

Durham E-Theses

The design and implementation of the Durham university seismic processing system

Michael John Poulter

How to cite:

Poulter, Michael John (1982) The design and implementation of the Durham university seismic processing system. Doctoral thesis, Durham University.

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 <https://etheses.durham.ac.uk/id/eprint/7778/> 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

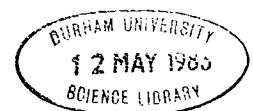
Michael John Poulter

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.



Abstract

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 Asberfy, 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

<u>Chapter 1:</u> Introduction	1
Introduction To Hardware	1
Project Aims	2
Scope of the Work	4
<u>Chapter 2:</u> Seismic Reflection Principles	6
Data Acquisition	7
Recording	9
Processing	11
Summary	18
<u>Chapter 3:</u> 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

Tape Handling	29
Tape Archiving	32
Processing System Tape Subroutines	33
Memory Management	35
Virtual Memory Input/Output	38
Plotting Software	42
Contouring	43
Seismic Displays	44
IBM Software	46
Summary	47
<u>Chapter 4:</u> Seismic Processing Software	48
Overview	48
Overall Design Considerations	49
Data Format	50
Data Input and Output	53
Seismic Processing System	54
Synthetics	55
Demultiplex	57

Sort	63
Pre-Stack Data Analysis	65
Pre-Stack Processing	67
Edit	68
Polarity Reversal	69
Gain Application	69
Muting	71
Frequency Filtering	72
Bandpass Filtering	73
Bandreject Filtering	73
Deconvolution	74
Wiener Spiking Deconvolution	75
Prediction Error Deconvolution	80
Trace Normalisation	82
Pre-Stack Processing Summary	83
Velocity Analysis and Stacking	84
Coherence Velocity Analysis	87
Velocity Analysis Display	89
NMO corrected gathers	90

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
<u>Chapter 5: Using The System</u>	120

Data Flow	121
Jan Mayen Data Processing	122
Jan Mayen Processing	125
Caribbean Arc 1980	126
Caribbean Processing Parameters	131
Summary	132
<u>Chapter 6: Conclusions</u>	133
Future Software Development	133
Demultiplex	134
Data Exchange Format	135
Land Data	136
Other Possible Software Additions	138
Vibroiseis	138
Improved Filtering	139
Amplitudes	141
Summary	141
Hardware Evolution	142
Future Evolution Path	147

Summary	149
<u>References</u>	151
<u>Appendix 1</u>	162
Demultiplex:- MPDMXA	163
Sort :- MPSORT	167
Pre-Stack Processing :- MPPRST	170
Interactive Spectral Analysis Program :- MPFANL	176
Velocity Analysis :- MPVEL	177
NMO corrected gathers :- MPCDP	179
CMP Stacking :- MPSTAK	181
Post-Stack Processing :- MPPOST	185
Post-Stack Trace Mix :- MPTMIX	191
Trace Sequential - Time Slice :- MPSLIC	193
Time Slice to Trace Sequential :- MPUSLC	194
Finite Difference Migration :- MPFD15 and MPFD45	195
Kirchhoff Migration Operator Design :- MPOGEN	197
Kirchhoff Migration :- MPKMIG	199

Stand Alone Interpolation :- ANINT	201
Internal Header Interrogation :- MPHIST	202
Synthetic CMP Gathers :- ANSEI	203
Synthetic Sections :- MPSYNS	205
Velocity Analysis Display :- MPVCON	208
Section Plotting :- MPSPLT	209
Section Plot Background :- MPPBLK	211
Section Plotting Merge and Post Process :- MPMERG	214
Gather Plotting(Small Trace separation) :- MPSPLI	216
Gather Plotting(Large Separation) :- MPGPLI	217
Quick Raster Plot Processor :- MPPROC	218
General Purpose Raster Merge and Output :- MPMERP	219
<u>Appendix 2:</u>	221
Tape Subroutine Descriptions	222
Floating Point Transfers	226
Extended Memory Input/Output Routines	228
Microcode Routines	230
Plotting Subroutines	233

Utilities 235

IBM NUMAC MTS Software 238

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.

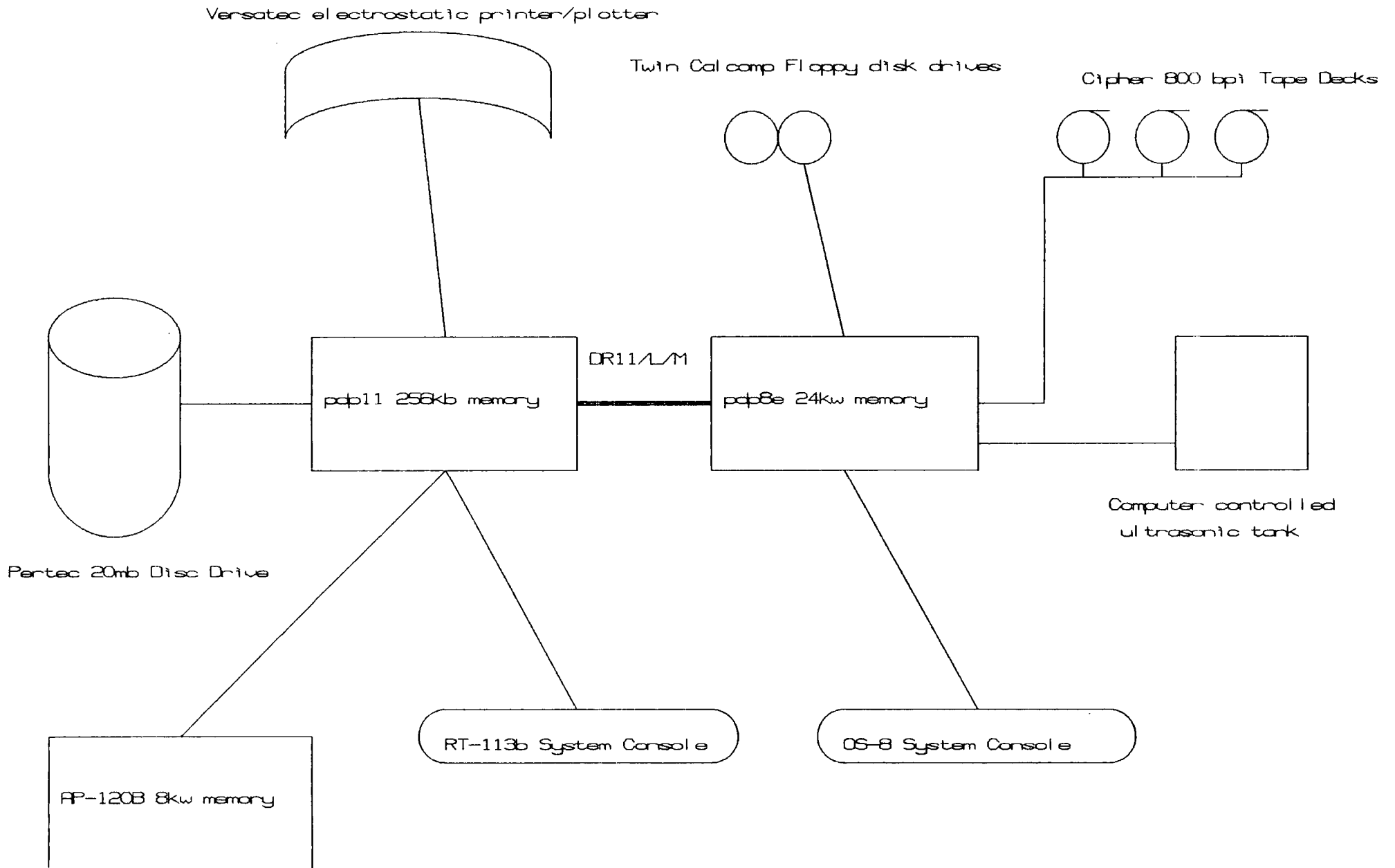


Fig 1.1: Hardware configuration of the Durham University seismic processing system

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.

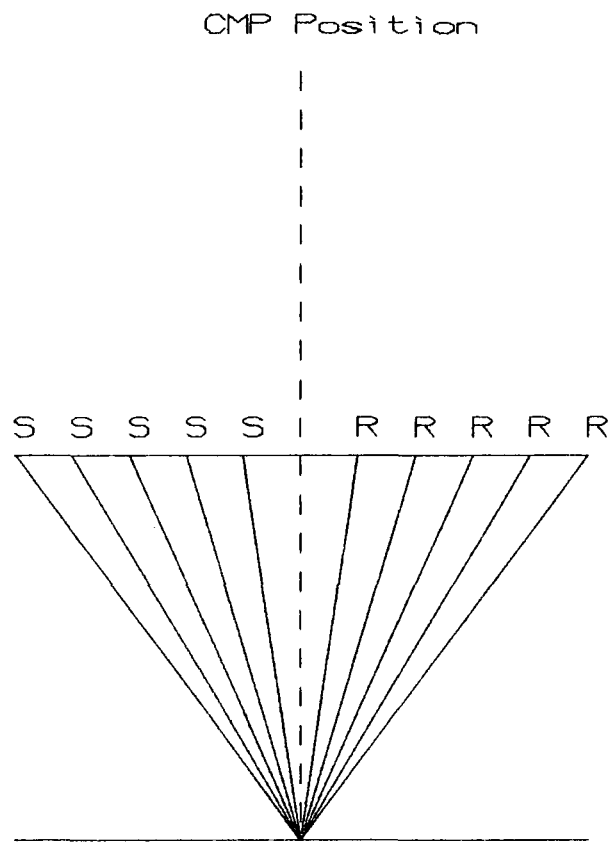


Fig 2.1 : Common Mid-Point Geometry

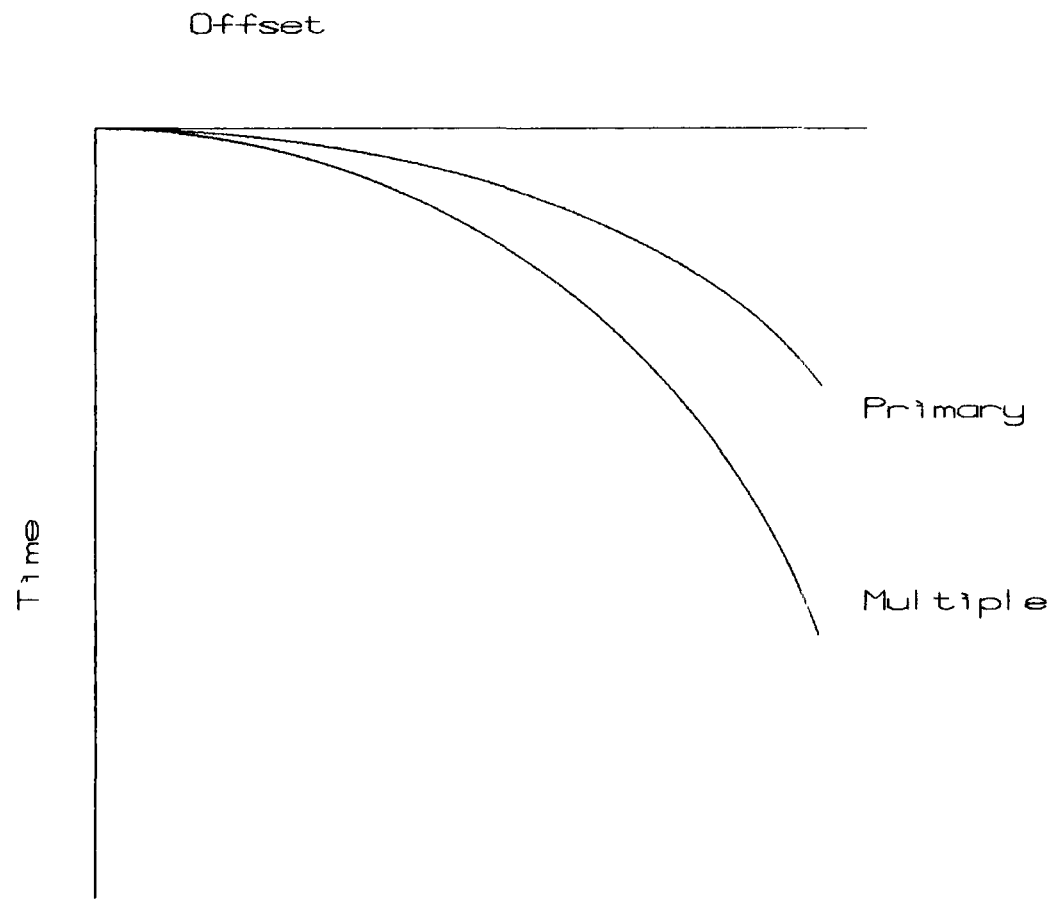


Fig 2.2 : Travel time relation in CMP geometry

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.

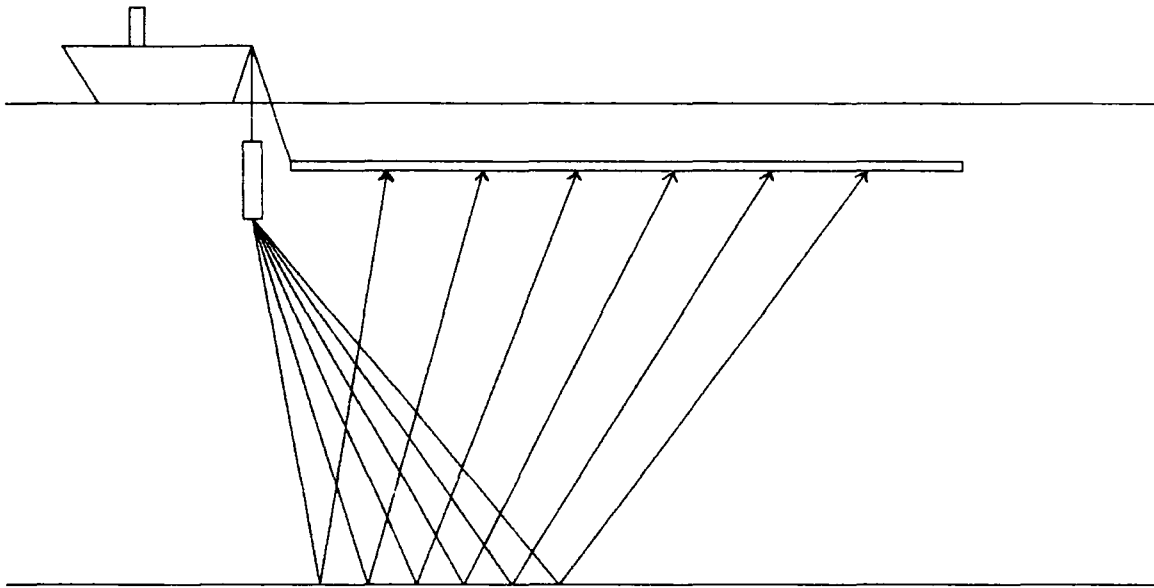


Fig 2.3 : Marine acquisition configuration

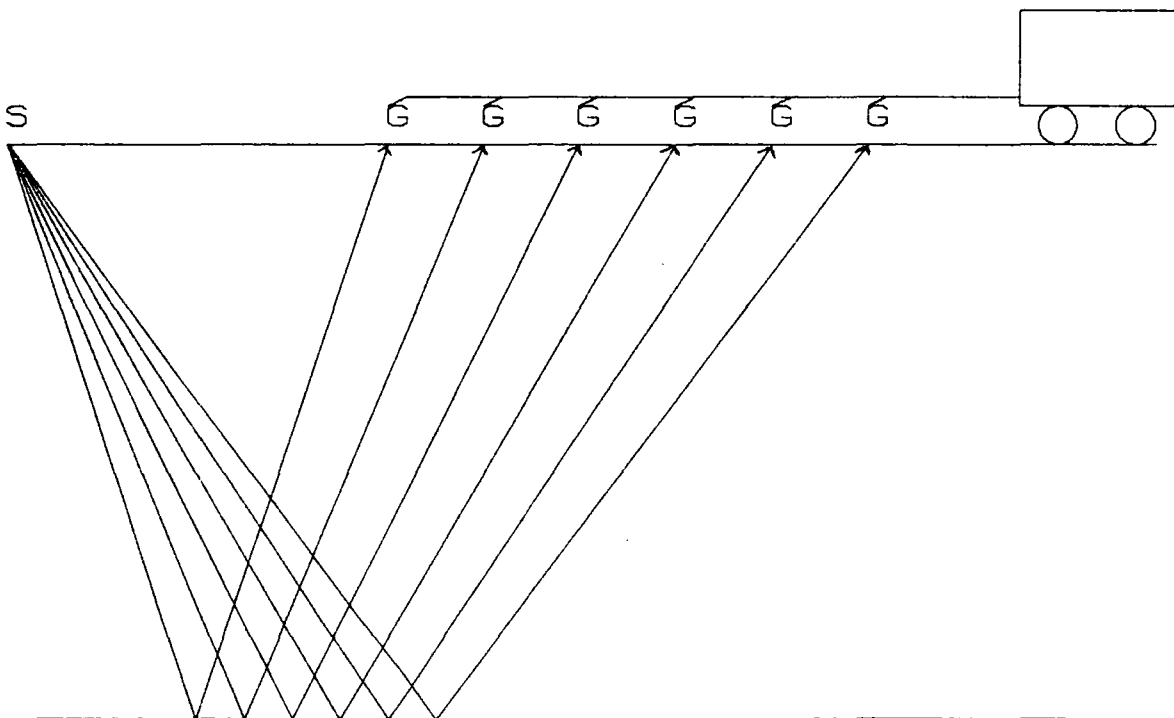


Fig 2.4 : Land acquisition configuration

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:-

$$dV = 0.5dX/dS \text{ where}$$

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

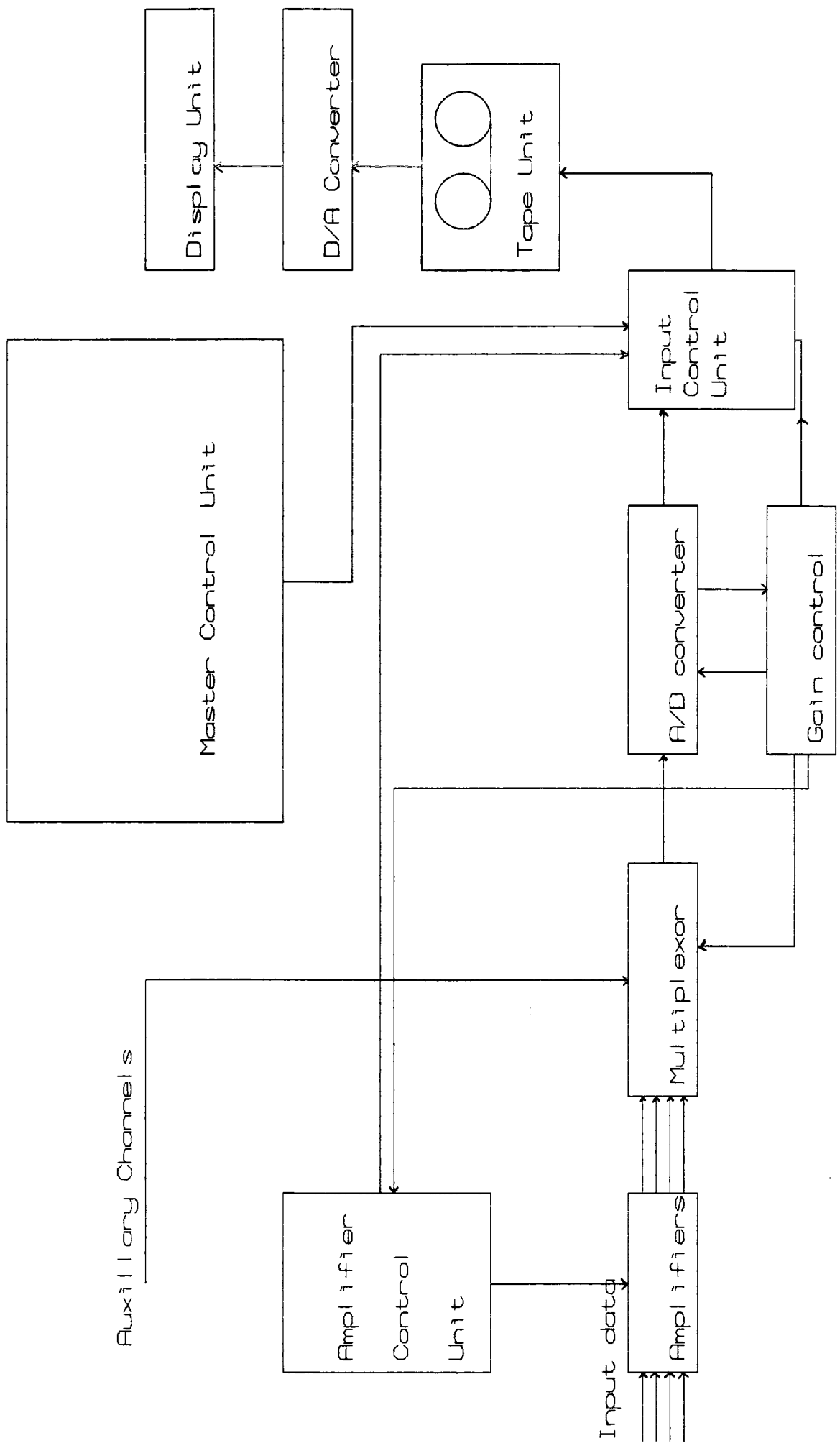
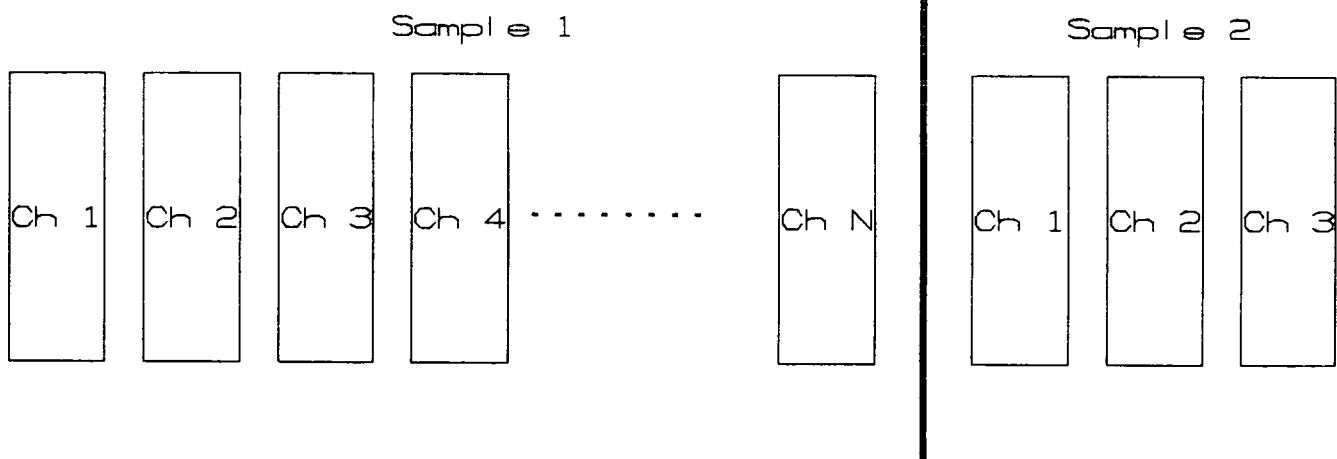


Fig 2.5 : Block diagram of a seismic data acquisition system

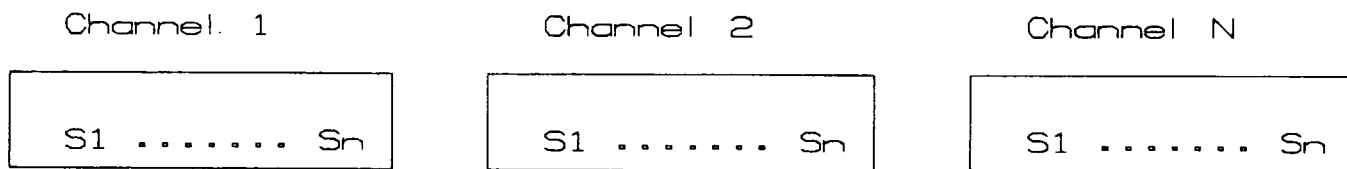
bit integer value with an associated 4 bits of gain information, although instantaneous floating point values are generated by some systems. This output is written to 9 track digital tape in a multiplexed order. The format used to write the data to tape usually adheres to one of the accepted tape formats 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 ~~file~~ ^{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

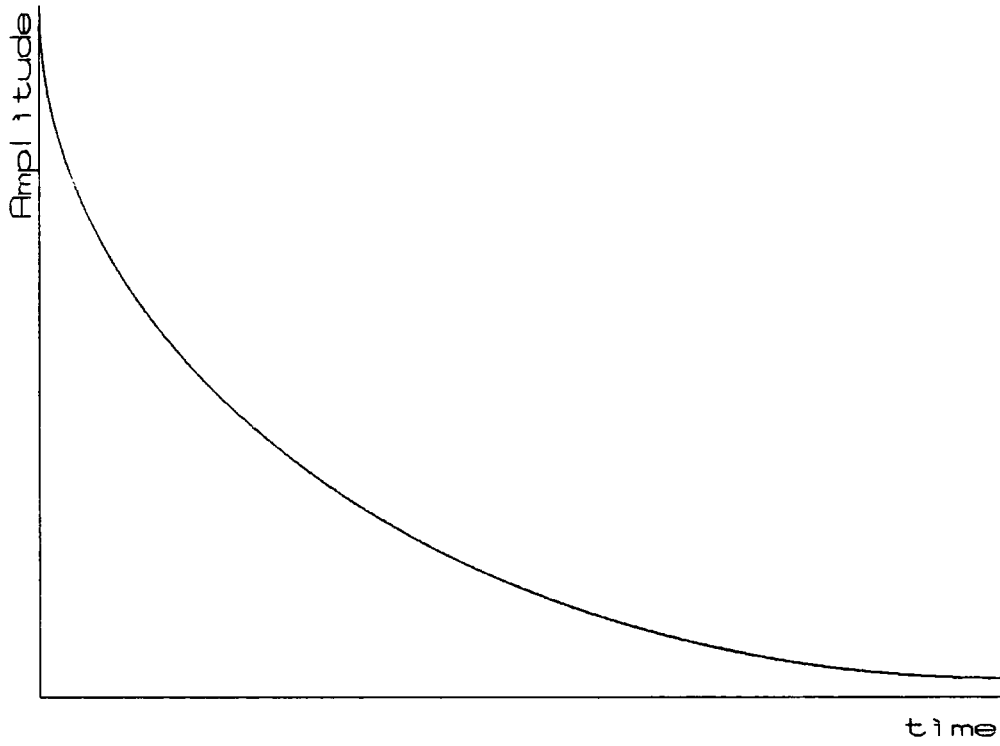
Fig 2.6 : Data organisations

Migration.....Image data to correct location

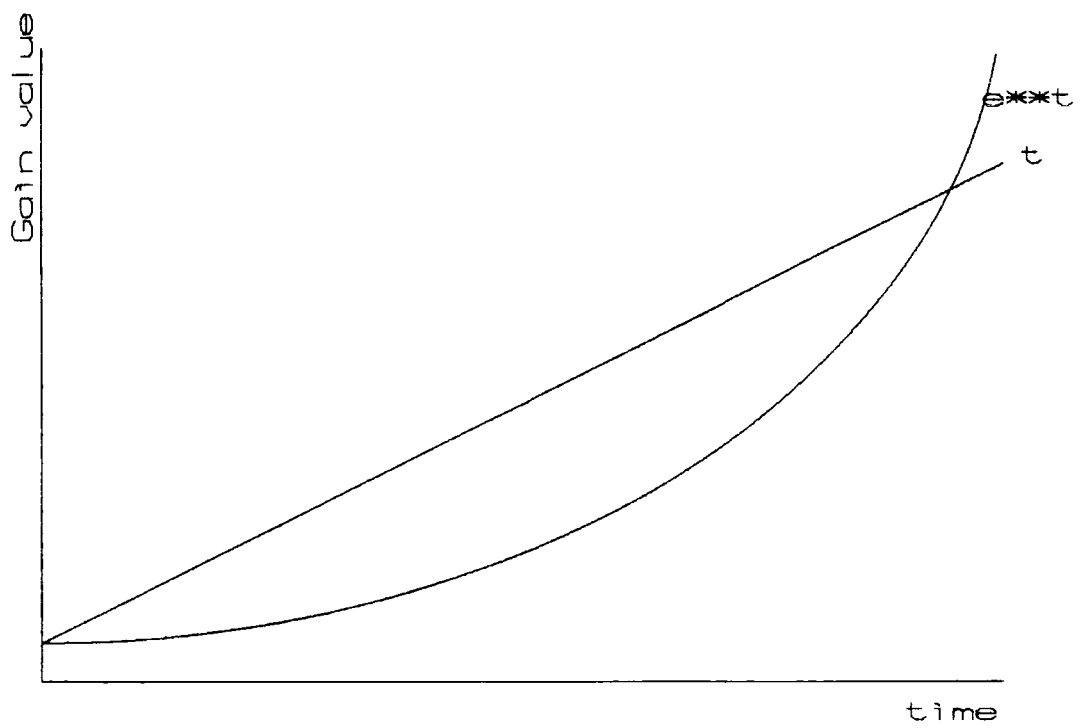
Processing

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 been found to work well in practice.



Data amplitude



Amplitude Recovery curves

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.

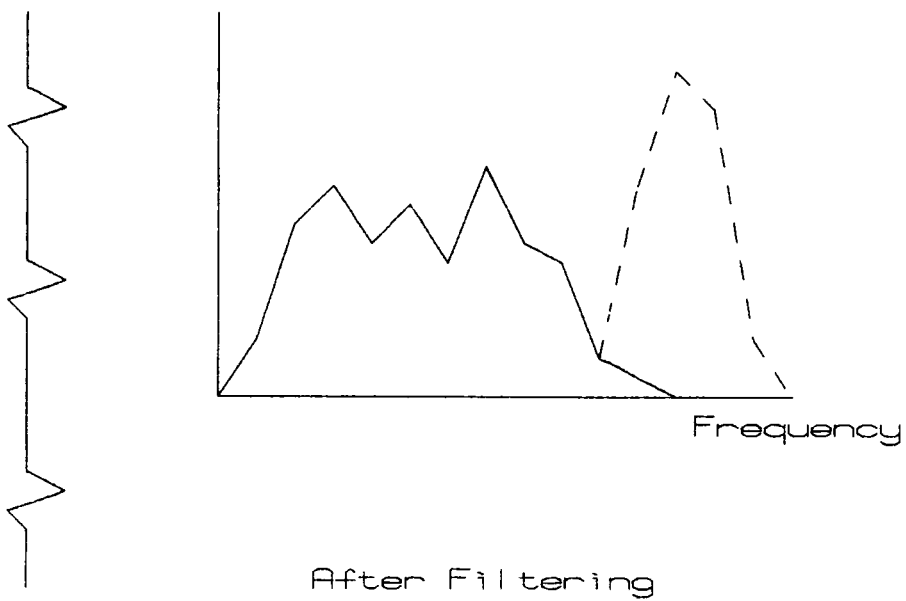
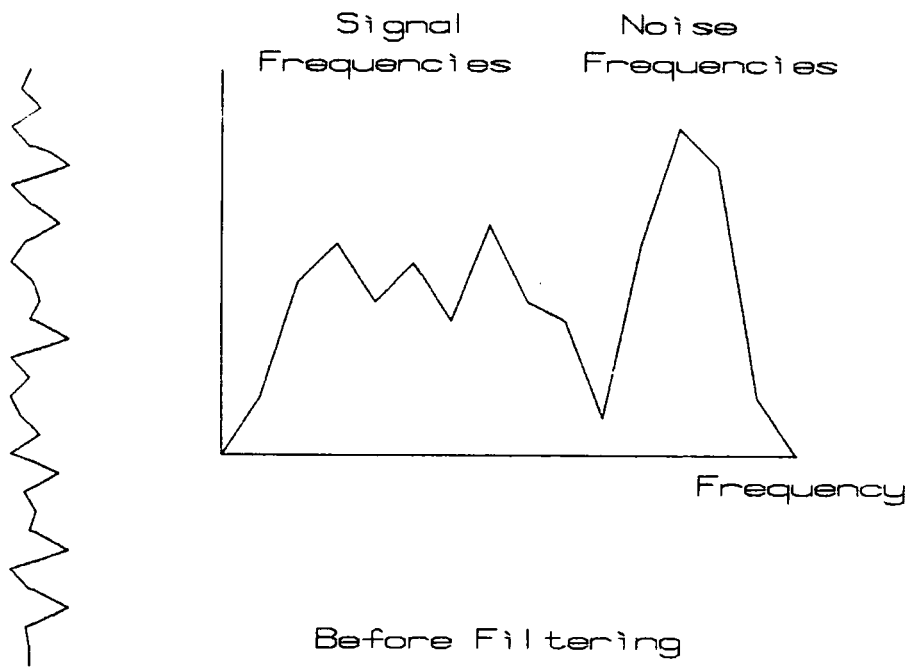
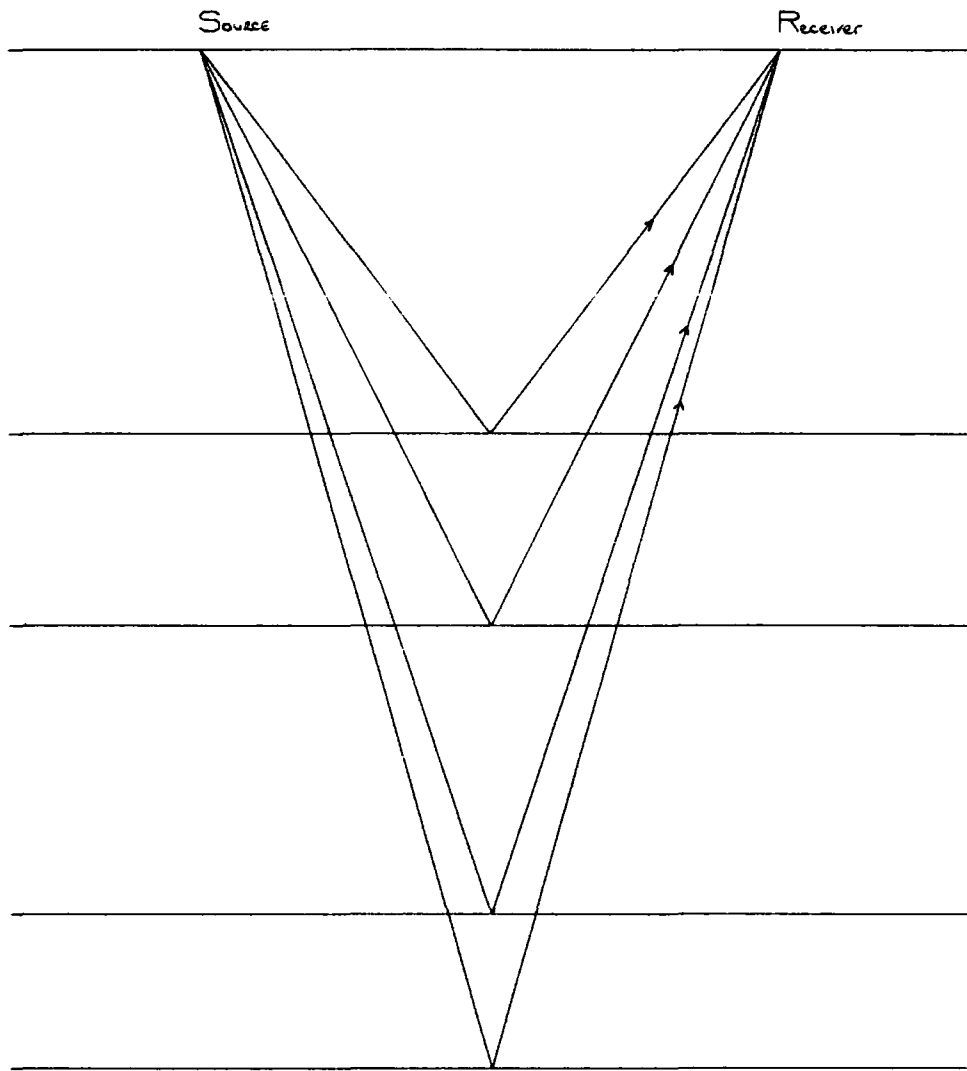


Fig 2.8 : Example of Frequency Filtering

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

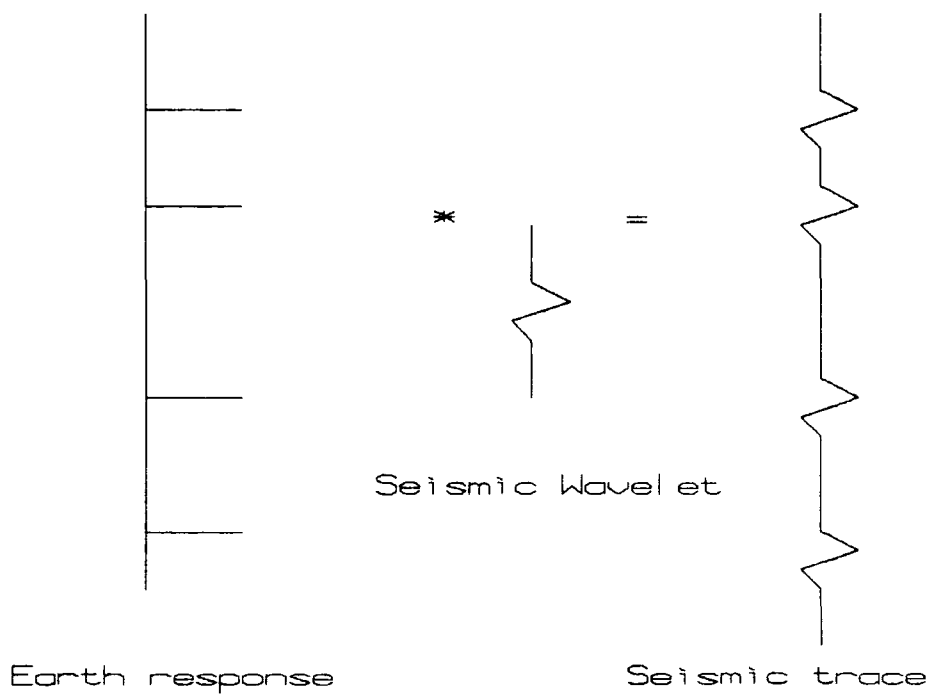


Fig 2.9 : Convolutional model of the seismic trace

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.

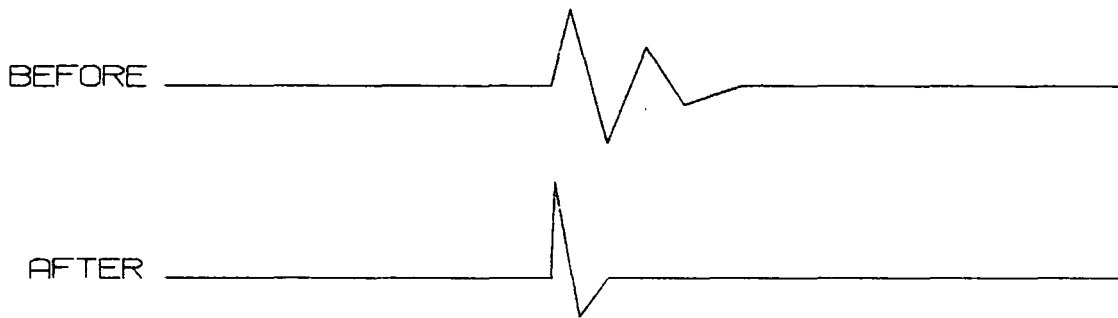
The spiking filter is a special case of a range of filters, known as prediction error filters, which can be derived from the statistics of the data. These filters record the error in a prediction of the trace a certain distance ahead from the statistics of the trace. This leads to predictable, events 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)

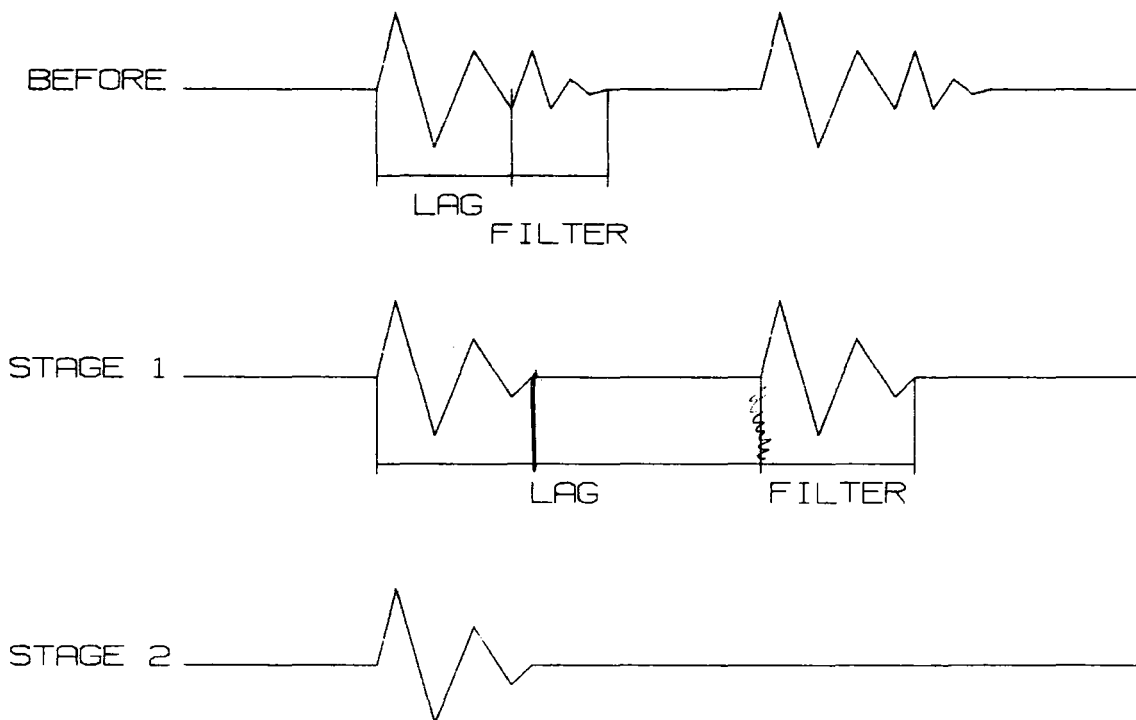
If the prediction distance is set up to be the same as the period of a long period multiple, then the prediction error filter can be used to dereverberate the trace. Also it can be used to compress an airgun wavelet, to improve resolution, by having a prediction distance just less than one wavelength 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.

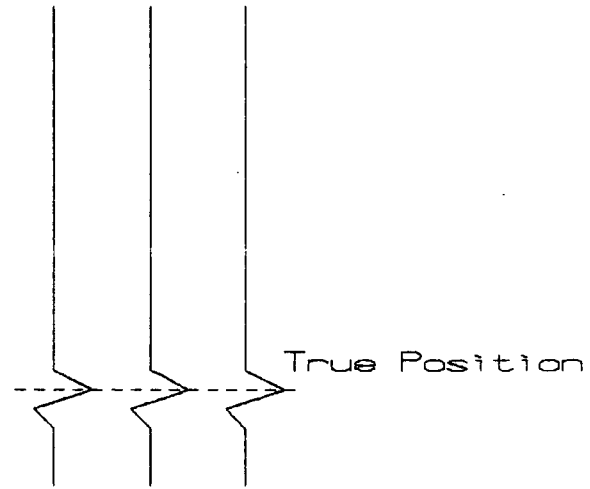
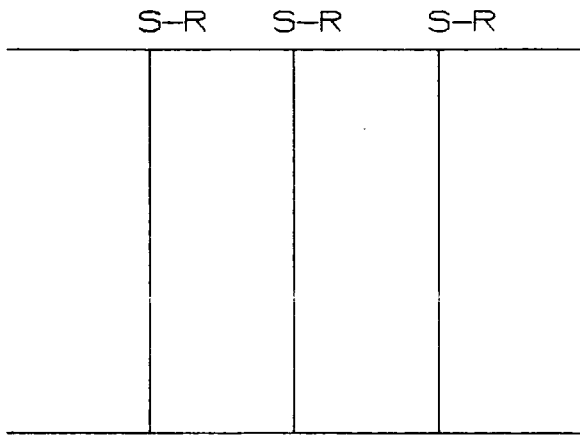
In the case of simple geological structures, where the horizons are near horizontally layered, the processed CMP stacked section is adequate for a geological interpretation. However, if the data are more complicated, a final procedure, migration, is necessary to produce an interpretable section.

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.

