, IFILE, HBLK, 20)
2189 CALL TPHAND(5, IDRV, ISTAT, , ILEN, IFILE, )
2199 IF (NVNO.NE.VBUF) CALL TPHAND(5, IDRV, ISTAT, 2, ILEN, IFILE, )
2209 IF (NVNO.NE.VBUF) GOTO 2042
2219 ISIZE = SBUF + 16
2229 IF (ENTER(IRD, FSBUF, ISIZE).LT.0) GOTO 9003
2239 CALL TPHAND(1, IDRV, ISTAT, , ILEN, IFILE, )
2249 CALL TPHAND(5, IDRV, ISTAT, , ILEN, IFILE, )
2259 CALL R5ASC(12, FSPECW, FBUF)
2269 TYPE 2041, FBUF
2279 FORMAT(' ENTER NEW NAME FOR TAPE FILE: ', 3A4)
2289 ACCEPT 2029, FBUF
2299 CALL IRADS(EH, 12, FBUF, FSPECW)
2309 ISIZE = SBUF
2319 IF (ENTER(ITR, FSPECW, ISIZE).LT.0) GOTO 9003
2329 IBLK = 0
2339 DO 2045 I = 1, ISIZE
2349 IF (IREADW(256, BUF, IBLK, IRD).LT.0) GOTO 9004
2359 IF (IWRITEW(256, BUF, IBLK, ITR).LT.0) GOTO 9005
2369 IBLK = IBLK + 1
2379 CONTINUE
2389 CALL CLOSEC(IRD)
2399 CALL CLOSEC(ITR)
2409 GOTO 1
C PUT NAME IN THE QUEUE
C
2419 IQNUM = IQNUM + 1
2429 FSPEC(IQNUM) = FSPECW
2439 IQMOD(IQNUM) = IFNUM(I)
2449 IF (.NOT.EOR) GOTO 1
C COME HERE AT END OF RUN
C
CALL TPHAND(7, 1DRV, ISTAT, , IFILE, , )
IQ=0
IFILE=1
CALL TPHAND(5, 1DRV, ISTAT, 2, , IFILE, , )
N2=1

C SORT QUEUE INTO ORDER
C
IGAP=IQNUM
2081 IF(IGAP .LE. 1) GOTO 2100
IGAP=IGAP/2
IMAX=IQNUM-IGAP
2085 IEX=0
DO 2086 I=1, IMAX
IPLUSG=I+IGAP
IF(QNO(I) .LE. QNO(IPLUSG)) GOTO 2086
ISAVE=QNO(I)
FSPECW=QSPEC(I)
QNO(I)=QNO(IPLUSG)
QSPEC(I)=QSPEC(IPLUSG)
QNO(IPLUSG)=ISAVE
QSPEC(IPLUSG)=FSPECW
IEX=IEX+1
2485 CONTINUE
IF(IEX .GT. Z) GOTO 2085
GOTO 2081

C MAIN RESTORE LOOP
C
2100 CONTINUE
IQ=IQ+1
N1=N2
N2=QNO(IQ)
NSEP=N2-N1
IF(1FILE=1FILE+NSEP
NFILE=NSEP*4

C VIND FOWARD TO CORRECT FILE.
C
IF(NFILE. GT. 0) CALL TPHAND(5, 1DRV, ISTAT, NFILE, ILEN, IFILE, , )
READ HEADER BLOCK AND CHECK IF FOUND CORRECT FILE
C
2231 CALL TPHAND(2, 1DRV, ISTAT, , ILEN, IFILE, HBLK, 20)
2232 IF(NBUF .NE. QSPEC(IQ)) GOTO 2080
2254 CALL TPHAND(5, 1DRV, ISTAT, 1, ILEN, IFILE, , )
2255 FSPECW=NBUF
2256 ISIZE=SBUF+16