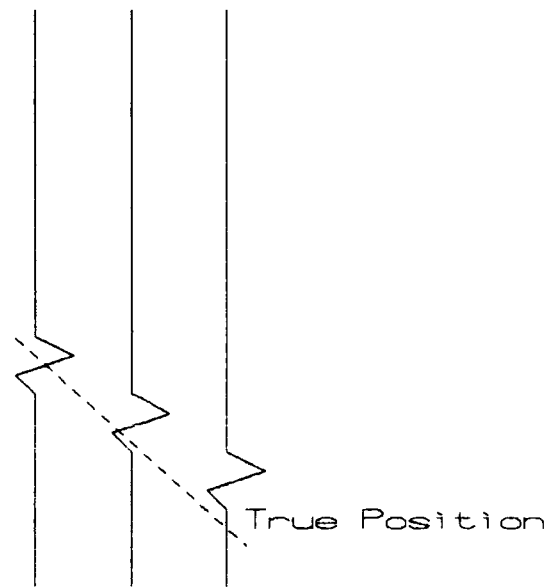
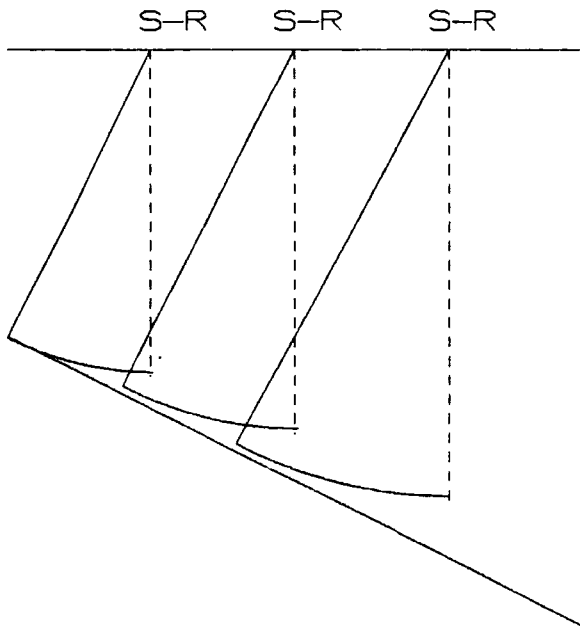
For the purpose of migration the CMP traces are considered as being the recording of the wave field produced with a coincident shot and receiver. In this mode of data collection, upward and downward paths are coincident and so the recording is the same as would be obtained by having a source at the reflector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

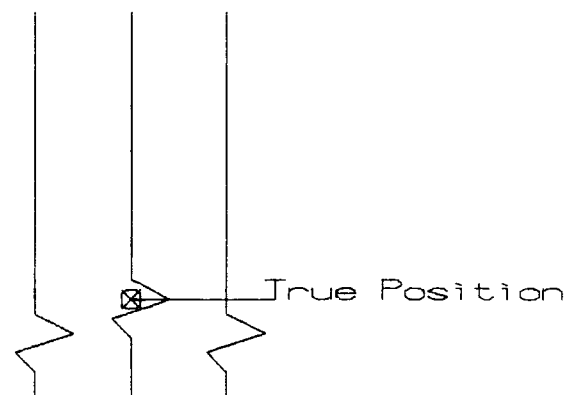
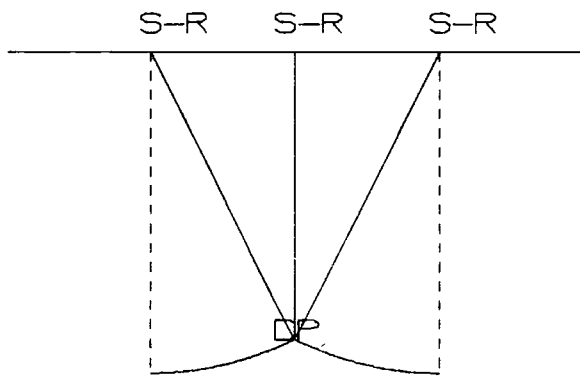
This mathematical reconstruction is performed by solving approximations to the acoustic wave equation. There are three methods of approach which are most frequently used; Kirchhoff



Horizontal Beds



Dipping Beds



Diffracting Point

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 3Hardware and Systems SoftwareHardware

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 64Kbytes, 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 ~~gapped~~^{long-record} 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 AP120~~0~~ 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(APDEBUG). 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 ^{long-recor-d}~~gapless~~ 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:MPDMLXA.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 ^{long-record}~~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 TWRIT 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

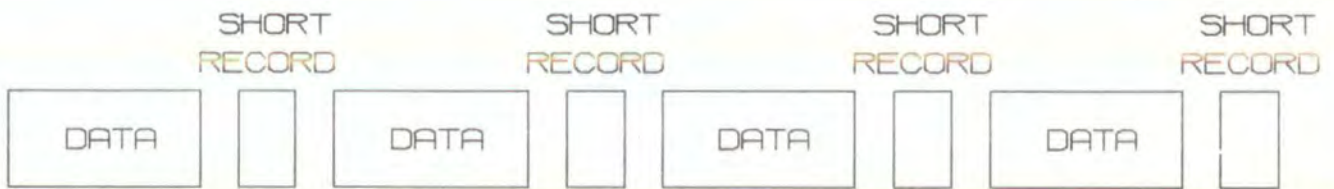
It was realised that the programs in the processing stream would need to have a tape handling capability and that this capability would have to be consistent, from process to process, in its treatment of errors and other processing conditions. Therefore it was decided to produce two Fortran subroutines 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

<u>STATUS</u>	<u>ACTION</u>
BOT	Ignore
BUSY	Loop until not busy then continue
OFFLINE	Write a message to the console Loop until online then continue
EOT	Write a message to the console Set status = -1 and return

READ ERRORS

NORMAL	Return
SHORT RECORD	Read next record
PARITY	Wind back one record and retry 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} ~~virtual~~ 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 ⁿ ~~man~~agement 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 8 ~~4~~ 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. Additionally¹ 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.

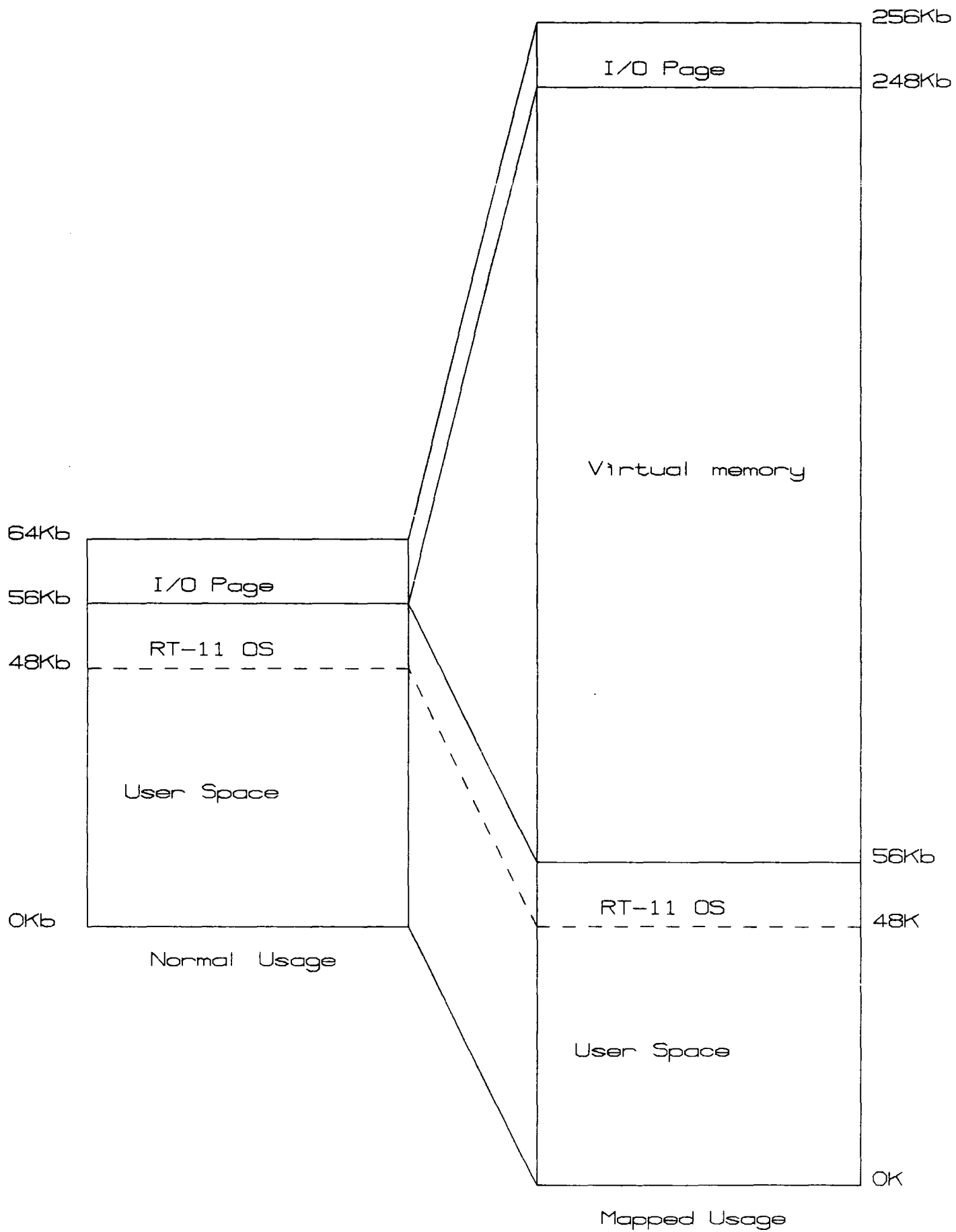


Fig 3.2 : Memory configuration of the pdp 11/34

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 upto 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 longwinded₂ 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

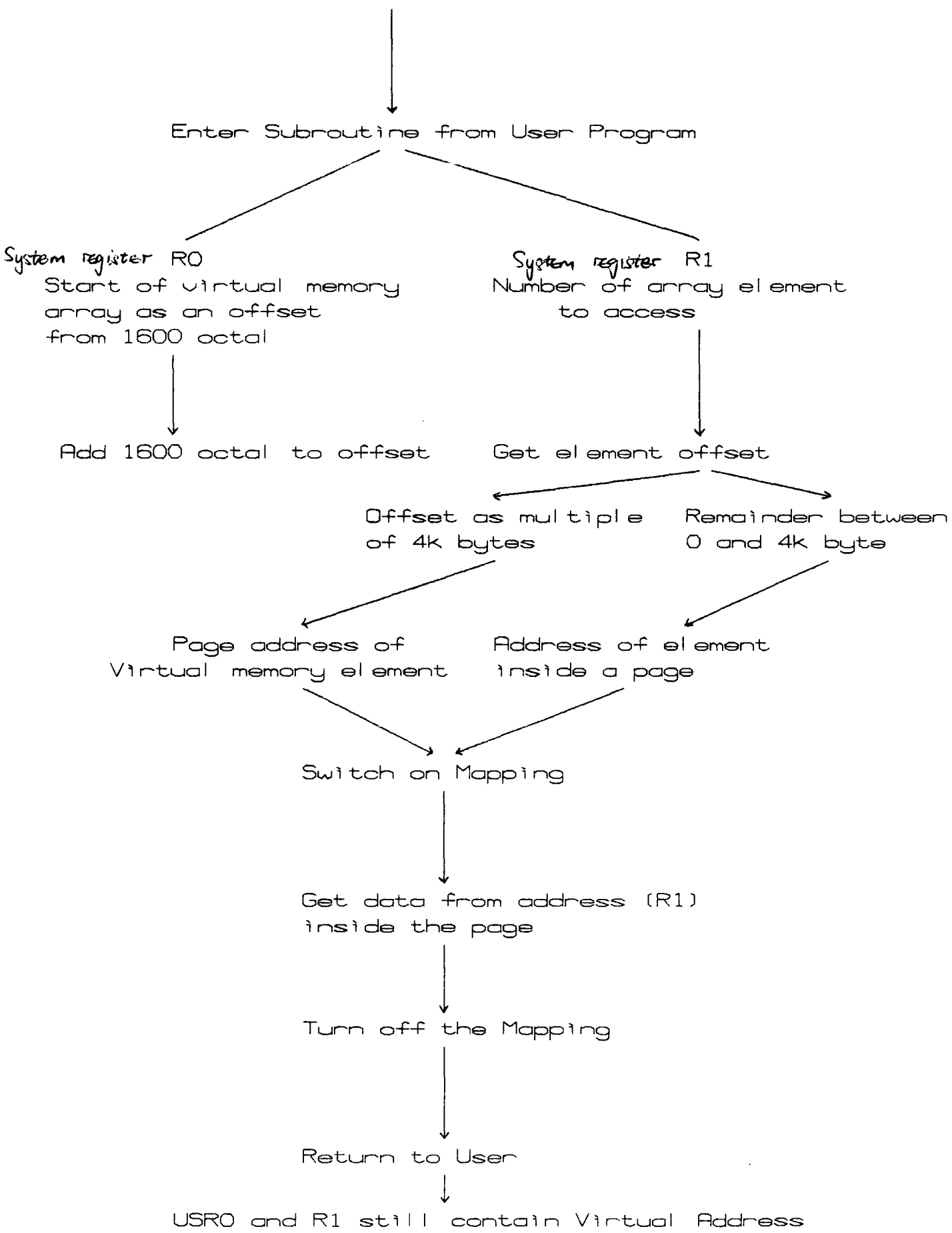


Fig 3.3 : Virtual Memory access in RT-11 Fortran

element.

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 0 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. ^(Northumbrian Universities' Multiple Access Computer) 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_x 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 transferred 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.

IBM Software

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 ^qacquired_^ 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

PLOT OF WAVE THEORY RAYS TO ONE SHOT POINT
KNEE MODEL PHANTOM DIFFR
FEET

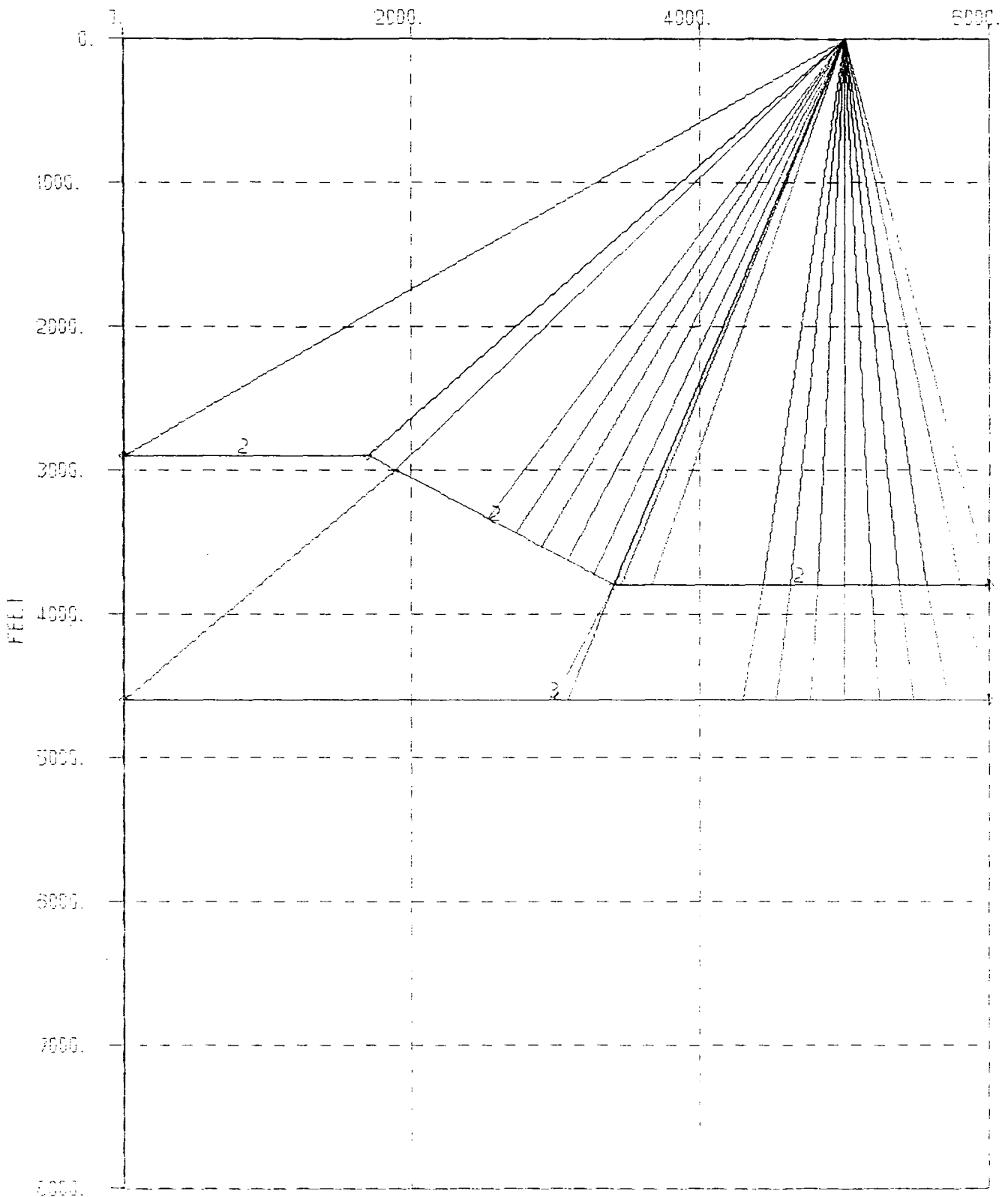


Fig 3.5a: Example of an AIMS input model

SECTION FOR KNEE MODEL WITH PHANT DIFFR

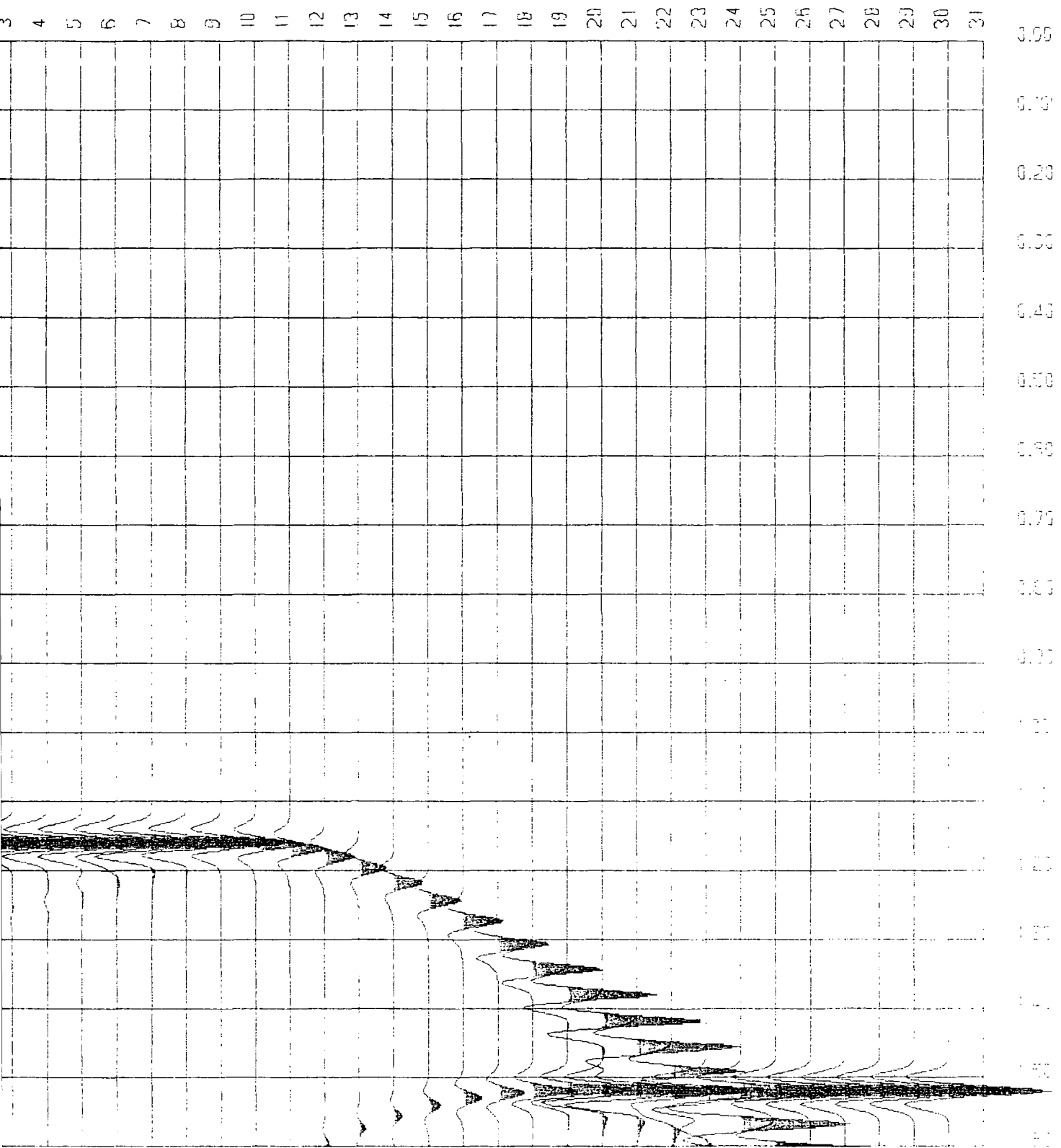


Fig 3.5b: AIMS output from the model in Fig 3.5a

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.

(in this case ^{ANSI}FORTRAN 66)

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 4Seismic Processing SoftwareOverview

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~~^{long-record} 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.

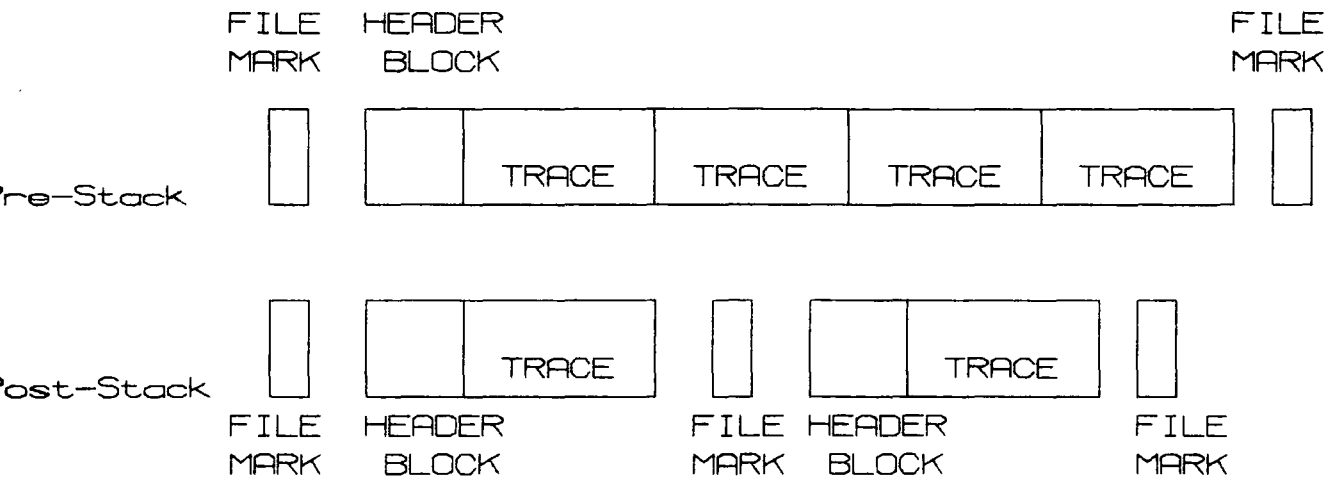
It was decided that in order to keep the structure of the data as simple and straightforward as possible, logical units of data would be separated by file marks on tape and put into separate files on disc. Hence, before stack, common shot gathers or CMP gathers would be contained in separate files and 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 upto 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

<u>Byte position</u>	<u>Function</u>	
1-3 4-5 6-8	ASCII values from field tape header block	
9		Sampling interval in msec(s)(from field tape)
10		Equipment serial number(from field tape)
11	Recorded data length	
12	Number of channels recorded	
13-14	Number of channels in the file	
15-16	Starting position of first data sample	
17-18	Ending position of last data sample	
19	Gather code: 0-common shot/reciever. 1-CMP. 2-Stacked trace. 3-Single trace.	
20	Units code: 1-metric. 2-Imperial	
21-24	Shot to 1st Reciever offset	
25-28	Reciever spacing	
29-32	Shot spacing	
33-36	Shot interval in seconds(Marine)	
37-40	Source depth	
41-44	Reciever Array depth	
45-46	Source code: 0-Airgun. 1-Explosive. 2-Vibroseis. 3-Weights.	
47-48	Next useable address in process header	
49-50	Delimiter with all bits set(%0177777)	
51-254	User comments inserted at DEMUX time	
255-256	Delimiter with all bits set(%0177777)	
257-512	Process Header : A process history inserted into Header at the next free slot by each process	

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

Header Block Analysis

Migration

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

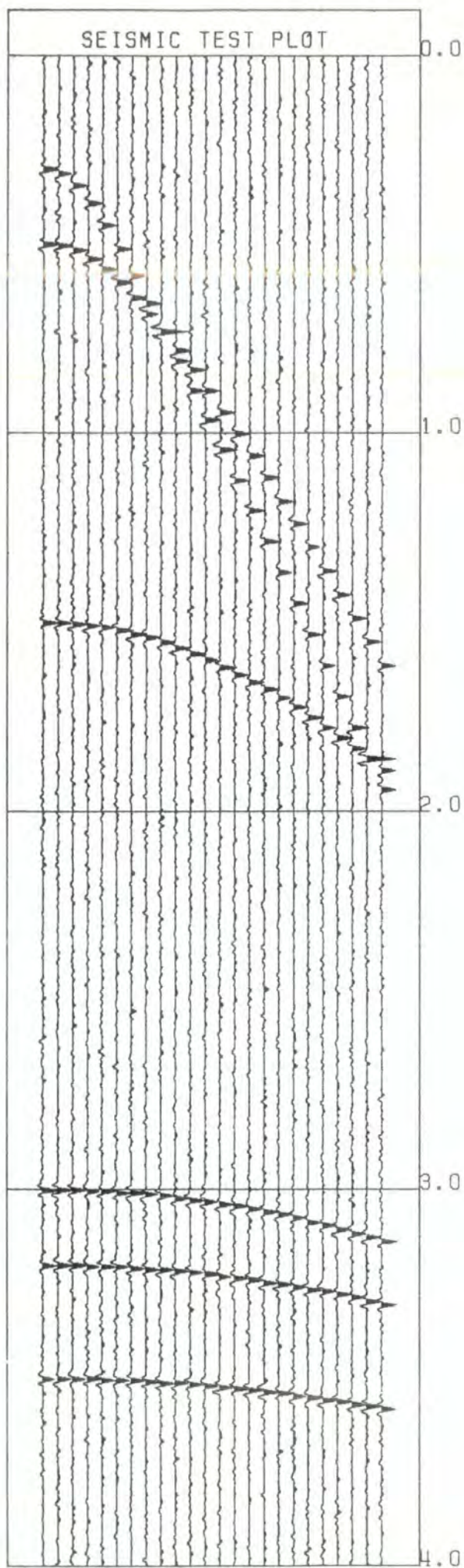


Fig 4.2: A Synthetic CMP gather generated by ANSEL

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, upto 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

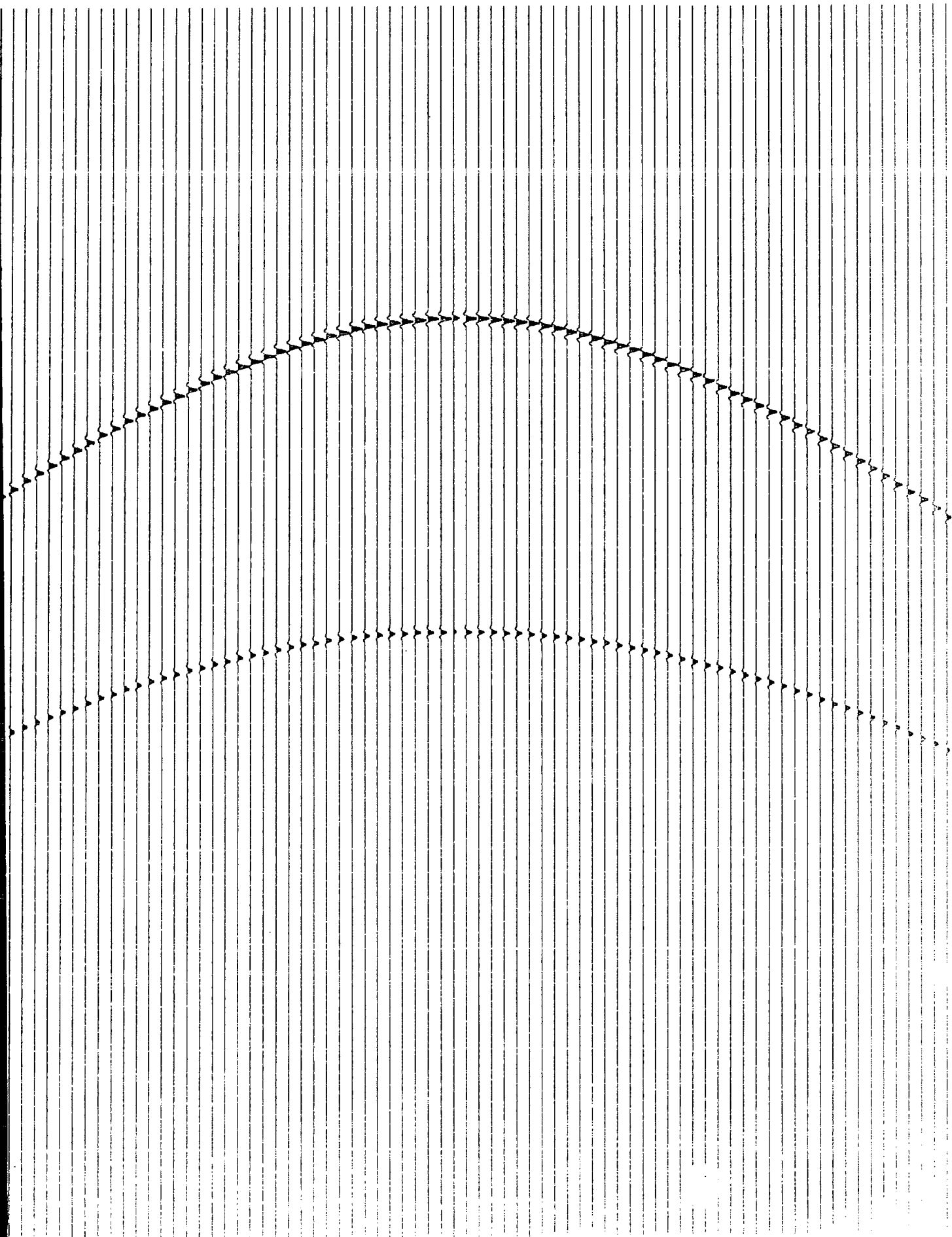


Fig 4.3: A synthetic Stacked Section Showing Diffractors

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.

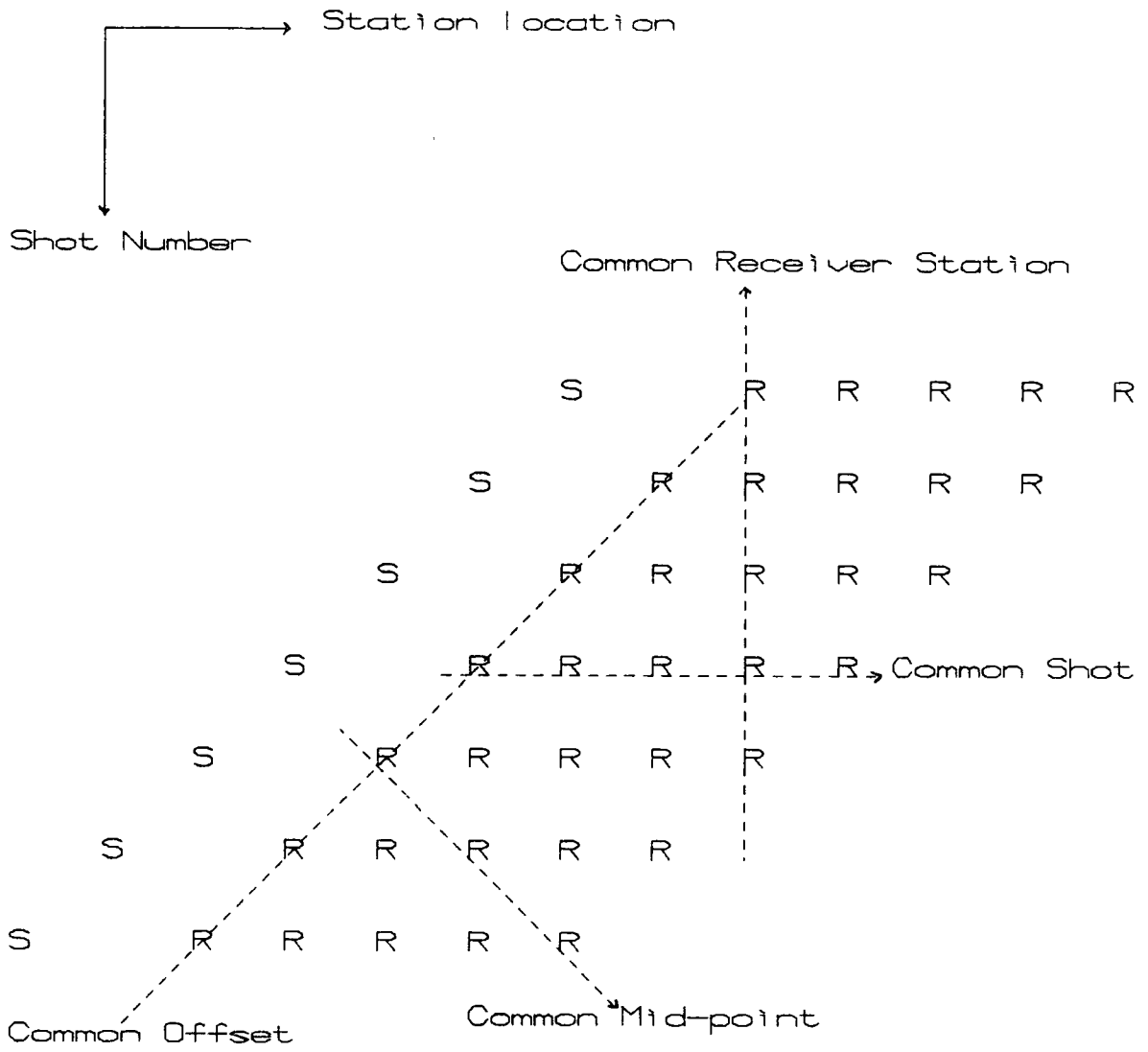


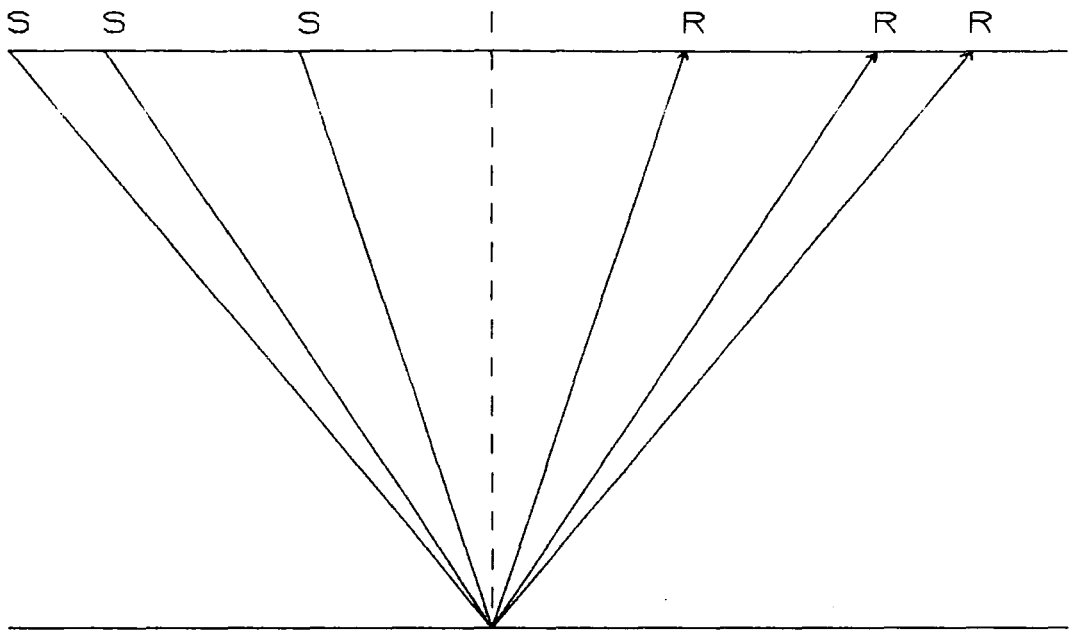
Fig 4.6 : CMP Acquisition Geometry

Sort

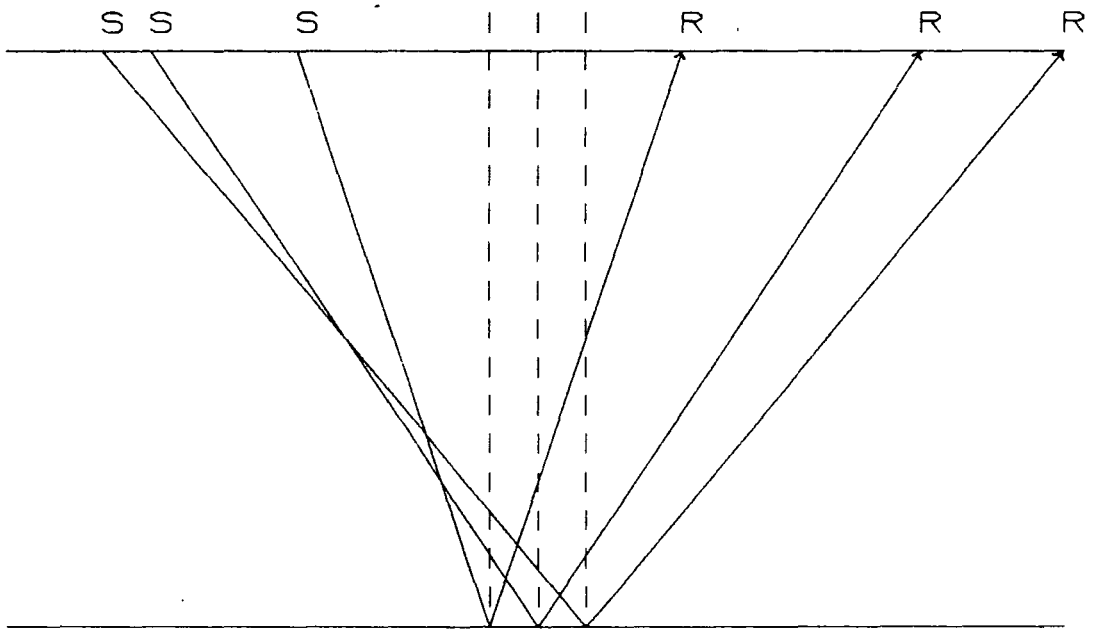
It was decided that during demultiplex all the data on the field tape should be demultiplexed and written to tape as common shot gathers, as mentioned previously, so that once it has been completed the field tapes would not have to be referred to again. Therefore, in order to select the segment of the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



CMP Smear

Incorrect CMP positioning

Fig 4.7 : The Smear Effect of Incorrect CMP positioning

An example with CMP gathers as input files

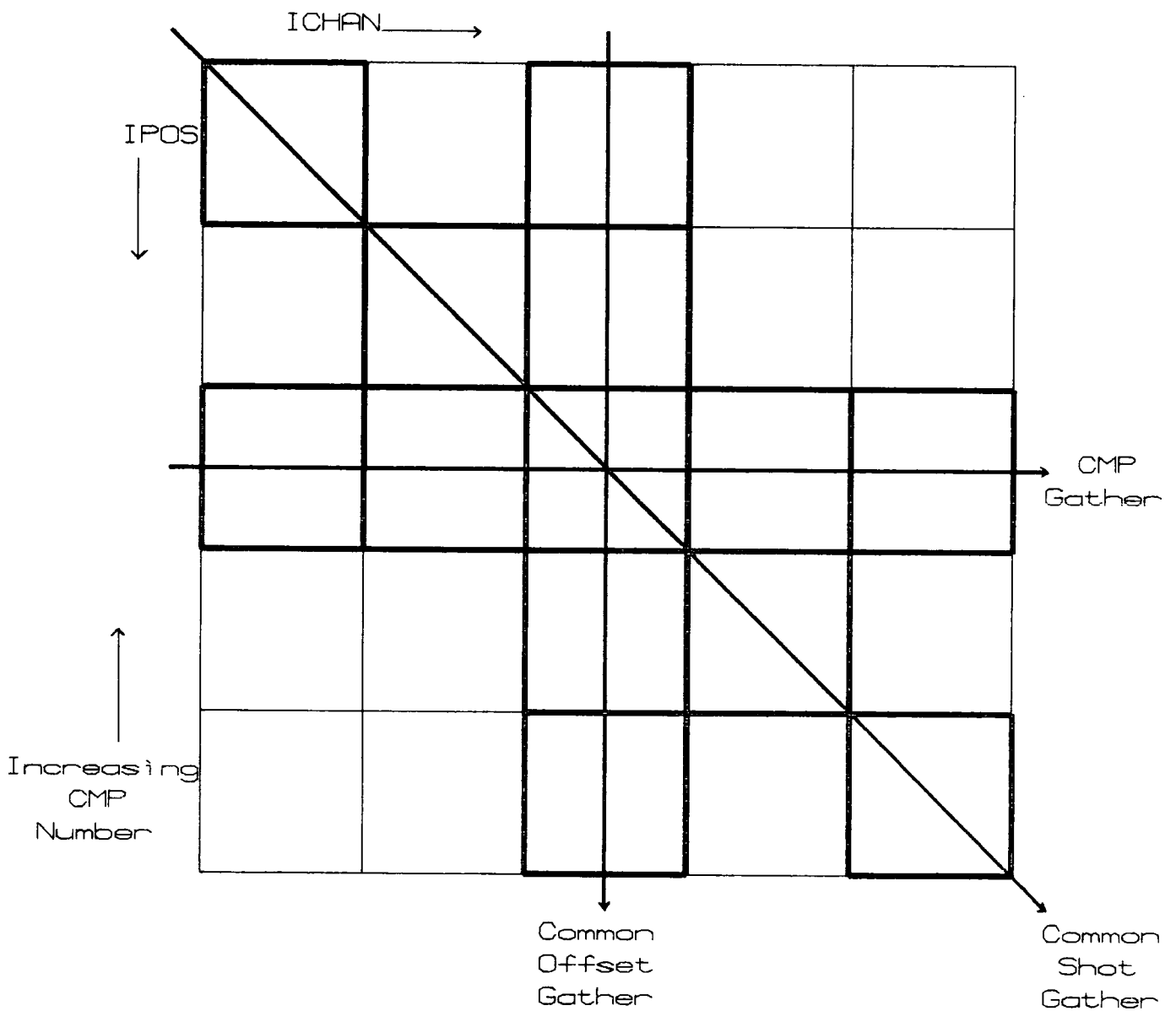


Fig 4.8 : Generating Different Trace Gathers using SORT

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.

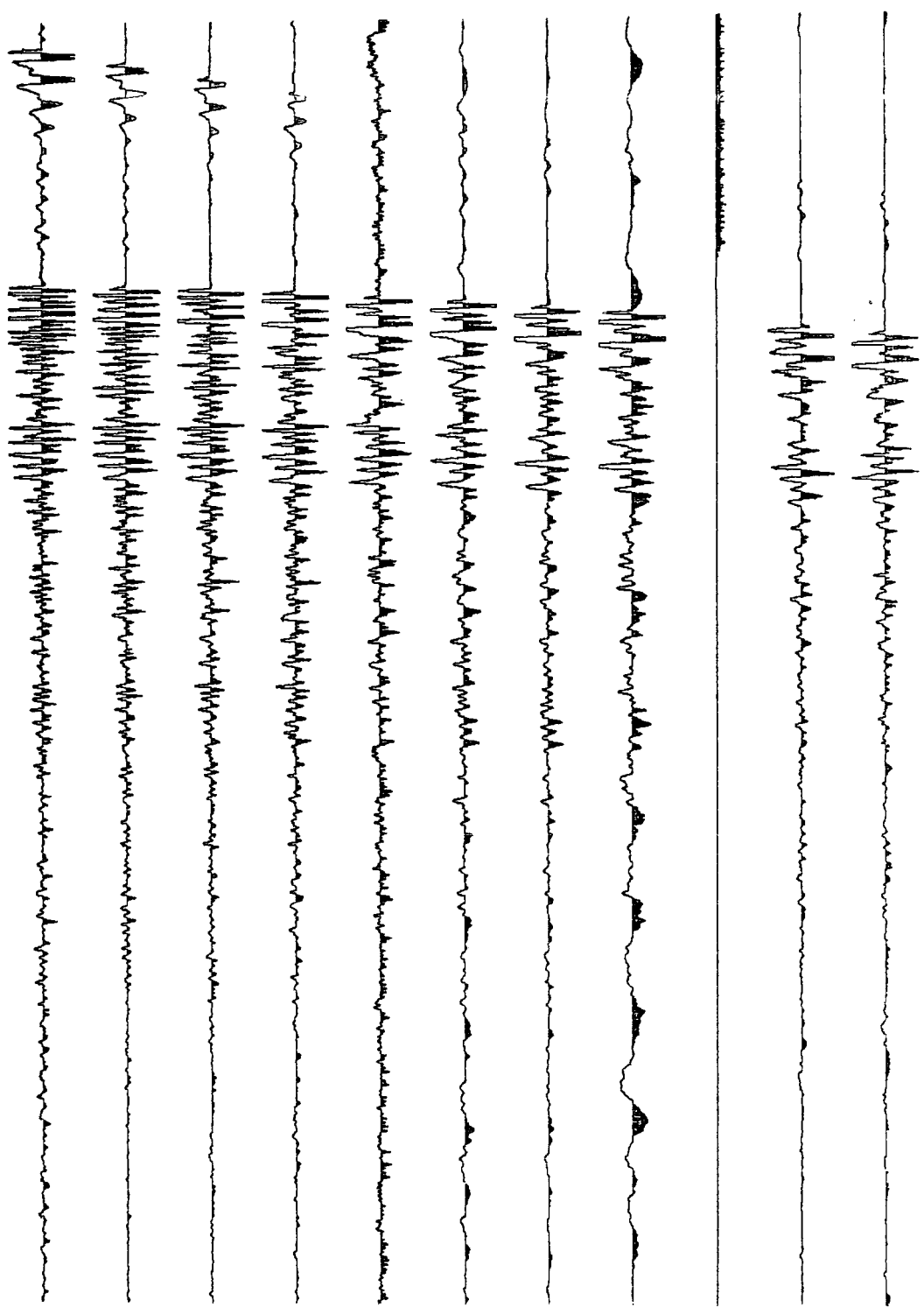


Fig 4.9: Example of data plotted with a large trace

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



SCALING FACTOR = 1.4×10^{-4}

FREQUENCY $\times 10^2$ ENERGY SPECTRUM

Fig 4.10: Energy Spectrum of a noisy seismic trace produced by MPFANT

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 ^{in a homogeneous medium,} 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 $\frac{\exp(-at)}{e^{-at}}$. Therefore the inverse function $\frac{\exp(at)}{e^{-at}}$ 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.

$$\exp(0.2t) \quad e^{-0.2t}$$

$$t \exp(0.2t) \quad t e^{-0.2t} \dots \text{one usually applied}$$

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 $t \exp(0.2t)$ ~~$t e^{-0.2t}$~~ gave the best results, of the ones available, in producing equalisation of amplitudes down the trace. It was also felt that

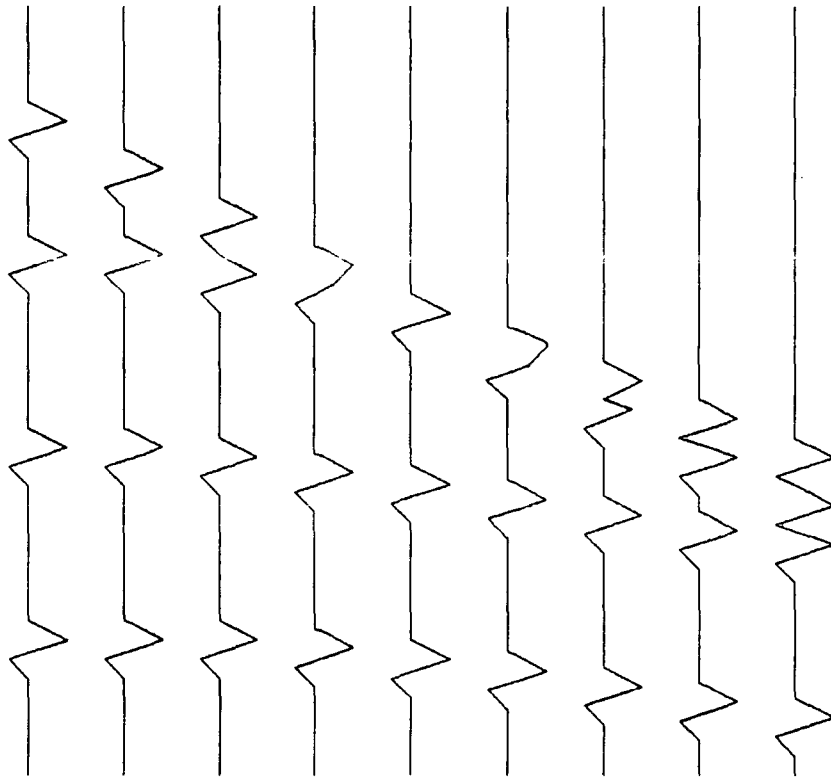
these deterministic methods are preferable to AGC functions at this stage in the processing because their effects can be easily removed, which is not possible with AGC. In fact one of the options in the program is to be able to remove one of the specified functions, perhaps to replace it with an alternative, 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.

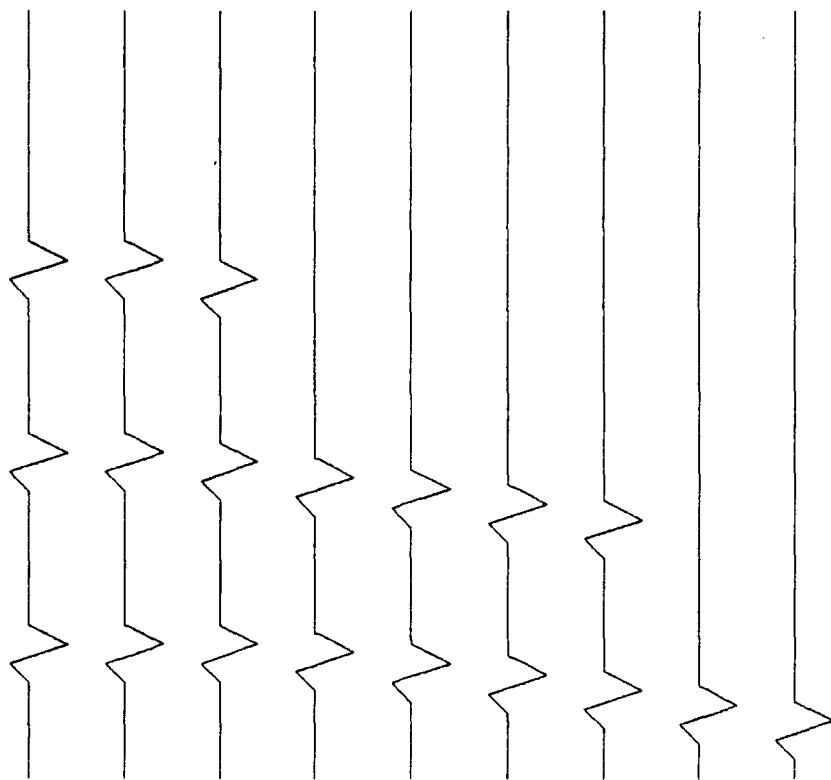
Muting

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.



Before Mute



After Mute

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 Schaffer, 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

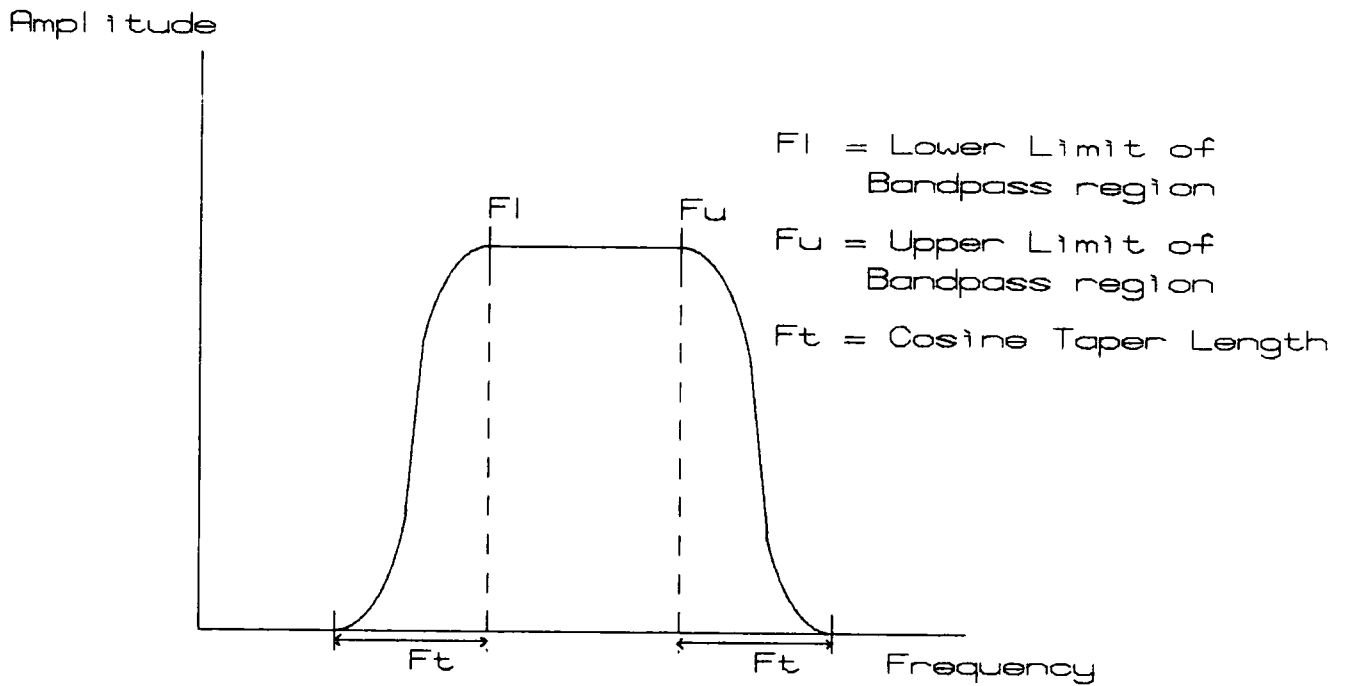


Fig 4.12 : Bandpass Filter Representation

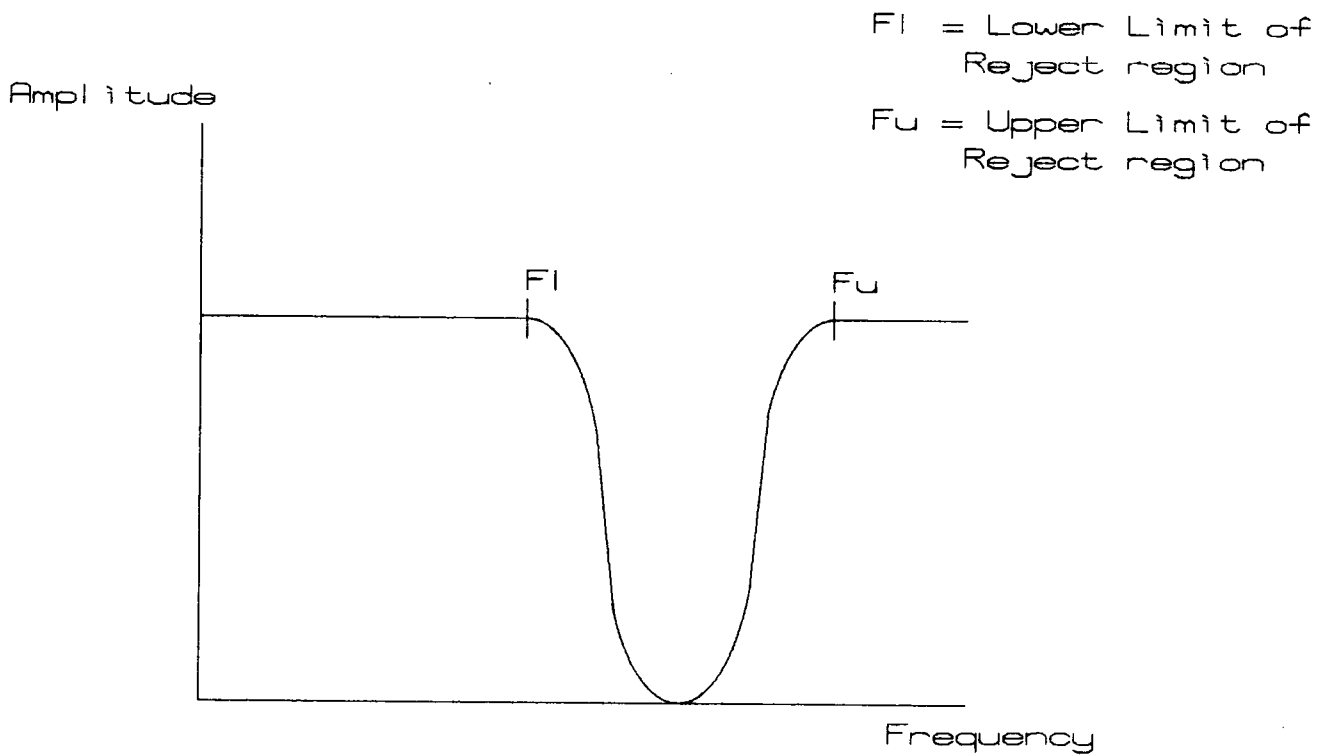


Fig 4.13 : Bandreject Filter Representation

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 .

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

and the error energy is given by:-

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

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

Therefore:-

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

and this reduces to

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

now

$$\text{eqn 5} \quad \sum_{t=0}^{M+N} B_{t-s} B_{t-i} = \cancel{\phi_{BB}(s-i)} \phi_{BB}(s-i)$$

This is the autocorrelation function of the input wavelet

$$\text{eqn 6} \quad \sum_{t=0}^{M+N} D_t B_{t-i} = \cancel{\phi_{DB}(i)} \phi_{DB}(i)$$

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):-

$$\text{eqn 7} \quad \begin{bmatrix} \phi_{BB} \end{bmatrix} \begin{bmatrix} F \end{bmatrix} = \begin{bmatrix} \phi_{DB} \end{bmatrix}$$

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

In order to apply this to a seismic trace several assumptions have to be made. The purpose of ^{this} 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.

$$\text{eqn 8} \quad \text{TRACE} = \mathcal{D}(t) = S(t) * r(t) + n(t)$$

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 ~~can be obtained from~~ ^{is a scalar multiple of} the autocorrelation of the trace, ~~except for the zero-lag coefficient.~~

$$\text{eqn 9} \quad \sum_{s=0}^{M+N} \left(\sum_s * r_s + n_s \right) \left(\sum_{s_1 \tau} * c_{s_1 \tau} + n_{s_1 \tau} \right) = \sum_{s=0}^{M+N} \sum_s \sum_{s_1 \tau}$$

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.

$$\begin{aligned} \text{eqn 10}_a \quad I &= 1 - 2c_0 + \sum_{c=0}^{M+N} C_c^2 \\ &= 1 - 2B_c F_0 + \sum_{c=0}^{M+N} C_c^2 \end{aligned}$$

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, $\sum C_c^2$ being a constant for B convolved with any

filter having the same autocorrelation function.

In a seismic application the filter is derived as follows:-

$$\text{eqn 10}_b \quad [\phi_{xx}] [F_k] = [\phi_{xy}]$$

$[\phi_{xx}]$ is the autocorrelation of the seismic trace for lags 0-M.

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

$[\phi_{xy}]$ is the cross correlation function between a spike at $T=0$ which is the desired output, $y(t)$ seismic trace, and the source wavelet $x(t)$.

As the spike series can be represented by 1,0,0,0,0,...0 the crosscorrelation can be seen to be A,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 ^{control noise amplification, and it will also} 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 $t+\alpha$ 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.

$$\text{eqn 1} \quad \sum_{s=0}^t \alpha_s P_{t-s} = \hat{\alpha}_{t+\alpha}$$

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.

$$\text{eqn 2} \quad \begin{bmatrix} \Phi_{xx} \end{bmatrix} \begin{bmatrix} P_m \end{bmatrix} = \begin{bmatrix} \Phi_{xy} \end{bmatrix}$$

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:-

$$\text{eqn 1} \quad T_x^2 = T_0^2 + \frac{X^2}{V^2}$$

T_x is the observed time on the trace .

T_0 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.

$$\text{eqn 2 Normal Moveout } = \Delta T = \left(T_0^2 - \frac{x^2}{v^2} \right)^{1/2} - T_0$$

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_0 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.

$$\text{eqn 1 } S(v,t) = \frac{\sum_{K=-N/2}^{N/2} \left(\sum_{J=1}^M A_{L+K,J} \right)^2}{M \sum_{K=-N/2}^{N/2} \sum_{J=1}^M A_{L+K,J}^2}$$

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

times.

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

PROCESSING PARAMETERS

VELOCITY ANALYSIS CONTOURS

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

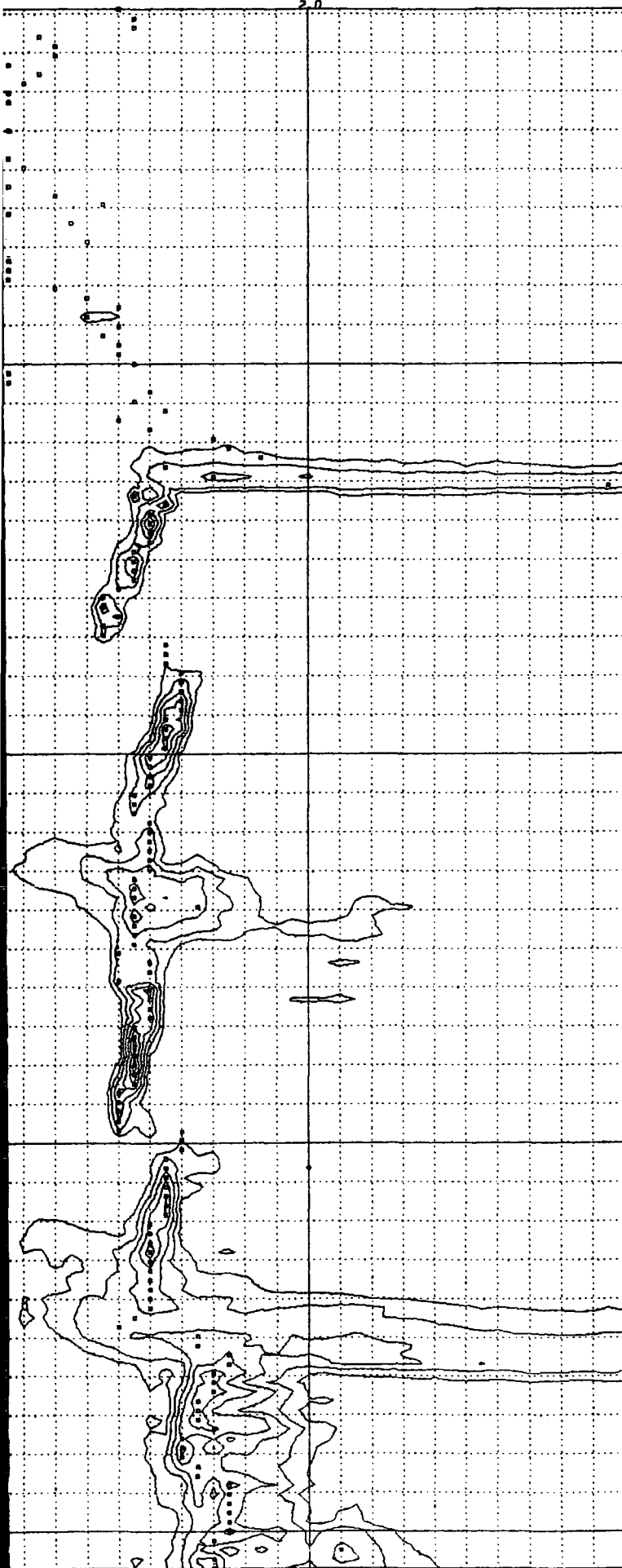


Fig 4.14: Contoured Velocity Analysis Display

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 ^t_A 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:-

$$T_i = (i-1)DT \text{ for } i=1 \dots \text{length of trace}$$

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:-

$$i = 1 + \text{INT}(\text{SQRT}(T_j^{**2} + X_k^{**2}/V_j^{**2})/DT)$$

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.

eqn 1 stacked value

$$S_j = \sum_{j=1}^M C_j A_{ij}/M$$

eqn 2 for constant energy stack

$$C_j = \left(\sum_{j=1}^M A_{ij}^2 \right)^{-1}$$

eqn 3 for diversity stack

$$C_j = \left(\sum_{j=1}^M A_{ij}^2 \right)^{-2}$$

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, ^{(c.f. Dunkin and Levin, 1973). Furthermore,} as the correction can only be made to the nearest sample, ~~it~~ it can be shown that the moveout corrected signal suffers a power loss which varies with frequency, given by:-

$$1 - (\sin(\pi Fdt)/\pi Fdt)^{**2}$$

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 $\exp(-2\pi F dt)$, 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 of N the transformed trace is multiplied by the factor $\exp(-2\pi F DT/N)$. 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

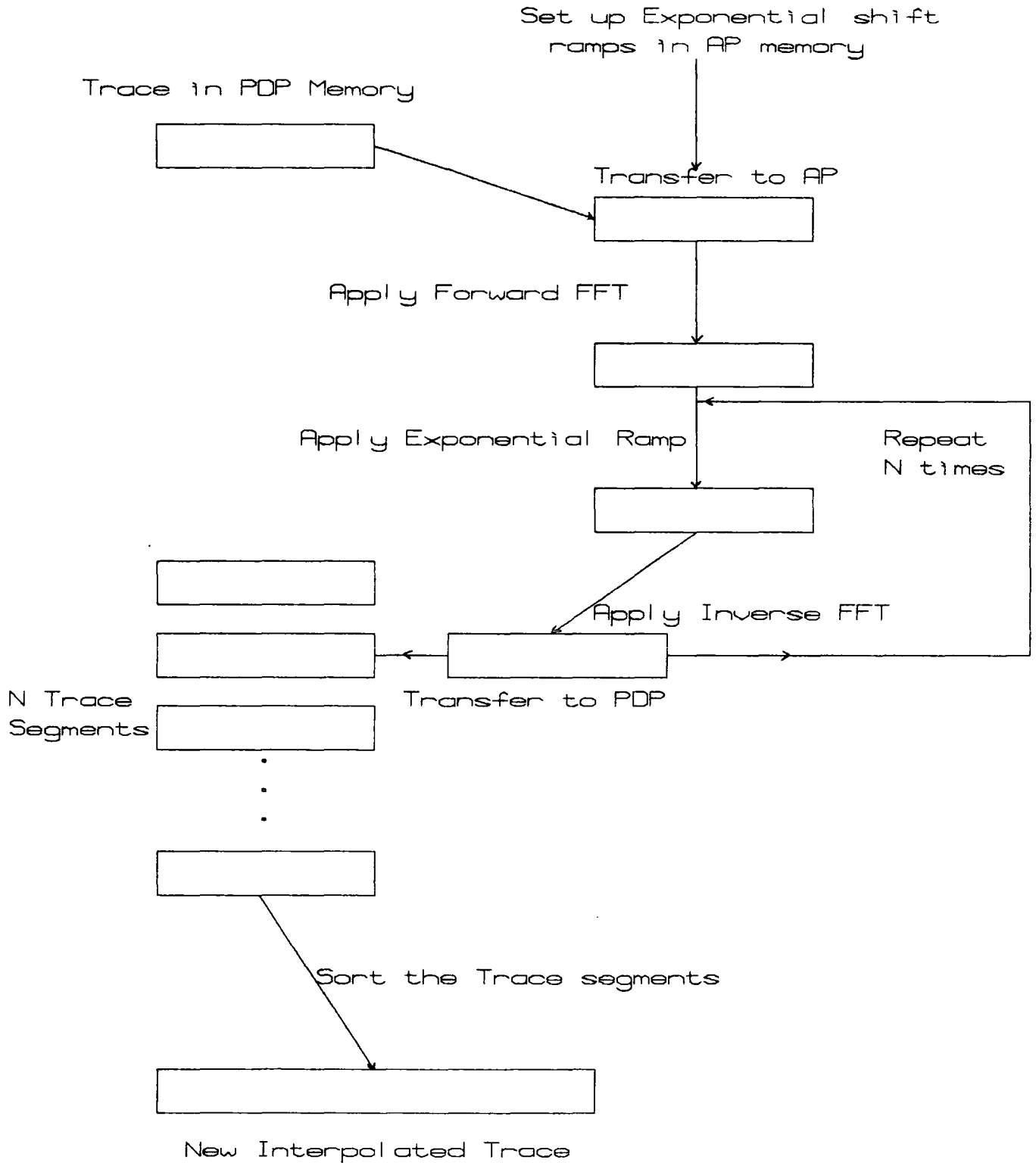


Fig 4.15 : Flow of the Interpolation Algorithm

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 upto 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

used.

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 DK3TSTPRSDAT TO BE ANALYSED, LENGTH= 145 BLOCKS
FILE NO:484 TAPE NO:02202
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: 0
AND ENDING WITH SAMPLE NO: 1536
TYPE OF SOURCE= AIRGUNS
AND THE UNITS OF LENGTH USED= METRES
RECIEVER DEPTH= 10.70
SOURCE DEPTH= 7.30
SOURCE-RECIEVER OFFSET= 297.00
RECIEVER SPACING= 100.00
SHOT SPACING= 50.00
SHOT REPITITION RATE= 20.00 SECS
USER INFO PUT INTO HEADER BLOCK
IS GIVEN BELOW
DURHAM UNIVERSITY 1980 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
T*EXP(0.2T) 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:-

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

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 $U'(Kx, \frac{z}{\lambda}, \omega t)$ to be the Fourier counterpart of $U(x,z,t)$, then the scalar wave equation can be written as:-

$$\text{eqn 1} \quad \frac{\partial^2 u'}{\partial z^2} + \left(\frac{\omega_t^2}{V^2} - K_x^2 \right) u' = 0$$

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 ^{ed} as noise and just the first equation used for the wave energy. This equation is in fact the basis of F-K migration.

$$u'(K_x, z, \omega_t) = u'(K_x, 0, \omega_t) e^{i z \text{sgn}(\omega) \left(\frac{\omega^2}{V^2} - K^2 \right)^{1/2}} \quad \text{for } |\omega| > V|K|$$

eqn 2

$$u'(K_x, 0, \omega_t) e^{z \left(\frac{\omega^2}{V^2} - K^2 \right)^{1/2}} \quad \text{for } |\omega| < V|K|$$

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):-

$$U(x, z, t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U(x-x', 0, (t-t')) M_z(x, t) dx' dt'$$

eqn 3

$$\text{WHERE } M_z = \frac{-1}{\pi} \frac{\partial}{\partial t} \left[\frac{H(-t' - r'/V)}{(t^2 - r'^2/V^2)^{1/2}} \right] dx' dt'$$

$$H(t) = \begin{cases} 1 & t > 0 \\ 0 & t < 0 \end{cases}$$

$$r' = (x'^2 + z^2)^{1/2}$$

It is interesting to transform the operator M_z into the Fourier domain and examine its counterpart M'_z , which is given below.

$$\text{eqn 4} \quad M'_z = \begin{cases} e^{i z \operatorname{sgn}(\omega) (\omega^2/v^2 - k^2)^{1/2}} & |\omega| > v|k| \\ e^{-z (k^2 - \omega^2/v^2)^{1/2}} & |\omega| < v|k| \end{cases}$$

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.

$$\text{eqn 5} \quad u(x, z, 0) = \iint_{-\infty}^{\infty} u(x-x', 0, (t-t')) M_z \partial x' dt'$$

$$M_z = \frac{1}{\pi} \frac{\partial}{\partial t} \left[\frac{H(t'+t_0)}{(t'^2 - t_0^2)^{1/2}} \right]$$

$$t_0 = r/v$$

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~~^{one}-way travel time τ is defined as $\tau = z/v$ and so the migration equations

can be couched in terms of x and τ rather than x and z .

$$U_m(x, \frac{\tau}{2}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U(\omega-x', 0, t') M_{\tau} dx' dt'$$

eqn 6

$$M_{\tau} = \frac{-1}{\sqrt{\pi}} \frac{\partial}{\partial \tau} \left[\frac{H(t'+t_0)}{(t'^2-t_0^2)^{1/2}} \right]$$

$$t_0 = \left(\frac{x'^2}{v^2} + \tau^2 \right)^{1/2}$$

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_{τ} 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.

$$t_0 = \left[\left(\frac{n \Delta x}{v} \right)^2 + \tau^2 \right]^{1/2}$$

$$t_0 \Psi_{nk} = H(k \Delta t - t_0) t_0 \left[\left(\frac{k \Delta t}{t_0} \right)^2 - 1 \right]^{1/2}$$

eqn 7

$$k_0 = \left(\frac{t_0}{\Delta t} \right)$$

$$\therefore \Psi_{nk} = 0 \quad \text{for } k \leq k_0$$

$$= \left[\left(\frac{k \Delta t}{t_0} \right)^2 - 1 \right]^{1/2} \quad k > k_0$$

this gives the alternative

$$M_{\tau} = \frac{\tau}{\pi v t_0} \frac{\partial^2}{\partial t^2} \left[H(t-t_0) (t^2 - t_0^2)^{1/2} \right] \cancel{\partial t \partial t}$$

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

$$\text{eqn 8 } M_{nk} = \left(2\psi_{nk} - \psi_{n,k-1} - \psi_{n,k+1} \right) \frac{\tau \Delta x}{\pi t_0 v \Delta t}$$

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

$$\text{eqn 9 } U_m \left(n \Delta x, \frac{\tau}{2} \right) = \sum_{n'=-N}^N \sum_{k'=0}^K U \left((n-n') \Delta x, 0, k' \Delta t \right) M_{n'k'}$$

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.

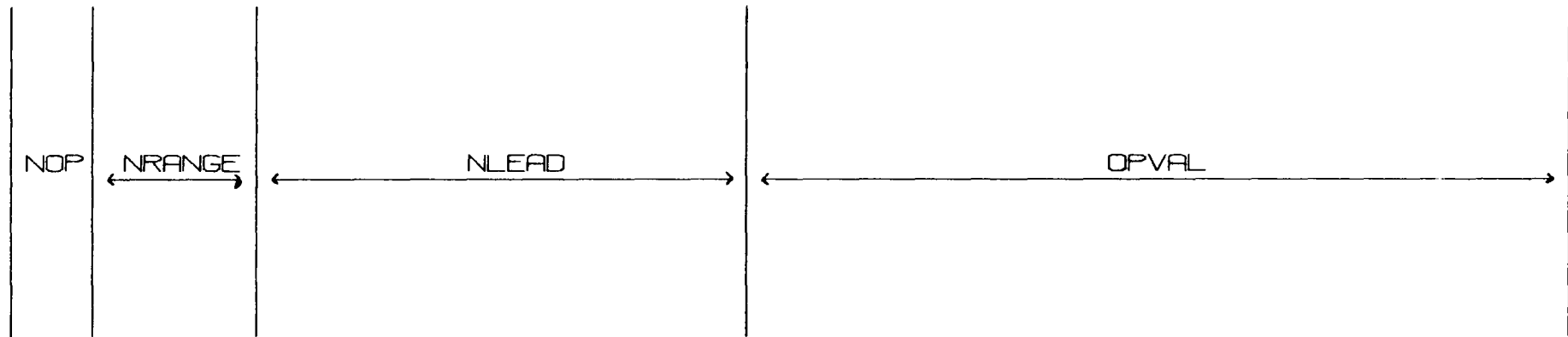
The constraints on the migration parameters were determined from the amount of available disc space and AP memory. The operator values obviously have to be written out to a disc file once they have been calculated, and this file is limited in size by the storage available on a single disc platter. Also 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 halfwidth 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 halfwidth * 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 MPTPDK 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.

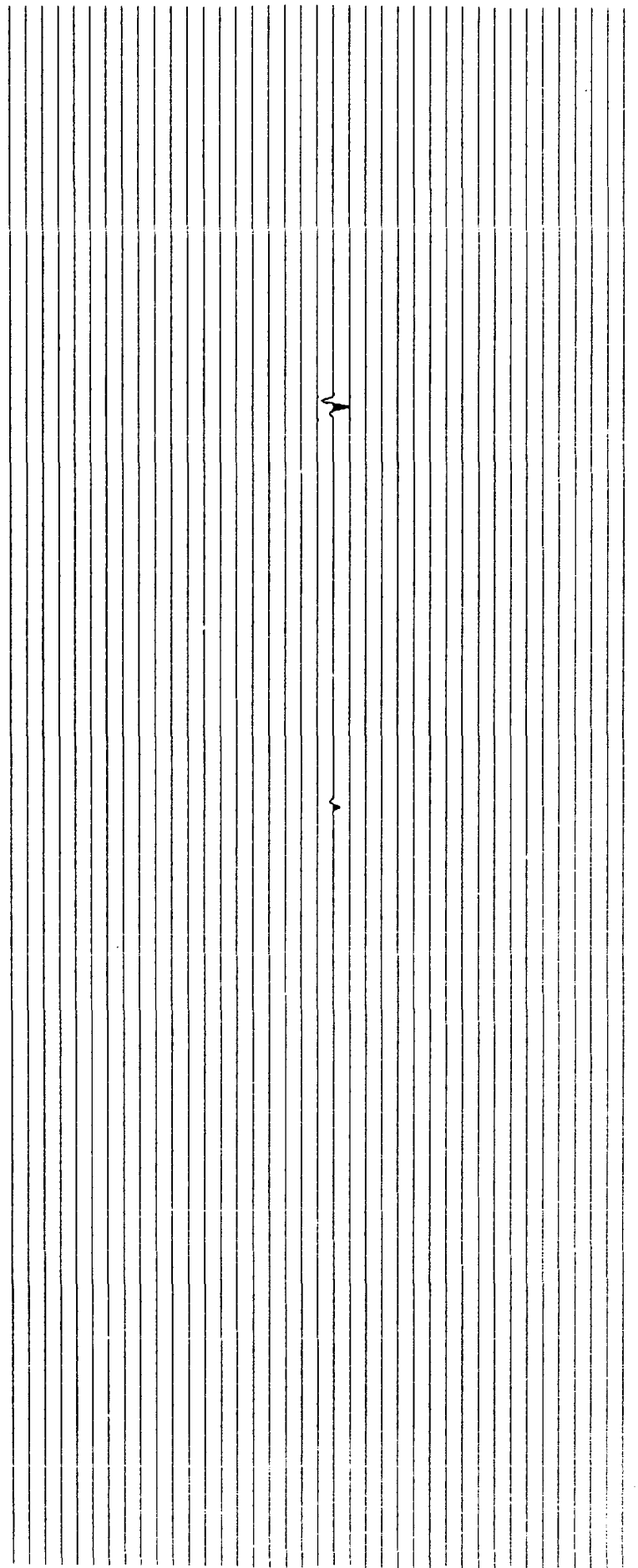


Fig 4.18: The Diffractor Model of Fig 4.3 after
Kirchhoff Migration using MPKMIG

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.

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

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:-

$$t' = t - \int \frac{dz}{V}$$

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

For small dips it is possible to neglect the terms in $\partial^2 u / \partial z^2$ and so we are left with the well known 15 degree approximation to the wave equation.

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

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 $\partial^2 u / \partial z^2$ we can derive:

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

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

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

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 τ which is not so susceptible to errors in the velocity specification. Hence the equations below can be formed.

$$\tau = \frac{z}{v}$$

15 degree equation
$$\frac{v^2}{2} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial t^2} = 0$$

45 degree equation
$$\frac{v^2}{4} \frac{\partial^3 u}{\partial x^2 \partial t} - \frac{v^2}{2} \frac{\partial^3 u}{\partial x^2 \partial t} - \frac{\partial^3 u}{\partial t^2 \partial t} = 0$$

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:-

$$v' = v/2$$

$$t = n \Delta t$$

$$\tau = k \Delta \tau$$

$$x = j \Delta x$$

$$* A = \frac{0.518 v^2 \Delta t \Delta \tau}{16 \Delta x^2} \approx \frac{v^2 \Delta t \Delta \tau}{32 \Delta x^2}$$

$$\frac{\partial^2}{\partial x^2} = \frac{T}{\Delta x^2 (I + bT)}$$

$$T = (E_x^{-1}, -2, E_x)$$

$$b = 1/6 \quad \ddagger$$

$$\frac{\partial}{\partial t} = \frac{2}{\Delta t} \left(\frac{(E_t - I)}{(E_t + I)} \right) \quad \frac{\partial}{\partial \tau} = \left(\frac{(E_\tau - I)}{(E_\tau + I)} \right)$$

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

$$[(b-A)T + I] U_{k+1, j}^n = [(b+A)T + I] U_{k+1, j}^{n+1} + [(b+A)T + I] U_{k, j}^n - [(b-A)T + I] U_{k, j}^{n+1}$$

* 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 $U_{k+1, J}^n$ 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:

$$A = \frac{0.279 V^2 \Delta t^2}{4 \Delta x^2} \approx \frac{V^2 \Delta t^2}{4 \Delta x^2}$$

$$B = \frac{V^2 \Delta t \Delta \tau}{32 \Delta x^2} = \frac{V^2 \Delta t \Delta \tau}{2 \Delta x^2}$$

$$\frac{\partial^2}{\partial x^2} = \frac{T_x}{\Delta x^2 (I + b T_x)} \quad \frac{\partial^2}{\partial t^2} = \frac{T_t}{\Delta t^2}$$

$$\frac{\partial}{\partial x} = \frac{2}{\Delta x} \left(\frac{(E_x - I)}{(E_x + I)} \right) \quad \frac{\partial}{\partial t} = \frac{1}{2 \Delta t} (E_t - E_t^{-1})$$

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

$$[I + (b \cdot B) T_x] E_x E_t^{-1} = [I + (b \cdot B) T_x] E_t - [I + (b \cdot B) T_x] E_x E_t - [2 + (2b + A) T_x] \\ + [2 + (2b + A) T_x] E_x + [I + (B + b) T_x] E_t^{-1}$$

$$\therefore [I + (b \cdot B) T_x] U_{k+1, J}^{n-1} = [I + (b \cdot B) T_x] U_{k, J}^{n+1} - [I + (b \cdot B) T_x] U_{k+1, J}^{n+1} - [2 + (2b + A) T_x] U_{k, J}^n \\ + [2 + (2b + A) T_x] U_{k+1, J}^n + [I + (B + b) T_x] U_{k, J}^{n-1}$$

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 $U_{K,I,J}^{n-1}$ 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 $\Delta\tau$ 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.

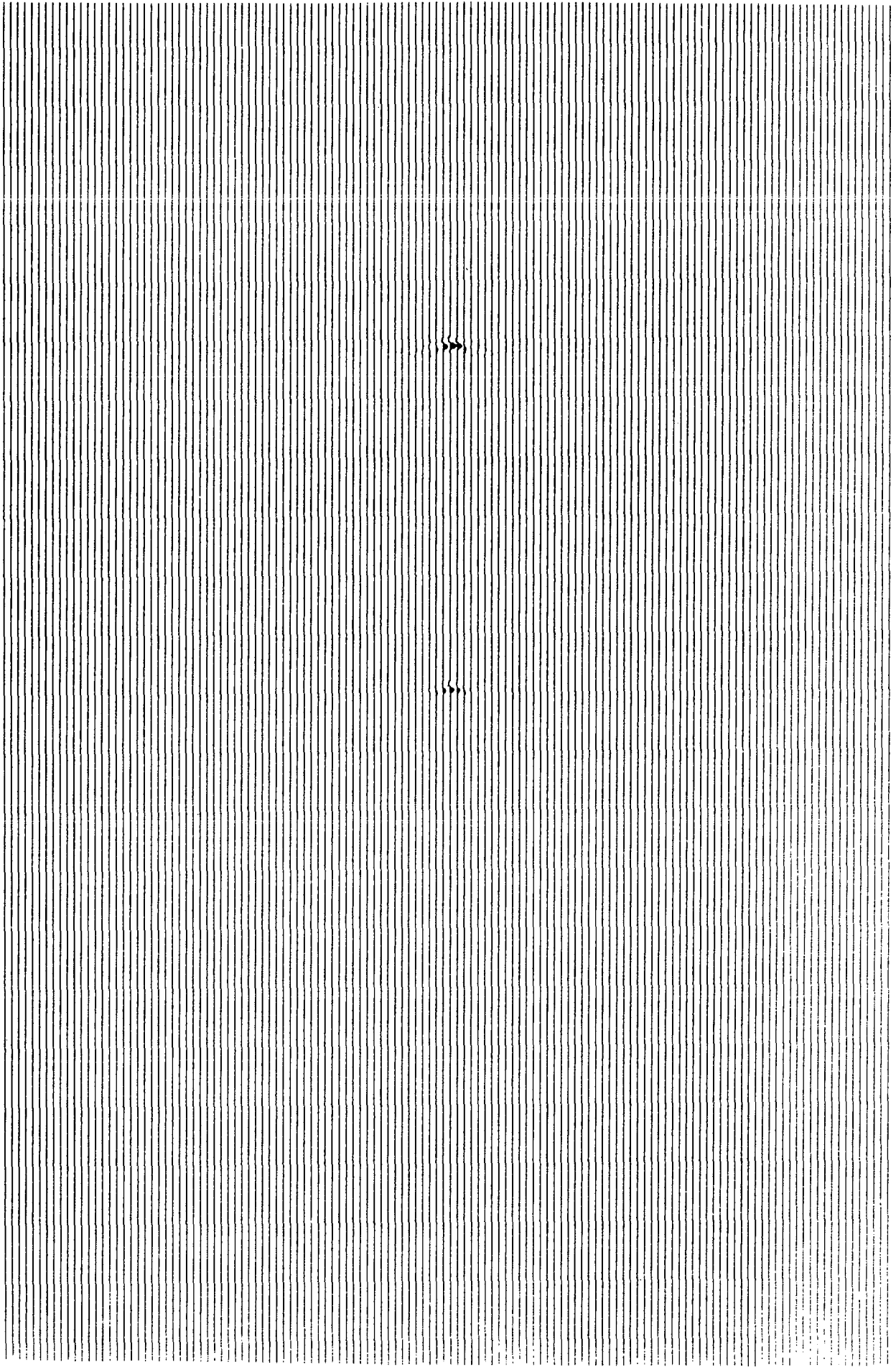


Fig 4.19: The Diffractor Model of Fig 4.3 after
15° Finite Difference Migration with MPFD15

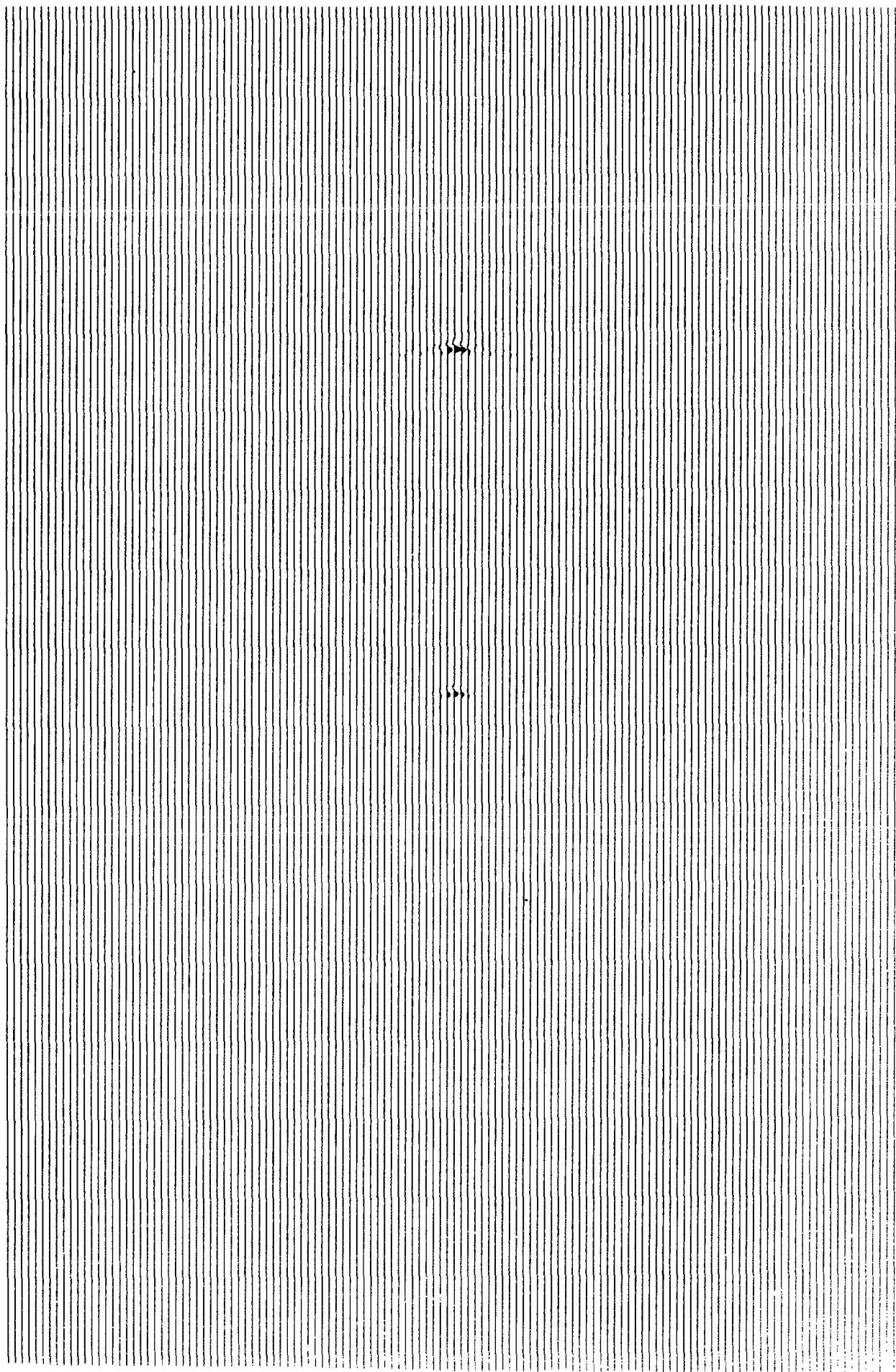


Fig 4.20: The Diffraction Model of Fig 4.3 after
45° Finite Difference Migration with MPD45

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 $\Delta\tau$ 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 $\Delta\tau$ step at a time to determine the values of $U(x, n\Delta\tau, 0)$ 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 $n\Delta\tau$.

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 $\Delta\tau$ step by M , where M is given by $M = \Delta\tau / \Delta t$, as the step $\Delta\tau$ can be larger than Δt .

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.

Chapter 5

Using The System

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.

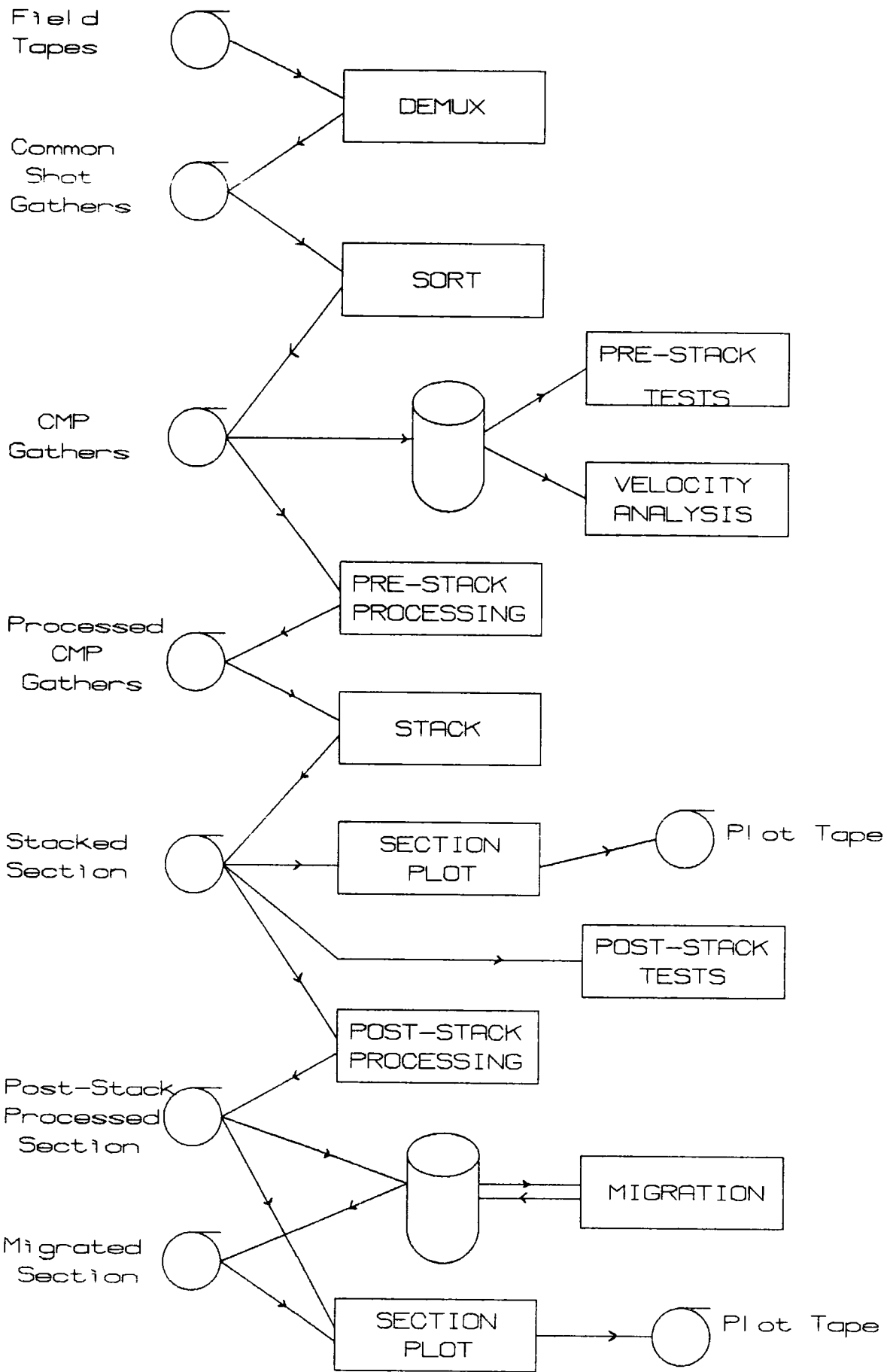


Fig 5.1 : Processing System data flow

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

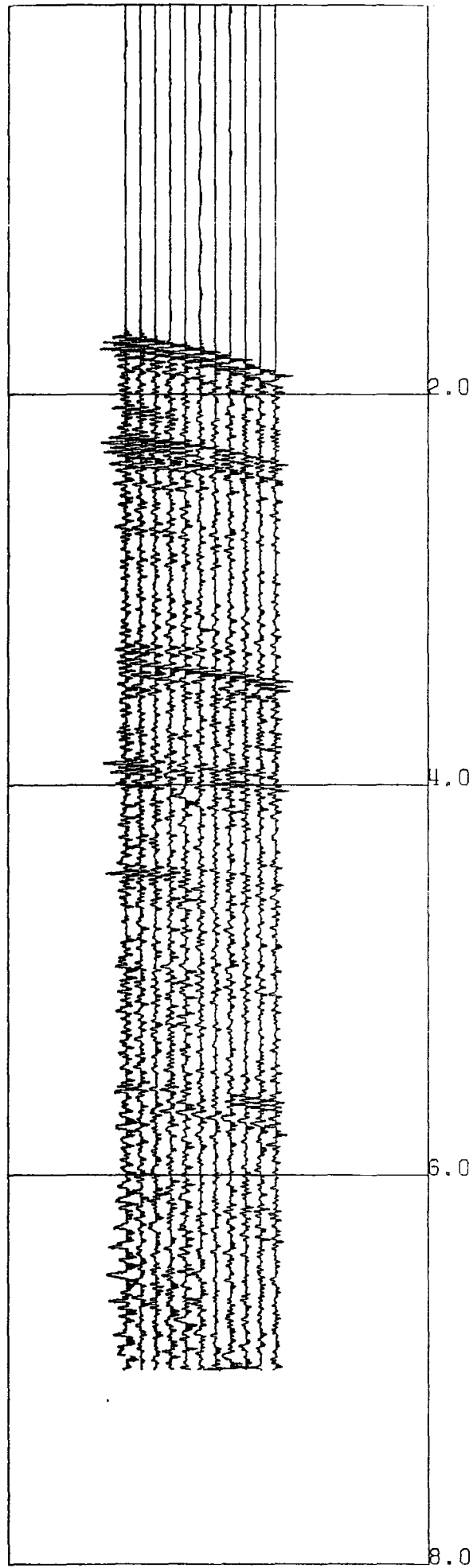


Fig 5.2a: An Example of a Jan Mayen Shot Record

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 5.2**b**. 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.