C READ ONTO TEMPORARY FILE
C
2257 IF(IENTER(IRD, FSBUF, ISIZE). LT. 0) GOTO 9993
CALL THAND(1, IDRV, ISTAT, ILEN, IFILE, )
CALL THAND(5, IDRV, ISTAT, ILEN, IFILE, )
TYPE R$ASC(12, FSPECW, FBUF)
ACCEPT 2020, FBUF
CALL IRAD59(12, FBUF, FSPECW)
IF (ENTER(ISTR, FSPECW, SBUF) .LT. 0) GOTO 9003
TRANSFER TO APPROPRIATE PERMANENT FILE
IBLK = 0
DO 2101 I = 1, SBUF
IF (IREADW(256, BUF, IBLK, IRD) .LT. 0) GOTO 9004
IF (IWRITE(256, BUF, IBLK, ITR) .LT. 0) GOTO 9005
IBLK = IBLK + 1
2101 CONTINUE
CALL CLOSE(IRD)
CALL CLOSE(ITR)
IQNUM = IQNUM - 1
N2 = N2 + 1
IF (IQNUM .NE. 0) GOTO 2100
STOP 'NORMAL TERMINATION'
WRONG FILE FOUND
CALL THAND(7, IDRV, ISTAT, , IFILE, )
CALL R$ASC(12, NSPECW, FBUF)
TYPE 2810, FBUF
2810 FORMAT(' FILE FOUND ON TAPE=', 3A4)
CALL R$ASC(12, O$SPEC(IO), FBUF)
TYPE 2820, FBUF
2820 FORMAT(' FILE REQUIRED=', 3A4)
STOP 'WRONG FILE FOUND FOR RESTORE'
FILE NOT FOUND
CALL FORMAT(' FILE NOT FOUND')
GOTO 1
CODE FOR STOP COMMAND
IF (IQNUM .NE. 0) GOTO 2000
CALL THAND(7, IDRV, ISTAT, , IFILE, )
CALL CLOSE(IRD)
CALL CLOSE(ITR)
STOP 'NORMAL TERMINATION'
CODE FOR DIRECTORY
IF (.NOT. EOT) CALL FSERCH (EOTCOD, IDRV, EOT)
EOT = .TRUE.
TYPE 4070
4070 FORMAT (DIR ON TERMINAL Y/N: ', S)
ACCEPT 4060, ANS
4080 FORMAT (A1)
LUNIT = 6
IF (.NOT. ANS.EQ. 'YES') LUNIT = 7
4010 WRITE (LUNIT, 4020) TPNUM, IDRV
4020 FORMAT (I TAPE NO = ', A8, ' DRIVE NO', I3, ' FILE DIRECTORY')
WRITE (LUNIT, 4030)
4030 FORMAT (FILE NAME ', 5X, ' DATE SAVED ', 5X, ' VERSION ',
%5X, ' SIZE')
DO 4040 I = 1, IFILE
CALL RSBASC (12, FSPEC (I), FBUF)
4040 WRITE (LUNIT, 4050) FBUF, DATE (I), VNUM (I), SIZE (I)
4050 FORMAT (1X, 'TAPE NO = ', AS, ' DRIVE NO ', I3, ' FILE',
'DIRECTORY')
WRITE (LUNIT, 4060) FILE
4060 FORMAT (' TOTAL NUMBER OF FILES ON TAPE = ', I3)
GOTO 1

ERROR FINISHES
3901 NBUF = EOTCOD
3902 ILEN = 1
3903 CALL TPHand (4, IDRV, ISTAT, , ILEN, IFILE, HBLK, 20)
3904 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3905 CALL CLOSEC (IRD)
3906 CALL CLOSEC (IWR)
3907 CALL CLOSEC (ITR)
3908 STOP 'LOOKUP ERROR'
3909 STOP 'LOOKUP ERROR'
3912 NBUF = EOTCOD
3913 ILEN = I
3914 CALL TPHand (4, IDRV, ISTAT, , ILEN, IFILE, HBLK, 20)
3915 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3916 CALL CLOSEC (IRD)
3917 CALL CLOSEC (IWR)
3918 CALL CLOSEC (ITR)
3919 STOP 'READ ERR FOR SAVE'
3920 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3921 CALL CLOSEC (IRD)
3922 CALL CLOSEC (IWR)
3923 CALL CLOSEC (ITR)
3924 STOP 'READ ERR FOR SAVE'
3925 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3926 CALL CLOSEC (IRD)
3927 CALL CLOSEC (IWR)
3928 CALL CLOSEC (ITR)
3929 STOP 'READ ERR FOR SAVE'
3930 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3931 CALL CLOSEC (IRD)
3932 CALL CLOSEC (IWR)
3933 CALL CLOSEC (ITR)
3934 STOP 'READ ERR FOR RESTORE'
3935 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3936 CALL CLOSEC (IRD)
3937 CALL CLOSEC (IWR)
3938 CALL CLOSEC (ITR)
3939 STOP 'READ ERR FOR RESTORE'
3940 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3941 CALL CLOSEC (IRD)
3942 CALL CLOSEC (IWR)
3943 CALL CLOSEC (ITR)
3944 STOP 'READ ERR ON RESTORE TRANSFER'
3945 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3946 CALL CLOSEC (IRD)
3947 CALL CLOSEC (IWR)
3948 CALL CLOSEC (ITR)
3949 STOP 'READ ERR ON RESTORE TRANSFER'
3950 CALL TPHand (7, IDRV, ISTAT, , IFILE, , )
3951 CALL CLOSEC (IRD)
3952 CALL CLOSEC (IWR)
3953 CALL CLOSEC (ITR)
3954 STOP 'WRITE ERR ON RESTORE TRANSFER'

END
SUBROUTINE FSERCH(FNAME,IDRV,EOT)

C SUBROUTINE WHICH SEARCHES FORWARD TO SPECIFIED
C FILE NAME ON THE TAPE

REAL*8 FNAME,FSPEC(1000),DATE(1000),NBUF,dbuf,EOTCOD
INTEGER*2 SIZE(1000),VNO(1000),VBUF,SBUF
LOGICAL*1 HBLK(20),IDRV,EOT,ISTAT
COMMON/SERCH/ FSPEC,DATE,VNO,SIZE,HBLK,IFILE
EQUIVALENCE(HBLK(1),NBUF),(HBLK(9),DBUF),(HBLK(17),VBUF),
%(HBLK(19),SBUF)
DATA EOTCOD/12REOTEOTEOTEOT/
13 CALL TPHAND(2,IDRV,ISTAT,.ILEN,IFILE,HBLK,20)
IF(NBUF.EQ.EOTCOD)GOTO 15

C STICK HEADER INFO IN DIRECTORY

IFILE=IFILE+1
FSPEC(IFILE)=NBUF
DATE(IFILE)=DBUF
VNO(IFILE)=VBUF
SIZE(IFILE)=SBUF
IF(NBUF.EQ.FNAME)GOTO 20

C POSITION TAPE NEXT TO NEXT HEADER

CALL TPHAND(5,IDRV,ISTAT,.ILEN,IFILE,.)
GOTO 10

C REWIND TO BEGINNING OF FOUND HEADER

EOT=.TRUE.

CALL TPHAND(6,IDRV,ISTAT,1,.ILEN,IFILE,.)
RETURN
END
SUBROUTINE TPHAND(ICOM, IDRV, ISTAT, ITLEN, ILEN, IFNUM, BUF, NBYTE)

TAPE HANDLING SUBROUTINE

ICOM IS THE COMMAND SIGNAL

1 = READ FROM TAPE TO DISC

2 = READ FROM TAPE TO MEMORY

3 = WRITE TO TAPE FROM DISC

4 = WRITE TO TAPE FROM MEMORY

5 = WIND FORWARD

6 = WIND REVERSE

7 = REWIND TO START

IDRV IS THE DRIVE BEING USED

ISTAT IS THE STATUS ON RETURN

ITLEN IS THE NO OF TAPE FILES TO MOVE PAST

ILEN IS THE BLOCK LENGTH OF A FILE READ OR WRITTEN

BUF IS THE MEMORY AREA USED BY TWRIT AND TREAD

NBYTE IS THE SIZE OF BUF

INTEGER*2 MASK(8), ESTAT

LOGICAL*1 ISTAT, COM(4), SDSCOM(8), IDRV, ITLEN, ECOM(4),

IFLEN, ESTAT, BUF(1)

DATA MASK//1, 2, 4, 10, 20, 40, 100, 200//

DATA SDSCOM//0, 1, 2, 3, 4, 5, 6, 7//

ITRV = 0

GOTO (1000, 2000, 3000, 4000, 5000, 6000, 7000) ICOM

SECTION CONTROLLING A READ

CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED

1000 ITRY = ITRY + 1

SET UP COMMAND FOR READ

COM(1) = SDSCOM(4)

COM(2) = 1

COM(3) = IDRV

COM(4) = -1

CALL SDSI0(COM, ISTAT, ITLEN, ILEN)

IF(ISTAT.EQ.0) RETURN

ERROR DETECTED ON READ

ISTAT = ISTAT

GOTO 40

READ FROM TAPE TO MEMORY

NBUFF = NBYTE

CALL TREAD(BUF, NBUF, ISTAT, IDRV)

IF(ISTAT.EQ.0) RETURN
ISTAT1 = ISTAT
GOTO 40

IF SHORT RECORD FOUND REREAD TAPE

50 ITMP = ISTAT1 .AND. MASK(6)
IF(ITMP .NE. 0) GOTO (1000, 2000) ICOM
ISTAT1 = ISTAT1 .AND. MASK(2)
IF(ITMP .EQ. 0) RETURN