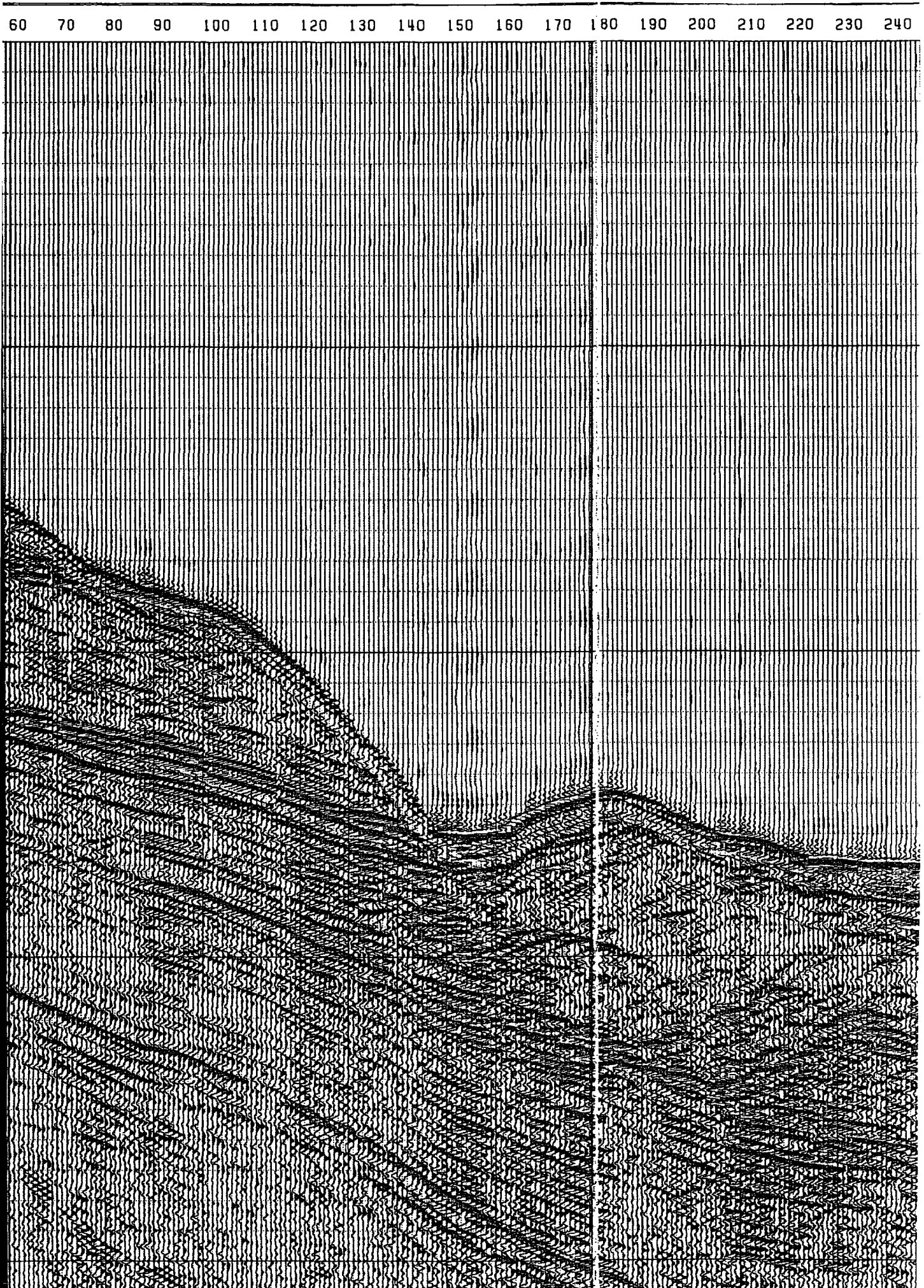


Fig 5.2.b: A Portion of the Final Section for the Jan Mayen line

Jan-Mayen Processing Details

- 1...Demultiplex and Sort into 11 channel CMP gathers
- 2...Amplitude correction- $t e^{**0.2t}$
- 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 systems 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

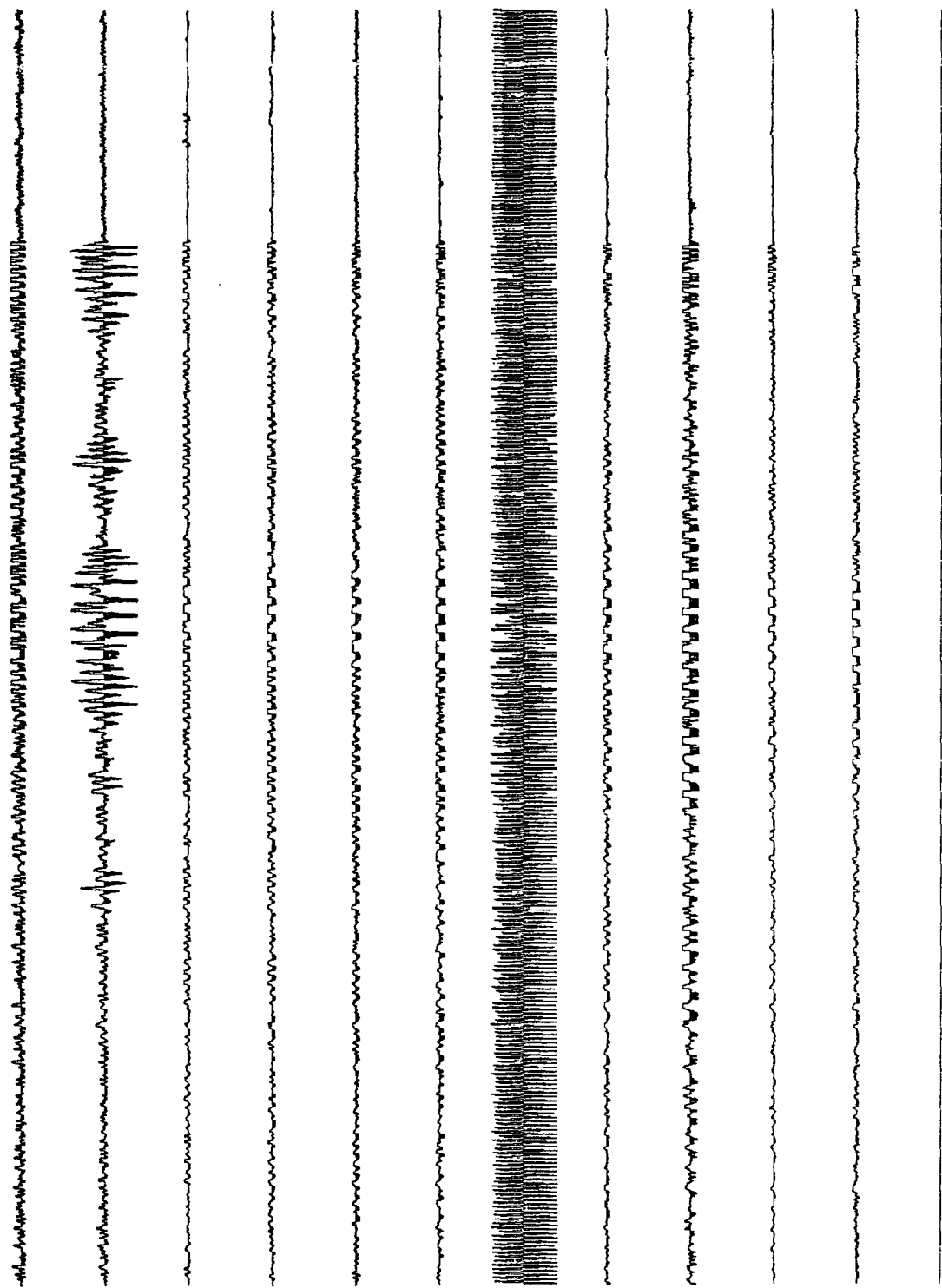


Fig 5.3: Display of a Caribbean Arc Shot Record

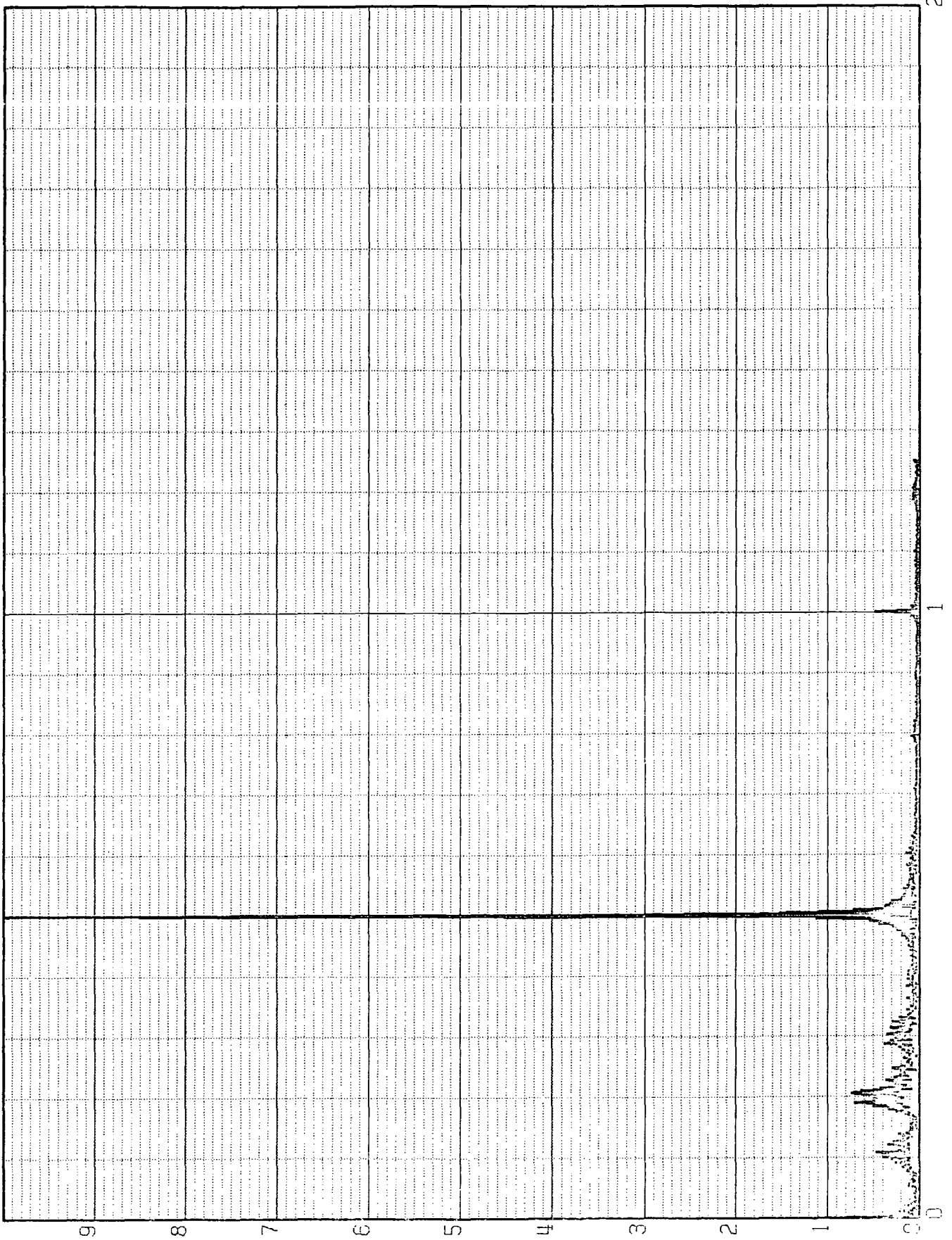


Fig 5.4: Spectral Analysis of channel 7 from
Displayed Shot Record

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.

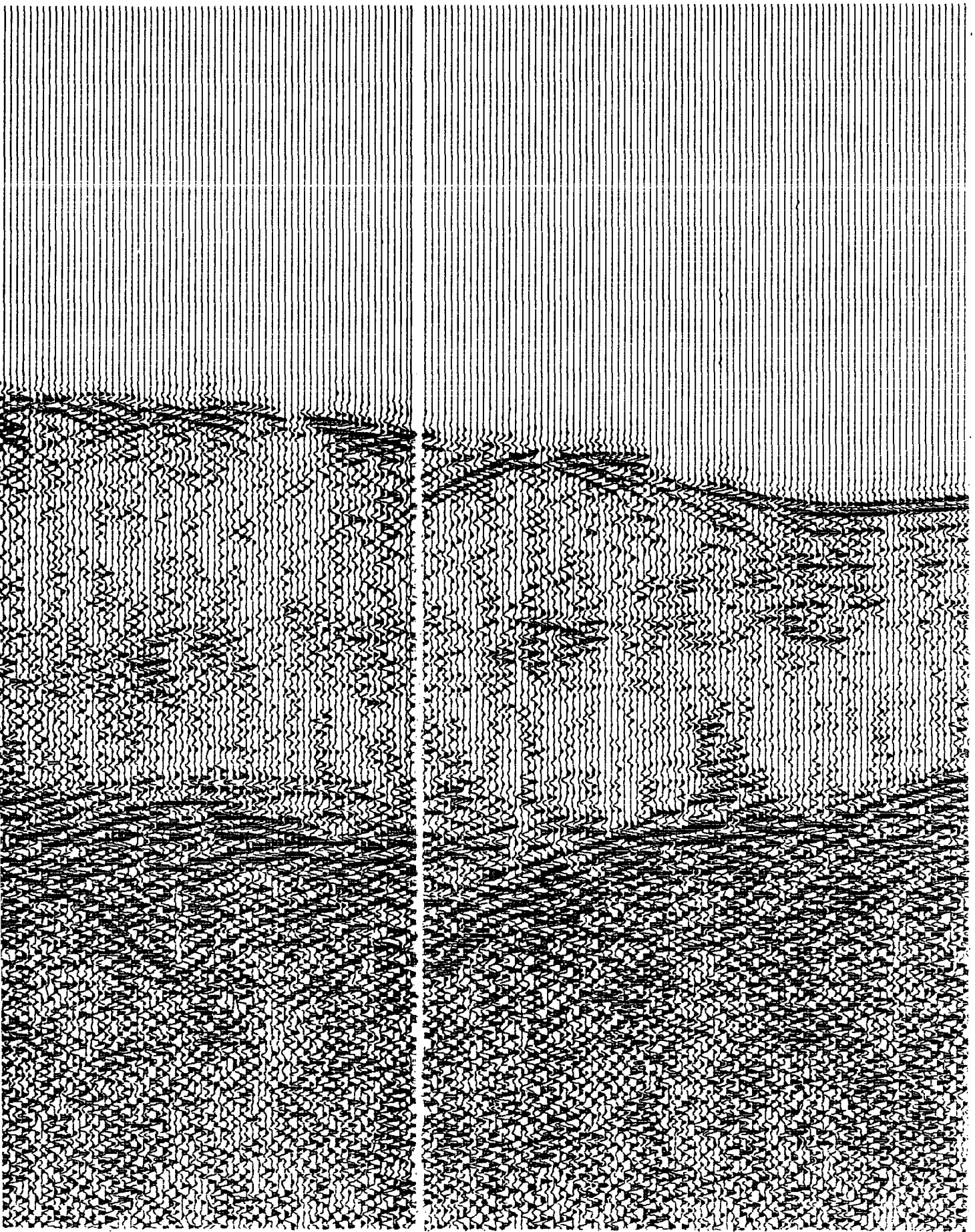


Fig 5.5: A single channel Display of a Portion of
the Caribbean line after Pre-Stack Processing

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

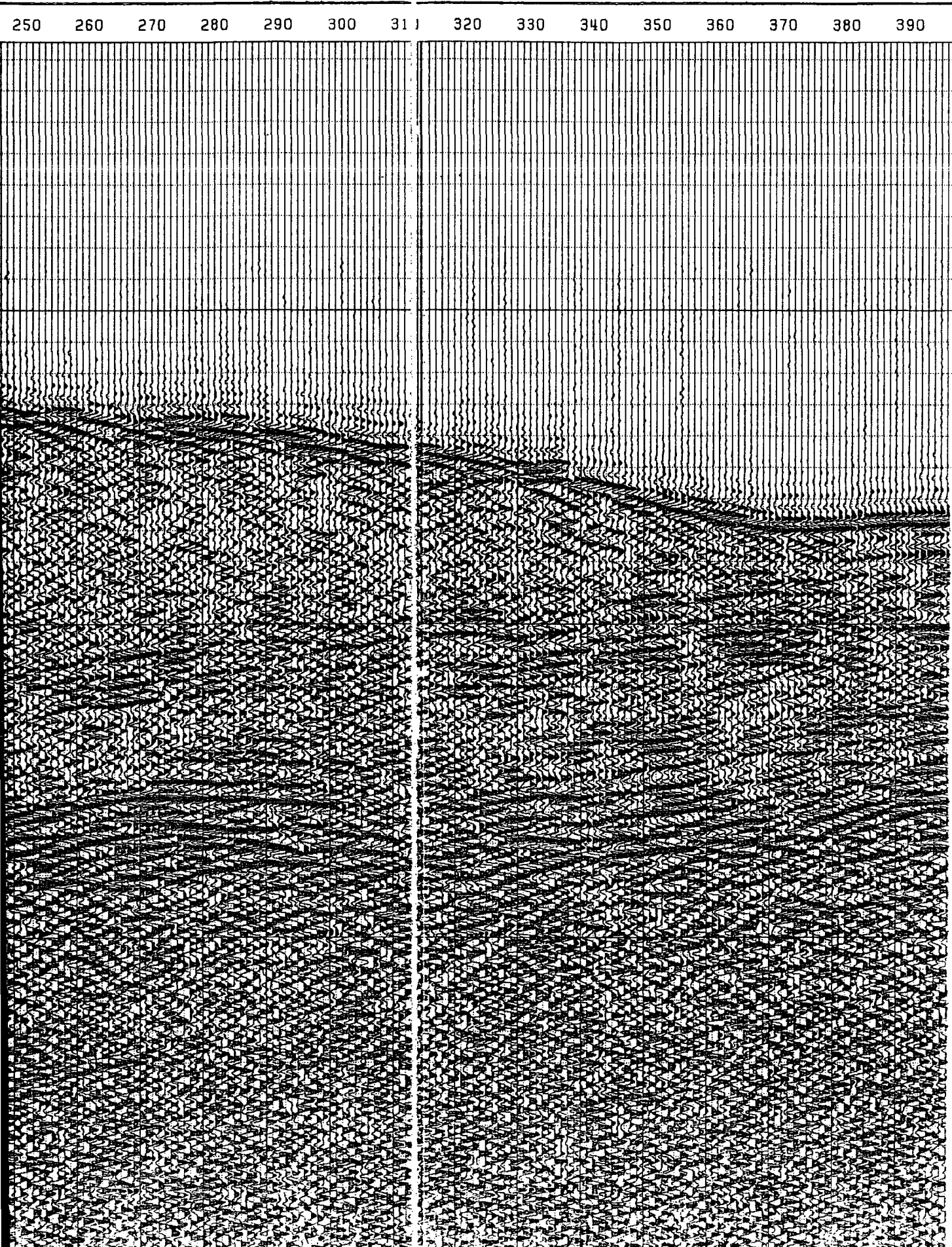


Fig 5.6: A Portion of the Caribbean line After Stack

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.

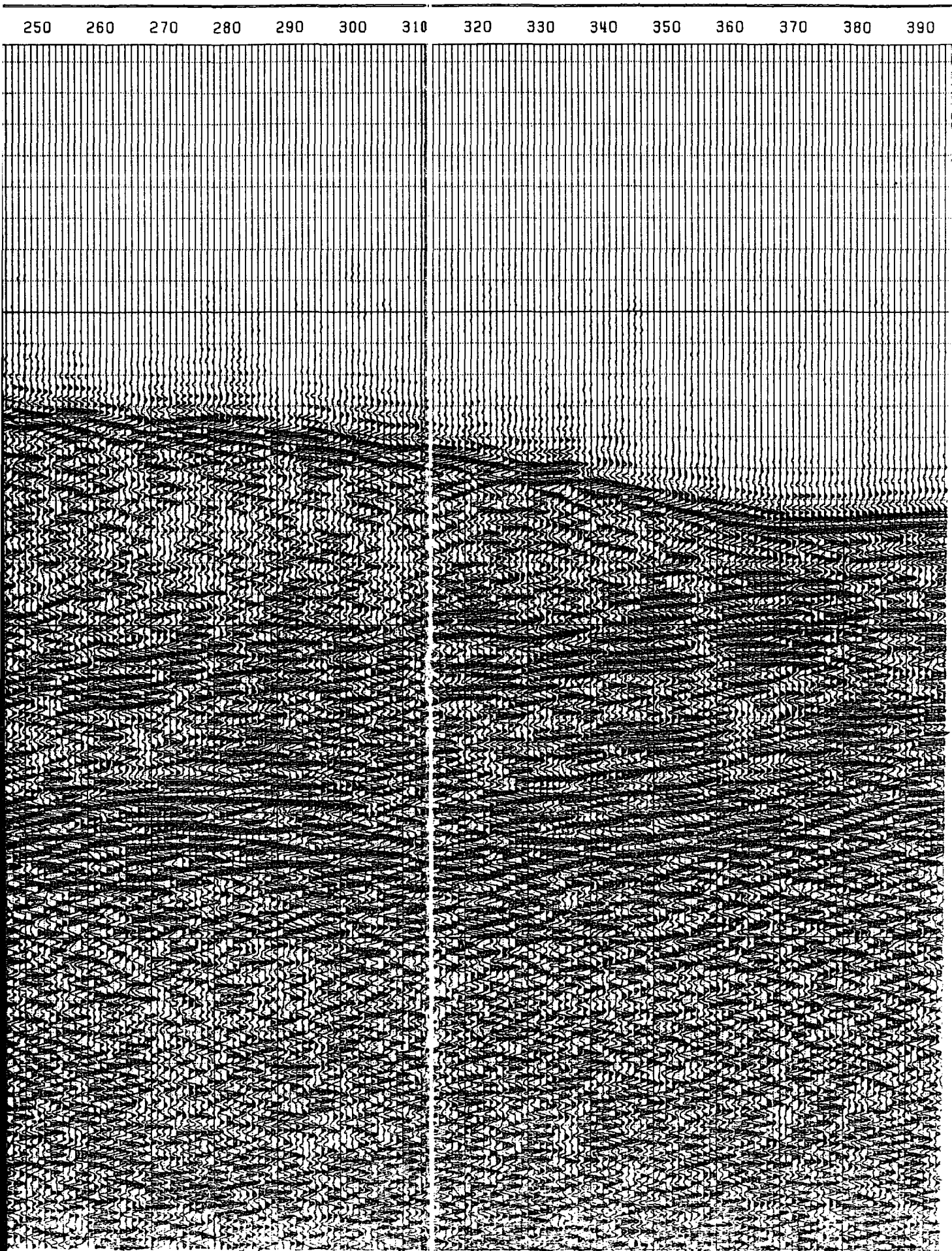


Fig 5.7: A Portion of the Caribbean line after all Processing

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 tE0.2t
- 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 be 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)

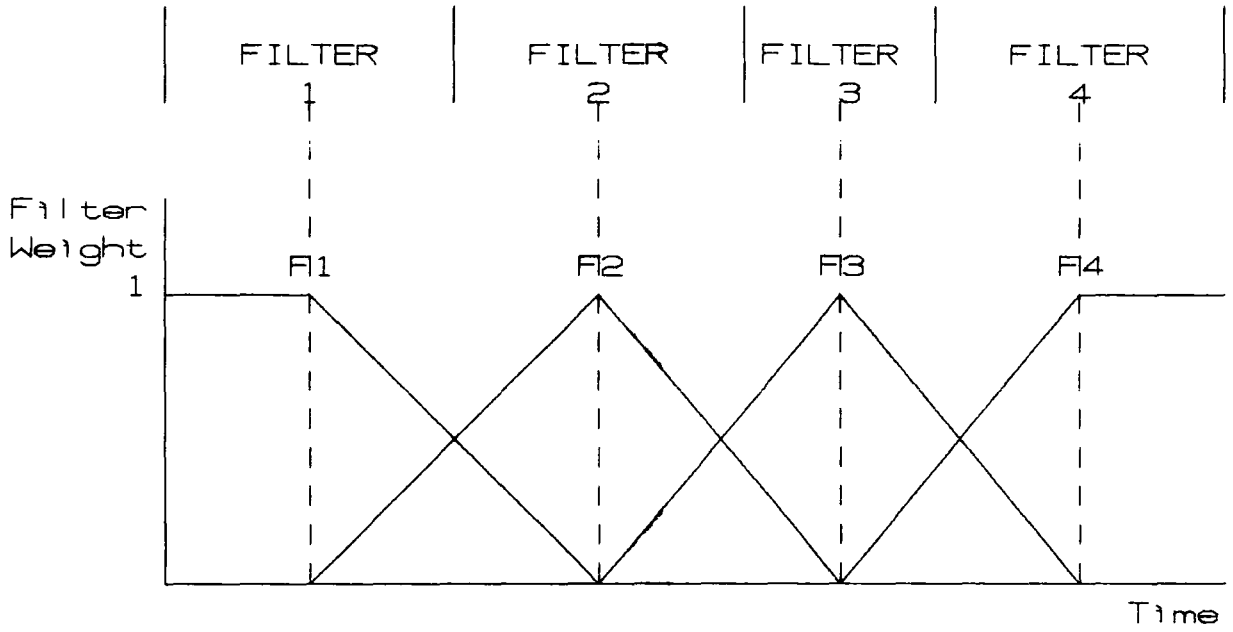


Fig 6.1 : Application of Time Variant Filters

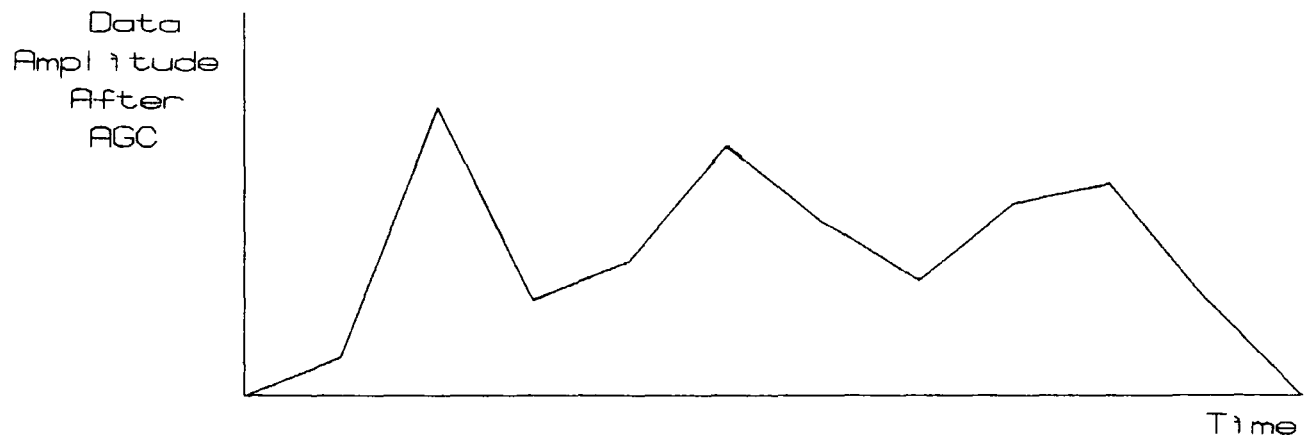
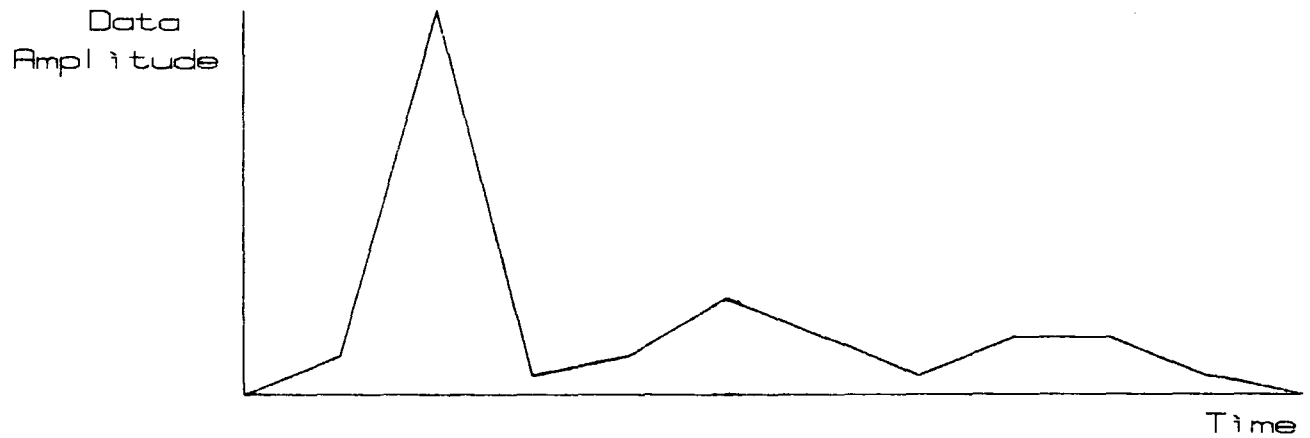


Fig 6.2 : Example of AGC for Trace scaling

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 ^{long-record}/~~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 ^{long-record}/~~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

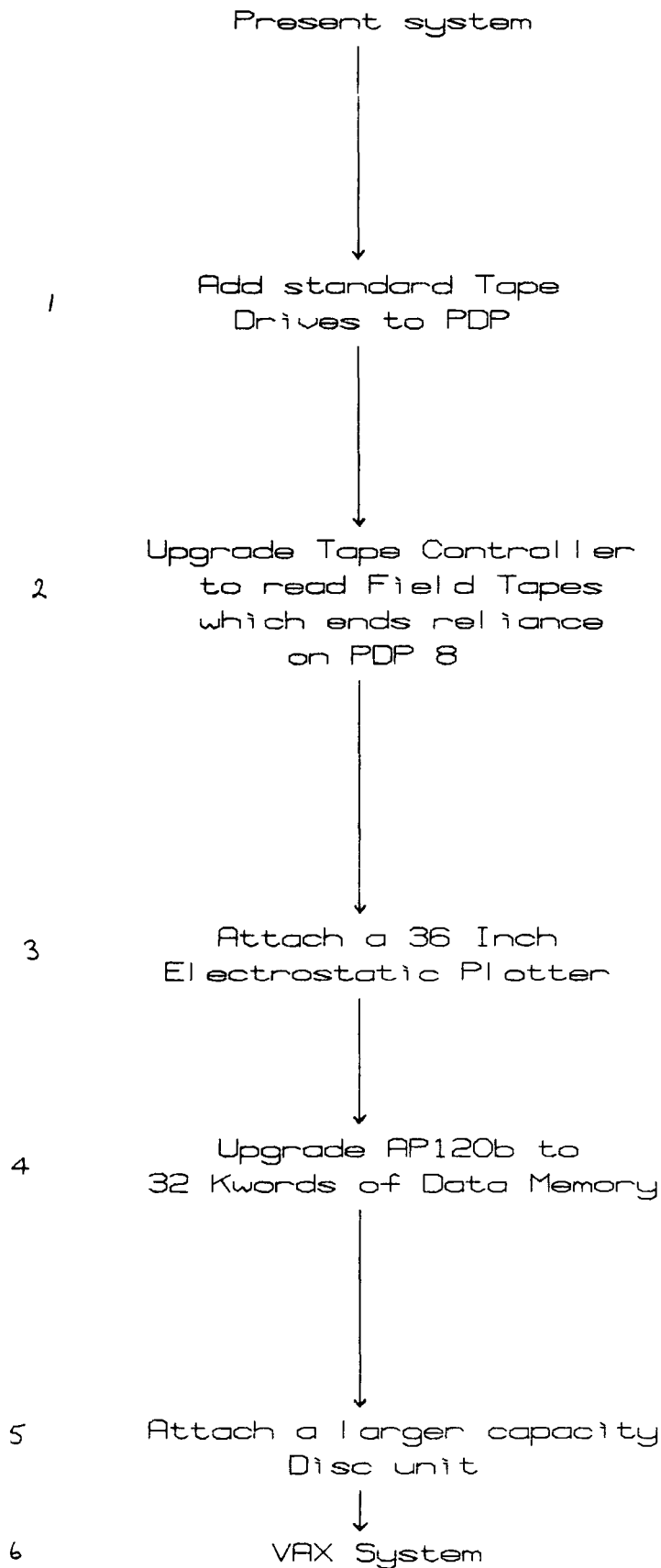


Fig 6.3 : Envisaged System upgrade path

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 upto 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.

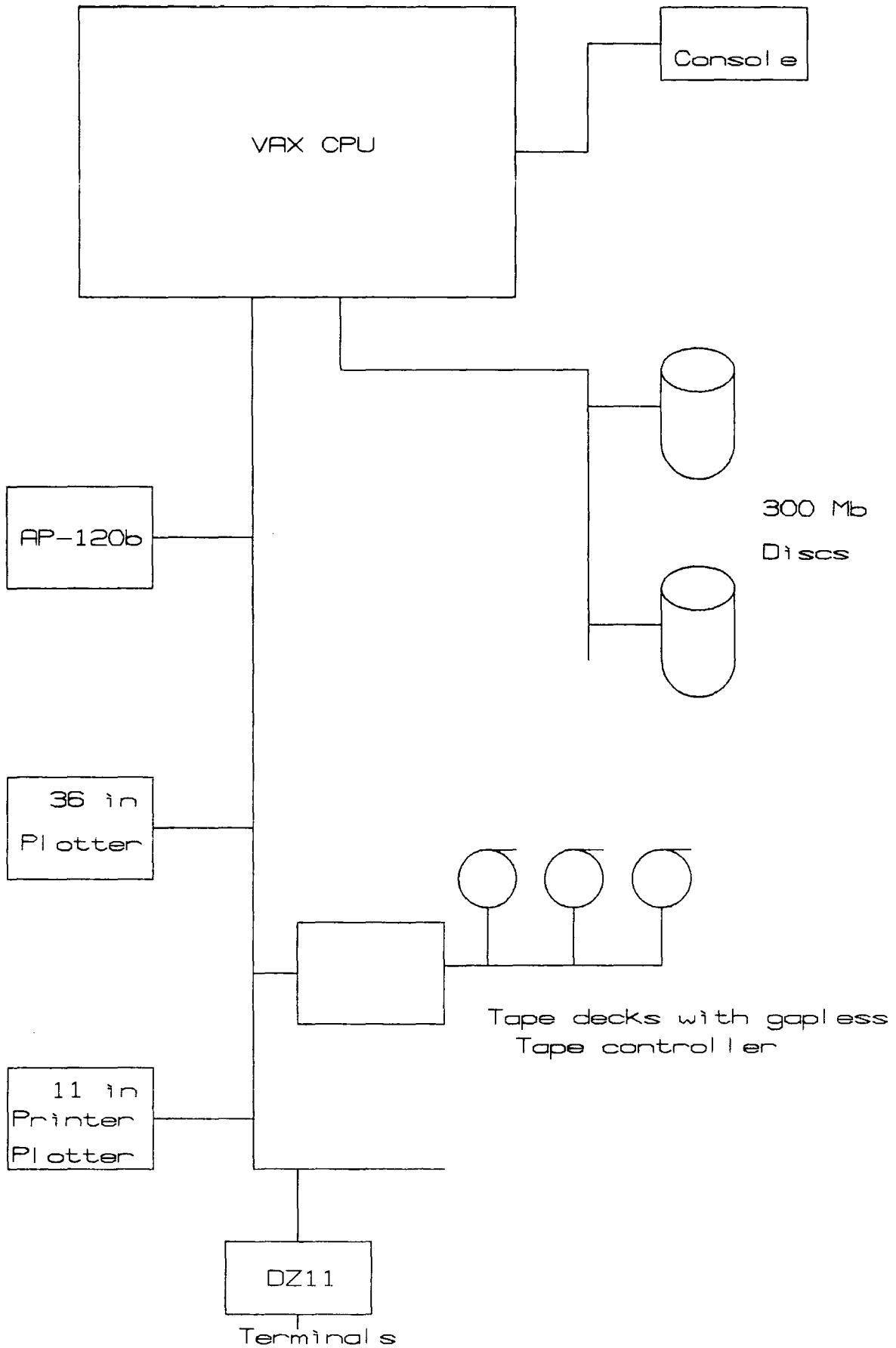


Fig 6.4 : Suggested Future Computer System

The VAX is a 32-bit word computer and, depending on model, has upto 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 26 750-772

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 propogation

Geophys 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

3rd Edition New York McGraw-Hill

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

Wiggins R A, Larner K L and Wisecup R D 1976

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.....NFILES 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

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

FPOS.....Output sort position for each input channel

READ(1,1001)(VELAN(I),I=1,VELNUM)

VELAN....File numbers to be saved on disc

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

FNBUF....File names for sort file buffers, always at least 1

READ(1,1000)(VELNAM(I),I=1,VELNUM)

VELNAM...File names for files to be saved on disc

READ(1,1001)NOCHAN,IGCODE,IUNITS,ISCODE

NOCHAN...Number of active channels recorded in field data

IGCODE...Output gather code

- 0 - Common shot gather
- 1 - CMP gather
- 2 - Single channel stacked
- 3 - Single channel unstacked

IUNITS...Units of measurement

- 1 - Metres
- 2 - Feet

ISCODE...Acquisition 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

C DEMULTIPLEX PROGRAM

C THIS PROGRAM TAKES DATA OFF TAPE(VIA PDP-8), OR DISC
 C IN THE MULTIPLEXED FORMAT USED AT DURAM UNIV
 C AND DEMULTIPLEXES THE CHANNELS AND REFORMATS THE DATA
 C VALUES. THESE ARE THEN WRITTEN TO DISC FOR STORAGE
 C BEFORE BEING ASSEMBLED INTO A STACK POINT GATHER.
 C THEY ARE THEN EITHER LEFT ON DISC FOR A VELOCITY ANALYSIS
 C OR WRITTEN (VIA THE PDP-8) TO TAPE FOR STORAGE.

C DATA STORAGE DECLARATIONS.

```

0001 VIRTUAL BUFF(8448),FNAMES(30),VELNAM(60),RTNAM(60)
0002 REAL*8 FNAMR,FNAMES,VELNAM,DBLK(2),FMBUF,RTNAM
0003 REAL*4 DEVNAM,BUFOUT(256),FNBUF(3)
0004 INTEGER*2 FNUM,CHOFF(30),FPOS(30),NBLKOF(30),IBLKOF(30),INDEX(31),
%USEFLG,QNUM,VNUM,BLK,BSST,EGAINS(30),GCNT,
%SPOS,FST,BSST1,BLST,OLAP,RST,ICHAN(30),
%GSAVE(30),EOFFLG,MASK,SYNC,BUFF,HSBLKW(9),VELAN(60),GAINS(3840),
%OUTFLG,INFLG,FLEN,VELNUM
0005 LOGICAL*1 STATUS,RC,ITLEN,IGCODE,IUNITS,NOCHAN,
%TLEN,TPDRR,TPDRW,HSBLK(256)
0006 COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
%IERFLG,SPOS,RC
0007 COMMON /DECOM/BUFOUT,CHOFF,ICHAN,FPOS,NCHAN,NBLKOF,INDEX,GCNT,
#IHSTRT,IHSEC
0008 COMMON /BUFCOM/FBSST1,FBLST,F256,F256D,FBSST,F1,F4097,
%BSST1,BLST,OLAP,RST,BLK
0009 COMMON/BUFS/NUERR,NALOW,ITIC
0010 EQUIVALENCE(HSBLKW(1),HSBLK(1)),(SROFF,HSBLK(21)),
%(SLSPAC,HSBLK(29)),(STSPAC,HSBLK(33)),(SDEPT,HSBLK(37)),
%(RDEPT,HSBLK(41)),(NOCHAN,HSBLK(12)),(IGCODE,HSBLK(19)),
%(IUNITS,HSBLK(20)),(ISCODE,HSBLK(45)),(IBFREE,HSBLK(47)),
%(RSPAC,HSBLK(25))
0011 DATA DEVNAM/3RDK /
0012 DATA CHOFF/4352,4480,4608,4736,4864,4992,
%5120,5248,5376,5504,5632,5760,5888,6016,6144,6272,
%6400,6528,6656,6784,6912,7040,7168,7296,7424,7552,
%7680,7808,7936,8064/
0013 DATA MASK,SYNC/"17,"177777/
0014 DATA HSBLK(49)/"377/,HSBLK(50)/"377/,HSBLK(255)/"377/,
#HSBLK(256)/"377/

```

C
 C CONSTANTS USED IN PROGRAM

```

0015 IPADNO=0
0016 IEOTR=0
0017 IEOTW=0
0018 BSST=4096+256
0019 IBIAS=15
0020 FNUM=1
0021 IAD=BSST+1

```

```

0022      VNUM=1
0023      F1=ADGET(BUFF(1))
0024      F4097=ADGET(BUFF(4097))
0025      FBSST=ADGET(BUFF(IAD))
0026      F256D=ADGET(BUFF(257))
0027      F256=APGAD(BUFF(257))

C
C   SET UP THE RT-11 INPUT-OUTPUT PROCEDURES
C
0028      IF(ICDFN(50).NE.0)STOP'INSUFFICIENT CHANNEL FREE SPACE'
C   INITIALIZE THE AP
C
0030      CALL APINIT
C   READ IN THE NECESSARY INPUT DATA
C
0031      CALL ASSIGN(1,'DK1:MPDMXD.SPF',14)
0032      CALL ASSIGN(2,'DK1:MPDMXD.LOG',14)
0033      READ(1,1001)NCHAN,NFILES,ITSIZ,IHSTRT,NROW,TPDRR,TPDRW,VELNUM
0034      1001  FORMAT(12I5)
0035      READ(1,1001)USEFLG,OUTFLG,INFLG,IERFLG,NRECOV,NUERR,NALOW
0036      IF(INFLG.NE.0)GOTO 1
0038      READ(1,1002)FNBUF
0039      1002  FORMAT(3A4)
0040      CALL IRAD50(12,FNBUF,FNAMR)
0041      GOTO 2
0042      1 DO 3 IRD=1,NFILES
0043      READ(1,1002)FNBUF
0044      CALL IRAD50(12,FNBUF,FMBUF)
0045      RTNAM(IRD)=FMBUF
0046      3 CONTINUE
0047      2 CONTINUE
0048      READ(1,1001)(INDEX(I),I=1,NROW)
0049      READ(1,1001)(ICHAN(I),I=1,NCHAN)
0050      READ(1,1001)(FPOS(I),I=1,NCHAN)
0051      READ(1,1001)(VELAN(I),I=1,VELNUM)
0052      DO 5 IPL=1,NROW
0053      READ(1,1002)FNBUF
0054      CALL IRAD50(12,FNBUF,FMBUF)
0055      5 FNAME(IPL)=FMBUF
0056      DO 6 ILP=1,VELNUM
0057      READ(1,1002)FNBUF
0058      CALL IRAD50(12,FNBUF,FMBUF)
0059      6 VELNAM(ILP)=FMBUF

C
C   READ IN HEADER INFO
C
0060      READ(1,1001)NOCHAN,IGCODE,IUNITS,ISCODE
0061      READ(1,1003)SROFF,RESPAC,SLSPAC,STSPAC,SDEPT,RDEPT
0062      1003  FORMAT(6F10.0)
0063      READ(1,1004)(HSBLK(I),I=51,254)
0064      1004  FORMAT(80A1)
0065      HSBLKW(7)=NCHAN
0066      HSBLKW(8)=IHSTRT*128-128
0067      HSBLKW(9)=ITSIZ*128

```

```

C
C   SET UP CONSTANTS AND REST OF RT-11
C   INPUT OUTPUT ROUTINES.
C
0063     ITSIZO=ITSIZ-IHSTRT+1
0069     IBLKOF(1)=1
0070     DO 10 J=2,NCHAN
0071     10 IBLKOF(J)=IBLKOF(J-1)+ITSIZO
0072     LSTBLK=(ITSIZO*NCHAN)
0073     IFSIZ=LSTBLK+1
0074     ITLEN=ITSIZO/2
0075     QNUM=NCHAN+2
0076     IF(IQSET(QNUM).NE.0)STOP'QSET ERROR'
0078     IFET=IFETCH(DEVNAM)
0079     IF(IFET.NE.0)TYPE 1009,IFET
0081     1009 FORMAT(' FETCH RETURN=',I2)
0082     IF(IFET.NE.0)STOP ' BAD HANDLER FETCH'
C
C   STACK FILE ORGANISATION
C
0084     IF(USEFLG.NE.0)GOTO 20
0086     DO 15 JJ=1,NROW
0087     FMBUF=FAMES(JJ)
0088     IF(IENTER(22+JJ,FMBUF,IFSIZ).LT.0)STOP' ENTER ERROR'
0090     IF(IWRITE(256,BUFOUT,LSTBLK,22+JJ).LT.0)STOP'WRITE ERROR'
0092     CALL CLOSEC(22+JJ)
0093     15 CONTINUE
0094     20 DO 30 L=1,NROW
0095     FMBUF=FAMES(L)
0096     30 IF(LOOKUP(22+L,FMBUF).LT.0)STOP'LOOKUP ERROR'
C
C   START OF THE MAIN DEMULTIPLEX LOOP
C
0098     DO 999 I=1,NFILES
0099     IERR=0
0100     IFNUM=I
C
C   FILE ORGANISATION ON A NORMAL RUN
C
0101     DO 35 M=1,NCHAN
0102     35 NBLKOF(M)=IBLKOF(M)
0103     INDEX(NROW+1)=INDEX(1)
0104     DO 40 MM=1,NROW
0105     40 INDEX(MM)=INDEX(MM+1)
C
C   SEE IF FILES TO BE ZEROED
C
0106     IF(IPADNO.GT.0)GOTO 100
0108     IF(INFLG.NE.0)GOTO 45
C
C   IF THE DATA IS TO BE READ FROM TAPE THE ROUTINE TAPRED
C   IS USED IN ORDER TO COMMUNICATE WITH THE PDP-8 AND ALSO
C   TO DO A FAST FILE TRANSFER BOTH 8->11 AND 11->8.
C   THE ROUTINE ALSO RETURNS THE STATUS BYTE FOR ERROR