IF CRC ERROR FOUND REWIND TAPE AND RETRY

TYPE 2010, INUM
FORMAT(' FILE NO , I4 , CRC ERROR REWINDING')
IF(ITRY .GE. 2) GOTO 130
ECOM(1) = SDSCOM(6)
ECOM(2) = 1
ECOM(3) = IDR1
ECOM(4) = 0
CALL SDS10(ECOM, ISTAT, )
GOTO (1000, 2000) ICOM

WRITE SECTION

WRITE ERROR DETECTED

ISTAT1 = ISTAT
GOTO 40

MEMORY TO TAPE WRITE

CONTINUE
NB11 = NBYT
IFLEN = (ILEN + 3) / 4
IF(IFLEN .LT. 2) ILEN = 2
IPAD = (IFLEN * 2048) - NBUF
CALL TWRIT(BUF, NBUF, ISTAT, IPAD, IFL1EN, IDR1)
IF((ISTAT .EQ. 0)) RETURN
ISTAT1 = ISTAT
GOTO 40

ERROR RETURN POSITION

70 ITMP = ISTAT1 .AND. MASK(6)
REPORT AND RETRY

TYPE 2028,IFNUM

FORMAT(' FILE NO ',I4,' WRITE CRC ERR ')

RETURN

C WIND FOWARD ONE FILE

IF(icom.eq.5)com(1)=SDSCOM(5)
IF(icom.eq.6)com(1)=SDSCOM(6)
IF(icom.eq.7)com(1)=SDSCOM(2)
com(2)=ITLEN
com(3)=IDRV
com(4)=

CALL SDS18CECOM,ESTAT,

C CLEAR IRRELEVANT BITS FROM ERROR BYTE

ISTAT=ISTAT.AND..NOT.MASK(6)
ISTAT=ISTAT.AND..NOT.MASK(2)
IF(ISTAT.EQ.6)RETURN
ISTAT=ISTAT

GOTO 40

RETURN

CLEAR BUSY SECTION...AFTER CLEARING BOT FLAG

FILE NO=' ,I4)
ISTAT=ISTAT.AND..NOT.MASK(4)
ITMP=ISTAT.AND.MASK(5)
IF(ITMP.EQ.6)GOTO 80
ECOM(1)=SDSCOM(1)
ECOM(2)=
ECOM(3)=IDRV
ECOM(4)=
CALL SDS18CECOM,ESTAT,

HAVING EXAMINED STATUS IF TAPE STILL BUSY, LOOP AGAIN, IF NOT TRY COMMAND AGAIN

ESTAT=ESTAT
ITMP=ESTAT.AND.MASK(5)
IF(ITMP.NE.6)GOTO 90
ITMP=ESTAT.AND.MASK(4)
IF(ITMP.EQ.6)GOTO 90

RETURN

GOTO 35
GOTO (1000, 2000, 3000, 4000, 5000, 5000, 5000) ICOM

C TAPE OFFLINE

60 IF (ITMP .EQ. 0) GOTO 100

10 TYPE 1001, IDR

101 FORMAT(' ERROR EXIT RETURN')

102 HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED

103 ECOM(1) = SDSICOM(1)

104 ECOM(2) = IDR

105 ECOM(3) = IDR

106 CALL SDSICOM(ECOM, ESTAT, )

107 ECOM(3) = IDR

108 IF (ITMP .EQ. 0) GOTO 100

109 GOTO (1000, 2000, 3000, 4000, 5000, 5000, 5000) ICOM

C EOT

115 IF (ITMP .EQ. 0) GOTO 120

116 TYPE 1002, IDR

117 FORMAT(' ERROR EXIT RETURN')

118 IDR = 3

119 RETURN

120 GOTO (50, 50, 70, 70, 35, 35) ICOM

C ERROR EXIT RETURN

130 ISTAT = -1

131 RETURN

132 END
**Tape Handling Utility:—MPTAPH**

This is a fully interactive program allowing the user to perform any tape function from the keyboard, giving the user total control of all tape functions. The command sequence is completely prompted by the program, with the command menu being presented every time.

**Tape to Disc Transfer Utility:— MPTPDK**

This program allows files to be read from tape to disc and seismic channels to be selected for putting into a trace sequential file, for migration or plotting.