```

```

C ANALYSIS BY THE ROUTINE TAPRED.
C
C OPEN FILE ON CH20 FOR SDS10
C
0110 IN=IENTER(20,FNAMR,-1)
0111 IF(IN.LT.0)WRITE(7,*)IN
0112 IF(IN.LT.0)STOP'FNAMR ENTER ERROR'
C
C CHECK ARNT AT EOT
C
0115 IF(IEOTR.GE.0)GOTO 43
0117 WRITE(7,1060)TPDRR,IFNUM
0118 1060 FORMAT(' EOT ENCOUNTERED ON DRIVE:',I2,' FILE NO:',I4)
0119 WRITE(7,1061)
0120 1061 FORMAT(' ENTER NEW READ DRIVE NO:',S)
0121 READ(5,1062)TPDRR
0122 1062 FORMAT(I1)
0123 IEOTR=0
0124 IF(TPDRR.GT.2)GOTO 265
C
C CHECK IF ZERO FILES TO BE ADDED AT END OF TAPE
C
0125 WRITE(7,1063)
0127 1063 FORMAT(' ENTER NO OF ZEO FILES TO BE ADDED(I2):',S)
0128 READ(5,1064)IPADNO
0129 1064 FORMAT(I2)
0130 IF(IPADNO.GT.0)GOTO 100
C
C DO A READ
C
0132 43 CALL TAPRED(-1,TPDRR,STATUS,TLEN,FLEN,IFNUM,IEOTR)
0133 IF(STATUS.LT.0)WRITE(2,1050)IFNUM
0135 1050 FORMAT(' WARNING FILE NO ',I4,' RETRIES FAILED')
C
C DO A WIND
C
0136 IF(IEOTR.LT.0)GOTO 46
0138 CALL TAPRED(0,TPDRR,STATUS, , ,IFNUM,IEOTR)
C
C START OF MAIN BUSINESS
C
0139 46 CALL IWAIT(20)
0140 CALL CLOSEC(20)
0141 45 BLK=0
0142 GCNT=0
0143 IHSEC=0
0144 EOFFLG=0
0145 RC=.FALSE.
0146 IF(INFLG.EQ.0)IOPEN=LOOKUP(20,FNAMR)
0147 IF(INFLG.NE.0)FMBUF=RTNAM(FNUM)
0148 IF(INFLG.NE.0)IOPEN=LOOKUP(20,FMBUF)
0149 IF(IOPEN.LT.0)WRITE(7,*)IOPEN
0150 IF(IOPEN.LT.0)STOP'FNAMR LOOKUP ERROR'

```

```

C
C READ IN FIRST DATA BLOCK FROM DISC AND EXTRACT THE
C HEADER INFORMATION
C
0156 IF(IREADA(20,BSST,BLK,F1).LT.0)STOP ' READ ERROR'
0158 IF(IWAIT(20).LT.0)STOP ' WAIT ERROR'
0160 BLK=BLK+17
0161 IF(IREADA(20,4096,BLK,FBSST).LT.0)STOP 'READX ERROR'
0163 BLK=BLK+16
C
C END OF INITIAL READS
C BEGINNING OF HEADER BLOCK SCANS
C
0164 DO 50 LL=1,4
0165 50 HSBLKW(LL)=ISWAP(BUFF(LL)).OR."30060
0166 HSBLKW(5)=ISWAP(BUFF(5))
0167 HSBLK(11)=TLEN
0168 ITIC=2*HSBLK(9)
0169 IGJL=1
0170 DO 60 JL=1,30
0171 EGAINS(JL)=BUFF(5+JL).AND.MASK
0172 GSAVE(JL)=EGAINS(JL)+IBIAS
0173 GAINS(IGJL)=GSAVE(JL)
0174 IGJL=IGJL+128
0175 60 CONTINUE
0176 SPOS=35
0177 70 SPOS=SPOS+1
0178 IF(BUFF(SPOS).EQ.0)GO TO 70
C
C BUFFER CONSTANTS SET UP AFTER END OF HEADER BLOCK IS LOCATED
C
0180 FST=SPOS
0181 BSST1=4096+FST
0182 BLST=8192+FST
0183 OLAP=257-FST
0184 RST=4096-OLAP
C
C CHECK SYNC WORDS FOR ERRORS AND EXTRACT THE GAINS READY
C FOR USE AS INTEGERS.
C
0185 IFDIR=IDIRG(BUFF(SPOS+1))
0186 IF(IERFLG.NE.0.OR.IERR.NE.0)CALL BUFSCN(BUFF,0,IFNUM)
0188 IF(IERR.LT.0)GOTO 100
C
C PUT FIRST BUFFER INTO AP
C
0190 F=APGAD(BUFF(FST))
0191 CALL APPUTX(0,4096,1)
C
C FIND ADDRESS PAIRS FOR XM OPERATIONS
C
0192 FBSST1=APGAD(BUFF(BSST1))
0193 FBLST=APGAD(BUFF(BLST))
0194 IF(IERFLG.NE.0.OR.IERR.NE.0)GOTO 220

```

```

C-----
C   MAIN DEMUX LOOP FOR A SINGLE FILE
C-----
C
C   COME HERE IF WANT TO DEMUX ALL THE FILE
C   WITHOUT FULL ERROR CHECKING SWITCHED ON?
0196 210 CONTINUE
0197     IHSEC=IHSEC+1
0198     CALL DMX
0199     IF(IERR.NE.0)GOTO 215
0201     IF(IHSEC.GE.ITSIZ)GOTO 230
0203     IF(EOFFLG.EQ.2)GO TO 230
0205     CALL DBLBUF(BUFF)
0206     GO TO 210
0207 215 CALL IWAIT(20)
0208     WRITE(2,1030)IFNUM
0209 1030 FORMAT(' ERROR ON FILE:',I4,' FOUND GOING INTO RECOVERY MODE')
0210     IF(NRECOV.EQ.0)GOTO 100
0212     DO 216 LLZ=1,NCHAN
0213 216 NBLKOF(LLZ)=IBLKOF(LLZ)
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     IHSEC=IHSEC+1
0217     IF (IHSEC.GE.IHSTRT)CALL DEMUX
0219     IF(IHSEC.GE.ITSIZ)GOTO 230
0221     IF(EOFFLG.EQ.2)GOTO 230
0223     CALL BUFSCN(BUFF,1,IFNUM)
0224     IF(IERR.GE.0)GOTO 220
C-----
C   END OF MAIN DEMUX LOOP
C-----
C
C   BLANK PARTS OF FILES WITH FATAL ERRORS
0226 100 CALL VCLR(0,1,128)
0227     CALL APWR
0228     CALL APGET(BUFOUT,0,128,2)
0229     WRITE(2,1020)IFNUM
0230 1020 FORMAT(' FILE NUMBER ',I4,' DELETED')
0231     CALL APWD
0232     DO 125 JZ=1,NCHAN
0233     NCH=INDEX(FPOS(JZ))
0234     IBLK=IBLKOF(JZ)
0235     DO 125 LZ=1,ITSIZ0
0236     IF(IWRITW(256,BUFOUT,IBLK,22+NCH).LT.0)STOP 'CLEAR ERR'
0238     IBLK=IBLK+1
0239 125 CONTINUE
0240     IPADNO=IPADNO-1
C
C   WRITE OUT HEADER BLOCK AND CLOSE DOWN COMPLETED GATHER FILE

```

```

C
0241 230 FMBUF=FNAMES(INDEX(1))
0242 IF(IWRITE(128,HSBLK,0,22+INDEX(NROW)).LT.0)STOP'HSBLK ERROR'
0244 IF(USEFLG.EQ.0.AND.FNUM.LT.NROW)GO TO 998
0246 CALL CLOSEC(22+INDEX(1))
0247 IF(OUTFLG.NE.0)GOTO 240
C
C WRITE OUT GATHER FILE TO TAPE
C
0249 FLEN=LOOKUP(21,FMBUF)
0250 IF(FLEN.LT.0)STOP' FMBUF LOOKUP ERR'
0252 CALL TAPRED(1,TPDRW,STATUS,TLEN,FLEN,IFNUM,IEOTW)
0253 CALL CLOSEC(21)
0254 IF(IEOTW.GE.0)GOTO 235
0256 WRITE(7,1080)TPDRW,IFNUM
0257 1080 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
0258 WRITE(7,1081)
0259 1081 FORMAT(' ENTER NO OF NEW WRITE DRIVE:',S)
0260 READ(5,1062)TPDRW
0261 IEOTW=0
0262 IF(TPDRW.GT.2)GOTO 265
0264 235 IF(STATUS.GE.0)GOTO 240
0265 WRITE(2,1070)IFNUM
0267 1070 FORMAT(' WRITE ON FILE ',I4,' FATAL ERROR')
0268 GOTO 265
0269 240 IF(FNUM.NE.VELAN(VNUM))GOTO 250
C
C SET UP A VELOCITY ANALYSIS FILE
C
0271 DBLK(1)=FMBUF
0272 DBLK(2)=VELNAM(VNUM)
0273 IF(IRENAM(21,DBLK).GT.0)STOP 'RENAME ERROR'
0275 VNUM=VNUM+1
0276 IF(IENTER(22+INDEX(1),FMBUF,IFSIZ).LT.0)STOP'ENTER ERROR'
0278 IF(IWRITE(256,BUFOUT,LSTBLK,22+INDEX(1)).LT.0)STOP'WRITE ERROR'
0280 CALL CLOSEC(22+INDEX(1))
0281 250 CONTINUE
0282 IF(LOOKUP(22+INDEX(1),FMBUF).LT.0)STOP'LOOKUP IND ERROR'
C
C CLOSE DOWN FILE 20 AND GO TO NEXT INPUT FILE IF REQU'D
C
0284 998 FNUM=FNUM+1
0285 CALL CLOSEC(20)
0286 999 CONTINUE
0287 265 DO 260 LL=1,NROW
0288 260 CALL CLOSEC(22+LL)
0289 WRITE(2,1001)(INDEX(I),I=1,NROW)
0290 STOP'NORMAL TERMINATION'
0291 END
```

```

0001      SUBROUTINE DEMUX
0002      INTEGER*2 CHOFF(30),GANADD,FPOS(30),NBLKOF(30),
%EOFFLG,GAINS(3840),ICHAN(30),GSAVE(30),INDEX(31),GCNT
0003      REAL*4 BUFOUT(256)
0004      LOGICAL*1 RC
0005      COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
%IERFLG,ISPOS,RC
0006      COMMON /DECOM/BUFOUT,CHOFF,ICHAN,FPOS,NCHAN,NBLKOF,INDEX,GCNT,
%IHSTRT,IHSEC
C
C      DO DEMUX AND BRING IN THE SAVED GAINS
C
0007      NCH=0
0008      NIN=128
0009      IF(EOFFLG.EQ.2)NIN=NSMPIN
0010      NOUT=2*NIN
0011      IF(NOUT.EQ.0)RETURN
0012      CALL APWD
0013      CALL VFLT(4352,1,4352,1,3840)
0014      CALL APWR
0015      CALL APPUT(GAINS,0,3840,1)
C
C      FORM THE DEMUXED NOS INTO R*4 RERPRESENTATION
C      FOR EACH CHANNEL IN TURN AND THEN WRITE THEM
C      OUT TO DISC. THIS IS DONE FOR 128 SAMPLES OF
C      EACH CHANNEL WHICH ARE EXPECTED TO BE IN THE A.P.
C
0018      CALL APWD
0019      DO 10 NJ=1,NCHAN
0020      NCHADD=CHOFF(ICHAN(NJ))
0021      GANADD=NCHADD-4352
0022      CALL VBINSC(NCHADD,1,NCHADD,1,GANADD,1,NIN)
0023      CALL APWR
0024      CALL IWAIT(22+NCH)
0025      NCH=INDEX(FPOS(NJ))
0026      CALL APGET(BUFOUT,NCHADD,NIN,2)
0027      CALL APWD
0028      IF(IWRITE(NOUT,BUFOUT,NBLKOF(NJ),22+NCH).LT.0)STOP'DEMUX ERROR'
0029      NBLKOF(NJ)=NBLKOF(NJ)+1
0030      10 CONTINUE
0031      CALL IWAIT(22+NCH)
0032      RETURN
0033      END
0034

```

```

0001      SUBROUTINE DMX
0002      INTEGER*2 CHOFF(30),FPOS(30),NBLKOF(30),GCNT,EOFFLG,
%GAINS(3840),ICHAN(30),GSAVE(30),INDEX(31)
0003      REAL*4 BUFOUT(256)
0004      LOGICAL*1 RC
0005      COMMON /SUBS/ GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
%IERFLG,ISPOS,RC
0006      COMMON /DECOM/BUFOUT,CHOFF,ICHAN,FPOS,NCHAN,NBLKOF,INDEX,GCNT,
#IHSTR, IHSEC
C
C      DO DEMUX AND BINARY SCALING
C
0007      NCH=0
0008      NIN=128
0009      IF(EOFFLG.EQ.2)NIN=NSMPIN
0011      NOUT=2*NIN
0012      IF(NOUT.EQ.0)RETURN
0014      CALL APWD
0015      CALL APPUT(GSAVE,4096,30,1)
0016      CALL APWD
0017      CALL DMXA(4096,4352,128,GCNT,IFDIR,NIN)
0018      CALL APWR
0019      CALL APGSP(IERR,15)
0020      IF(IERR.NE.0)IERR=1
0022      IF(IERR.NE.0)RETURN
0024      CALL APGSP(GCNT,3)
0025      CALL APGSP(IFDIR,4)
0026      CALL APGET(GSAVE,4096,30,1)
0027      CALL IWAIT(20)
0028      IF(IHSEC.LT.IHSTR)GOTO 30
0030      NCHADD=CHOFF(ICHAN(1))
0031      CALL APGET(BUFOUT(1),NCHADD,NIN,2)
0032      IIN=129
0033      IOUT=1
0034      IF(NCHAN.EQ.1)GOTO 20
C
C      EXTRACT EACH WANTED CHANNEL AND PUT ON DISC
C
0036      DO 10 NJ=2,NCHAN
0037      NJ1=NJ-1
0038      NCHADD=CHOFF(ICHAN(NJ))
0039      CALL IWAIT(22+NCH)
0040      CALL APGET(BUFOUT(IIN),NCHADD,NIN,2)
0041      NCH=INDEX(FPOS(NJ1))
0042      IF(IWRITE(NOUT,BUFOUT(IOUT),NBLKOF(NJ1),22+NCH).LT.0)STOP 'DMX'
0044      NBLKOF(NJ1)=NBLKOF(NJ1)+1
0045      IT=IIN
0046      IIN=IOUT
0047      IOUT=IT
0048      10 CONTINUE
0049      20 NCH=INDEX(FPOS(NCHAN))
0050      CALL APWD
0051      IF(IWRITE(NOUT,BUFOUT(IOUT),NBLKOF(NCHAN),22+NCH).LT.0)STOP 'DMX'
0052      NBLKOF(NCHAN)=NBLKOF(NCHAN)+1

```

```

0054      30 RC=.NOT.RC
0055      CALL IWAIT(22+NCH)
0056      RETURN
0057      END

```

```

0001      SUBROUTINE DBLBUF(BUFF)
0002      VIRTUAL BUFF(8448)
0003      INTEGER*2 BUFF,EOFFLG,BSST,BLST,SPOS,BSST1,
0004      %RST,SYNC,OLAP,BLK,GSAVE(30),GAINS(3840)
0005      LOGICAL*1 RC
0006      COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
0007      %IERFLG,ISPOS,RC
0008      COMMON /BUFCOM/FBSST1,FBLST,F256,F256D,FBSST,F1,F4097,
0009      %BSST1,BLST,OLAP,RST,BLK
0010      DATA SYNC /"177777"/
0011
0012      C
0013      C THIS ROUTINE CONTROLS THE DOUBLE BUFFERING SCHEME USED
0014      C TO TAKE DATA FROM DISC AND PUT IT IN TO THE AP
0015      C WHEN IT IS NEEDED. IT IS ALSO RESPONSIBLE FOR CHECKING
0016      C THE SYNC WORDS AND EXTRACTING THE GAINS AS INTEGERS FOR USE
0017      C
0018      C
0019      C SET UP THE START OF THE BUFFER
0020      C
0021      SPOS=BSST1
0022      IF(.NOT.RC)SPOS=BLST
0023
0024      C DO A BUFFER SCAN WHEN THE INPUT PROCEDURE HAS NOTIFIED EOF
0025      C
0026      IF(EOFFLG.LE.0)GOTO 50
0027      ISMPIN=0
0028      IPOS=SPOS
0029      DO 40 L=1,NSMPIN
0030      IF(BUFF(IPOS).NE.SYNC)GO TO 60
0031      IPOS=IPOS+32
0032      IF(IPOS.GT.8448)IPOS=IPOS-8192
0033      ISMPIN=ISMPIN+1
0034      40 CONTINUE
0035      GOTO 50
0036      60 NSMPIN=ISMPIN
0037      50 CONTINUE
0038
0039      C
0040      C PUT A BUFFER INTO THE A.P. AND START THE READ TO FILL
0041      C THE SECOND BUFFER FOR USE NEXT TIME
0042      C
0043      IF(RC)CALL APPUTA(0,4096,1,FBSST1)
0044      IF(.NOT.RC)CALL APPUTA(0,OLAP,1,FBLST)
0045      IF(.NOT.RC)CALL APPUTA(OLAP,RST,1,F256)
0046      IF(EOFFLG.GT.0)EOFFLG=2
0047      IF(EOFFLG.GT.0)RETURN
0048      IF(EOFFLG.LT.0)EOFFLG=1
0049      IF(EOFFLG.GT.0)RETURN
0050      FINP=F256D
0051      IF(.NOT.RC)FINP=FBSST
0052      IIN=IREADA(20,4096,BLK,FINP)
0053      BLK=BLK+16
0054
0055      C
0056      C CHECK INFO RETURNED FROM INPUT ROUTINE
0057      C FOR ERRORS AND AN EOF SITUATION

```

```

0058      C
0059      IF(IIN.EQ.4096)RETURN
0060      IF(IIN.GT.RST)GOTO 140
0061      IF(IIN.EQ.-1)IIN=0
0062      IF(IIN.LT.0)WRITE(7,*)IIN,BLK
0063      IF(IIN.LT.0)STOP 'READ ERROR'
0064      EOFFLG=1
0065      NSMPIN=(OLAP+IIN)/32
0066      RETURN
0067      140 EOFFLG=-1
0068      IIN=IIN-RST
0069      NSMPIN=IIN/32
0070      RETURN
0071      END

```

```

0001 SUBROUTINE BUFSCN(BUFF,ICODE,FDONE)
0002 VIRTUAL BUFF(8448)
0003 INTEGER*2 BUFF,GAINS(3840),GSAVE(30),SPOS,EOFFLG,DMXBUF(3840),
%FRAME(33),FDONE
0004 LOGICAL*1 RC,IF(2),IBYT(2),IBYTE(2),FRAMEB(66),IBSYNC
0005 EQUIVALENCE (IWORD,IF(1)),(IWORDF,IBYT(1)),(IWORDB,IBYTE(1)),
%(FRAME(1),FRAMEB(1))
0006 COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
%IERFLG,SPOS,RC
0007 COMMON/BUFS/NUERR,NALOW,ITIC
0008 DATA ISYNC/"177777"/,IBIAS/15/
0009 DATA IBSYNC/"377/"
0010 IF(ICODE.GT.0)GOTO 1
0011 ICHCK=1
0012 ICD=0
0013 LPINT=0
0014 FRAME(33)=BUFF(SPOS+1)
0015 SPOS=SPOS+2
0016 ITBIAS=BUFF(SPOS+24)/2*2
0017 ITCONT=0
0018 NERR=0
0019 GSAVE(30)=30
0020 CALL FRAMFL(BUFF,FRAME,1,33,-1)

```

```

C
C DATA CHECK AND GAIN PREPARATION SUBROUTINE
C

```

```

0022 1 IGCHK=GSAVE(ICHCK)
0023 LDONE=0
0024 5 LDONE=LDONE+1
0025 IF(LPINT.GT.0)GOTO 100
0026 FRAME(1)=FRAME(33)
0027 CALL FRAMFL(BUFF,FRAME,2,33,ICD)

```

```

C
C SYNC TEST
C

```

```

0029 IF(EOFFLG.EQ.2)GOTO 90
0030 5 IF(FRAME(32).NE.ISYNC)GOTO 20
0031 IF(FRAMEB(66).EQ.IBSYNC)GOTO 30
0032 WRITE(2,1010)FDONE
0033 1010 FORMAT(' FILE NO:',I4,' ERROR DETECTED')

```

```

C
C TEST FOR TYPE OF DATA CORRUPTION
C

```

```

0037 20 IF(LDONE.EQ.128)GOTO 90
0038 IF(NERR.GT.NUERR)GOTO 100
0039 NERR=NERR+1
0040 NFER=0
0041 25 DO 40 I=1,32
0042 IWORDB=FRAME(I)
0043 IF(IWORDB.NE.ISYNC)GOTO 50
0044 IWORDB=FRAME(I+1)
0045 IF(IBYTE(2).NE.IBSYNC)GOTO 50

```

```

C
C DETECT PATTERN OF BYTES LOST

```

```

C
0053     IND=1
0051     DO 60 L=I+1,33
0052     FRAME(IND)=FRAME(L)
0053     IND=IND+1
0054     60 CONTINUE
0055     IF(ICD.EQ.3)GOTO 86
0057     65 CALL FRAMFL(BUFF,FRAME,IND,33,ICD)
0058     ICD=0
0059     IF(EOFFLG.EQ.2)GOTO 90
0061     GOTO 6
0062     50 IF(IBYTE(1).NE.IBSYNC)GOTO 70
0064     IWORDF=FRAME(I+1)
0065     IF(IWORDF.NE.ISYNC)GOTO 70
C
C ODD BYTE LOSS OR GAIN DETECTED
C
0067     IND=1
0068     IF(I.EQ.32)GOTO 85
0070     DO 80 L=I+2,33
0071     IWORDB=FRAME(L)
0072     IF(2)=IBYT(1)
0073     IF(1)=IBYTE(2)
0074     FRAME(IND)=IWORD
0075     IWORDF=IWORDB
0076     IND=IND+1
0077     80 CONTINUE
0079     85 IF(ICD.EQ.3)ICD=1
0080     IF(ICD.EQ.1)GOTO 65
0082     86 CALL FRAMFL(BUFF,FRAME,IND,33,2)
0083     ICD=3
0084     IF(EOFFLG.EQ.2)GOTO 90
0085     GOTO 6
0087     70 IF(IBYTE(2).NE.IBSYNC)GOTO 40
0089     IWORDF=FRAME(I+1)
0090     IF(IBYT(1).NE.IBSYNC)GOTO 40
C
C COMMUNICATION ERR POSSIBLE
C
0092     WRITE(2,1020)FDONE
0093     1020 FORMAT(' POSSIBLE COMMUNICATION LOSS FILE NO:',I4)
C
C CHECK TO SEE IF OK TO SEARCH FURTHER
C
C AHEAD IN ATTEMPTING TO REESTABLISH CONTACT
C
0094     40 CONTINUE
0095     IF(NFER.GT.NALOW)GOTO 160
0097     NFER=NFER+1
0096     FRAME(1)=FRAME(33)
0099     CALL FRAMFL(BUFF,FRAME,2,33,0)
0100     IF(EOFFLG.EQ.2)GOTO 150
0102     GOTO 25
C

```

C TIME CHECK SECTION

C

```

0103      30 ITN=FRAME(26)/2*2
0104      ITP=ITCONT.OR.ITBIAS
0105      IF(ITN.EQ.ITP)GOTO 90
0107      IF(NERR.GT.NUERR)GOTO 150
0109      WRITE(2,1030)FDONE
0110 1030  FORMAT(' TIME CHECK ERROR ON FILE NO:',I4)
0111      NERR=NERR+1
0112      INTVAL=(ITN-ITP)/ITIC
0113      IF(INTVAL.LT.0)GOTO 105
0115      LPINT=INTVAL
0116 100  IGPOS=LDONE
0117      DO 110 IVI=1,29
0118      DMXBUF(IGPOS)=1
0119      GAINS(IGPOS)=GSAVE(IVI)
0120      IGPOS=IGPOS+128
0121 110  CONTINUE
0122      GAINS(IGPOS)=30
0123      LDONE=LDONE+1
0124      LPINT=LPINT-1
0125      ITCONT=ITCONT+ITIC
0126      ICHCK=ICHCK+1
0127      IF(ICHCK.GT.30)ICHCK=1
0129      IF(LDONE.GE.128)GOTO 150
0131      IF(LPINT.GT.0)GOTO 100
0133 105  CONTINUE
0134      90  ITCONT=ITCONT+ITIC

```

C

C GAINS CORRECTION

C

```

0135      IGSAVE=IGCHK(FRAME(1),IDIR)+IBIAS
0136      IF(IGSAVE.EQ.IGSCHK)GOTO 120
0138      WRITE(2,1040)FDONE
0139 1040  FORMAT(' GAIN CHECK ERROR FILE NO:',I4)
0140      IF(NERR.GT.NUERR)GOTO 160
0142      NERR=NERR+1
0143      GSAVE(ICHCK)=IGSAVE
0144 120  IF(IFDIR.EQ.IDIR)GOTO 130
0145      WRITE(2,1050)FDONE
0147 1050  FORMAT(' GAIN DIRECTION ERROR FILE NO:',I4)
0148      IF(NERR.GT.NUERR)GOTO 160
0150      NERR=NERR+1
0151      IFDIR=IDIR

```

C

C NORMAL WORK

C

```

0152 130  ICHCK=ICHCK+1
0153      IF(ICHCK.GT.30)ICHCK=1
0155      IGPOS=LDONE
0156      DO 140 L=1,29
0157      DMXBUF(IGPOS)=IGAIN(FRAME(L+1),GSAVE(L),IFDIR)
0158      GAINS(IGPOS)=GSAVE(L)
0159      IGPOS=IGPOS+128

```

```

0160 140  CONTINUE
0161      GAINS(IGPOS)=30
0162      IGSCHK=GSAVE(ICHCK)
0163      IFDIR=-IFDIR
0164      IF(LDONE.LT.128.AND.EOFFLG.NE.2)GOTO 5
0165 150  NSMPIN=LDONE
0167      CALL APPUT(DMXBUF,4352,3840,1)
0168      RETURN
0169 160  WRITE(2,1060)FDONE
0170 1060  FORMAT(' FILE NO ',I4,' DECLARED DEAD')
0171      IERR=-1
0172      RETURN
0173      END

```

```

0001 SUBROUTINE FRAMFL(BUFF,FRAME,IST,IFIN,ICODE)
0002 VIRTUAL BUFF(8448)
0003 INTEGER*2 FRAME(33),SPOS,BUFEND,EOF,BUFF,GAINS(3840),GSAVE(30),
%EOFFLG,BSST1,BLST,OLAP,RST
0004 LOGICAL*1 RC,IF(2),IBYT(2),IBYTE(2)
0005 EQUIVALENCE (IWORD,IF(1)),(IWORDB,IBYTE(1)),(IWORDF,IBYT(1))
0006 COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
%IERFLG,SPOS,RC
0007 COMMON/BUFCOM/FBSST1,FBLST,F256,F256D,FBSST,F1,F4097,
%BSST1,BLST,OLAP,RST,IBLK
0008 IF(ICODE.GE.0)GOTO 1
0009 BUFEND=8449
0010 IBEG=4097
0011 EOF=0
0012 RETURN
0013

```

```

C
C FILL FRAME IN NORMAL CIRCUMSTANCES
C

```

```

0014 1 IF(EOFFLG.EQ.2)RETURN
0016 IF(ICODE.GT.1)GOTO 10
0018 IF(ICODE.EQ.1)FRAME(IST)=IWORDF
0020 IF(ICODE.EQ.1)IST=IST+1
0022 DO 20 I=IST,IFIN
0023 FRAME(I)=BUFF(SPOS)
0024 SPOS=SPOS+1
0025 IF(SPOS.LT.BUFEND)GOTO 20
0027 IF(EOF.GT.0)GOTO 50
0029 IBEG=4096-IBEG+2
0030 FINP=F1
0031 IF( IBEG.EQ.4097)FINP=F4097
0033 IN=IREADA(20,4096,IBLK,FINP)
0034 CALL IWAIT(20)
0035 IBLK=IBLK+16
0036 SPOS=IBEG
0037 BUFEND=SPOS+IN
0038 EOF=0
0039 IF(IN.EQ.4096)GOTO 20
0041 IN=IN+1
0042 IF(IN.LT.0)STOP 'READ ERR'
0044 EOF=1
0045 20 CONTINUE
0046 RETURN

```

```

C
C BYTE LOST PATTERN FRAME FILL
C

```

```

0047 10 IF(ICODE.EQ.2)IWORDF=FRAME(33)
0049 DO 40 I=IST,IFIN
0050 IWORDB=BUFF(SPOS)
0051 IF(2)=IBYT(1)
0052 IF(1)=IBYTE(2)
0053 FRAME(I)=IWORD
0054 IWORDF=IWORDB
0055 SPOS=SPOS+1
0056 IF(SPOS.LT.BUFEND)GOTO 40

```

```

0058 IF(EOF.GT.0)GOTO 50
0059 IBEG=4096-IBEG+2
0061 FINP=F1
0062 IF( IBEG.EQ.4097)FINP=F4097
0064 IN=IREADA(20,4096,IBLK,FINP)
0065 CALL IWAIT(20)
0066 IBLK=IBLK+16
0067 SPOS=IBEG
0068 BUFEND=SPOS+IN
0069 EOF=0
0070 IF(IN.EQ.4096)GOTO 40
0072 IN=IN+1
0073 IF(IN.LT.0)STOP ' READ ERRW'
0075 EOF=1
0076 40 CONTINUE
0077 RETURN

```

```

C
C EOF RETURN
C

```

```

0078 50 EOFLG=2
0079 RETURN
0080 END

```

0001 SUBROUTINE TAPRED(ICOM,IDRV,ISTAT,ITLEN,ILEN,IFNUM,IEOT)

C
C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ,0 IS A WIND,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
C

0002 INTEGER*2 MASK(8),ESTATI
0003 LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
%IFLEN,ESTAT,ERRS(8)
0004 DATA MASK/"1","2","4","10","20","40","100","200/
0005 DATA SDSCOM/"0","1","2","3","4","5","6","7/
0006 DATA ERRS/"377","377","377","377","377","377","377","377/
0007 ITRY=0
0008 IF(ICOM) 10,30,20

C
C SECTION CONTROLLING A READ
C

C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
C

0009 10 ITRY=ITRY+1

C
C SET UP COMMAND FOR READ
C

0010 COM(1)=SDSCOM(4)
0011 COM(2)=0
0012 COM(3)=IDRV
0013 COM(4)=-1
0014 CALL SDS10(COM,ISTAT,ITLEN,ILEN)
0015 IF(ISTAT.EQ.0)RETURN

C
C ERROR DETECTED ON READ
C

0017 ISTATI=ISTAT
0018 GOTO 40

C
C IF SHORT RECORD FOUND REREAD TAPE
C

0019 50 ITMP=ISTATI.AND.MASK(6)
0020 IF(ITMP.NE.0)GOTO 10
0021 ITMP=ISTATI.AND.MASK(2)
0022 IF(ITMP.EQ.0)RETURN

C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C

0025 WRITE(2,2010)IFNUM
0026 2010 FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
0027 IF(ITRY.GE.2)GOTO 130
0028 ECOM(1)=SDSCOM(6)
0029 ECOM(2)=1

```
0031      ECOM(3)=IDRV
0032      ECOM(4)=0
0033      CALL SDS10(ECOM,ESTAT, , )
0034      GOTO 10
C
C WRITE SECTION
C
0035      20 ITRY=ITRY+1
0036      IF(ITRY.GT.3)GOTO 130
0037      COM(1)=SDSCOM(7)
0038      IFLEN=(ILEN+3)/4
0039      COM(2)=IFLEN
0040      COM(3)=IDRV
0041      COM(4)=1
0042      CALL SDS10(COM,ISTAT, , )
0043      IF(ISTAT.EQ.0)RETURN
C
C WRITE ERROR DETECTED
C
0046      ISTATI=ISTAT
0047      GOTO 40
0048      70 ITMP=ISTATI.AND.MASK(6)
0049      ITMPI=ISTATI.AND.MASK(2)
0050      IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
C
C REPORT AND RETRY
C
0052      WRITE(2,2020)IFNUM
0053      2020 FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
0054      ECOM(1)=SDSCOM(6)
0055      ECOM(2)=2
0056      ECOM(3)=IDRV
0057      ECOM(4)=0
0058      CALL SDS10(ECOM,ESTAT, , )
0059      NBUF=8
0060      IFLENE=16
0061      IPAD=32760
0062      CALL TWRT(ERRS,NBUF,ESTAT,IPAD,IFLENE,IDRV)
0063      GOTO 20
C
C WIND FOWARD ONE FILE
C
0064      30 COM(1)=SDSCOM(5)
0065      COM(2)=1
0066      COM(3)=IDRV
0067      COM(4)=0
0068      CALL SDS10(COM,ISTAT, , )
C
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
C
0069      ISTAT=ISTAT.AND..NOT.MASK(6)
0070      IF(ISTAT.EQ.2)RETURN
0071      ISTATI=ISTAT
0072      IF(ISTAT.NE.0)GOTO 40
```

```

C
C IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
C
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
C
0075      ECOM(1)=SDSCOM(6)
0076      ECOM(2)=1
0077      ECOM(3)=IDRV
0078      ECOM(4)=0
0079      CALL SDS10(ECOM,ESTAT, , )
0080      35 RETURN
C
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS:= TAPE BUSY,TAPE OFFLINE
C BOT,EOT
C
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
C
0081      40 WRITE(2,1010)ISTATI,IFNUM
0082      1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
0083      ISTATI=ISTATI.AND..NOT.MASK(4)
0084      ITMP=ISTATI.AND.MASK(5)
0085      IF(ITMP.EQ.0)GOTO 80
0087      90 ECOM(1)=SDSCOM(1)
0088      ECOM(2)=0
0089      ECOM(3)=IDRV
0090      ECOM(4)=0
0091      CALL SDS10(ECOM,ESTAT, , )
C
C HAVING EXAMINED STATUS IF TAPE STILL
C BUSY, LOOP AGAIN,IF NOT TRY COMMAND AGAIN
C
0092      ESTATI=ESTAT
0093      ITMP=ESTATI.AND.MASK(5)
0094      IF(ITMP.NE.0)GOTO 90
0095      IF(ICOM) 10,30,20
C
C TAPE OFFLINE
C
0097      80 ITMP=ISTATI.AND.MASK(1)
0098      IF(ITMP.EQ.0)GOTO 100
0100      TYPE 1001,IDRV
0101      1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
0102      110 ECOM(1)=SDSCOM(1)
0103      ECOM(2)=0
0104      ECOM(3)=IDRV
0105      ECOM(4)=0
0106      CALL SDS10(ECOM,ESTAT, , )
0107      ESTATI=ESTAT
0108      ITMP=ESTATI.AND.MASK(1)
0109      IF(ITMP.NE.0)GOTO 110

```

```

0111      IF(ICOM) 10,30,20
C
C EOT
C
0112      100 ITMP=ISTATI.AND.MASK(3)
0113      IF(ITMP.EQ.0)GOTO 120
0115      TYPE 1002,IDRV
0116      1002 FORMAT(' EOT ON DRIVE ',I1)
0117      IFOT=-1
0118      RETURN
0119      120 IF(ICOM) 50,35,70
C
C ERROR EXIT RETURN
C
0120      130 ISTATI=-1
0121      RETURN
0122      END

```

```

        .TITLE   ISWAP
ISWAP:  .GLOBL   ISWAP,IGCHK,IGAIN,DIRG
        MOV     @2(R5),R0
        SWAB
        RTS     PC

IGCHK:  MOV     @2(R5),R0
        TSTB   R0
        BPL    1$
        MOV     #1,@4(R5)
        BR     2$
1$:     MOV     #-1.,@4(R5)
2$:     BIC    #177760,R0
        RTS     PC

IDIRG:  TST     @2(R5)
        BPL    3$
        MOV     #1,R0
        RTS     PC
3$:     MOV     #-1.,R0
        RTS     PC

IGAIN:  MOV     @2(R5),R0
        ASR    R0
        BCC    4$
        ADD    @6(R5),@4(R5)
4$:     TST     R0
        BPL    5$
        ADD    #1,R0
5$:     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

```
C
C M J POULTER OCT 80
C MPSORT.FOR... THIS IS A
C GENERAL PURPOSE SORTING PROGRAM FOR
C SEISMIC DATA FILES
C
0001 VIRTUAL RDNAM(200),WRTNAM(200),TPNAM(24)
0002 REAL *8 FSPECR,FSPECW,FMBUF,RDNAM,WRTNAM,TPNAM
0003 REAL *4 DEV,FNBUF(3),SEIS(2048)
0004 INTEGER*2 IHBLK(256),IBLKO(24),IBLKI(24),ICHANO(24),
%IPOS(24),USEFLG,INFLG,OUTFLG,INDEX(25),FLEN
0005 LOGICAL*1 ISTAT,TLEN,TPDRR,TPDRW,LBLK(512),IGCODE
0006 EQUIVALENCE (LBLK(1),IHBLK(1))
0007 DATA DEV/3RRK /
C
C SET UP I/O
C
0008 IF(ICDFN(50).NE.0)STOP' CHANNEL SET ERROR'
0009 IF(IFETCH(DEV).NE.0)STOP' FETCH ERROR'
0010 CALL ASSIGN(1,'DK2:MPSORT.DAT',14)
0011 CALL ASSIGN(2,'DK2:MPSORT.LOG',14)
0012 IRD=20
0013 IWRT=21
0014 IEOTR=0
0015 IEOTW=0
C
C READ INPUT DATA
C
0016 READ(1,1000)NFILES,NCHANI,NCHANO,NROW,TPDRR,TPDRW
0017 1000 FORMAT(12I5)
0018 READ(1,1001)ISECIN,ISBLKO,IFBLKO,USEFLG,INFLG,OUTFLG,IGCODE
0019 READ(1,1002)(INDEX(I),I=1,NROW)
0020 READ(1,1003)(ICHANO(I),I=1,NCHANO)
0021 READ(1,1004)(IPOS(I),I=1,NCHANO)
C
C READ FILE SPECS
C
0022 IF(INFLG.NE.0)GOTO 10
0023 READ(1,1001)FNBUF
0024 1001 FORMAT(3A4)
0025 CALL IRAD50(12,FNBUF,FSPECR)
0026 GOTO 15
0027 10 DO 20 J=1,NFILES
0028 READ(1,1001)FNBUF
0029 CALL IRAD50(12,FNBUF,FMBUF)
0030 RDNAM(J)=FMBUF
0031 20 CONTINUE
0032 15 IF(OUTFLG.EQ.0)GOTO 30
0033 DO 35 J=1,NFILES
0034 READ(1,1001)FNBUF
0035 CALL IRAD50(12,FNBUF,FMBUF)
0036 WRTNAM(J)=FMBUF
0037 35 CONTINUE
C
```

```
C READ ARRAY SORT FILE SPECS
C
0042 30 DO 40 J=1,NROW
0043     READ(1,1001)FNBUF
0044     CALL IRAD50(12,FNBUF,FMBUF)
0045     TPNAM(J)=FMBUF
0046 40 CONTINUE
C
C SET UP DATA CONSTANTS
C
0047     ISV=0
0048     NBLKR=(NCHANI*ISECIN)+9
0049     NBLKW=IFBLKO-ISBLKO+1
0050     IFSIZO=(NBLKW*NCHANO)+1
0051     LSTBLK=IFSIZO-1
0052     NSAMPW=NBLKW*256
0053     NBEG=(ISBLKO-1)*128
0054     NFIN=(IFBLKO*128)
C
C SET UP BLOCK POSITIONS IN FILES
C
0055     DO 45 J=1,NCHANO
0056 45 IBLKI(J)=(ICHANO(J)-1)*ISECIN+ISBLKO
0057     IBLKO(1)=1
0058     DO 50 J=2,NCHANO
0059 50 IBLKO(J)=IBLKO(J-1)+NBLKW
C
C SET UP ARRAY SORT FILES
C
0060     IF(USEFLG.NE.0)GOTO 60
0061     DO 65 L=1,NROW
0062     FMBUF=TPNAM(L)
0063     ITCH=22+L
0064     IF(IENTER(ITCH,FMBUF,IFSIZO).LT.0)STOP'ENTER ERR'
0065     IF(IWRITE(256,IHBLK,LSTBLK,ITCH).LT.0)STOP'WRITE ERR'
0066     CALL CLOSEC(ITCH)
0067 65 CONTINUE
0068     DO 70 L=1,NROW
0069     ITCH=22+L
0070     FMBUF=TPNAM(L)
0071     IF(LOOKUP(ITCH,FMBUF).LT.0)STOP'LOOKUP ERR'
0072 70 CONTINUE
C
C START OF MAIN WORK LOOP
C
0073     DO 999 IFIL=1,NFILES
0074     IFNUM=IFIL
0075     INDEX(NROW+1)=INDEX(1)
0076     DO 100 J=1,NROW
0077 100 INDEX(J)=INDEX(J+1)
0078     IF(INFLG.NE.0)GOTO 105
C
C TAPE READ CONTROL
C
```

```

0084      IF(IENTER(IRD,FSPECR,NBLKR).LT.0)STOP'ENTER ERR'
0086      ITRY=1
      C
      C EOT CONTROL
      C
0087      120 IF(ITRY.GT.3)GOTO 125
0088      IF(IEOTR.GE.0)GOTO 110
0091      125 WRITE(7,1010)TPDRR,IFNUM
0092      1010 FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I5)
0093      WRITE(7,1020)
0094      1020 FORMAT(' ENTER NEW READ TAPE DRIVE:',S)
0095      READ(5,1030)TPDRR
0096      1030 FORMAT(I1)
0097      IEOTR=0
0098      IF(TPDRR.GT.2)STOP'EOTR TERMINATE'
      C
      C TAPE READ
      C
0100      110 CALL TAPRED(-1,TPDRR,ISTAT,TLEN,FLEN,IFNUM,IEOTR)
0101      IF(ISTAT.LT.0)WRITE(2,1070)IFNUM
0103      1070 FORMAT(' WARNING RETRY FAILED FOR FILE:',I5)
0104      IF(IEOTR.LT.0)GOTO 115
0106      CALL TAPRED(0,TPDRR,ISTAT, , ,IFNUM,IEOTR)
      C
      C CHECK IF READ A BAD FILE
      C
0107      115 CALL IWAIT(IRD)
0108      IERR=0
0109      ITRY=ITRY+1
0110      IF(IREADW(1,IERR,0,IRD).LT.0)STOP'ERR READ ERR'
0112      IF(IERR.EQ."177777")GOTO 120
0114      CALL CLOSEC(IRD)
      C
      C OPEN UP INPUT FILE FOR USE
      C
0115      105 FMBUF=FSPECR
0116      IF(INFLG.NE.0) FMBUF=RDNAM(IFNUM)
0118      IF(LOOKUP(IRD,FMBUF).LT.0)STOP'LOOKUP ERR'
      C
      C HEADER BLOCK MANIPULATION
      C
0120      ITCH=22+INDEX(NROW)
0121      IF(IREADW(256,IHBLK,0,IRD).LT.0)STOP'READ ERR'
0123      LBLK(19)=IGCODE
0124      IHBLK(7)=NCHANO
0125      IHBLK(8)=IHBLK(8)+NBEG
0126      IHBLK(9)=IHBLK(9)+NFIN
0127      IF(IWRITE(256,IHBLK,0,ITCH).LT.0)STOP'WRITE ERR'
      C
      C MAIN TRANSFER LOOP
      C
0129      DO 200 ICH=1,NCHANO
0130      ITCH=22+INDEX(IPOS(ICH))
0131      ITBLKI=IBLKI(ICH)

```

```

0132      ITBLKO=IBLKO(ICH)
0133      IF(IREADW(NSAMPW,SEIS,ITBLKI,IRD).LT.0)STOP'READ ERR'
0135      IF(IWRITW(NSAMPW,SEIS,ITBLKO,ITCH).LT.0)STOP'WRITE ERR'
0137      200 CONTINUE
0138      IF(USEFLG.EQ.0.AND.IFNUM.LT.NROW)GOTO 210
0140      ITCH=INDEX(1)+22
0141      FMBUF=TPNAM(INDEX(1))
0142      IF(OUTFLG.NE.0)GOTO 220

C
C TAPE OUTPUT
C
0144      CALL CLOSEC(ITCH)
0145      IFLEN=LOOKUP(IWRT,FMBUF)
0146      IF(IFLEN.LT.0)STOP'FMBUF LOOKUP ERR'
0148      CALL TAPRED(1,TPDRW,ISTAT,TLEN,IFLEN,IFNUM,IEOTW)
0149      CALL CLOSEC(IWRT)
0150      IF(LOOKUP(ITCH,FMBUF).LT.0)STOP'LOOKUP ERR'

C
C EOT DETECTION
C
0152      IF(IEOTW.GE.0)GOTO 230
0154      WRITE(7,1040)TPDRW,IFNUM
0155      1040 FORMAT(' EOT ON WRITE DRIVE:',I2,' FILE NO:',I5)
0156      WRITE(7,1050)
0157      1050 FORMAT(' ENTER NEW WRITE DRIVE NO:',S)
0158      READ(5,1030)TPDRW
0159      IEOTW=0
0160      IF(TPDRW.GT.2)STOP'EOTW TERMINATION'
0162      230 IF(ISTAT.GE.0)GOTO 210
0164      WRITE(2,1060)IFNUM
0165      1060 FORMAT(' FATAL WRITE ERROR ON FILE:',I5)
0165      STOP'WRITE ERROR TERMINATION'

C
C STORAGE ON DISC
C
0167      220 ISV=ISV+1
0168      FSPECW=WRTNAM(ISV)
0169      IF(IENTER(IWRT,FSPECW,IFSIZE).LT.0)STOP'ENTER ERROR'
0171      IF(IREADW(256,IHBLK,0,ITCH).LT.0)STOP'TR READ ERR'
0173      IF(IWRITW(256,IHBLK,0,IWRT).LT.0)STOP'TR WRIT ERR'
0175      DO 240 J=1,NCHANO
0176      ITBLKO=IBLKO(J)
0177      IF(IREADW(NSAMPW,SEIS,ITBLKO,ITCH).LT.0)STOP'TR READ ERR'
0179      IF(IWRITW(NSAMPW,SEIS,ITBLKO,IWRT).LT.0)STOP'TR WRIT ERR'
0181      240 CONTINUE
0182      CALL CLOSEC(IWRT)