Input file....DK2:MPTPDK.DAT

Log file.....DK2:MPTPDK.LOG

**Input Parameters**

\[\text{READ}(1,1000)\text{NFILIN,NBLKS,IBLKST,TPDRR}\]

1000 FORMAT(4I5)

NFILIN...Number of files to read from tape
NBLKS....Number of blocks to select from each file
IBLKST....Starting block for selection
TPDRR....Tape drive
READ(1,1001)FSPECW

1001 FORMAT(3A4)

FSPECW...output file

READ(1,1001)FSPECR

FSPECR...Temporary file for tape to disc read
REAL*8 FSPEC

REAL*4 DEVNAM,FMBUF(3)

INTEGER*4 COM(8),LUN,HBLK(I),FSPEC,ITYP

LOGICAL*1 SDS.COM(4),COM(8),LUN,HBLK(I),FNUM,ITYP

INTEGER*2 STATUS,TLEN,FLEN

DATA COM("0","1","2","3","4","5","6","7")

DATA DEVNAM/3RRK/

IF(IFETCH(DEVNAM).NE.0)STOP'FETCH ERROR'

IF(ICDFN(30).NE.0)STOP'CDFN ERROR'

CONTINUE

1 CONTINUE

LOGICAL*1 SDSCOM(4),COM(8),LUN,HBLK(I),FSPEC,ITYP

INTEGER*2 STATUS,TLEN,FLEN

DATA COM("0","1","2","3","4","5","6","7")

DATA DEVNAM/3RRK/

IF(IFETCH(DEVNAM).NE.0)STOP'FETCH ERROR'

IF(ICDFN(30).NE.0)STOP'CDFN ERROR'

CONTINUE

1 CONTINUE

READ

IFIENT=ENTER(20,FSPEC,40)

IF(IENT.LT.0)STOP'EXECUTION TERMINATED'

TYPE 70

FORMAT('ENTER THE FILE SPEC, NO COLONS OR DOTS',/,X'BUT WITH IMPLIED BLANKS INCLUDED:','$)

ACCEPT 40,FMBUF

40 FORMAT(3A4)

CALL IRADSB(H2,FMBUF,FSPEC)

IF(I2G0 TO 60)

WRITE

FORMAT('FILE NO=',13)

ACCEPT 50,LUN

TYPE 71

71 FORMAT('ENTER 0 FOR FIELD TAPE 1 FOR INTERNAL:','$)

ACCEPT 20,IYTP

30 FORMAT('ENTER ERROR IENT=',13)

IF(IENT.LT.0)STOP

TYPE 70

FORMAT('ENTER TAPE UNIT TO BE READ FROM:','$)

ACCEPT 20,LUN

TYPE 71

71 FORMAT('ENTER 0 FOR FIELD TAPE 1 FOR INTERNAL:','$)

ACCEPT 20,IYTP

30 FORMAT('ENTER ERROR IENT=',13)

IF(IENT.LT.0)STOP

TYPE 70

FORMAT('ENTER TAPE UNIT TO BE READ FROM:','$)

ACCEPT 20,LUN

TYPE 71

71 FORMAT('ENTER 0 FOR FIELD TAPE 1 FOR INTERNAL:','$)

ACCEPT 20,IYTP

30 FORMAT('ENTER ERROR IENT=',13)

IF(IENT.LT.0)STOP

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

GOTO 10

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)

40 FORMAT('STATUS EQUALS...','13)

IF(STATUS.EQ.91)GOTO 91

DO 90 S=1,10

90 FORMAT('READ HBLK ERROR'

CALL SDS1.0(20,SDS.COM,STATUS,TLEN,FLEN)
DATA CONSTANTS = ',5A1, /
SAMPLING PERIOD = ',A1, /
DATA TIME LENGTH = ',12)

91 CALL CLOSEC(20)
GO TO 1

WRITE

56 FLEN=LOOKUP(21,FSPEC)
57 IF(FLEN.LT.0)STOP'LOOKUP ERROR'
59 FLEN=(FLEN+3)/4
52 TYPE 35,STATUS,FLEN
52 TYPE 95
52 95 FORMAT( ' ENTER THE TAPE UNIT TO BE WRITTEN TO: ',S)
023 ACCEPT 28,LUN
024 SDSCOM(1)=COM(7)
025 SDSCOM(2)=FLEN
026 SDSCOM(3)=LUN
027 SDSCOM(4)=1
029 CALL SDS10(SDSCOM,STATUS,FLEN)
030 TYPE 35,STATUS,FLEN
032 CALL CLOSEC(21)
037 GOTO 1