C
C END OF MAIN LOOP
C
0183      210 CALL CLOSEC(IRD)
0184      999 CONTINUE

C
C CLOSE DOWN CODE
C

```

```

0185      DO 250 J=1,NROW
0185      250 CALL CLOSEC(22+J)
0187      WRITE(2,1000)(INDEX(1),I=1,NROW)
0189      STOP' NORMAL TERMINATION'
0189      END

```

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 NFILES

READ(1,1200)FBUF

FBUF....If OUTFLG = 0, Temporary file for tape write

OUTFLG = 1, Output files, from 1 to NFILES

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 RampREAD(1,1000)IAPLTX

IAPLTX...Application flag

0 - apply ramp

1 - remove ramp

TV**2 RampREAD(1,1000)IAPLTV,NLYRREAD(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....MuteREAD(1,1000)NTAPREAD(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
C PRE STACK UTILITY PROGRAM
C THIS INVOLVES THE FOLLOWING
C 1:EDIT
C 2:POLARITY REVERSAL
C 3:EXP(0.2T) AMP RECOVERY
C 4:T*EXP(0.2T) AMP RECOVERY
C 5:TV**2 AMP RECOVERY
C 6:MUTING
C 7:DECONVOLUTION
C 8:BANDPASS FILTERING
C 9:BANDREJECT FILTERING
C 10:PREDICTION ERROR FILTERING
C 11:NORMALISATION TO UNIT ENERGY OR AMPLITUDE
C
0001 REAL*8 FSPECR,FSPECW,FNAMR,FNAMO
0002 VIRTUAL FNAMR(100),FNAMO(100),EXPT(2048),TEXPT(2048),
%TVSQ(2048),TAPER(512),BPASS(2049),BRJCT(2049)
0003 REAL*4 FBUF(3),TOLYR(20),VLYR(20),CONST(3),SEISM(2048)
0004 INTEGER*2 IORD(11),UTLFLG(11),IFILED(100),ICHAND(100),
%ICHANP(24),MUTE(24),MUTET(24),IHBLK(256),OUTFLG
0005 LOGICAL*1 TPD RR,TPDRW,ISTAT,ITLEN
0006 DATA DEV/3RRK /
C
C SET UP VIRTUAL ADDRESSES
C
0007 IEOTR=0
0008 IEOTW=0
0009 ATAP=APGAD(TAPER(1))
0010 ATBP=APGAD(BPASS(1))
0011 ATVSQ=APGAD(TVSQ(1))
0012 ATEXPT=APGAD(TEXPT(1))
0013 AEXPT=APGAD(EXPT(1))
0014 ATBR=APGAD(BRJCT(1))
C
C SET UP I/O CHANNELS AND READ IN CONTROL DATA
C
0015 IF(ICDFN(25).NE.0)STOP 'CHANNEL OVERFLOW'
0016 CALL ASSIGN(1,'DK2:MPPRST.DAT',14)
0017 CALL ASSIGN(2,'DK2:MPPRST.LOG',14)
0018 IDCH=20
0019 IDCH1=21
0020 READ(1,1000)NFILES,NCHAN,NSAMP,NSTART,INFLG,OUTFLG
0021 1000 FORMAT(12I5)
0022 READ(1,1000)TPDRR,TPDRW
0023 READ(1,1100)FSAMP
0024 1100 FORMAT(2F10.0)
C
C READ IN FILE SPECS FOR INPUT
C
0025 IF(INFLG.NE.0)GOTO 10
0026 READ(1,1200)FBUF
0027 1200 FORMAT(3A4)
0028 CALL IRAD50(12,FBUF,FSPECR)

```

```

0031      GOTO 20
0032      10 DO 30 I=1,NFILES
0033          READ(1,1200)FBUF
0034          CALL IRAD50(12,FBUF,FSPECR)
0035          FNAMR(I)=FSPECR
0036      30 CONTINUE
C
C READ IN FILE SPECS FOR OUTPUT
C
0037      20 IF(OUTFLG.NE.0)GOTO 40
0038          READ(1,1200)FBUF
0039          CALL IRAD50(12,FBUF,FSPECW)
0040          GOTO 50
0041
0042      40 DO 60 I=1,NFILES
0043          READ(1,1200)FBUF
0044          CALL IRAD50(12,FBUF,FSPECW)
0045          FNAMO(I)=FSPECW
0046      60 CONTINUE
C
C READ IN JOB SPECIFIC DATA AND SET UP
C THE FILTERS TO BE USED
C
0047      50 NUTIL=11
0048          ITX=0
0049          CONST(1)=FLOAT(NSTART)
0050          CONST(2)=0.2
0051          CONST(3)=1.0/(1000.0*FSAMP)
0052          CALL APINIT
0053          NSAMP2=2
0054
0055      51 IF(NSAMP2.GE.NSAMP)GOTO 52
0056          NSAMP2=NSAMP2*2
0057          GOTO 51
0058      52 CONTINUE
C
C READ IN FLAGS FOR PROCESSES AND EXECUTION ORDER
C
0059          READ(1,1000)NPROC
0060          READ(1,1000)(UTLFLG(I),I=1,NUTIL)
0061          READ(1,1000)(IORD(I),I=1,NPROC)
C
C TRACE EDIT DATA
C
0062          IF(UTLFLG(1).EQ.0)GOTO 65
0063          READ(1,1000)NFILED
0064          READ(1,1000)(IFILED(I),ICHAND(I),I=1,NFILED)
C
C POLARITY REVERSAL DATA
C
0065      65 IF(UTLFLG(2).EQ.0)GOTO 70
0066          READ(1,1000)NCHANP
0067          READ(1,1000)(ICHANP(I),I=1,NCHANP)
C
C EXP(0.2T) RAMP DATA
C

```

```
0070      70 IF(UTLFLG(3).EQ.0)GOTO 80
0072      READ(1,1000)IAPLX
0073      CALL TEXRMP(EXPT,0,ITX,NSAMP,CONST,AEXPT)
0074      ITX=1
C
C T*EXP(0.2T) RAMP DATA
C
0075      80 IF(UTLFLG(4).EQ.0)GOTO 90
0077      READ(1,1000)IAPLTX
0078      CALL TEXRMP(TEXPT,1,ITX,NSAMP,CONST,ATEXPT)
0079      ITX=1
C
C TV**2 RAMP
C
0080      90 IF(UTLFLG(5).EQ.0)GOTO 100
0082      READ(1,1000)IAPLTV,NLYR
0083      READ(1,1100)(T0LYR(I),VLVR(I),I=1,NLYR)
0084      CALL TVRMP(TVSQ,T0LYR,VLVR,NLYR,ITX,NSAMP,NSTART,CONST,
%FSAMP,ATVSQ)
C
C MUTE DATA
C
0085      100 IF(UTLFLG(6).EQ.0)GOTO 110
0087      READ(1,1000)NTAP
0088      READ(1,1000)(MUTE(I),I=1,NCHAN)
0089      READ(1,1000)(MUTET(I),I=1,NCHAN)
0090      CALL COTAP(TAPER,NTAP,ATAP)
C
C DECON INPUT
C
0091      110 IF(UTLFLG(7).EQ.0)GOTO 120
0093      READ(1,1000)NFILT,ISPIKE,INORM
0094      READ(1,1100)WHITE
C
C BANDPASS FILTER
C
0095      120 IF(UTLFLG(8).EQ.0)GOTO 130
0097      READ(1,1100)FL,FU
0098      READ(1,1100)FTPR1,FTPR2
0099      DFI=FLOAT(NSAMP2/2+1)/(FSAMP*500.0)
0100      FL1=FL-FTPR1
0101      FU4=FU+FTPR2
0102      CALL BANDPS(ATBP,BPASS,FL1,FL,FU,FU4,DFI,NSAMP2)
0103      NTRANF=2*NSAMP2
0104      NBFILT=NSAMP2+1
0105      NBEXP=(2*NSAMP2)+2-NSAMP
C
C BANDREJECT FILTER
C
0106      130 IF(UTLFLG(9).EQ.0)GOTO 140
0108      READ(1,1100)FLR,FUR
0109      DFI=FLOAT(NSAMP2/2+1)/(FSAMP*500.0)
0110      NTRANF=2*NSAMP2
0111      NRFILT=NSAMP2+1
```

```
0112      NBEXP=(2*NSAMP2)+2-NSAMP
0113      CALL BANDRJ(ATBR,BRJCT,FLR,FUR,DFI,NSAMP2)
      C
      C PREDICTION ERROR FILTER
      C
0114      140 IF(UTLFLG(10).EQ.0)GOTO 150
0116      READ(1,1000)NPFILT,NLAG,IPNORM
0117      READ(1,1100)PRWHIT
      C
      C TRACE NORMALISATION
      C
0118      150 IF(UTLFLG(11).EQ.0)GOTO 160
0120      READ(1,1000)NRMFLG
      C
      C BLOCKING PARAMETERS
      C
0121      160 CONTINUE
0122      IFED=1
0123      NBLKW=NSAMP/128*NCHAN+1
0124      NBLKR=NBLKW+5
0125      NBLKTR=NSAMP/128
      C
      C START OF LOOP ON DIFFERENT FILES
      C
0126      DO 200 IFNUM=1,NFILES
0127      IFIL=IFNUM
0128      IF(INFLG.NE.0)GOTO 210
      C
      C TAPE INPUT HANDLING
      C
0130      IF(IENTER(IDCH,FSPECR,NBLKR).LT.0)STOP'ENTER ERROR'
0132      ITRY=1
      C
      C EOT CHECK
      C
0133      225 IF(ITRY.GT.3)GOTO 227
0135      IF(IEOTR.GE.0)GOTO 220
0137      227 WRITE(7,1800)TPDRR,IFIL
0138      1800 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
0139      WRITE(7,1801)
0141      1801 FORMAT(' ENTER NEW READ DRIVE NO:',S)
0141      READ(5,1802)TPDRR
0142      1802 FORMAT(I1)
0143      IEOTR=0
0144      IF(TPDRR.GT.2)STOP' EOT TERMINATION'
      C
      C READ FROM TAPE
      C
0145      200 CALL TAPRED(-1,TPDRR,ISTAT,ITLEN,IFLEN,IFIL,IEOTR)
0147      IF(ISTAT.LT.0)WRITE(2,1500)IFIL
0148      1500 FORMAT(' RETRIES ON READ FAILED ON FILE',I5)
0150      IF(IEOTR.LT.0)GOTO 230
      C
      C WIND OVER EOF MARK
```

```

C
0152 CALL TAPRED(0,TPDRR,ISTAT, , ,IFIL,IEOTR)
0153 230 CALL IWAIT(IDCH)
0154 IERR=0
0155 ITRY=ITRY+1
0156 IF(IREADW(1,IERR,0,IDCH).LT.0)STOP'ERR READ ERR'
0158 IF(IERR.EQ."177777")GOTO 225
0160 CALL CLOSEC(IDCH)
C
C OPEN UP READING FILES
C
0161 210 IF(INFLG.NE.0)FSPECR=FNAMR(IFNUM)
0163 IF(LOOKUP(IDCH,FSPECR).LT.0)STOP'LOOKUP ERR'
C
C OPEN UP OUTPUT FILES
C
0166 IF(OUTFLG.NE.0)FSPECW=FNAMO(IFNUM)
0167 IF(ENTER(IDCH1,FSPECW,NBLKW).LT.0)STOP'ENTER ERR2'
C
C HEADER BLOCK MANAGEMENT
C
0169 IF(IREADW(256,IHBLK,0,IDCH).LT.0)STOP'READW ERR'
C
C PUT IN CORRECT ORDER OF PROCESSING
C
0171 IBFREE=IHBLK(24)
0172 IHBLK(129+IBFREE)=1
0173 IBFREE=IBFREE+1
0174 IHBLK(129+IBFREE)=NPROC
0175 IBFREE=IBFREE+1
0176 DO 215 J=1,NPROC
0177 IHBLK(129+IBFREE)=IORD(J)
0178 IBFREE=IBFREE+1
0179 215 CONTINUE
0180 IHBLK(24)=IBFREE
C
C WRITE OUT UPDATED HEADER
C
0181 IF(IWRITW(256,IHBLK,0,IDCH1).LT.0)STOP'WRITW ERR'
0182 JBLK=1
0183 DO 300 ICHNUM=1,NCHAN
0185 IF(IREADW(2*NSAMP,SEISM,JBLK,IDCH).LT.0)STOP'READW ERR2'
0187 CALL APPUT(SEISM,0,NSAMP,2)
0188 CALL APWD
0189 DO 400 IPCNUM=1,NPROC
0190 GOTO(410,420,430,440,450,460,470,480,
%490,500,510)IORD(IPCNUM)
C
C PROCESS EDIT COMMANDS
C
0191 410 IF(IFED.GT.NFILED)GOTO 400
0192 IF(IFILED(IFED).NE.IFIL)GOTO 400
0193 IF(ICHAND(IFED).NE.ICHNUM)GOTO 400
0197 IFED=IFED+1

```

```

0198 CALL VCLR(0,1,NSAMP)
0199 CALL APWR
0200 CALL APGET(SEISM,0,NSAMP,2)
0201 CALL APWD
0202 GOTO 310
C
C POLARITY REVERSAL
C
0203 420 DO 421 IP=1,NCHANP
0204     IF(ICHANP(IP).EQ.ICHNUM)GOTO 425
0205 421 CONTINUE
0206     GOTO 400
0207 425 CALL VNEG(0,1,0,1,NSAMP)
0208     CALL APWR
0209     GOTO 400
C
C AMP RECOVERY FILTERS APPLICATION AND REMOVAL
C
C EXP(0.2T) FILTER
C
0211 430 IAPL=IAPLX
0212     CALL APPUTA(NSAMP,NSAMP,2,AEXPT)
0213     GOTO 455
C
C T*EXP(0.2T) FILTER
C
0214 440 IAPL=IAPLTX
0215     CALL APPUTA(NSAMP,NSAMP,2,ATEXPT)
0216     GOTO 455
C
C TV**2 FILTER
C
0217 450 IAPL=IAPLTV
0218     CALL APPUTA(NSAMP,NSAMP,2,ATVSQ)
C
C COMMON CODE
C
0219 455 CALL APWD
0220     IF(IAPL.EQ.0)CALL VMUL(0,1,NSAMP,1,0,1,NSAMP)
0221     IF(IAPL.NE.0)CALL VDIV(NSAMP,1,0,1,0,1,NSAMP)
0222     CALL APWR
0223     GOTO 400
C
C MUTE APPLICATION
C
0224 460 CALL APPUTA(NSAMP,NTAP,2,ATAP)
0225     CALL APWD
0226     LMUT=MUTE(ICHNUM)
0227     CALL VCLR(0,1,LMUT)
0228     CALL VMUL(LMUT,1,NSAMP,1,LMUT,1,NTAP)
0229     CALL APWR
0230     LMUT=MUTET(ICHNUM)
0231     IF(LMUT.GE.NSAMP)GOTO 400

```

```
0235      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
C DECON
C
0240      470 CALL SPIKE(NSAMP,NSAMP2,NFILT,WHITE,INORM,ISPIKE)
0241      GOTO 400

C
C BANDPASS FILTER
C
0242      480 CALL APPUTA(4100,NBFILT,2,ATBP)
0243      CALL APWD
0244      CALL VCLR(NSAMP,1,NBEXP)
0245      CALL RFFT(0,NTRANF,+1)
0246      CALL RFFTSC(0,NTRANF,3,1)
0247      CALL VMUL(0,2,4100,1,0,2,NBFILT)
0248      CALL VMUL(1,2,4100,1,1,2,NBFILT)
0249      CALL RFFTSC(0,NTRANF,-3,0)
0250      CALL RFFT(0,NTRANF,-1)
0251      CALL APWR
0252      GOTO 400

C
C BANDREJECT FILTER
C
0253      490 CALL APPUTA(4100,NRFILT,2,ATBR)
0254      CALL APWD
0255      CALL VCLR(NSAMP,1,NBEXP)
0256      CALL RFFT(0,NTRANF,1)
0257      CALL RFFTSC(0,NTRANF,3,1)
0258      CALL VMUL(0,2,4100,1,0,2,NRFILT)
0259      CALL VMUL(1,2,4100,1,1,2,NRFILT)
0260      CALL RFFTSC(0,NTRANF,-3,0)
0261      CALL RFFT(0,NTRANF,-1)
0262      CALL APWR
0263      GOTO 400

C
C PREDICTION ERROR FILTER
C
0264      500 CALL PRDICT(NSAMP,NSAMP2,NPFILT,PRWHIT,IPNORM,NLAG)
0265      GOTO 400

C
C NORMALISATION
C
0266      510 IF(NRMFLG.EQ.0)GOTO 515
0268      CALL MAXMGV(0,1,2050,NSAMP)
0269      GOTO 516
0270      515 CALL SVESQ(0,1,2050,NSAMP)
0271      CALL VSQRT(2050,1,2050,1,1)
0272      515 CALL VDIV(2050,0,0,1,0,1,NSAMP)
0273      CALL APWR
0274      GOTO 400
```

```
0275 400 CONTINUE
      C
      C END OF PROCESS LOOP
      C
0276 CALL APWAIT
0277 CALL APGET(SEISM,0,NSAMP,2)
0278 CALL APWD
0279 310 IF(IWRITE(2*NSAMP,SEISM,JBLK,IDCH1).LT.0)STOP'WRITW ERR2'
0281 JBLK=JBLK+NBLKTR
0282 300 CONTINUE
      C
      C END OF CHANNEL LOOP
      C
0283 CALL CLOSEC(IDCH)
0284 CALL CLOSEC(IDCH1)
      C
      C OUTPUT TO TAPE
      C
0285 IF(OUTFLG.NE.0)GOTO 200
0287 IFLEN=LOOKUP(IDCH1,FSPECW)
0288 IF(IFLEN.LT.0)STOP'LOOKUP ERR3'
0290 CALL TAPRED(1,TPDRW,ISTAT,ITLEN,IFLEN,IFIL,IEOTW)
0291 CALL CLOSEC(IDCH1)
      C
      C CHECK FOR ERRORS
      C
0292 IF(IEOTW.GE.0)GOTO 250
0294 WRITE(7,1600)TPDRW,IFIL
0295 1600 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
0296 WRITE(7,1601)
0297 1601 FORMAT(' ENTER DRIVE NO FOR NEW WRITE TAPE:',S)
0298 READ(5,1802)TPDRW
0299 IEOTW=0
0300 IF(TPDRW.GT.2)STOP ' EOT WRITE TERMINATION'
0302 250 IF(ISTAT.GE.0)GOTO 200
0304 WRITE(2,1700) IFIL
0306 1700 FORMAT(' FATAL ERROR ON WRITE FILE NO',I5)
0308 STOP ' WRITE ERROR'
0307 200 CONTINUE
0308 CALL CLOSEC(IDCH)
0309 CALL CLOSEC(IDCH1)
0310 STOP'NORMAL TERMINATION'
0311 END
```

```

0001      SUBROUTINE TEXRMP(FILT,IFTYP,IFLG,NSAMP,CONST,AFILT,FS)
C
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
C      CONST(1)=NSART
C      CONST(2)=0.2
C      CONST(3)=1.0/(1000.0*FSAMP)
C
0002      VIRTUAL FILT(2048)
0003      DIMENSION CONST(3)
0004      IF(IFLG.NE.0)GOTO 10
C
C FORM THE T RAMP
C
0005      CALL APWAIT
0007      CALL APPUT(CONST,8189,3,2)
0008      CALL APWD
0009      CALL VCLR(0,1,NSAMP)
0010      CALL VRAMP(8189,8191,0,1,NSAMP)
C
C FORM EXP(0.2T)
C
0011      CALL VSMUL(0,1,8190,NSAMP,1,NSAMP)
0012      CALL VEXP(NSAMP,1,NSAMP,1,NSAMP)
0013      IF(IFTYP.EQ.0)GOTO 20
C
C FORM T*EXP(0.2T)
C
0015      10 CALL VMUL(NSAMP,1,0,1,NSAMP,1,NSAMP)
0016      20 CALL APWR
0017      CALL APGETA(NSAMP,NSAMP,2,AFILT)
0018      RETURN
0019      END
    
```

```

0001      SUBROUTINE COTAP(TAPER,NTAP,ATAP)
C
C THIS ROUTINE PRODUCES A COSINE TAPER NTAP
C SAMPLES LONG
C
0002      VIRTUAL TAPER(512)
0003      CALL APWAIT
0004      CALL APPUT(1.0/FLOAT(NTAP),0,1,2)
0005      CALL VCLR(1,1,NTAP)
0006      CALL VTSADD(1,1,2306,1,1,1)
0007      CALL VTSMUL(0,1,2306,0,1,1)
0008      CALL VRAMP(1,0,0,1,NTAP)
0009      CALL VCOS(0,1,0,1,NTAP)
0010      CALL VTSADD(0,1,2049,0,1,NTAP)
0011      CALL VTSMUL(0,1,2327,0,1,NTAP)
0012      CALL APWR
0013      CALL APGETA(0,NTAP,2,ATAP)
0014      CALL APWD
0015      RETURN
0016      END
    
```

0001 SUBROUTINE TVRMP(FILT,T0LYR,VLYR,NLYR,IFLG,NSAMP,NSTART,
%CONST,FSAMP,AFILT)

C
C THIS ROUTINE PRODUCES A TV**2 RAMP
C FROM VELOCITY INFO IN T0LYR,VLYR

0002 VIRTUAL FILT(2048)
0003 DIMENSION T0LYR(20),VLYR(20),CONST(3)

C
C CHECK IF T RAMP ALREADY FORMED

0004 IF(IFLG.NE.0)GOTO 10
0006 CALL APWAIT
0007 CALL APPUT(CONST,8189,3,2)
0008 CALL APWD
0009 CALL VCLR(0,1,NSAMP)
0010 CALL VRAMP(8189,8191,0,1,NSAMP)

C
C FORM VELOCITY RAMP IN FILT

0011 10 N1=1
0012 N2=IFIX(FSAMP*T0LYR(1))-NSTART
0013 V1=VLYR(1)
0014 DO 15 I=N1,N2
0015 15 FILT(I)=V1
0016 IF(NLYR.EQ.1)GOTO 40
0018 DO 20 J=2,NLYR
0019 N1=N2+1
0020 N2=IFIX(FSAMP*T0LYR(J))-NSTART
0021 DELV=(VLYR(J)-VLYR(J-1))/(N2-N1+2)
0022 V=VLYR(J-1)
0023 DO 30 I=N1,N2
0024 30 FILT(I)=V
0025 V=V+DELV
0026 30 CONTINUE
0027 20 CONTINUE
0028 40 N1=N2+1
0029 N2=NSAMP
0030 VN=VLYR(NLYR)
0031 DO 50 I=N1,N2
0032 50 FILT(I)=VN

C
C PUT V RAMP IN AP AND FORM TV**2

0033 CALL APWAIT
0034 CALL APPUTA(NSAMP,NSAMP,2,AFILT)
0035 CALL APWD
0036 CALL VSQ(NSAMP,1,NSAMP,1,NSAMP)
0037 CALL VMUL(0,1,NSAMP,1,NSAMP,1,NSAMP)
0038 CALL APWR
0039 CALL APGETA(NSAMP,NSAMP,2,AFILT)
0040 CALL APWD
0041 RETURN
0042 END

```
0002 SUBROUTINE BANDPS(ATBP,BPASS,F1,F2,F3,F4,DFI,NSAMP)
```

```
C
```

```
C SUBROUTINE WHICH CREATES A BANDPASS FILTER
```

```
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
```

```
C
```

```
0002 VIRTUAL BPASS(2049)
```

```
C
```

```
C SET UP CONSTANTS
```

```
C
```

```
0003 N1=2*F1*DFI
```

```
0004 N2=2*F2*DFI
```

```
0005 N3=2*F3*DFI
```

```
0006 N4=2*F4*DFI
```

```
0007 NFILT=NSAMP+1
```

```
0008 NTP1=N2-N1
```

```
0009 NTP2=N4-N3
```

```
0010 NOK=N3-N2
```

```
0011 CALL APWAIT
```

```
C
```

```
C SET UP THE FILTER IN THE AP
```

```
C
```

```
0012 DO 10 I=1,2
```

```
0013 NTAPE=NTP1
```

```
0014 IF(I.EQ.2)NTAPE=NTP2
```

```
0015 RNTAPE=1.0/FLOAT(NTAPE)
```

```
0016 CALL APPUT(RNTAPE,0,1,2)
```

```
0017 CALL VCLR(1,1,NTAPE)
```

```
0018 CALL VTSADD(1,1,2306,1,1,1)
```

```
0019 CALL VTSMUL(0,1,2306,0,1,1)
```

```
0020 CALL VRAMP(1,0,0,1,NTAPE)
```

```
0021 CALL VCOS(0,1,0,1,NTAPE)
```

```
0022 CALL VTSADD(0,1,2049,0,1,NTAPE)
```

```
0023 CALL VTSMUL(0,1,2327,0,1,NTAPE)
```

```
0024 IF(I.EQ.1)CALL VMOV(0,1,2050,1,NTAPE)
```

```
0025 IF(I.EQ.2)CALL VMOV(0,1,4100,1,NTAPE)
```

```
0026 10 CONTINUE
```

```
C
```

```
C NOW HAVE TAPERS FORM REST OF FILTER
```

```
C
```

```
0027 CALL VCLR(0,1,NFILT)
```

```
0028 CALL VADD(N1,1,2050,1,N1,1,NTP1)
```

```
0029 CALL VADD(N4,-1,4100,1,N4,-1,NTP2)
```

```
0030 CALL VTSADD(N2,1,2049,N2,1,NOK)
```

```
0031 CALL APWR
```

```
0032 CALL APGETA(0,NFILT,2,ATBP)
```

```
0033 CALL APWD
```

```
0034 RETURN
```

```
0035 END
```

```
0001      SUBROUTINE BANDRJ(ATBR,BRJCT,F1,F2,DFI,NSAMP)
C
C SUBROUTINE TO CREATE A BANDREJECT FILTER
C
C      F1=LOWER CUTOFF POSITION
C      F2=UPPER CUTOFF POSITION
C
C      THE FILTER TAKES THE FORM OF
C      A SINE BELL CENTERED ON THE FREQUENCY TO
C      BE REMOVED COMPLETELY
C
0002      VIRTUAL BRJCT(2049)
C
C SET UP CONSTANTS
C
0003      N1=2.0*F1*DFI
0004      N2=2.0*F2*DFI
0005      NTAP=N2-N1
0006      CALL APWAIT
C
C SET UP FILTER IN AP
C
0007      CALL VCLR(0,1,NTAP)
0008      NFILT=NSAMP+1
0009      FTAP=1.0/FLOAT(NTAP)
0010      CALL APPUT(FTAP,1,1,2)
0011      CALL APWD
0012      CALL VTSMUL(1,1,2317,1,1,1)
0013      CALL VRAMP(0,1,0,1,NTAP)
0014      CALL VCOS(0,1,0,1,NTAP)
0015      CALL VTSADD(0,1,2049,0,1,NTAP)
0016      CALL VTSMUL(0,1,2327,0,1,NTAP)
0017      CALL VMOV(0,1,4096,1,NTAP)
C
C SET UP FULL FILTER NOW TAPER FINISHED
C
0018      CALL VCLR(0,1,NFILT)
0019      CALL VTSADD(0,1,2049,0,1,NFILT)
0020      CALL VMUL(N1,1,4096,1,N1,1,NTAP)
0021      CALL APWR
0022      CALL APGETA(0,NFILT,2,ATBR)
0023      CALL APWD
0024      RETURN
0025      END
```

```
0001      SUBROUTINE SPIKE(NSAMP,NSAMP2,ILENGTH,WHITE,IFLAG,ISPIKE)
C
C WIENER SPIKING FILTER ROUTINE
C NSAMP=DATA LENGTH
C NSAMP2=NEAREST POWER OF 2 TO NSAMP
C ILENGTH=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
C
0002      NTRAN=2*NSAMP2
0003      NCLR=NTRAN-NSAMP
0004      NFCLR=NTRAN-ILENGTH
0005      NTRAN2=NTRAN+2
0006      NM1=NSAMP2-1
0007      I6=NTRAN+ILENGTH
0008      I7=I6+ILENGTH
0009      I8=I7+ILENGTH
0010      ISP=I6+ISPIKE
C
C HAVING SET UP CONSTANTS GET INPUT TRACE
C ENERGY IF REQD FOR NORMALISATION
C
0011      IF(IFLAG.NE.2)GOTO 10
0012      CALL SVESQ(0,1,8191,NSAMP)
0013      CALL VSQRT(8191,1,8191,1,1)
0014      CALL APWR
0015      CALL APGET(EN,8191,1,2)
0016      CALL APWD
C
C GET AUTOCORRELATION FUNCTION
C
0017      10 CALL VCLR(NSAMP,1,NCLR)
0018      CALL RFFT(0,NTRAN,1)
0019      CALL VMOV(0,1,NTRAN,1,NTRAN)
0020      CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
0021      CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
0022      CALL VCLR(NTRAN+3,2,NM1)
0023      CALL RFFTSC(NTRAN,NTRAN,-1,-1)
0024      CALL RFFT(NTRAN,NTRAN,-1)
C
C AUTO FUNCTION NOW 2N LONG FROM NTRAN
C ORIGINAL FUNCTION TRANSFORMED 0-NTRAN
C
C NOW WHITEN
C
0025      CALL APWR
0026      CALL APPUT(WHITE,8191,1,2)
0027      CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)
C
C SET UP SPIKE CC FUNCTION
C
```

```
0029      CALL VCLR(I6,1,ILENTN)
0030      CALL VTSADD(ISP,1,2049,ISP,1,1)
      C
      C SOLVE EQNS
      C
0031      CALL WIENER(ILENTN,NTRAN,I6,I7,I8,1)
0032      CALL APCHK(IER)
0033      IF(IER.NE.0)STOP'LEVINSON FAILURE'
0035      CALL VMOV(I7,1,NTRAN,1,ILENTN)
      C
      C NORMALISE FILTER IF ASKED FOR
      C
0036      IF(IFLAG.NE.1)GOTO 20
0037      CALL SVESQ(NTRAN,1,I6,ILENTN)
0039      CALL VSQRT(I6,1,I6,1,1)
0040      CALL VDIV(I6,0,NTRAN,1,NTRAN,1,ILENTN)
      C
      C APPLY FILTER
      C
0041      20 CALL VCLR(I6,1,NFCLR)
0042      CALL RFFT(NTRAN,NTRAN,1)
0043      CALL VMUL(0,1,NTRAN,1,0,1,2)
0044      CALL CVMUL(2,2,NTRAN2,2,2,2,NM1,1)
0045      CALL RFFTSC(0,NTRAN,0,-1)
0046      CALL RFFT(0,NTRAN,-1)
      C
      C DO SCALING IF REQD
      C
0047      IF(IFLAG.NE.2)GOTO 30
0049      CALL SVESQ(0,1,8191,NSAMP)
0050      CALL VSQRT(8191,1,8191,1,1)
0051      CALL APWR
0052      CALL APPUT(EN,8190,1,2)
0053      CALL APWD
0054      CALL VDIV(8191,1,8190,1,8190,1,1)
0055      CALL VSMUL(0,1,8190,0,1,NSAMP)
0056      30 CALL APWR
0057      RETURN
0058      END
```

```
0001 SUBROUTINE PRDICT(NSAMP,NSAMP2,ILENGTH,WHITE,IFLAG,NLAG)
```

```
C
C THIS ROUTINE DESIGNS AND APPLIES
C A PREDICTION ERROR FILTER
C NSAMP=DATA LENGTH
C NSAMP2=NEAREST POWER OF 2 TO NSAMP
C ILENGTH=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
```

```
0002 NTRAN=2*NSAMP2
0003 NCLR=NTRAN-NSAMP
0004 NM1=NSAMP2-1
0005 NTRAN2=NTRAN+2
0006 NLG=NTRAN+NLAG
0007 NFILT=ILENGTH+NLAG
0008 I7=NLG+ILENGTH
0009 I8=I7+ILENGTH
```

```
C
C GET INPUT TRACE ENERGY
```

```
0010 IF(IFLAG.NE.2)GOTO 10
0011 CALL SVESQ(0,1,8191,NSAMP)
0012 CALL VSQRT(8191,1,8191,1,1)
0013 CALL APGET(EN,8191,1,2)
0014 CALL APWD
```

```
C
C GET AUTOCORRELATION FUNCTION
```

```
0015 10 CALL VCLR(NSAMP,1,NCLR)
0016 CALL RFFT(0,NTRAN,1)
0017 CALL VMOV(0,1,NTRAN,1,NTRAN)
0018 CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
0019 CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
0020 CALL VCLR(NTRAN+3,2,NM1)
0021 CALL RFFTSC(NTRAN,NTRAN,-1,-1)
0022 CALL RFFT(NTRAN,NTRAN,-1)
```

```
C
C NOW WHITEN IT
```

```
0023 CALL APWR
0024 CALL APPUT(WHITE,8191,1,2)
0025 CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)
```

```
C
C NOW SOLVE EQNS
```

```
0026 CALL WIENER(ILENGTH,NTRAN,NLG,I7,I8,1)
0027 CALL APCHK(IER)
0028 IF(IER.NE.0)STOP 'LEVINSON FAILURE'
0029 IF(IFLAG.NE.1)GOTO 20
0030 CALL SVESQ(I7,1,I8,ILENGTH)
```

```
0034      CALL VSQRT(I8,1,I8,1,1)
0035      CALL VDIV(I8,0,I7,1,I7,1,ILENGTH)
      C
      C APPLY FILTER
      C
0036      20 CALL VCLR(NTRAN,1,NFILT)
0037      CALL VTSADD(NTRAN,1,2049,NTRAN,1,1)
0038      CALL VSUB(I7,1,NLG,1,NLG,1,ILENGTH)
0039      CALL VCLR(I7,1,NTRAN-NFILT)
0040      CALL RFFT(NTRAN,NTRAN,1)
0041      CALL VMUL(0,1,NTRAN,1,0,1,2)
0042      CALL CVMUL(2,2,NTRAN2,2,2,2,NM1,1)
0043      CALL RFFTSC(0,NTRAN,0,-1)
0044      CALL RFFT(0,NTRAN,-1)
      C
      C DO SCALING
      C
0045      IF(IFLAG.NE.2)GOTO 30
0047      CALL SVESQ(0,1,8191,NSAMP)
0048      CALL VSQRT(8191,1,8191,1,1)
0049      CALL APWR
0050      CALL APPUT(EN,8190,1,2)
0051      CALL APWD
0052      CALL VDIV(8191,1,8190,1,8190,1,1)
0053      CALL VSMUL(0,1,8190,0,1,NSAMP)
0054      30 CALL APWR
0055      RETURN
0056      END
```

```
0001      SUBROUTINE TAPRED(ICOM,IDRV,ISTAT,ITLEN,ILEN,IFNUM,IEOT)
C
C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ,0 IS A WIND,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
C
0002      INTEGER*2 MASK(8),ESTATI
0003      LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
      %IFLEN,ESTAT,ERRS(8)
0004      DATA MASK/'1','2','4','10','20','40','100','200'/
0005      DATA SDSCOM/'0','1','2','3','4','5','6','7'/
0006      DATA ERRS/'377','377','377','377','377','377','377','377'/
0007      ITRY=0
0008      IF(ICOM) 10,30,20
C
C SECTION CONTROLLING A READ
C
C
C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
0009      10 ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
0010      COM(1)=SDSCOM(4)
0011      COM(2)=1
0012      COM(3)=IDRV
0013      COM(4)=-1
0014      CALL SDS10(COM,ISTAT,ITLEN,ILEN)
0015      IF(ISTAT.EQ.0)RETURN
C
C ERROR DETECTED ON READ
C
0017      ISTATI=ISTAT
0018      GOTO 40
C
C IF SHORT RECORD FOUND REREAD TAPE
C
0019      50 ITMP=ISTATI.AND.MASK(6)
0020      IF(ITMP.NE.0)GOTO 10
0021      ITMP=ISTATI.AND.MASK(2)
0022      IF(ITMP.EQ.0)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
0025      WRITE(2,2010)IFNUM
0026      2010 FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
0027      IF(ITRY.GE.2)GOTO 130
0028      ECOM(1)=SDSCOM(6)
0029      ECOM(2)=1
```

```
0031      ECOM(3)=IDRV
0032      ECOM(4)=0
0033      CALL SDS10(ECOM,ESTAT, , )
0034      GOTO 10
C
C WRITE SECTION
C
0035      20 ITRY=ITRY+1
0036      IF(ITRY.GT.3)GOTO 130
0039      COM(1)=SDSCOM(7)
0039      IFLEN=(ILEN+3)/4
0040      COM(2)=IFLEN
0041      COM(3)=IDRV
0042      COM(4)=1
0043      CALL SDS10(COM,ISTAT, , )
0044      IF(ISTAT.EQ.0)RETURN
C
C WRITE ERROR DETECTED
C
0046      ISTAT=ISTAT
0047      GOTO 40
0048      70 ITMP=ISTATI.AND.MASK(6)
0049      ITMPI=ISTATI.AND.MASK(2)
0050      IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
C
C REPORT AND RETRY
C
0052      WRITE(2,2020)IFNUM
0052      2020 FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
0054      ECOM(1)=SDSCOM(6)
0055      ECOM(2)=2
0056      ECOM(3)=IDRV
0057      ECOM(4)=0
0058      CALL SDS10(ECOM,ESTAT, , )
0059      NBUF=8
0060      IFLENE=16
0061      IPAD=32760
0062      CALL TWRIT(ERRS,NBUF,ESTAT,IPAD,IFLENE,IDRV)
0063      GOTO 20
C
C WIND FOWARD ONE FILE
C
0064      30 COM(1)=SDSCOM(5)
0065      COM(2)=1
0066      COM(3)=IDRV
0067      COM(4)=0
0068      CALL SDS10(COM,ISTAT, , )
C
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
C
0070      ISTAT=ISTAT.AND..NOT.MASK(6)
0071      IF(ISTAT.EQ.2)RETURN
0072      ISTAT=ISTAT
0073      IF(ISTAT.NE.0)GOTO 40
```

```

C
C IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
C
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
C
0075      ECOM(1)=SDSCOM(6)
0076      ECOM(2)=1
0077      ECOM(3)=IDRV
0078      ECOM(4)=0
0079      CALL SDS10(ECOM,ESTAT, , )
0080      35 RETURN
C
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS:= TAPE BUSY,TAPE OFFLINE
C BOT,EOT
C
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
C
0081      40 WRITE(2,1010)ISTATI,IFNUM
0082      1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
0083      ISTATI=ISTATI.AND..NOT.MASK(4)
0084      ITMP=ISTATI.AND.MASK(5)
0085      IF(ITMP.EQ.0)GOTO 80
0086      90 ECOM(1)=SDSCOM(1)
0087      ECOM(2)=0
0088      ECOM(3)=IDRV
0089      ECOM(4)=0
0090      CALL SDS10(ECOM,ESTAT, , )
C
C HAVING EXAMINED STATUS IF TAPE STILL
C BUSY, LOOP AGAIN,IF NOT TRY COMMAND AGAIN
C
0091      ESTATI=ESTAT
0092      ITMP=ESTATI.AND.MASK(5)
0093      IF(ITMP.NE.0)GOTO 90
0094      IF(ICOM) 10,30,20
C
C TAPE OFFLINE
C
0097      30 ITMP=ISTATI.AND.MASK(1)
0098      IF(ITMP.EQ.0)GOTO 100
0099      TYPE 1001,IDRV
0100      1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
0101      110 ECOM(1)=SDSCOM(1)
0102      ECOM(2)=0
0103      ECOM(3)=IDRV
0104      ECOM(4)=0
0105      CALL SDS10(ECOM,ESTAT, , )
0106      ESTATI=ESTAT
0107      ITMP=ESTATI.AND.MASK(1)
0108      IF(ITMP.NE.0)GOTO 110

```

```

0111      IF(ICOM) 10,30,20
C
C EOT
C
0112      100 ITMP=ISTATI.AND.MASK(3)
0113      IF(ITMP.EQ.0)GOTO 120
0114      TYPE 1002,IDRV
0115      1002 FORMAT(' EOT ON DRIVE ',I1)
0116      IEOT=-1
0117      RETURN
0118      120 IF(ICOM) 50,35,70
C
C ERROR EXIT RETURN
C
0120      130 ISTATI=-1
0121      RETURN
0122      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

```

0001      REAL*8 FSPECR
0002      REAL*4 SEIS(2060),FNAMR(3)
0003      INTEGER*2 ICHAN(24)
0004      DATA DEV/3RRK /
C
C THIS IS A SPECTRAL ANALYSIS PROGRAM
C
0005      WRITE(7,1000)
0006      1000 FORMAT(' ENTER NO OF SAMPLES(I4):',S)
0007      READ(5,1001)NSAMP
0008      1001 FORMAT(I4)
0009      WRITE(7,1002)
0010      1002 FORMAT(' ENTER NO OF CHANNELS(I2):',S)
0011      READ(5,1007)NCHAN
0012      1007 FORMAT(I2)
0013      WRITE(7,1008)
0014      1008 FORMAT(' ENTER CHANNELS REQD IN ORDER(12I3)',/, ' >',S)
0015      READ(5,1009)(ICHAN(I),I=1,NCHAN)
0016      1009 FORMAT(12I3)
0017      WRITE(7,1003)
0018      1003 FORMAT(' ENTER SAMPLING PERIOD(F10.0):',S)
0019      READ(5,1004)TSEC
0020      1004 FORMAT(F10.0)
0021      WRITE(7,1005)
0022      1005 FORMAT(' ENTER FILE NAME:',S)
0023      READ(5,1006)FNAMR
0024      1006 FORMAT(3A4)
0025      CALL IRAD50(12,FNAMR,FSPECR)
C
C SET UP READ IN
C
0026      ICHR=IGETC( )
0027      IF(IFETCH(DEV).NE.0)STOP ' FETCH ERR'
0028      IF(LOOKUP(ICHR,FSPECR).LT.0)STOP 'LOOKUP ERR'
0029      NWDS=2*NSAMP
0030      NBLKS=NSAMP/128
C
C PROCESS FIRST CHANNEL
C
0031      NBLKST=(ICHAN(1)-1)*NBLKS+1
0032      IF(IREADW(NWDS,SEIS,NBLKST,ICHR).LT.0)STOP 'READ ERR'
0033      IFLAG=+1
0034      CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(1))
0035      IF(NCHAN.EQ.1)GOTO 10
C
C DO MOST OF CHANNELS
C
0040      DO 20 J=2,NCHAN
0041      NBLKST=(ICHAN(J)-1)*NBLKS+1
0042      IF(IREADW(NWDS,SEIS,NBLKST,ICHR).LT.0)STOP 'READW ERR'
0043      IFLAG=-1
0044      CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(J))
0045      20 CONTINUE
0046      10 IFLAG=0
C
C
FORTRAN IV      V02.04      THU 08-JAN-81 00:13:29      PAGE 002
0047      CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(1))
0048      STOP ' NORMAL TERMINATION'
0049      END

```

```

0001 SUBROUTINE SPEC(AVAL, INUM, IPILOT, TSEC, ICHAN)
0002 DIMENSION AVAL(1)
0003 DATA MASK/0104210/
C*****
C*****
C     AVAL=SOURCE DATA TO BE PADDED TO NEAREST 2**N
C     INUM=DIMENSION AVAL
C     IPILOT=DISPLAY CONTROL
C
C             IPILOT>0 NEW OR FIRST PLOT
C             IPILOT<0 SUBSIDIARY PLOT
C             IPILOT=0 TERMINATE PLOT
C
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 THA SAMPLING RAMP
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
C     EXPAND INUM TO 2**N
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
0004     IF(IPILOT.EQ.0)GOTO 999
0005     N=2
0006     ICNTRL=0
0007     100 CONTINUE
0008     IF(N.GE.INUM) GOTO 99
0009     N=N*2
0010     GOTO 100
C     N=LOWEST 2**N GT INUM
0011     99 IF(N.GT.4096) STOP 'AP OVERFLOW ON SPECTRUM'
C     THIS FOR AP'S SAKE
0012     DELF=1.0/(N*TSEC)
C     DELF IS RETURNED TRANSFORM FREQUENCY
C     PAD OUT AVAL WITH ZEROS
0013     IF(INUM.EQ.N) GOTO 1011
0014     ISTART=INUM+1
0015     DO 101 I=ISTART, N
0016     101 AVAL(I)=0.0
0017     1011 FLIM=0.5*DELF*N
C     FLIM=MAX TRANSFORM FREQUENCY
C     SEEK FREQUENCY DECADE OF PLOT
0018     FBASE=0.1
0019     IBASE=-1
0020     102 CONTINUE
0021     IF(FLIM.LE.FBASE) GOTO 103
0022     FBASE=FBASE*10.
0023     IBASE=IBASE+1
0024     GOTO 102
0025     103 FBASE=FBASE*0.1
0026     IBASE=IBASE-1
0027     FMAX=FLIM/FBASE
0028     IFMAX=IFIX(FMAX)
0029     IFMAX=IFMAX+1
0030     FMAX=FLOAT(IFMAX)
C     FMAX=NEXT HIGHEST INTEGER DECADE=1 TO 9

```

```

C
FORM PLOTTING MASK
0036 YSIZE=9.0
0037 ASPECT=0.75
0038 XSIZE=YSIZE*ASPECT
0039 XOFST=XSIZE/20.0
0040 YOFST=YSIZE/20.0
0041 1031 CSIZE=YSIZE/80.0
0042 DELX=XSIZE/100.0
0043 IF(ICNTRL.NE.0) DELX=XSIZE/36.0
0044 Y=FMAX*10.0
0045 IY=IFIX(Y)
0046 DELY=YSIZE/IY
0047 IF(I PLOT.GT.0) CALL PLOTS(0,0,0)
0048 IF(I PLOT.LE.0) CALL PLOT(0.,0.,-999)
0052 CALL SETMSG(0)
0053 NX=100
0054 IF(ICNTRL.NE.0) NX=36
0055 CALL GRID(XOFST,YOFST,NX,DELX,IY,DELY,MASK)
0057 CALL PLOT(XOFST,YOFST,3)
0058 CALL NEWPEN(3)
0059 X=XOFST+XSIZE
0060 Y=YOFST+YSIZE
0061 CALL PLOT(X,YOFST,2)
0062 CALL PLOT(X,Y,2)
0063 CALL PLOT(XOFST,Y,2)
0064 CALL PLOT(XOFST,YOFST,2)
0065 CALL NEWPEN(1)
0066 CALL PLOT(X,YOFST,-3)
C
RE-ORIGIN THE PLOT
0067 XINC=XSIZE/10.0
0068 IF(ICNTRL.NE.0) XINC=XSIZE/36.0
0071 XNUM=0.0
0072 IF(ICNTRL.NE.0) XNUM=-180.0
0073 Y=-CSIZE
0074 IF(ICNTRL.NE.0) Y=-4.0*CSIZE
0075 DELX=1.0
0076 IF(ICNTRL.NE.0) DELX=10.0
0077 NX=10
0078 IF(ICNTRL.NE.0) NX=37
0079 X=0.0
0080 DO 104 I=1,NX
0081 IF(ICNTRL.NE.0) GOTO 1041
0082 CALL PLOT(X,0.,3)
0083 CALL PLOT(X,YSIZE,2)
0084 1041 CALL NUMBER(X,Y,CSIZE,XNUM,90.0,-1)
0085 XNUM=XNUM+DELX
0086 104 X=X-XINC
0087 IY=IY*0.1
0088 YINC=YSIZE/IY
0089 IY=IY+1
0090 YNUM=0.0
0091 Y=0.0
0092 X=1.5*CSIZE
0093 XINC=-XSIZE

```

```
0094 DO 105 I=1,IY
0095 CALL PLOT(0.0,Y,3)
0100 CALL PLOT(XINC,Y,2)
0101 CALL NUMBER(X,Y,CSIZE,YNUM,90.0,-1)
0102 YNUM=YNUM+1
0103 105 Y=Y+YINC
0104 SSIZE=CSIZE*1.5
0105 X=X+2.0*SSIZE
0106 Y=YSIZE*0.5-7.0*SSIZE
0107 CALL SYMBOL(X,Y,SSIZE,12HFREQUENCY*10,90.0,12)
0108 Y=Y+12.0*SSIZE
0109 X=X-SSIZE
0110 DELX=FLOAT(IBASE)
0111 CALL NUMBER(X,Y,CSIZE,DELX,90.0,-1)
0112 Y=0.65*YSIZE
0113 X=X+SSIZE
0114 SSIZE=SSIZE*1.33
0115 CALL NEWPEN(4)
0116 IF(ICNTRL.EQ.0) CALL SYMBOL(X,Y,SSIZE,15HENERGY SPECTRUM,90.0,15)
0118 IF(ICNTRL.NE.0) CALL SYMBOL(X,Y,SSIZE,15HPHASE SPECTRUM,90.0,15)
0120 FEND=FMAX*FBASE
0121 DFPLOT=YSIZE*DELF/FEND
0122 IF(ICNTRL.NE.0) GOTO 1051
C
C DFPLOT=PLOT UNITS PER FREQUENCY SAMPLE
C *****
C PROCESS TIME SERIES
C *****
0124 AVAL(N+1)=-XSIZE
0125 AVAL(N+2)=-XSIZE*7.0/44.0
0126 AVAL(N+3)=-0.5*XSIZE
0127 NX=N+3
C EXPAND AVAL TO CONTAIN DISPLAY CONSTANTS
C DO AP PROCESSING
0129 MN=N+2
0129 NN=MN/2
0131 LN=MN+1
0131 NNN=NN-1
0132 CALL APCLR
0133 CALL APPUT(AVAL,0,NX,2)
0134 CALL APWD
0135 CALL VMOV(N,1,8189,1,3)
C HOLD DISPLAY CONSTANTS IN HIGH MEMORY
0136 CALL RFFT(0,N,1)
0137 CALL RFFTSC(0,N,3,1)
C NN=COMPLEX TRANSFORM LENGTH
0138 CALL POLAR(0,2,0,2,NN)
0139 CALL MAXV(0,2,MN,NN)
C NORMALISE AMPLITUDE SPECTRUM
0140 CALL VFILL(MN,LN,1,NNN)
0141 CALL VDIV(MN,1,0,2,0,2,NN)
C SCALE SPECTRUM FOR PLOTTING
0142 CALL VSMUL(0,2,8189,0,2,NN)
C SCALE PHASE FOR PLOTTING
0143 CALL VSMUL(1,2,8190,1,2,NN)
```

```

0144 CALL VSADD(1,2,8191,1,2,NN)
0145 CALL APWR
0146 CALL APGET(AVAL,0,LN,2)
0147 CALL APWD
      C AVAL(1,MN)=AMPLITUDE:PHASE
      C AVAL(LN)=SCALING FACTOR FOR AMPLITUDE
      C PLOT AMPLITUDE SPECTRUM
0148 1051 CALL PLOT(0.,0.,3)
0149 CALL NEWPEN(3)
0150 Y=0.0
0151 ISTART=1
0152 IF(ICNTRL.NE.0) ISTART=2
0153 DO 106 I=ISTART,MN,2
0154 CALL PLOT(AVAL(I),Y,2)
0155 106 Y=Y+DFPLOT
0156 IF(ICNTRL.NE.0) GOTO 208
      C MARK IN SCALING FACTOR
0157 CALL NEWPEN(1)
0158 X=4.5*CSIZE
0159 Y=0.
      C FLOAT SCALING FOR DISPLAY
0160 ASCALE=1.0
0161 BSCALE=0.
0162 BINDEX=0.
0163 IF(AVAL(LN).LT.1.0) ASCALE=10.0
0164 IF(AVAL(LN).GT.1.0) ASCALE=0.1
0165 IF(ASCALE.EQ.10.0) BINDEX=-1.0
0166 IF(ASCALE.EQ.0.1) BINDEX=1.0
0167 206 CONTINUE
0168 IF(AVAL(LN).GE.1.0.AND.AVAL(LN).LT.10.0) GOTO 207
0169 AVAL(LN)=AVAL(LN)*ASCALE
0170 BSCALE=BSCALE+BINDEX
0171 GOTO 206
0172 207 CALL SYMBOL(X,Y,CSIZE,21HSCALING FACTOR= *10,90.0,21)
0173 Y=Y+15.0*CSIZE
0174 CALL NUMBER(X,Y,CSIZE,AVAL(LN),90.0,1)
0175 Y=Y+6.0*CSIZE
0176 DELX=0.5*CSIZE
0177 X=X-DELX
0178 CALL NUMBER(X,Y,DELX,BSCALE,90.0,-1)
0179 ICNTRL=1
0180 CALL PLOT(0.,0.,-999)
0181 GOTO 1031
0182 208 X=4.5*CSIZE
0183 Y=0.0
0184 CALL SYMBOL(X,Y,CSIZE,'CHANNEL NUMBER = ',90.0,17)
0185 Y=Y+16.0*CSIZE
0186 RCHAN=FLOAT(ICHAN)
0187 CALL NUMBER(X,Y,CSIZE,RCHAN,90.0,-1)
0188 ICNTRL=0
0189 999 IF(IPLT.EQ.0) CALL PLOT(0.,0.,999)
0190 RETURN
0191 END

```

Velocity Analysis :- MPVEL

Input file.....DK2:ANINV.DAT

Output file.....DK2:ANOUTV.DAT

Input ParametersREAD(1,1000)L,NCHAN,NSTART,M1000 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)FBUF1200 FORMAT(12X,3A4)

FBUF.....Input data file name

READ(1,1100)XSTART,XSTEP,FSAMP1100 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

```

0001      DIMENSION NMOFS(4096),T0SQ(512),NV(512),VINSQ(170),SMBLCE(170),
0002      1SQ(1024),S1(1024),XSQ(24),
0003      2JSTCH(24),JST(24),NWDS(24),NEL(24),JBLK(24),CONST(7),FBUF(3)
0004      LOGICAL*1 SW
0005      REAL*8 FSPEC
0006      DATA K1,K2,KNCHIN,KBLK,KBLKIN,KADD2,KADD1,KHF,KFS,NBUF
0007      1/1,2,8185,8186,8187,8188,8189,8190,8191,1024/
0008      DATA DEV/3RDK /
0009
0010      C
0011      C ASSIGN INPUT AND OUTPUT CHANNELS
0012      CALL ASSIGN(1,'DK2:ANINV.DAT',13)
0013      CALL ASSIGN(2,'DK2:ANOUTV.DAT',14)
0014      IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0015      IDCH=IGETC()
0016
0017      C READ IN REQUIRED INPUT PARAMETERS
0018      READ(1,1000) L,NCHAN,NSTART,M
0019      1000  FORMAT(4I5)
0020      READ(1,1200) FBUF
0021      READ(1,1100) XSTART,XSTEP,FSAMP
0022      1100  FORMAT(10F10.0)
0023      READ(1,1100) T01,T02,T0STEP,TGATE
0024      READ(1,1100) VSTEP,V1,V2MIN,V2MAX,VICPT,VGRAD
0025      1200  FORMAT(12X,3A4)
0026
0027      C VALIDATE DATA AND CALCULATE REQUIRED ARRAYS AND CONSTANTS.
0028      C FORM ARRAY JSTCH CORRESPONDING TO THE BEGINNING BLOCK
0029      C NUMBERS FOR EACH TRACE ON DISK.
0030      C AT SAME TIME FORM ARRAY XSQ OF SQUARED SHOT RECEIVER SEPARATIONS
0031      JSTCH(1)=1
0032      NREAD=L*M/128
0033      X=XSTART
0034      XSQ(1)=X**2
0035      DO 10 JCHAN=2,NCHAN
0036      JSTCH(JCHAN)=JSTCH(JCHAN-1)+NREAD
0037      X=X+XSTEP
0038      XSQ(JCHAN)=X**2
0039      10  CONTINUE
0040
0041      C CALCULATE FSAMPM,THE SAMPLING FREQUENCY AFTER INTERPOLATION
0042      FSAMPM=FLOAT(M)*FSAMP
0043
0044      C CALCULATE TSAMP THE SAMPLE INTERVAL
0045      TSAMP=1.0/FSAMP
0046
0047      C CALCULATE TSTART,THE BEGINNING TIME
0048      TSTART=NSTART*TSAMP
0049
0050      C CALCULATE NSTM THE INITIAL SAMPLE NUMBER AFTER INTERPOLATION.
0051      NSTM=M*NSTART
0052
0053      C TRUNCATE T0STEP TO BE INTEGRAL MULTIPLE OF TSAMP
0054      T0STEP=TSAMP*IFIX(T0STEP*FSAMP)
0055
0056      C ROUND TGATE TO BE EVEN INTEGRAL MULTIPLE OF TSAMP
0057      C NGATE IS THE NUMBER OF ELEMENTS IN GATE(BEFORE INTERPOLATION)
0058      C NGTM2 IS THE NUMBER IN HALF OF GATE,AFTER INTERPOLATION.
0059      NGATE=IFIX(TGATE*FSAMP+1.0)/2*2
0060      TGATE=FLOAT(NGATE)*TSAMP
0061      NGATE=NGATE+1
0062      NGTM2=M*(NGATE/2)
0063
0064      C ENSURE THAT T01 IS A LEAST TSTART+TGATE/2

```

```

0030      T01=AMAX1(T01,TSTART+0.5*TGATE)
0031      C ROUND T01 UP TO BE INTEGRAL MULTIPLE OF TSAMP
0032      T01=TSAMP*(IFIX(T01*FSAMP)+1)
0033      C TRUNCATE T02 IF NECESSARY SO THAT THERE IS SUFFICIENT
0034      C DATA TO DO VELOCITY ANALYSIS AT T02
0035      T02MAX=SQRT((TSTART+(L-1)*TSAMP-0.5*TGATE)**2-
0036      1XSQ(NCHAN)/V1**2)
0037      T02=AMIN1(T02,T02MAX)
0038      CALCULATE NT0, THE NUMBER OF TWO WAY TIMES CONSIDERED
0039      NT0=IFIX((T02-T01)/T0STEP)+1
0040      T02=T01+(NT0-1)*T0STEP
0041      IF(NT0.LE.0)STOP 'NT0 NOT POSITIVE'
0042      C FORM ARRAYS T0SQ (SQUARED TWO WAY TIME) AND NV (NUMBER
0043      C OF VELOCITY POINTS AT EACH TWO WAY TIME)
0044      T0J=T01
0045      DO 20 JT0=1,NT0
0046      T0SQ(JT0)=T0J**2
0047      V2=VICPT+VGRAD*T0J
0048      V2=AMAX1(V2MIN,V2)
0049      V2=AMIN1(V2,V2MAX)
0050      NV(JT0)=IFIX((V2-V1)/VSTEP)+1
0051      T0J=T0J+T0STEP
0052  20  CONTINUE
0053      C VINSQ IS THE ARRAY OF INVERSE SQUARED VELOCITIES
0054      NVMX=NV(NT0)
0055      IF(NVMX.GT.170)STOP 'NVMX GT 170'
0056      V=V1
0057      DO 30 JV=1,NVMX
0058      VINSQ(JV)=1.0/V**2
0059      V=V+VSTEP
0060  30  CONTINUE
0061      C NOW CALCULATE NWMX, AN UPPER BOUND ON THE SIZE OF THE
0062      C DATA WINDOW. IF THIS EXCEEDS NBUF PROGRAM STOPS.
0063      NWMX=IFIX(FSAMP*SQRT(T0SQ(1)+XSQ(NCHAN)*VINSQ(1))+0.5)
0064      1-IFIX(FSAMP*SQRT(T0SQ(1)+XSQ(NCHAN)*VINSQ(NVMX))+0.5)+2*NGTM2+1
0065      IF(NWMX.GT.NBUF-127)STOP 'NWMX TOO LARGE'
0066      C CONST IS ARRAY OF CONSTANTS USED IN AP
0067      CONST(1)=1.0/FLOAT(NCHAN)
0068      CONST(2)=-128.
0069      CONST(3)=1.0/128.
0070      CONST(4)=FLOAT(NGTM2-NSTM+1)
0071      CONST(5)=-FLOAT(NGTM2+NSTM)
0072      CONST(6)=0.5
0073      CONST(7)=FSAMP
0074      C INITIALISE AP AND PLACE CONSTANTS IN TOP 7 LOCATIONS
0075      CALL APINIT
0076      CALL APWR
0077      CALL APPUT(CONST,KNCHIN,7,K2)
0078      C CALCULATE THOSE ADDRESSES WHICH ARE CONSTANT IN NMO
0079      C AND SEMBLANCE COMPUTATIONS.
0080      IVINSQ=NCHAN+1
0081      ISSQ=NWMX
0082      IEN=ISSQ+1
0083      C LOOKUP FILE TO BE READ FROM

```

```

0070 CALL IRAD50(12,FBUF,FSPEC)
0071 IF(LOOKUP(IDCH,FSPEC).LT.0)STOP 'LOOKUP ERROR'
0081 WRITE(2) L,NCHAN,NSTART,M
0082 WRITE(2) XSTART,XSTEP,FSAMP
0083 WRITE(2) T01,T02,T0STEP,TGATE
0084 WRITE(2) VSTEP,V1,V2MIN,V2MAX
0085 WRITE(2) NT0

```

```

0086 DO 500 JT0=1,NT0
0087 NVD=NV(JT0)

```

```

C-----
C THE PURPOSE OF NMO IS TO CALCULATE ALL THE NORMAL MOVEOUTS AND
C AUXILIARY INFORMATION NECESSARY TO CALCULATE THE SEMBLANCE FOR
C THE REQUIRED RANGE OF VELOCITIES AT A GIVEN VERTICAL INCIDENCE
C TIME.
C-----

```

```

C OUTPUT ARGUMENTS AND METHOD OF COMPUTATION
C-----

```

```

C NMOFS IS ARRAY CONTAINING ALL THE REQUIRED NORMAL MOVEOUT OFFSETS.
C ALTHOUGH IT IS A LINEAR ARRAY IT IS USED IN A 'TWO DIMENSIONAL'
C MANNER TO STORE ALL THE OFFSETS AS A FUNCTION OF BOTH SEISMIC
C CHANNEL NUMBER AND VELOCITY. THUS THE FIRST NVD VALUES ARE
C THE OFFSETS FOR CHANNEL 1, IN ORDER OF INCREASING VELOCITY, THE NEXT
C NVD VALUES THE OFFSETS FOR CHANNEL 2 AND SO ON. THE OFFSETS
C ARE ROUNDED TO THE NEAREST (INTERPOLATED) SAMPLE AND, FOR A GIVEN
C SEISMIC CHANNEL, ARE RELATIVE TO THE BEGINNING OF THE DATA WINDOW
C THAT IS TO BE TAKEN INTO THE AP TO DO THE PARTIAL SEMBLANCE
C CALCULATIONS FOR THAT CHANNEL, AT TIME JT0.

```

```

C AT A TIME JT0, AND FOR A CHANNEL JCHAN THE DATA WINDOW THAT IS
C TAKEN INTO THE AP (IN ROUTINE SEMB) MUST COVER ALL TIMES FROM
C THE CROSSING OF THE SHALLOWEST ARRIVAL TRAJECTORY (VELOCITY OF
C V(NVD)) TO THE CROSSING OF THE STEEPEST ARRIVAL TRAJECTORY
C (VELOCITY OF V(NV(1))). IN ADDITION A HALF GATEWIDTH OF DATA IS
C REQUIRED AT EITHER END.

```

```

C SET UP INITIAL ADDRESSES

```

```

0088 NVNC=NVD*NCHAN
0089 NTOT=NVNC+4*NCHAN
0090 INMC=IVINSQ+NVD
0091 I1=INMO+NVNC
0092 I2=I1+NCHAN
0093 I3=I2+NCHAN
0094 I4=I3+NCHAN
0095 I5=I4+NCHAN
0096 IADD=I5+NCHAN

```

```

C TRANSFER T0SQ TO 0

```

```

0097 CALL APPUT(T0SQ(JT0),0,K1,K2)

```

```

C TRANSFER XSQ TO (1,NCHAN)

```

```

0098 CALL APPUT(XSQ,1,NCHAN,K2)

```

```

C TRANSFER VINSQ TO (IVINSQ,NVD)

```

```

0099 CALL APPUT(VINSQ,IVINSQ,NVD,K2)

```

```

C SET UP INITIAL ADDRESSES FOR RESULTS

```

```

0102      IRES=INMO
0103      IRNV1=IRES+NVD-1
0104      I1J=I1
0105      I2J=I2
0106      CALL APWD
0107      C ITERATE THROUGH CHANNELS
0108      DO 40 JCHAN=1,NCHAN
0109      C FORM XSQ(JCHAN)*VINSQ(JV)+T0SQ(JT0) IN (RES,NVD)
0110      CALL VSMSA(IVINSQ,K1,JCHAN,0,IRES,K1,NVD)
0111      C FORM SQRT(RES,NVD)
0112      CALL VSQRT(IRES,K1,IRES,K1,NVD)
0113      C FORM INT(FSAMPM*SQRT(T0SQ(JT0)+XSQ(JCHAN)*VINSQ(JV))+0.5) USING FSAMPM
0114      C IN KFS AND 0.5 IN KHF
0115      CALL VSMSA(IRES,K1,KFS,KHF,IRES,K1,NVD)
0116      CALL VINT(IRES,K1,IRES,K1,NVD)
0117      C FILL IADD WITH VALUE FROM IRNV1 AND NEGATE IT
0118      CALL VFILL(IRNV1,IADD,K1,K1)
0119      CALL VNEG(IADD,K1,IADD,K1,K1)
0120      C FILL I1J WITH VALUE FROM IRNV1 (VELOCITY V2)
0121      C AND I2J WITH VALUE FROM IRES (VELOCITY V1)
0122      CALL VFILL(IRNV1,I1J,K1,K1)
0123      CALL VFILL(IRES,I2J,K1,K1)
0124      C ADD VALUE IN IADD TO (IRES,NVD), I.E. SUBTRACT MOVEOUT FOR VELOCITY V2
0125      C TO GIVE NMOFS
0126      CALL VSADD(IRES,K1,IADD,IRES,K1,NVD)
0127      C REDEFINE ADDRESSES SO THAT RESULTS FOR NEXT CHANNEL FOLLOW ON
0128      IRES=IRES+NVD
0129      IRNV1=IRNV1+NVD
0130      I1J=I1J+1
0131      I2J=I2J+1
0132      40  CONTINUE
0133      C ADD -NSTM-NGTM2 TO (I1,NCHAN) USING VALUE IN KADD1
0134      CALL VSADD(I1,K1,KADD1,I1,K1,NCHAN)
0135      C ADD -NSTM+NGTM2+1 TO (I2,NCHAN) USING VALUE IN KADD2
0136      CALL VSADD(I2,K1,KADD2,I2,K1,NCHAN)
0137      C SUBTRACT (I1,NCHAN) FROM (I2,NCHAN) TO FORM (I3,NCHAN)
0138      CALL VSUB(I1,K1,I2,K1,I3,K1,NCHAN)
0139      C MULTIPLY (I1,NCHAN) BY 1.0/NSBLK FROM KBLKIN TO FORM (I4,NCHAN)
0140      CALL VSMUL(I1,K1,KBLKIN,I4,K1,NCHAN)
0141      C TRUNCATE (I4,NCHAN) TO INTEGER VALUES
0142      CALL VINT(I4,K1,I4,K1,NCHAN)
0143      C MULTIPLY (I4,NCHAN) BY -NSBLK FROM KBLK TO FORM (I5,NCHAN)
0144      CALL VSMUL(I4,K1,KBLK,I5,K1,NCHAN)
0145      C ADD (I5,NCHAN) TO (I1,NCHAN)
0146      CALL VADD(I1,K1,I5,K1,I1,K1,NCHAN)
0147      C ADD 1.0 FROM TM TO (I1,NCHAN)
0148      CALL VTSADD(I1,K1,2049,I1,K1,NCHAN)
0149      C ADD (I5,NCHAN) TO (I2,NCHAN) AND MULTIPLY BY 2.0 FROM TM
0150      CALL VADD(I2,K1,I5,K1,I2,K1,NCHAN)
0151      CALL VTSMUL(I2,K1,2050,I2,K1,NCHAN)
0152      C FIX ALL VALUES FROM INMO TO I5 PREPARATORY T
0153      C TRANSFER TO HOST
0154      CALL VFIX(INMO,K1,INMO,K1,NTOT)
0155      C TRANSFER TO HOST ARRAYS

```

```

0130 CALL APWR
0131 CALL APGET(NMOFS,INMO,NVNC,K1)
0132 CALL APGET(JST,I1,NCHAN,K1)
0133 CALL APGET(NWDS,I2,NCHAN,K1)
0134 CALL APGET(NEL,I3,NCHAN,K1)
0135 CALL APGET(JBLK,I4,NCHAN,K1)
0136 C SET UP INITIAL ADDRESSES FOR SEMBLANCE CALCULATIONS
0137 ISEMB=IEN+NVD
0138 ISEMB1=ISEMB-1
0139 ISMAX=ISEMB+NVD
0140 ISMAX1=ISMAX+1
0141 ISTK=ISMAX+2
0142 CALL APWD
0143 C CLEAR SECTIONS OF AP MEMORY THAT WILL HAVE DATA ADDED.
0144 CALL VCLR(IEN,K1,NVD)
0145 NVNG=NVD*NGATE
0146 CALL VCLR(ISTK,K1,NVNG)
0147 C
0148 C FILL BUFFER S0 WITH DATA WINDOW FOR CHANNEL 1
0149 IF(IREAD(NWDS(1),S0(1),JBLK(1)+1,IDCH).LT.0)STOP'READ ERROR'
0150 C DEFINE INITIAL VALUE OF BUFFER SWITCH;
0151 C SW=.TRUE.,READ FROM S0,INTO S1
0152 C SW=.FALSE.,READ FROM S1,INTO S0.
0153 SW=.TRUE.
0154 C
0155 C ITERATE THROUGH CHANNELS
0156 JNMO=1
0157 DO 200 JCHAN=1,NCHAN
0158 JCHAN1=JCHAN+1
0159 C TRANSFER DATA FROM BUFFER TO AP
0160 CALL APWR
0161 CALL IWAIT(IDCH)
0162 IF(SW)CALL APPUT(S0(JST(JCHAN)),0,NEL(JCHAN),K2)
0163 IF(.NOT.SW)CALL APPUT(S1(JST(JCHAN)),0,NEL(JCHAN),K2)
0164 IF(JCHAN1.GT.NCHAN)GOTO500
0165 C TRANSFER NEW DATA FROM DISK TO BUFFERS
0166 IF(SW)IERR=IREAD(NWDS(JCHAN1),S1(1),JSTCH(JCHAN1)+JBLK(JCHAN1),
0167 IDCH)
0168 IF(.NOT.SW)IERR=IREAD(NWDS(JCHAN1),S0(1),JSTCH(JCHAN1)+
0169 JBLK(JCHAN1),IDCH)
0170 IF(IERR.LT.0)STOP'READ ERROR'
0171 SW=.NOT.SW
0172 C NOW DO SEMBLANCE CALCULATIONS ON DATA WITHIN AP
0173 DO 100 JV=1,NVD
0174 C ADD APPROPRIATELY MOVED OUT GATE ONTO STACK USING ONLY EVERY MTH
0175 C DATA POINT
0176 CALL VADD(ISTK,K1,NMOFS(JNMO),M,ISTK,K1,NGATE)
0177 C FORM SUM OF SQUARES OF ELEMENTS ADDED ONTO STACK
0178 CALL SVESQ(NMOFS(JNMO),M,ISSQ,NGATE)
0179 IENJ=ISSQ+JV
0180 CALL VADD(IENJ,K1,ISSQ,K1,IENJ,K1,K1)
0181 C REDEFINE ISTK SO THAT NEXT STACK FOLLOWS ON
0182 ISTK=ISTK+NGATE

```

```
0174      JNMO=JNMO+1
0175 1000   CONTINUE
0176      ISTK=ISMAX+2
0177 1000   CONTINUE
C
C NOW ACCUMULATE SEMBLANCE
0178      DO 300 JV=1,NVD
C FIND MEAN SUM OF SQUARES FOR EACH STACK
0179      CALL SVESQ(ISTK,K1,ISEMB1+JV,NGATE)
0180      ISTK=ISTK+NGATE
0181 300    CONTINUE
C DIVIDE MEAN STACK ENERGIES BY SUMMED ENERGY
0182      CALL VDIV(IEN,K1,ISEMB,K1,ISEMB,K1,NVD)
C FIND MAXIMUM VALUE OF SEMBLANCE
0183      CALL MAXV(ISEMB,K1,ISMAX,NVD)
C DIVIDE BY NCHAN TO NORMALISE SEMBLANCE AND ITS MAX VALUE
0184      NV1=NVD+1
0185      CALL VSMUL(ISEMB,K1,KNCHIN,ISEMB,K1,NV1)
C WRITE OUT SEMBLANCE AND ITS MAXIMUM VALUE TO SMLCE
0186      CALL APWR
0187      CALL APGET(SMLCE,ISEMB,NV1,K2)
C WRITE OUT ADDRESS OF MAXIMUM VALUE TO IMAX
0188      CALL APGET(IMAX,ISMAX1,K1,K1)
C CALCULATE JVSMX, VALUE OF JV FOR WHICH SEMBLANCE IS MAXIMUM
0189      JVSMX=IMAX-ISEMB1
C WRITE OUT SEMBLANCE TO OUTPUT DEVICE 2
0190      WRITE(2) NVD,JVSMX,(SMLCE(JV),JV=1,NVD)
0191 5000   CONTINUE
0192      CALL CLOSEC(IDCH)
0193      STOP
0194      END
```

NMO corrected gathers :- MPCDP

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=====

C MAIN PROGRAM

C-----

```

0001 REAL*8 FSPECR,FSPECW
0002 VIRTUAL T0SQ(2048),VINSQ(2048)
0003 DIMENSION MUTE(24),XSQ(24),FBUF(3)
0004 COMMON /LAYER/T0LYR(99),VLWR(99)
0005 DATA DEV/3RRK /
0006 CALL ASSIGN(1,'DK1:ANCDP.DAT',13)
0007 IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0008 IDCH=IGETC()
0009 IDCH1=IGETC()
0010 READ(1,1000)L,NCHAN,NSTART,NLYR,M
0011 1000 FORMAT(12I5)
0012 READ(1,1100)XSTART,XSTEP,FSAMP
0013 1100 FORMAT(3F10.0)
0014 READ(1,1200)(T0LYR(I),VLWR(I),I=1,NLYR)
0015 1200 FORMAT(2F10.0)
0016 READ(1,1000)INDEN,INDMUT
0017 IF(INDMUT.GE.0)READ(1,1000)(MUTE(I),I=1,NCHAN)
0018 READ(1,1300)FBUF
0019 1300 FORMAT(3A4)
0020 CALL IRAD50(12,FBUF,FSPECR)
0021 READ(1,1300)FBUF
0022 CALL IRAD50(12,FBUF,FSPECW)
0023 IF(LOOKUP(IDCH,FSPECR).LT.0)STOP'LOOKUP ERROR'
0024 NBLKS=L*(NCHAN+1)/128+1
0025 IF(IENTER(IDCH1,FSPECW,NBLKS).LT.0)STOP'ENTER ERROR'
0026 L2INT=2
0027 DO 5 K=1,1000
0028 IF(L2INT.GE.L) GOTO 10
0029 L2INT=2*L2INT
0030 5 CONTINUE
0031 10 X=XSTART
0032 DO 20 JCHAN=1,NCHAN
0033 XSQ(JCHAN)=X**2
0034 X=X+XSTEP
0035 20 CONTINUE
0036 TSAMP=1./FSAMP
0037 T0=NSTART*TSAMP
0038 DO 30 JT0=1,L
0039 T0SQ(JT0)=T0**2
0040 T0=T0+TSAMP
0041 30 CONTINUE
0042 CALL SETAP(L2INT,M)
0043 CALL INVSQ(FSAMP,NSTART,NLYR,L,VINSQ)
0044 CALL CDP(L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,
0045 1INDEN,INDMUT,IDCH,IDCH1,T0SQ,VINSQ,XSQ)
0046 STOP
0047 END
    
```

C-----

BLOCK DATA

C-----

```

0001 IMPLICIT INTEGER*2(I-N)
0002 COMMON /KONST/ K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
0003 1IA,IB,IC,ID,ITOP
0004 DATA K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,IA,IB,IC,ID,ITOP
0005 1/-1,0,1,2,3,4,5,6,7,8,9,10,0,2048,4096,6144,8191/
0006 END
    
```

```

C-----
0001      SUBROUTINE INVSQ(FSAMP,NSTART,NLYR,L,VINSQ)
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.
0002      VIRTUAL VINSQ(2048)
0003      COMMON /LAYER/T0LYR(99),VLYR(99)
0004      N1=1
0005      N2=IFIX(FSAMP*T0LYR(1))-NSTART
0006      VINSQ1=1.0/VLYR(1)**2
0007      DO 10 I=N1,N2
0008      VINSQ(I)=VINSQ1
0009  10    CONTINUE
0010      IF(NLYR.EQ.1) GOTO 40
0011      DO 30 J=2,NLYR
0012      N1=N2+1
0013      N2=IFIX(FSAMP*T0LYR(J))-NSTART
0014      DELV=(VLYR(J)-VLYR(J-1))/(N2-N1+2)
0015      V=VLYR(J-1)
0016      DO 20 I=N1,N2
0017      VINSQ(I)=1.0/V**2
0018      V=V+DELV
0019  20    CONTINUE
0020  30    CONTINUE
0021  40    N1=N2+1
0022      N2=L
0023      VINSQN=1.0/VLYR(NLYR)**2
0024      DO 50 I=N1,N2
0025      VINSQ(I)=VINSQN
0026  50    CONTINUE
0027      RETURN
0028      END

```

```

C-----
0001      SUBROUTINE SETAP(L2INT,M)
C-----
C SETS UP COMPLEX EXPONENTIAL TABLE IN D+1
C AND PUTS 1.0 IN ID
0002      COMMON /KONST/ K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
0003      11A,IB,IC,ID,ITOP
C INITIALISE AP
0004      CALL APINIT
C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS.
0005      ID1=ID+1
0006      ID2=ID+2
0007      L2I21=L2INT/2-1
C TRANSFER 1.0/(L2INT*M) TO K0 AND MULTIPLY BY 2PI FROM TM.
0008      CONST=1.0/FLOAT(L2INT*M)
0009      CALL APWR
0010      CALL APPUT(CONST,K0,K1,K2)
0011      CALL APWD
C FORM VECTOR RAMP AND TAKE SIN AND COS OF IT
0012      CALL VTSMUL(K0,K1,2317,K0,K1,K1)
0013      CALL VRAMP(K0,K0,IB,K1,L2I21)
0014      CALL VCOS(IB,K1,ID1,K2,L2I21)
0015      CALL VSIN(IB,K1,ID2,K2,L2I21)
C PUT 1.0 IN ID
0016      CALL VCLR(ID,K1,K1)
0017      CALL VTSADD(ID,K1,2049,ID,K1,K1)
0018      RETURN
0019      END

```

```

0007 SUBROUTINE CDP(L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,
0008 IINDEN,INDMUT,IDCH,IDCH1,T0SQ,VINSQ,XSQ)
0009 VIRTUAL SEISM(32767),T0SQ(2048),VINSQ(2048)
0010 DIMENSION INDEX(2048),MUTE(24),STACK(2048),
0011 IDUM(10),CONST(10),FSTAP(20),XSQ(24)
0012 COMMON /KONST/ K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
0013 1IA,IB,IC,ID,ITOP
0014
0015 C
0016 C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS
0017 IB1=IB+1
0018 IC1=IC+1
0019 IC2=IC+2
0020 ID1=ID+1
0021 L1=L+1
0022 L2I2I=L2INT/2-1
0023 CONST(2)=M*FSAMP
0024 CONST(3)=0.5
0025 CONST(4)=-M*NSTART
0026 CONST(5)=0.0
0027 CONST(6)=L*M
0028 CONST(7)=L+1
0029 CONST(8)=1.0
0030 CONST(9)=1.0/M
0031 CONST(10)=(L+1)*M-1
0032 JBLK=1
0033 NBLKTR=L/128
0034 FSTD=ADGET(SEISM(1))
0035 FT0SQ=APGAD(T0SQ(1))
0036 FVINSQ=APGAD(VINSQ(1))
0037 ISTART=1
0038 DO 50 K=1,M
0039 FSTAP(K)=APGAD(SEISM(ISTART))
0040 ISTART=ISTART+1
0041 50 CONTINUE
0042 C SET SEISM(L+1) TO 0.0 TO COPE WITH TIME OVERFLOW AND CLEAR A IN AP
0043 SEISM(L1)=0.0
0044 CALL VCLR(IA,K1,L)
0045 C ITERATE THROUGH CHANNELS
0046 DO 500 JCHAN=1,NCHAN
0047 C READ IN TRACE SEISM FROM UNIT 2
0048 CALL IWAIT(IDCH1)
0049 IF(IREADA(IDCH,2*L,JBLK,FSTD).LT.0)STOP'READA ERROR'
0050 CALL IWAIT(IDCH)
0051 C COMPUTE INDEX ARRAY
0052 CONST(1)=XSQ(JCHAN)
0053 CALL APWR
0054 C TRANSFER (K1,K10) TO HOST DUMMY ARRAY DUM AND REPLACE BY CONST
0055 CALL APGET(DUM,K1,K10,K2)
0056 CALL APPUT(CONST,K1,K10,K2)
0057 C TRANSFER T0SQ TO B AND VINSQ TO IC1
0058 CALL APPUTA(IB,L,K2,FT0SQ)
0059 CALL APPUTA(IC1,L,K2,FVINSQ)
0060 CALL APWD
0061 C FORM XSQ(JCHAN)*VINSQ IN C1

```

```

0044      CALL VSMUL(IC1,K1,K1,IC1,K1,L)
C FORM SQRT(T0SQ+XSQ(JCHAN)*VINSQ) IN B
0045      CALL VADD(IB,K1,IC1,K1,IB,K1,L)
0046      CALL VSQRT(IB,K1,IB,K1,L)
C FORM IFIX(FSAMP*M*SQRT(T0SQ+XSQ(JCHAN)*VINSQ)+0.5)-NSTART*M IN B
0047      CALL VSMSA(IB,K1,K2,K3,IB,K1,L)
0048      CALL VINT(IB,K1,IB,K1,L)
0049      CALL VSADD(IB,K1,K4,IB,K1,L)
C CLIP B BETWEEN 0.0 AND L*M
0050      CALL VCLIP(IB,K1,K5,K6,IB,K1,L)
C MULTIPLY B BY L+1 ,ADD 1.0 AND PLACE RESULT IN C1
0051      CALL VSMSA(IB,K1,K7,K8,IC1,K1,L)
C MULTIPLY B BY 1.0/M AND TAKE INTEGER PART
0052      CALL VSMUL(IB,K1,K9,IB,K1,L)
0053      CALL VINT(IB,K1,IB,K1,L)
C MULTIPLY B BY (L+1)*M-1
0054      CALL VSMUL(IB,K1,K10,IB,K1,L)
C SUBTRACT B FROM C1 AND PUT RESULT IN B
0055      CALL VSUB(IB,K1,IC1,K1,IB,K1,L)
C FIX B AND TRANSFER BACK AS HOST ARRAY INDEX
0056      CALL VFIX(IB,K1,IB,K1,L)
0057      CALL APWR
0058      CALL APGET(INDEX,IB,L,K1)
C TRANSFER DUM BACK TO K1
0059      CALL APPUT(DUM,K1,K10,K2)
0060      CALL APWD
C
C HAVING COMPUTED INDEX ARRAY NOW START INTERPOLATION OF TRACE
C
C CLEAR B IN AP
0061      CALL VCLR(IB,K1,L2INT)
C TRANSFER TRACE TO B
0062      CALL APWR
0063      CALL APPUTA(IB,L,K2,FSTAP(1))
0064      CALL APWD
C FIND MEAN VALUE OF TRACE AND PLACE IN AP(ITOP)
0065      CALL MEANV(IB,K1,ITOP,L)
C SUBTRACT MEAN FROM TRACE
0066      CALL VNEG(ITOP,K1,ITOP,K1,K1)
0067      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)
0068      IF(INDEN.LT.0) GOTO 100
0069      CALL RMSQV(IB,K1,ITOP,L)
0070      IF(INDEN.GT.0) CALL VSQ(ITOP,K1,ITOP,K1,K1)
C USE 1.0 RESIDING IN ID (FROM SETAP)
0071      CALL VDIV(ITOP,K1,ID,K1,ITOP,K1,K1)
0072      CALL VSMUL(IB,K1,ITOP,IB,K1,L)
C IF INDMUT IS NEGATIVE USE NO MUTING
0073      100 IF(INDMUT.LT.0) GOTO 200
0074      CALL VCLR(IB,K1,MUTE(ICHAN))
C TRANSFER MODIFIED TRACE BACK TO SEISM
0075      200 CALL APWR

```

```

      CALL APGETA(IB,L,K2,FSTAP(1))
      CALL APWD
      I=(N,EO,1) GOTO 350
C MOVE DATA FORM OF TRACE AND PUT IN C
      CALL RFFYB(IB,IC,L2INT,K1)
      CALL RFFYSC(IC,L2INT,K0,K1)
      CALL VCLR(IC,K1,K2)
C INTERFERE FROM 2 TO M
      DO 350 K=2,M
C MULTIPLY TRANSFORM BY COMPLEX EXPONENTIAL ARRAY
      CALL CVMUL(IC2,K2,IB1,K2,IC2,K2,L2I21,K1)
C MOVE C TO B AND TAKE IN PLACE IFFT
      CALL VMOV(IC,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 APWD
      CONTINUE
C MOVE REQUIRED ELEMENTS OUT OF SEISM AND PLACE IN STACK
      DO 400 I=1,L
      STACK(I)=SEISM(INDEX(I))
      CONTINUE
C TRANSFER STACK INTO B1 AND ADD TO STACK IN A
      CALL APWR
      CALL APPUT(STACK,IB1,L,K2)
      CALL APWD
      IF(IWRITE(2*L,STACK,JBLK,IDCH1).LT.0)STOP'WRITE ERROR'
      JBLK=JBLK+NBLKTR
      CALL VADD(IA,K1,IB1,K1,IA,K1,L)
      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 IWAIT(IDCH1)
      CALL APGET(STACK,IA,L,K2)
      CALL APWD
      IF(IWRITE(2*L,STACK,JBLK,IDCH1).LT.0)STOP'WRITE ERROR 2'
      CALL IWAIT(IDCH1)
      CALL CLOSEC(IDCH1)
      RETURN
      END

```

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-----
0001 REAL*8 FSPECR,FSPECW,FNAMR,FNAMO
0002 VIRTUAL T0SQ(2048),VINSQ(2048),FNAMR(100),FNAMO(100),
      %T0LYR(20),VLYR(20),T0INT(20),VINT(20),IVELAN(100),
      %T02LYR(20),V2LYR(20)
0003 DIMENSION MUTE(24),XSQ(24),FBUF(3),IHBLK(256)
0004 INTEGER*2 OUTFLG
0005 LOGICAL*1 TPDRR,TPDRW,STATUS,ITLEN,LBLK(512)
0006 EQUIVALENCE (IHBLK(1),LBLK(1))
0007 COMMON/STK/ L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,INDEN,
      %INDMUT,IDCH,IDCH1,XSQ
0008 DATA DEV/3RRK /
C
C SET UP I/O CHANNELS AND READ IN CONTROL DATA
C
0009 IF(ICDFN(25).NE.0)STOP 'CHANNELS FULL '
0011 CALL ASSIGN(1,'DK1:MPSTAK.DAT',14)
0012 CALL ASSIGN(2,'DK1:MPSTAK.LOG',14)
0013 IF(IFETCH(DEV).NE.0)STOP 'FETCH ERROR'
0015 IEOTR=0
0016 IEOTW=0
0017 IDCH=20
0018 IDCH1=21
0019 READ(1,1000)NFILES,NVEL,L,NCHAN,NSTART,N2LYR,M
0020 1000 FORMAT(12I5)
0021 READ(1,1000)TPDRR,TPDRW,INTSW
0022 READ(1,1000)(IVELAN(I),I=1,NVEL)
0023 READ(1,1100)XSTART,XSTEP,FSAMP
0024 1100 FORMAT(3F10.0)
0025 READ(1,1200)(T02LYR(I),V2LYR(I),I=1,N2LYR)
0026 1200 FORMAT(2F10.0)
0027 READ(1,1000)INDEN,INDMUT,INFLG,OUTFLG
0028 IF(INDMUT.GE.0)READ(1,1000)(MUTE(I),I=1,NCHAN)
C
C READ IN FILE SPECS FOR INPUT DEPENDING ON
C IF IT IS FROM TAPE OF DISC
C
0030 IF(INFLG.NE.0)GOTO 40
0032 READ(1,1300)FBUF
0033 1300 FORMAT(3A4)
0034 CALL IRAD50(12,FBUF,FSPECR)
0035 GOTO 50
0036 40 DO 60 I=1,NFILES
0037 READ(1,1200)FBUF
0038 CALL IRAD50(12,FBUF,FSPECR)
0039 FNAMR(I)=FSPECR
0040 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

```

0041 50 IF(OUTFLG.NE.0)GOTO 70
0042 READ(1,1300)FBUF
0043 CALL IRAD50(12,FBUF,FSPECW)
0044 GOTO 80
0045
0046 70 DO 90 I=1,NFILES
0047 READ(1,1300)FBUF
0048 CALL IRAD50(12,FBUF,FSPECW)
0049 FNAMO(I)=FSPECW
0050 90 CONTINUE

```

C

C SET UP CONSTANTS ARRAYS

C

```

0051 80 L2INT=2
0052 DO 5 K=1,1000
0053 IF(L2INT.GE.L) GOTO 10
0054 L2INT=2*L2INT
0055 5 CONTINUE
0056 10 X=XSTART
0057 DO 20 JCHAN=1,NCHAN
0058 XSQ(JCHAN)=X**2
0059 X=X+XSTEP
0060 20 CONTINUE
0061 TSAMP=1./FSAMP
0062 T0=NSTART*TSAMP
0063 DO 30 JT0=1,L
0064 T0SQ(JT0)=T0**2
0065 T0=T0+TSAMP
0066 30 CONTINUE
0067

```

C

C SET UP AP CONSTANTS AND CDP SUBROUTINE

C OUTSIDE OF LOOP

C

```

0068 CALL SETAP(L2INT,M)
0069 CALL CDP(T0SQ,VINSQ,0)
0070 IVEL=1
0071 NBLKR=L*NCHAN/128+6
0072 NBLKW=L/128+1

```

C

C START OF MAIN PROCESSING LOOP

C

```

0073 DO 999 IFNUM=1,NFILES
0074 IFIL=IFNUM
0075 IF(INFLG.NE.0)GOTO 100

```

C

C COME HERE IF INPUT FROM TAPE

C

```

0077 IF(IENTER(IDCH,FSPECR,NBLKR).LT.0)STOP'ENTER ERR'

```

C

C EOT CHECK

C

```

0079 ITRY=1
0080 200 IF(ITRY.GT.3)GOTO 210

```

```
0082      IF(IEOTR.GE.0)GOTO 105
0084      210 WRITE(7,1800)TPDRR,IFIL
0085      1800 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
0086      WRITE(7,1801)
0087      1801 FORMAT(' ENTER NEW READ DRIVE NUMBER:',S)
0088      READ(5,1802)TPDRR
0089      1802 FORMAT(I1)
0090      IEOTR=0
0091      IF(TPDRR.GT.2)STOP ' EOT READ TERMINATE '
C
C DO A TAPE READ
C
0093      105 CALL TAPRED(-1,TPDRR,STATUS,ITLEN,IFLEN,IFIL,IEOTR)
C
C FATAL ERROR DETECTION
C
0094      IF(STATUS.LT.0)WRITE(2,1500)IFNUM
0096      1500 FORMAT(' WARNING FILE ',I3,' RETRIES FAILED')
0097      IF(IEOTR.LT.0)GOTO 106
C
C WIND OVER EOF MARK
C
0099      CALL TAPRED(0,TPDRR,STATUS, , ,IFIL,IEOTR)
0100      106 CALL IWAIT(IDCH)
0101      IERR=0
0102      ITRY=ITRY+1
0103      IF(IREADW(1,IERR,0,IDCH).LT.0)STOP ' ERR READ ERR '
0105      IF(IERR.EQ."177777")GOTO 200
0107      CALL CLOSEC(IDCH)
C
C OPEN UP FILES FOR READING IN
C
0108      100 IF(INFLG.NE.0)FSPECR=FNAMR(IFNUM)
0110      IF(LOOKUP(IDCH,FSPECR).LT.0)STOP 'LOOKUP ERR'
C
C OPEN UP OUTPUT FILES
C
0112      IF(OUTFLG.NE.0)FSPECW=FNAMR(IFNUM)
0114      IF(IENTER(IDCH1,FSPECW,NBLKW).LT.0)STOP 'ENTER ERR'
C
C HEADER BLOCK MANAGEMENT
C
0116      IF(IREADW(256,IHBLK,0,IDCH).LT.0)STOP 'HBLK ERR'
C
C UPDATE HEADER
C
0118      IBFREE=IHBLK(24)
0119      IHBLK(7)=1
0120      LBLK(19)=3
0121      IHBLK(129+IBFREE)=2
0122      IBFREE=IBFREE+1
0123      IHBLK(129+IBFREE)=NCHAN
0124      IBFREE=IBFREE+1
0125      IHBLK(129+IBFREE)=NVEL
```