WIND

65 FNUM=0
67 LNUM=0
671 TYPE 188
675 188 FORMAT( 'UTILITY COMMAND TABLE: ',/,
% '----------------------',/,
% '1: STATUS ',/,
% '2: REWIND',/,
% '3: REWIND OFF LINE',/,
% '5: FORWARD WIND N FILES',/,
% '6: REVERSE WIND N FILES',/,
% '8: RETURN PDP-BE TO OS8',/,
% 'ENTER YOUR OPTION: ',S)
77 ACCEPT 28,NOPT
77 IF(NOPT.EQ.8)GOTO 118
979 TYPE 128
083 128 FORMAT( ' ENTER TAPE UNIT NO: ',S)
081 ACCEPT 28,LUN
082 IF(NOPT.EQ.5.OR.NOPT.EQ.6)TYPE 138
084 138 FORMAT( ' ENTER NO OF FILES TO BE WOUND PAST: ',S)
088 IF(NOPT.EQ.5.OR.NOPT.EQ.6)ACCEPT 28,FNUM
093 118 SDSCOM(1)=COM(NOPT)
093 SDSCOM(2)=FNUM
093 SDSCOM(3)=LUN
093 SDSCOM(4)=0
093 CALL SDS10(SDSCOM,STATUS)
133 TYPE 35,STATUS,FLEN
137 GOTO 1
139 END
REAL*8 FSPECW, FSPECW
REAL*4 FBUF(3), SEISM(2640)
LOGICAL*1 TPDRR, ISTAT, ITLEN
DATA DEV/3RRK /
IF(ICDFN(25).NE.0) STOP 'CHAN ERR'
IF(IFETCH(DEV).NE.0) STOP 'FETCH ERR'
CALL ASSIGN(1,'DK2:MPTPDK.DAT',14)
CALL ASSIGN(2,'DK2:MPTPDK.LOG',14)
IRD=20
IRWT=21
READ(1,1000) NFILIN, NBLKS, IBLKST, TPDRR
1000 FORMAT(4I5)
READ(1,1200) FBUF
1200 FORMAT(3A4)
CALL IRAD50(12, FBUF, FSPECW)
READ(1,1200) FBUF
CALL IRAD50(12, FBUF, FSPECW)
NWDS=NBLKS*256
IBLKOT=1
IBLKSZ=NFILIN*NBLKS+1
IF(ENTER(IWRT,FSP ECW, IBLKST).LT.0) STOP 'ENTER ERR'
DO 20 I=1, NFILIN
IFIL=I
IF(ENTER(IRD, FSPECW, 300).LT.0) STOP 'ENT2 ERR'
IF(TPDRR.LE.2) GOTO 30
WRITE(2,1400) IFIL
1400 FORMAT(' FILE NO ', IS, ' EOT ')
CALL CLOSEC(IWRT)
20 STOP 'EOT'
30 CALL TAPRED(-1, TPDRR, ISTAT, ITLEN, IFLEN, IFIL)
   IF(ISTAT.LT.0) WRITE(2,1500) IFIL
1500 FORMAT(' FILE NO ', IS, ' RETRIES FAILED LAST READ USED ')
   IF(TPDRR.GT.2) GOTO 40
CALL TAPRED(0, TPDRR, ISTAT, , IFIL)
40 CALL CLOSEC(IRD)
   IF(LOOKUP(IRD, FSPECW).LT.0) STOP 'LOOKUP ERR'
   IF(IREAD(NWDS, SEISM, IBLKST, IRD).LT.0) STOP 'READ ERR'
   IF(IWRITE(NWDS, SEISM, IBLKST, IRD).LT.0) STOP 'WRITE ERR'
   IBLKOT=IBLKOT+NBLKS
CALL CLOSEC(IRD)
20 CONTINUE
CALL CLOSEC(IWRT)
STOP 'NORMAL TERMINATION'
END
The two pieces of software put onto the IBM have their own manuals describing their operation, mentioned here are their locations and manner of execution.

**MATHSIM**

To use the AP maths library simulator the program is written in the normal manner except any references in AP calls must have the variables defined as INTEGER*2 to be compatible with the pdp.

```
$RUN PROG+GPT9:MTHSIMLIB
```

**AIMS**

Aims is fully documented in its own manual. Shown below is the run command with the file, logical unit assignments which have to be made.

```
$RUN GPT9:AIMS+*PLOTSYS
```

**Logical Units**

5..Input Deck
6..output listing
7,8....Temporary files
9..Plot output
10 to 18...Temporary work files in different jobs