```
0126      IBFREE=IBFREE+1
0127      IHBLK(129+IBFREE)=N2LYR
0128      IBFREE=IBFREE+1
0129      IHBLK(129+IBFREE)=M
0130      IBFREE=IBFREE+1
0131      IHBLK(129+IBFREE)=INTSW
0132      IBFREE=IBFREE+1
0133      IHBLK(24)=IBFREE
C
C WRITE OUT BLOCK
C
0134      IF(IWRITW(256,IHBLK,0,IDCH1).LT.0)STOP'HBLKW ERR'
C
C SEE IF NEW VELOCITY ANALYSIS REQUIRED
C
0135      IF(IVELAN(IVEL).NE.IFNUM)GOTO 108
0136      IVEL=IVEL+1
0137      NLYR=N2LYR
0138      DO 110 I=1,NLYR
0139      T0LYR(I)=T02LYR(I)
0140      VLYR(I)=V2LYR(I)
0141      110 CONTINUE
C
C SET UP NEW VINSQ TABLE
C
0142      CALL INVSQ(FSAMP,NSTART,NLYR,L,VINSQ,T0LYR,VLYR)
0143      IF(NVEL.LT.IVEL)INTSW=0
0144      IF(NVEL.LT.IVEL)GOTO 120
C
C READ IN NEXT ANALYSIS AND SET UP INTERPOLATION
C
0145      READ(1,1000)N2LYR
0146      READ(1,1200)(T02LYR(I),V2LYR(I),I=1,N2LYR)
0147      IF(INTSW.EQ.0)GOTO 120
0148      INT=IVELAN(IVEL)-IVELAN(IVEL-1)
0149      IF(INT.LE.1)GOTO 120
0150      RINT=FLOAT(INT)
0151      DO 130 I=1,NLYR
0152      T0INT(I)=(T02LYR(I)-T0LYR(I))/RINT
0153      VINT(I)=(V2LYR(I)-VLYR(I))/RINT
0154      130 CONTINUE
0155      GOTO 120
C
C COME HERE WHEN NOT A NEW ANALYSIS
C
0156      108 IF(INTSW.EQ.0)GOTO 120
C
C ADD ON INTERPOLATING PARAMETERS
C
0157      DO 140 I=1,NLYR
0158      T0LYR(I)=T0LYR(I)+T0INT(I)
0159      VLYR(I)=VLYR(I)+VINT(I)
0160      140 CONTINUE
C
```

```
C SET UP VINSQ TABLE AND THEN DO STACK
C
0168 CALL INVSQ(FSAMP,NSTART,NLYR,L,VINSQ,T0LYR,VLYR)
0169 120 CALL CDP(T0SQ,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,TPDRW,STATUS,ITLEN,IFLEN,IFIL,IEOTW)
0178 CALL CLOSEC(IDCH1)
0179 IF(IEOTW.GE.0)GOTO 150
0181 WRITE(7,1600)TPDRW,IFIL
0182 1500 FORMAT(' EOT ON WRITE DRIVE:',I2,' FILE NO:',I4)
0183 WRITE(7,1601)
0184 1601 FORMAT(' ENTER NUMBER OF NEW DRIVE:',S)
0185 READ(5,1802)TPDRW
0186 IEOTW=0
0187 IF(TPDRW.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)
0198 STOP ' NORMAL TERMINATION'
0199 END
```

```

C=====
0001 SUBROUTINE INVSQ(FSAMP,NSTART,NLYR,L,VINSQ,T0LYR,VLJR)
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.
0002 VIRTUAL VINSQ(2048),T0LYR(20),VLJR(20)
0003 N1=1
0004 N2=IFIX(FSAMP*T0LYR(1))-NSTART
0005 VINSQ1=1.0/VLJR(1)**2
0006 DO 10 I=N1,N2
0007 VINSQ(I)=VINSQ1
0008 10 CONTINUE
0009 IF(NLYR.EQ.1) GOTO 40
0010 DO 30 J=2,NLYR
0011 N1=N2+1
0012 N2=IFIX(FSAMP*T0LYR(J))-NSTART
0013 DELV=(VLJR(J)-VLJR(J-1))/(N2-N1+2)
0014 V=VLJR(J-1)
0015 DO 20 I=N1,N2
0016 VINSQ(I)=1.0/V**2
0017 V=V+DELV
0018 20 CONTINUE
0019 30 CONTINUE
0020 40 N1=N2+1
0021 N2=L
0022 VINSQN=1.0/VLJR(NLYR)**2
0023 DO 50 I=N1,N2
0024 VINSQ(I)=VINSQN
0025 50 CONTINUE
0026 RETURN
0027 END
0028

```

```

C=====
0001 BLOCK DATA
C-----
0002 IMPLICIT INTEGER*2(I-N)
0003 COMMON /KONST/ K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
0004 1IA,IB,IC,ID,ITOP
0005 DATA K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,IA,IB,IC,ID,ITOP
0006 1/-1,0.1,2,3,4,5,6,7,8,9,10,0,2048,4096,6144,8191/
0007 END

```

```
=====
0001  SUBROUTINE SETAP(L2INT,M)
-----
C
C SETS UP COMPLEX EXPONENTIAL TABLE IN D+1
C AND PUTS 1.0 IN ID
0002  COMMON /KONST/ K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
      1IA,IB,IC,ID,ITOP
C INITIALISE AP
0003  CALL APINIT
C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS.
0004  ID1=ID+1
0005  ID2=ID+2
0006  L2I21=L2INT/2-1
C TRANSFER 1.0/(L2INT*M) TO K0 AND MULTIPLY BY 2PI FROM TM.
0007  CONST=1.0/FLOAT(L2INT*M)
0008  CALL APWR
0009  CALL APPUT(CONST,K0,K1,K2)
0010  CALL APWD
C FORM VECTOR RAMP AND TAKE SIN AND COS OF IT
0011  CALL VTSMUL(K0,K1,2317,K0,K1,K1)
0012  CALL VRAMP(K0,K0,IB,K1,L2I21)
0013  CALL VCOS(IB,K1,ID1,K2,L2I21)
0014  CALL VSIN(IB,K1,ID2,K2,L2I21)
C PUT 1.0 IN ID
0015  CALL VCLR(ID,K1,K1)
0016  CALL VTSADD(ID,K1,2049,ID,K1,K1)
0017  RETURN
0018  END
```

```

C =====
0001   SUBROUTINE CDP(T0SQ,VINSQ,ICODE)
C
C MAIN STACKING SUBROUTINE
C =====
0002   VIRTUAL SEISM(32767),T0SQ(2048),VINSQ(2048)
0003   DIMENSION INDEX(2048),MUTE(24),STACK(2048),
0004   IDUM(10),CONST(10),FSTAP(20),XSQ(24)
0005   COMMON/STK/L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,INDEN,
%INDMUT,IDCH,IDCH1,XSQ
COMMON /KONST/ K1M,K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,
1IA,IB,IC,ID,ITOP
C
C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS
C
0006   IF(ICODE.GT.0)GOTO 20
0007   IB1=IB+1
0008   IC1=IC+1
0009   IC2=IC+2
0010   ID1=ID+1
0011   L1=L+1
0012   L2I2I=L2INT/2-1
0013   CONST(2)=M*FSAMP
0014   CONST(3)=0.5
0015   CONST(4)=-M*NSTART
0016   CONST(5)=0.0
0017   CONST(6)=L*M
0018   CONST(7)=L+1
0019   CONST(8)=1.0
0020   CONST(9)=1.0/M
0021   CONST(10)=(L+1)*M-1
0022   NBLKTR=L/128
0023   FSTD=ADGET(SEISM(1))
0024   FT0SQ=APGAD(T0SQ(1))
0025   FVINSQ=APGAD(VINSQ(1))
0026   ISTART=1
0027   DO 50 K=1,M
0028   FSTAP(K)=APGAD(SEISM(ISTART))
0029   ISTART=ISTART+L1
0030   50 CONTINUE
0031   RETURN
C SET SEISM(L+1) TO 0.0 TO COPE WITH TIME OVERFLOW AND CLEAR A IN AP
0032   20 JBLK=1
0033   IBLK=1
0034   SEISM(L1)=0.0
0035   CALL VCLR(IA,K1,L)
C ITERATE THROUGH CHANNELS
0036   DO 500 JCHAN=1,NCHAN
C READ IN TRACE SEISM FROM UNIT 2
0037   IF(IREADA(IDCH,2*L,JBLK,FSTD).LT.0)STOP'READA ERROR'
0038   JBLK=JBLK+NBLKTR
0039   CALL IWAIT(IDCH)
C COMPUTE INDEX ARRAY
0040   CONST(1)=XSQ(JCHAN)
0041
0042

```

```
0043      CALL APWR
0044      C TRANSFER (K1,K10) TO HOST DUMMY ARRAY DUM AND REPLACE BY CONST
0045      CALL APGET(DUM,K1,K10,K2)
0046      CALL APPUT(CONST,K1,K10,K2)
0047      C TRANSFER T0SQ TO B AND VINSQ TO IC1
0048      CALL APPUTA(IB,L,K2,FT0SQ)
0049      CALL APPUTA(IC1,L,K2,FVINSQ)
0050      CALL APWD
0051      C FORM XSQ(JCHAN)*VINSQ IN C1
0052      CALL VSMUL(IC1,K1,K1,IC1,K1,L)
0053      C FORM SQRT(T0SQ+XSQ(JCHAN)*VINSQ) IN B
0054      CALL VADD(IB,K1,IC1,K1,IB,K1,L)
0055      CALL VSQRT(IB,K1,IB,K1,L)
0056      C FORM IFIX(FSAMP*M*SQRT(T0SQ+XSQ(JCHAN)*VINSQ)+0.5)-NSTART*M IN B
0057      CALL VSMSA(IB,K1,K2,K3,IB,K1,L)
0058      CALL VINT(IB,K1,IB,K1,L)
0059      CALL VSADD(IB,K1,K4,IB,K1,L)
0060      C CLIP B BETWEEN 0.0 AND L*M
0061      CALL VCLIP(IB,K1,K5,K6,IB,K1,L)
0062      C MULTIPLY B BY L+1 ,ADD 1.0 AND PLACE RESULT IN C1
0063      CALL VSMSA(IB,K1,K7,K8,IC1,K1,L)
0064      C MULTIPLY B BY 1.0/M AND TAKE INTEGER PART
0065      CALL VSMUL(IB,K1,K9,IB,K1,L)
0066      CALL VINT(IB,K1,IB,K1,L)
0067      C MULTIPLY B BY (L+1)*M-1
0068      CALL VSMUL(IB,K1,K10,IB,K1,L)
0069      C SUBTRACT B FROM C1 AND PUT RESULT IN B
0070      CALL VSUB(IB,K1,IC1,K1,IB,K1,L)
0071      C FIX B AND TRANSFER BACK AS HOST ARRAY INDEX
0072      CALL VFIX(IB,K1,IB,K1,L)
0073      CALL APWR
0074      CALL APGET(INDEX,IB,L,K1)
0075      C TRANSFER DUM BACK TO K1
0076      CALL APPUT(DUM,K1,K10,K2)
0077      CALL APWD
0078      C
0079      C HAVING COMPUTED INDEX ARRAY NOW START INTERPOLATION OF TRACE
0080      C
0081      C CLEAR B IN AP
0082      CALL VCLR(IB,K1,L2INT)
0083      C TRANSFER TRACE TO B
0084      CALL APWR
0085      CALL APPUTA(IB,L,K2,FSTAP(1))
0086      CALL APWD
0087      C FIND MEAN VALUE OF TRACE AND PLACE IN AP(ITOP)
0088      CALL MEANV(IB,K1,ITOP,L)
0089      C SUBTRACT MEAN FROM TRACE
0090      CALL VNEG(ITOP,K1,ITOP,K1,K1)
0091      CALL VSADD(IB,K1,ITOP,IB,K1,L)
0092      C IF INDEN IS NEGATIVE,NO SCALING OF TRACE
0093      C IF INDEN IS ZERO SCALE TO UNIT R.M.S VALUE
0094      C IF INDEN IS POSITIVE,USE INVERSE ENERGY SCALING (DIVERSITY STACK)
0095      IF(INDEN.LT.0) GOTO 100
0096      CALL RMSQV(IB,K1,ITOP,L)
```

```
0076      IF(INDEN.GT.0) CALL VSQ(ITOP,K1,ITOP,K1,K1)
0077      C USE 1.0 RESIDING IN ID (FROM SETAP)
0078      CALL VDIV(ITOP,K1,ID,K1,ITOP,K1,K1)
0079      CALL VSMUL(IB,K1,ITOP,IB,K1,L)
0080      C IF INDMUT IS NEGATIVE USE NO MUTING
0081      100  IF(INDMUT.LT.0) GOTO 200
0082      CALL VCLR(IB,K1,MUTE(ICHAN))
0083      C TRANSFER MODIFIED TRACE BACK TO SEISM
0084      200  CALL APWR
0085      CALL APGETA(IB,L,K2,FSTAP(1))
0086      CALL APWD
0087      IF(M.EQ.1) GOTO 350
0088      C TAKE TRANSFORM OF TRACE AND PUT IN C
0089      CALL RFFT(B,IC,L2INT,K1)
0090      CALL RFFTSC(IC,L2INT,K0,K1)
0091      CALL VCLR(IC,K1,K2)
0092      C ITERATE FROM 2 TO M
0093      DO 300 K=2,M
0094      C MULTIPLY TRANSFORM BY COMPLEX EXPONENTIAL ARRAY
0095      CALL CVMUL(IC2,K2,ID1,K2,IC2,K2,L2I21,K1)
0096      C MOVE C TO B AND TAKE IN PLACE IFFT
0097      CALL VMOV(IC,K1,IB,K1,L2INT)
0098      CALL RFFT(IB,L2INT,K1M)
0099      C TRANSFER SHIFTED TRACE BACK TO SEISM
0100      CALL APWR
0101      CALL APGETA(IB,L,K2,FSTAP(K))
0102      CALL APWD
0103      300  CONTINUE
0104      C PICK REQUIRED ELEMENTS OUT OF SEISM AND PLACE IN STACK
0105      350  DO 400 I=1,L
0106      STACK(I)=SEISM(INDEX(I))
0107      400  CONTINUE
0108      C TRANSFER STACK INTO B1 AND ADD TO STACK IN A
0109      CALL APWR
0110      CALL APPUT(STACK,IB1,L,K2)
0111      CALL APWD
0112      CALL VADD(IA,K1,IB1,K1,IA,K1,L)
0113      500  CONTINUE
0114      C SCALE STACK BY 1.0/NCHAN
0115      FNCINV=1.0/NCHAN
0116      CALL APWR
0117      CALL APPUT(FNCINV,ITOP,K1,K2)
0118      CALL APWD
0119      CALL VSMUL(IA,K1,ITOP,IA,K1,L)
0120      C TRANSFER SCALED STACK BACK TO STACK
0121      CALL APWR
0122      CALL APGET(STACK,IA,L,K2)
0123      CALL APWD
0124      IF(IWRITE(2*L,STACK,IBLK,IDCH1).LT.0)STOP 'WRITE ERROR 2'
0125      RETURN
0126      END
```

```

0001      SUBROUTINE TAPRED(ICOM,IDRV,ISTAT,ITLEN,ILEN,IFNUM,IEOT)
C
C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ,0 IS A WIND,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
C
0002      INTEGER*2 MASK(8),ESTATI
0003      LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
          %IFLEN,ESTAT,ERRS(8)
0004      DATA MASK/'1','2','4','10','20','40','100','200'/
0005      DATA SDSCOM/'0','1','2','3','4','5','6','7'/
0006      DATA ERRS/'377','377','377','377','377','377','377','377'/
0007      ITRY=0
0008      IF(ICOM) 10,30,20
C
C SECTION CONTROLLING A READ
C
C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
0009      10 ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
0010      COM(1)=SDSCOM(4)
0011      COM(2)=1
0012      COM(3)=IDRV
0013      COM(4)=-1
0014      CALL SDS10(COM,ISTAT,ITLEN,ILEN)
0015      IF(ISTAT.EQ.0)RETURN
C
C ERROR DETECTED ON READ
C
0017      ISTATI=ISTAT
0018      GOTO 40
C
C IF SHORT RECORD FOUND REREAD TAPE
C
0019      50 ITMP=ISTATI.AND.MASK(6)
0020      IF(ITMP.NE.0)GOTO 10
0022      ITMP=ISTATI.AND.MASK(2)
0023      IF(ITMP.EQ.0)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
0025      WRITE(2,2010)IFNUM
0026      2010 FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
0027      IF(ITRY.GE.2)GOTO 130
0029      ECOM(1)=SDSCOM(6)
0030      ECOM(2)=1

```

```
0031      ECOM(3)=IDRV
0032      ECOM(4)=0
0033      CALL SDS10(ECOM,ESTAT, , )
0034      GOTO 10
C
C WRITE SECTION
C
0035      20 ITRY=ITRV+1
0036          IF(ITRY.GT.3)GOTO 130
0038      COM(1)=SDSCOM(7)
0039      IFLEN=(ILEN+3)/4
0040      COM(2)=IFLEN
0041      COM(3)=IDRV
0042      COM(4)=1
0043      CALL SDS10(COM,ISTAT, , )
0044      IF(ISTAT.EQ.0)RETURN
C
C WRITE ERROR DETECTED
C
0046      ISTAT=ISTAT
0047      GOTO 40
0048      70 ITMP=ISTATI.AND.MASK(6)
0049      ITMPI=ISTATI.AND.MASK(2)
0050      IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
C
C REPORT AND RETRY
C
0052      WRITE(2,2020)IFNUM
0053      2020 FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
0054      ECOM(1)=SDSCOM(6)
0055      ECOM(2)=2
0056      ECOM(3)=IDRV
0057      ECOM(4)=0
0058      CALL SDS10(ECOM,ESTAT, , )
0059      NBUF=8
0060      IPAD=(IFLEN*2048)-NBUF
0061      CALL TWRIT(ERRS,NBUF,ESTAT,IPAD,IFLEN,IDRV)
0062      GOTO 20
C
C WIND FOWARD ONE FILE
C
0063      30 COM(1)=SDSCOM(5)
0064      COM(2)=1
0065      COM(3)=IDRV
0066      COM(4)=0
0067      CALL SDS10(COM,ISTAT, , )
C
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
C
0068      ISTAT=ISTAT.AND..NOT.MASK(6)
0069      IF(ISTAT.EQ.2)RETURN
0071      ISTATI=ISTAT
0072      IF(ISTAT.NE.0)GOTO 40
C
```

```

C IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
C
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
C
0074      ECOM(1)=SDSCOM(6)
0075      ECOM(2)=1
0076      ECOM(3)=IDRV
0077      ECOM(4)=0
0078      CALL SDS10(ECOM,ESTAT, , )
0079      35 RETURN
C
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS:= TAPE BUSY,TAPE OFFLINE
C BOT,EOT
C
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
C
0080      40 WRITE(2,1010)ISTATI,IFNUM
0081      1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
0082      ISTATI=ISTATI.AND..NOT.MASK(4)
0083      ITMP=ISTATI.AND.MASK(5)
0084      IF(ITMP.EQ.0)GOTO 80
0086      90 ECOM(1)=SDSCOM(1)
0087      ECOM(2)=0
0088      ECOM(3)=IDRV
0089      ECOM(4)=0
0090      CALL SDS10(ECOM,ESTAT, , )
C
C HAVING EXAMINED STATUS IF TAPE STILL
C BUSY, LOOP AGAIN,IF NOT TRY COMMAND AGAIN
C
0091      ESTATI=ESTAT
0092      ITMP=ESTATI.AND.MASK(5)
0093      IF(ITMP.NE.0)GOTO 90
0095      IF(ICOM) 10,30,20
C
C TAPE OFFLINE
C
0096      80 ITMP=ISTATI.AND.MASK(1)
0097      IF(ITMP.EQ.0)GOTO 100
0099      TYPE 1001,IDRV
0100      1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
0101      110 ECOM(1)=SDSCOM(1)
0102      ECOM(2)=0
0103      ECOM(3)=IDRV
0104      ECOM(4)=0
0105      CALL SDS10(ECOM,ESTAT, , )
0106      ESTATI=ESTAT
0107      ITMP=ESTATI.AND.MASK(1)
0108      IF(ITMP.NE.0)GOTO 110
0110      IF(ICOM) 10,30,20

```

```

C
C EOT
C
0111      100 ITMP=ISTATI.AND.MASK(3)
0112      IF(ITMP.EQ.0)GOTO 120
0114      TYPE 1002,IDRV
0115      1002 FORMAT(' EOT ON DRIVE ',I1)
0116      IEOT=-1
0117      RETURN
0118      120 IF(ICOM) 50,35,70
C
C ERROR EXIT RETURN
C
0119      130 ISTATI=-1
0120      RETURN
0121      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 FilteringREAD(1,1100)FL,FUREAD(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 FilteringREAD(1,1100)FLR,FUR

FLR.....Lower cutoff frequency Hz

FUR.....Upper cutoff frequency Hz

7....Prediction Error DeconvolutionREAD(1,1000)NPFILT,NLAG,IPNORMREAD(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
C POST STACK UTILITY PROGRAM
C THIS INVOLVES THE FOLLOWING
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
C
0001 REAL*8 FSPECR,FSPECW,FNAMR,FNAMO
0002 VIRTUAL FNAMR(100),FNAMO(100),EXPT(2048),TEXPT(2048),
    %TVSQ(2048),TAPER(512),BPASS(2049),BRJCT(2049),MUTE(2000)
0003 REAL*4 FBUF(3),TOLYR(20),VLYR(20),CONST(3),SEISM(2048)
0004 INTEGER*2 IORD(8),UTLFLG(8),IFILED(100),DBUF(4352),
    %IHBLK(256),OUTFLG,MNPOS(30),MSAMP(30),MINC(30)
0005 LOGICAL*1 TPDRR,TPDRW,ISTAT
0006 EQUIVALENCE (IHBLK(1),DBUF(1)),(SEISM(1),DBUF(257))
0007 DATA DEV/3RRK /
C
C SET UP VIRTUAL ADDRESSES
C
0008 IEOTR=0
0009 IEOTW=0
0010 ATAP=APGAD(TAPER(1))
0011 ATBP=APGAD(BPASS(1))
0012 ATBR=APGAD(BRJCT(1))
0013 ATVSQ=APGAD(TVSQ(1))
0014 ATEXPT=APGAD(TEXPT(1))
0015 AEXPT=APGAD(EXPT(1))
C
C SET UP I/O CHANNELS AND READ IN CONTROL DATA
C
0016 IF(ICDFN(25).NE.0)STOP 'CHANNEL OVERFLOW'
0017 CALL ASSIGN(1,'DK2:MPPOST.DAT',14)
0018 CALL ASSIGN(2,'DK2:MPPOST.LOG',14)
0019 IDCH=20
0020 IDCH1=21
0021 READ(1,1000)NFILES,NSAMP,NSTART,INFLG,OUTFLG
0022 1000 FORMAT(12I5)
0023 READ(1,1000)TPDRR,TPDRW
0024 READ(1,1100)FSAMP
0025 1100 FORMAT(2F10.0)
C
C READ IN FILE SPECS FOR INPUT
C
0027 1200 FORMAT(3A4)
0028 IF(INFLG.EQ.0)GOTO 20
0029 DO 30 I=1,NFILES
0030 READ(1,1200)FBUF

```

```

0030      CALL IRAD50(12,FBUF,FSPECR)
0031      FNAMR(I)=FSPECR
0032      30 CONTINUE
C
C READ IN FILE SPECS FOR OUTPUT
C
0033      20 IF(OUTFLG.EQ.0)GOTO 50
0034      DO 60 I=1,NFILES
0035      READ(1,1200)FBUF
0036      CALL IRAD50(12,FBUF,FSPECW)
0037      FNAMO(I)=FSPECW
0038      60 CONTINUE
C
C READ IN JOB SPECIFIC DATA AND SET UP
C THE FILTERS TO BE USED
C
0039      50 NUTIL=10
0040      ITX=0
0041      CONST(1)=FLOAT(NSTART)
0042      CONST(2)=0.2
0043      CONST(3)=1.0/(1000.0*FSAMP)
0044      CALL APINIT
0045      NSAMP2=2
0046      51 IF(NSAMP2.GE.NSAMP)GOTO 52
0047      NSAMP2=NSAMP2*2
0048      GOTO 51
0049      52 CONTINUE
C
C READ IN FLAGS FOR PROCESSES AND EXECUTION ORDER
C
0050      READ(1,1000)NPROC
0051      READ(1,1000)(UTLFLG(I),I=1,NUTIL)
0052      READ(1,1000)(IORD(I),I=1,NPROC)
C
C TRACE EDIT DATA
C
0053      IF(UTLFLG(1).EQ.0)GOTO 70
0054      READ(1,1000)NFILED
0055      READ(1,1000)(IFILED(I),I=1,NFILED)
C
C EXP(0.2T) RAMP DATA
C
0056      70 IF(UTLFLG(2).EQ.0)GOTO 80
0057      READ(1,1000)IAPLX
0058      CALL TEXRMP(EXPT,0,ITX,NSAMP,CONST,AEXPT)
0059      ITX=1
C
C T*EXP(0.2T) RAMP DATA
C
0060      80 IF(UTLFLG(3).EQ.0)GOTO 90
0061      READ(1,1000)IAPLTX
0062      CALL TEXRMP(TEXPT,1,ITX,NSAMP,CONST,ATEXPT)
0063      ITX=1
C

```

C TV**2 RAMP

C

```
0071 90 IF(UTLFLG(4).EQ.0)GOTO 100
0073 READ(1,1000)IAPLTV,NLYR
0074 READ(1,1100)(T0LYR(I),VLYR(I),I=1,NLYR)
0075 CALL TVRMP(TVSQ,T0LYR,VLYR,NLYR,ITX,NSAMP,NSTART,CONST,
%FSAMP,ATVSQ)
```

C

C MUTE DATA

C

```
0077 100 IF(UTLFLG(5).EQ.0)GOTO 110
0078 READ(1,1000)NTAP,NMPTS
0079 READ(1,1000)(MNPOS(I),MSAMP(I),I=1,NMPTS)
0080 CALL COTAP(TAPER,NTAP,ATAP)
0081 DO 101 J=2,NMPTS
0082 101 MINC(J)=(MSAMP(J)-MSAMP(J-1))/(MNPOS(J)-MNPOS(J-1))
0083 MUTE(I)=MSAMP(I)
0084 IPOS=2
0085 DO 102 J=2,NFILES
0086 IF(J.LT.MNPOS(IPOS))GOTO 103
0087 MUTE(J)=MSAMP(IPOS)
0088 IPOS=IPOS+1
0089 GOTO 102
0090 103 MUTE(J)=MUTE(J-1)+MINC(IPOS)
0091 102 CONTINUE
```

C

C DECON INPUT

C

```
0092 110 IF(UTLFLG(6).EQ.0)GOTO 120
0093 READ(1,1000)NFILT,ICONT,ISPIKE,INORM
0094 READ(1,1100)WHITE
```

C

C SANDPASS FILTER

C

```
0097 120 IF(UTLFLG(7).EQ.0)GOTO 130
0098 READ(1,1100)FL,FU
0099 READ(1,1100)FTPR1,FTPR2
0100 DFI=FLOAT(NSAMP2/2+1)/(FSAMP*500.)
0101 FL1=FL-FTPR1
0102 FU4=FU+FTPR2
0103 CALL BANDPS(ATBP,BPASS,FL1,FL,FU,FU4,DFI,NSAMP2)
0104 NTRANF=2*NSAMP2
0105 NSFILT=NSAMP2+1
0106 NBEXP=(2*NSAMP2)+2-NSAMP
```

C

C SANDREJECT FILTER

C

```
0107 130 IF(UTLFLG(8).EQ.0)GOTO 140
0108 READ(1,1100)FLR,FUR
0109 DFI=FLOAT(NSAMP2/2+1)/(FSAMP*500.)
0110 NTRANF=2*NSAMP2
0111 NRFILT=NSAMP2+1
0112 NBEXP=(2*NSAMP2)+2-NSAMP
0113 CALL BANDRJ(ATBR,BRJCT,FLR,FUR,DFI,NSAMP2)
```

```

C
C PREDICTIVE DECONVOLUTION
C
0116 140 IF(UTLFLG(9).EQ.0)GOTO 150
0118 READ(1,1000)NPFILT,NLAG,IPNORM
0119 READ(1,1100)PRWHIT
C
C TRACE NORMALISATION
C
0120 150 IF(UTLFLG(10).EQ.0)GOTO 160
0122 READ(1,1000)NRMFLG
C
C BLOCKING PARAMETERS
C
0123 150 CONTINUE
0124 IFED=1
0125 NBLKTR=NSAMP/128
0126 NBLKW=NBLKTR+1
0127 NBYTR=NBLKW*512
0128 NWDR=NBYTR/2
C
C START OF LOOP ON DIFFERENT FILES
C
0129 DO 200 IFNUM=1,NFILES
0130 IFIL=IFNUM
0131 IF(INFLG.NE.0)GOTO 210
C
C TAPE INPUT HANDLING
C
C EOT CHECK
C
0133 ITRY=1
0134 211 IF(ITRY.GT.3)GOTO 212
0136 IF(IEOTR.GE.0)GOTO 220
0138 212 WRITE(7,1400)TPDRR,IFIL
0139 1400 FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I4)
0140 WRITE(7,1401)
0141 1401 FORMAT(' ENTER NEW DRIVE NO:',S)
0142 READ(5,1402)TPDRR
0143 1402 FORMAT(I1)
0144 IEOTR=0
0145 IF(TPDRR.GT.2)STOP ' EOT TERMINATION'
C
C READ FROM TAPE TO MEMORY
C
0147 220 CALL TAPSUB(-1,TPDRR,ISTAT,IFLEN,IFIL,DBUF,NBYTR,IEOTR)
0148 IF(ISTAT.LT.0)WRITE(2,1500)IFIL
0150 1500 FORMAT(' RETRIES ON READ FAILED ON FILE',I5)
0151 IF(IEOTR.LT.0)GOTO 230
C
C WIND OVER EOF MARK
C
0153 CALL TAPSUB(0,TPDRR,ISTAT, ,IFIL, ,IEOTR)

```

```

0154      230 ITRY=ITRY+1
0155      IF(IHBLK(1).EQ."177777")GOTO 211
0157      GOTO 280
C
C OPEN UP READING FILES
C
0158      210 FSPECR=FNAMR(IFNUM)
0159      CALL CLOSEC(IDCH)
0160      IF(LOOKUP(IDCH,FSPECR).LT.0)STOP'LOOKUP ERR'
0162      IF(IREADW(NWDR,DBUF,0,IDCH).LT.0)STOP'READW ERR'
0164      230 CONTINUE
C
C HEADER BLOCK MANAGEMENT
C
0165      IBFREE=IHBLK(24)
0166      IHBLK(129+IBFREE)=3
0167      IBFREE=IBFREE+1
0168      IHBLK(129+IBFREE)=NPROC
0169      IBFREE=IBFREE+1
0170      DO 215 J=1,NPROC
0171      IHBLK(129+IBFREE)=IORD(J)
0172      IBFREE=IBFREE+1
0173      215 CONTINUE
0174      IHBLK(24)=IBFREE
C
C OPEN UP OUTPUT FILES
C
0175      IF(OUTFLG.EQ.0)GOTO 300
0177      FSPECW=FNAMO(IFNUM)
0178      IF(IENTER(IDCH1,FSPECW,NBLKW).LT.0)STOP'ENTER ERR2'
0180      IF(IWRITW(256,IHBLK,0,IDCH1).LT.0)STOP'WRITW ERR'
0182      JBLK=1
0183      300 CALL APPUT(SEISM,0,NSAMP,2)
0194      CALL APWD
0185      DO 400 IPCNUM=1,NPROC
0186      GOTO(410,430,440,450,460,470,480,
          *490,500,510)IORD(IPCNUM)
C
C PROCESS EDIT COMMANDS
C
0187      410 IF(IFED.GT.NFILED)GOTO 400
0189      IF(IFILED(IFED).NE.IFIL)GOTO 400
0191      IFED=IFED+1
0192      CALL VCLR(0,1,NSAMP)
0193      CALL APWR
0194      CALL APGET(SEISM,0,NSAMP,2)
0195      CALL APWD
0196      GOTO 310
C
C AMP RECOVERY FILTERS APPLICATION AND REMOVAL
C
C EXP(0.2T) FILTER
C

```

```

0197 430 IAPL=IAPLX
0198 CALL APPUTA(NSAMP,NSAMP,2,AEXPT)
0199 GOTO 455
C
C T*EXP(0.2T) FILTER
C
0200 440 IAPL=IAPLTX
0201 CALL APPUTA(NSAMP,NSAMP,2,ATEXPT)
0202 GOTO 455
C
C TV**2 FILTER
C
0203 450 IAPL=IAPLTV
0204 CALL APPUTA(NSAMP,NSAMP,2,ATVSQ)
C
C COMMON CODE
C
0205 455 CALL APWD
0206 IF(IAPL.EQ.0)CALL VMUL(0,1,NSAMP,1,0,1,NSAMP)
0208 IF(IAPL.NE.0)CALL VDIV(NSAMP,1,0,1,0,1,NSAMP)
0210 CALL APWR
0211 GOTO 400
C
C MUTE APPLICATION
C
0212 460 CALL APPUTA(NSAMP,NTAP,2,ATAP)
0213 CALL APWD
0214 MNCLR=MUTE(IFIL)
0215 CALL VCLR(0,1,MNCLR)
0216 CALL VMUL(MNCLR,1,NSAMP,1,MNCLR,1,NTAP)
0217 CALL APWR
0218 GOTO 400
C
C SPIKE DECON
C
0219 470 CALL SPIKE(NSAMP,NSAMP2,NFILT,WHITE,INORM,ISPIKE)
0220 GOTO 400
C
C BANDPASS FILTER
C
0221 480 CALL APPUTA(4100,NBFILT,2,ATBP)
0222 CALL APWD
0223 CALL VCLR(NSAMP,1,NBEXP)
0224 CALL RFFT(0,NTRANF,1)
0225 CALL RFFTSC(0,NTRANF,3,1)
0226 CALL VMUL(0,2,4100,1,0,2,NBFILT)
0227 CALL VMUL(1,2,4100,1,1,2,NBFILT)
0228 CALL RFFTSC(0,1,NTRANF,-3,0)
0229 CALL RFFT(0,1,NTRANF,-1)
0230 CALL APWR
0231 GOTO 400
C
C BANDREJECT FILTER
C

```

```

0232 490 CALL APPUTA(4100,NRFILT,2,ATBR)
0233 CALL APWD
0234 CALL VCLR(NSAMP,1,NBEXP)
0235 CALL RFFT(0,NTRANF,1)
0236 CALL RFFTSC(0,NTRANF,3,1)
0237 CALL VMUL(0,2,4100,1,0,2,NRFILT)
0238 CALL VMUL(1,2,4100,1,1,2,NRFILT)
0239 CALL RFFTSC(0,NTRANF,-3,0)
0240 CALL RFFT(0,NTRANF,-1)
0241 CALL APWR
0242 GOTO 400
C
C PREDICT DECON
C
0243 500 CALL PRDICT(NSAMP,NSAMP2,NPFILT,PRWHIT,IPNORM,NLAG)
0244 GOTO 400
C
C NORMALISATION
C
0245 510 IF(NRMFLG.EQ.0)GOTO 515
0246 CALL MAXMGV(0,1,2050,NSAMP)
0247 GOTO 516
0248 515 CALL SVESQ(0,1,2050,NSAMP)
0249 CALL VSQRT(2050,1,2050,1,1)
0250 516 CALL VDIV(2050,0,0,1,0,1,NSAMP)
0251 CALL APWR
0252 CALL APWR
0253 400 CONTINUE
C
C END OF PROCESS LOOP
C
0254 CALL APWAIT
0255 CALL APGET(SEISM,0,NSAMP,2)
0256 CALL APWD
0257 310 IF(OUTFLG.EQ.0)GOTO 320
0258 IF(IWRITW(2*NSAMP,SEISM,JBLK,IDCH1).LT.0)STOP 'WRITW ERR2'
0259 CALL CLOSEC(IDCH1)
0260 GOTO 200
C
C OUTPUT TO TAPE
C
0261 320 IFLEN=NBLKW
0262 CALL TAPSUB(1,TPDRW,ISTAT,IFLEN,IFIL,DBUF,NBYTR,IEOTW)
C
C CHECK FOR ERRORS
C
0265 IF(IEOTW.GE.0)GOTO 250
0266 WRITE(7,1600)TPDRW,IFIL
0267 1600 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
0268 WRITE(7,1601)
0269 1601 FORMAT(' ENTER NEW WRITE DRIVE NO:',S)
0270 READ(5,1402)TPDRW
0271 IEOTW=0
0272 IF(TPDRW.GT.2)STOP 'EOT TERMINATE'
0273 250 IF(ISTAT.GE.0)GOTO 200

```

```

0274 WRITE(2,1700) IFIL
0275 1700 FORMAT(' FATAL ERROR ON WRITE FILE NO',I5)
0276 STOP ' WRITE ERROR'
0277 200 CONTINUE
0278 CALL CLOSEC(IDCH)
0279 CALL CLOSEC(IDCH1)
0280 STOP 'NORMAL TERMINATION'
0281 END

```

```
0001 SUBROUTINE TEXRMP(FILT,IFTYP,IFLG,NSAMP,CONST,AFILT,FS)
C
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
C   CONST(1)=NSART
C   CONST(2)=0.2
C   CONST(3)=1.0/(1000.0*FSAMP)
0002 VIRTUAL FILT(2048)
0003 DIMENSION CONST(3)
0004 IF(IFLG.NE.0)GOTO 10
C
C FORM THE T RAMP
C
0006 CALL APWAIT
0007 CALL APPUT(CONST,8189,3,2)
0008 CALL APWD
0009 CALL VCLR(0,1,NSAMP)
0010 CALL VRAMP(8189,8191,0,1,NSAMP)
C
C FORM EXP(0.2T)
C
0011 CALL VSMUL(0,1,8190,NSAMP,1,NSAMP)
0012 CALL VEXP(NSAMP,1,NSAMP,1,NSAMP)
0013 IF(IFTYP.EQ.0)GOTO 20
C
C FORM T*EXP(0.2T)
C
0015 10 CALL VMUL(NSAMP,1,0,1,NSAMP,1,NSAMP)
0016 20 CALL APWR
0017 CALL APGETA(NSAMP,NSAMP,2,AFILT)
0018 RETURN
0019 END
```

```
0001      SUBROUTINE TVRMP(FILT,T0LYR,VLJR,NLYR,IFLG,NSAMP,NSTART,  
      %CONST,FSAMP,AFILT)
```

```
      C  
      C THIS ROUTINE PRODUCES A TV**2 RAMP  
      C FROM VELOCITY INFO IN T0LYR,VLJR
```

```
      C
```

```
0002      VIRTUAL FILT(2048)  
0003      DIMENSION T0LYR(20),VLJR(20),CONST(3)
```

```
      C
```

```
      C CHECK IF T RAMP ALREADY FORMED
```

```
      C
```

```
0004      IF(IFLG.NE.0)GOTO 10  
0005      CALL APWAIT  
0006      CALL APPUT(CONST,8189,3,2)  
0007      CALL APWD  
0008      CALL VCLR(0,1,NSAMP)  
0009      CALL VRAMP(8189,8191,0,1,NSAMP)
```

```
      C
```

```
      C FORM VELOCITY RAMP IN FILT
```

```
      C
```

```
0011      10 N1=1  
0012         N2=IFIX(FSAMP*T0LYR(1))-NSTART  
0013         V1=VLJR(1)  
0014         DO 15 I=N1,N2  
0015           15 FILT(I)=V1  
0016             IF(NLYR.EQ.1)GOTO 40  
0017             DO 20 J=2,NLYR  
0018               N1=N2+1  
0019               N2=IFIX(FSAMP*T0LYR(J))-NSTART  
0020               DELV=(VLJR(J)-VLJR(J-1))/(N2-N1+2)  
0021               V=VLJR(J-1)  
0022               DO 30 I=N1,N2  
0023                 FILT(I)=V  
0024                 V=V+DELV  
0025           30 CONTINUE  
0026           20 CONTINUE  
0027           40 N1=N2+1  
0028             N2=NSAMP  
0029             VN=VLJR(NLYR)  
0030             DO 50 I=N1,N2  
0031               50 FILT(I)=VN
```

```
      C
```

```
      C PUT V RAMP IN AP AND FORM TV**2
```

```
      C
```

```
0033      CALL APWAIT  
0034      CALL APPUTA(NSAMP,NSAMP,2,AFILT)  
0035      CALL APWD  
0036      CALL VSQ(NSAMP,1,NSAMP,1,NSAMP)  
0037      CALL VMUL(0,1,NSAMP,1,NSAMP,1,NSAMP)  
0038      CALL APWR  
0039      CALL APGETA(NSAMP,NSAMP,2,AFILT)  
0040      CALL APWD  
0041      RETURN  
0042      END
```

```

0001      SUBROUTINE COTAP(TAPER,NTAP,ATAP)
C
C THIS ROUTINE PRODUCES A COSINE TAPER NTAP
C SAMPLES LONG
C
0002      VIRTUAL TAPER(512)
0003      CALL APWAIT
0004      CALL APPUT(1.0/FLOAT(NTAP),0,1,2)
0005      CALL VCLR(1,1,NTAP)
0006      CALL VTSADD(1,1,2306,1,1,1)
0007      CALL VTSMUL(0,1,2306,0,1,1)
0008      CALL VRAMP(1,0,0,1,NTAP)
0009      CALL VCOS(0,1,0,1,NTAP)
0010      CALL VTSADD(0,1,2049,0,1,NTAP)
0011      CALL VTSMUL(0,1,2327,0,1,NTAP)
0012      CALL APWR
0013      CALL APGETA(0,NTAP,2,ATAP)
0014      CALL APWD
0015      RETURN
0016      END
    
```

```
0001      SUBROUTINE BANDPS(ATBP,BPASS,F1,F2,F3,F4,DFI,NSAMP)
C
C SUBROUTINE WHICH CREATES A BANDPASS FILTER
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
C
0002      VIRTUAL BPASS(2049)
C
C SET UP CONSTANTS
C
0003      N1=2*F1*DFI
0004      N2=2*F2*DFI
0005      N3=2*F3*DFI
0006      N4=2*F4*DFI
0007      NFILT=NSAMP+1
0008      NTP1=N2-N1
0009      NTP2=N4-N3
0010      NOK=N3-N2
0011      CALL APWAIT
C
C SET UP THE FILTER IN THE AP
C
0012      DO 10 I=1,2
0013      NTAP=NTP1
0014      IF(I.EQ.2)NTAP=NTP2
0015      RNTAP=1.0/FLOAT(NTAP)
0016      CALL APPUT(RNTAP,0,1,2)
0017      CALL VCLR(1,1,NTAP)
0018      CALL VTSADD(1,1,2306,1,1,1)
0019      CALL VTSMUL(0,1,2306,0,1,1)
0020      CALL VRAMP(1,0,0,1,NTAP)
0021      CALL VCOS(0,1,0,1,NTAP)
0022      CALL VTSADD(0,1,2049,0,1,NTAP)
0023      CALL VTSMUL(0,1,2327,0,1,NTAP)
0024      IF(I.EQ.1)CALL VMOV(0,1,2050,1,NTAP)
0025      IF(I.EQ.2)CALL VMOV(0,1,4100,1,NTAP)
0026      10 CONTINUE
C
C NOW HAVE TAPERS FORM REST OF FILTER
C
0030      CALL VCLR(0,1,NFILT)
0031      CALL VADD(N1,1,2050,1,N1,1,NTP1)
0032      CALL VADD(N4,-1,4100,1,N4,-1,NTP2)
0033      CALL VTSADD(N2,1,2049,N2,1,NOK)
0034      CALL APWR
0035      CALL APGETA(0,NFILT,2,ATBP)
0036      CALL APWD
0037      RETURN
0038      END
```

```
0001      SUBROUTINE BANDRJ(ATBR,BRJCT,F1,F2,DFI,NSAMP)
C
C SUBROUTINE TO CREATE A BANDREJECT FILTER
C
C F1=LOWER CUTOFF POSITION
C F2=UPPER CUTOFF POSITION
C
C THE FILTER TAKES THE FORM OF
C A SINE BELL CENTERED ON THE FREQUENCY TO
C BE REMOVED COMPLETELY
C
0002      VIRTUAL BRJCT(2049)
C
C SET UP CONSTANTS
C
0003      N1=2.0*F1*DFI
0004      N2=2.0*F2*DFI
0005      NTAP=N2-N1
0006      CALL APWAIT
C
C SET UP FILTER IN AP
C
0007      CALL VCLR(0,1,NTAP)
0008      NFILT=NSAMP+1
0009      FTAP=1.0/FLOAT(NTAP)
0010      CALL APPUT(FTAP,1,1,2)
0011      CALL APWD
0012      CALL VTSMUL(1,1,2317,1,1,1)
0013      CALL VRAMP(0,1,0,1,NTAP)
0014      CALL VCOS(0,1,0,1,NTAP)
0015      CALL VTSADD(0,1,2049,0,1,NTAP)
0016      CALL VTSMUL(0,1,2327,0,1,NTAP)
0017      CALL VMOV(0,1,4096,1,NTAP)
C
C SET UP FULL FILTER NOW TAPER FINISHED
C
0018      CALL VCLR(0,1,NFILT)
0019      CALL VTSADD(0,1,2049,0,1,NFILT)
0020      CALL VMUL(N1,1,4096,1,N1,1,NTAP)
0021      CALL APWR
0022      CALL APGETA(0,NFILT,2,ATBR)
0023      CALL APWD
0024      RETURN
0025      END
```

```
0001      SUBROUTINE SPIKE(NSAMP,NSAMP2,ILENGTH,WHITE,IFLAG,ISPIKE)
```

```
      C
      C WIENER SPIKING FILTER ROUTINE
      C NSAMP=DATA LENGTH
      C NSAMP2=NEAREST POWER OF 2 TO NSAMP
      C ILENGTH=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
      C
```

```
0002      NTRAN=2*NSAMP2
0003      NCLR=NTRAN-NSAMP
0004      NFCLR=NTRAN-ILENGTH
0005      NTRAN2=NTRAN+2
0006      NM1=NSAMP2-1
0007      I6=NTRAN+ILENGTH
0008      I7=I6+ILENGTH
0009      I8=I7+ILENGTH
0010      ISP=I6+ISPIKE
```

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

```
0011      IF(IFLAG.NE.2)GOTO 10
0012      CALL SVESQ(0,1,8191,NSAMP)
0014      CALL VSQRT(8191,1,8191,1,1)
0015      CALL APWR
0016      CALL APGET(EN,8191,1,2)
0017      CALL APWD
```

```
      C
      C GET AUTOCORRELATION FUNCTION
      C
```

```
0018      10 CALL VCLR(NSAMP,1,NCLR)
0019      CALL RFFT(0,NTRAN,1)
0020      CALL VMOV(0,1,NTRAN,1,NTRAN)
0021      CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
0022      CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
0023      CALL VCLR(NTRAN+3,2,NM1)
0024      CALL RFFTSC(NTRAN,NTRAN,-1,-1)
0025      CALL RFFT(NTRAN,NTRAN,-1)
```

```
      C
      C AUTO FUNCTION NOW 2N LONG FROM NTRAN
      C ORIGINAL FUNCTION TRANSFORMED 0-NTRAN
      C
```

```
      C NOW WHITEN
      C
```

```
0026      CALL APWR
0027      CALL APPUT(WHITE,8191,1,2)
0028      CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)
```

```
      C
      C SET UP SPIKE CC FUNCTION
      C
```

```

0029      CALL VCLR(I6,1,ILENGTH)
0030      CALL VTSADD(ISP,1,2049,ISP,1,1)
      C
      C SOLVE EQNS
      C
0031      CALL WIENER(ILENGTH,NTRAN,I6,I7,I8,1)
0032      CALL APCHK(IER)
0033      IF(IER.NE.0)STOP 'LEVINSON FAILURE'
0035      CALL VMOV(I7,1,NTRAN,1,ILENGTH)
      C
      C NORMALISE FILTER IF ASKED FOR
      C
0036      IF(IFLAG.NE.1)GOTO 20
0038      CALL SVESQ(NTRAN,1,I6,ILENGTH)
0039      CALL VSQRT(I6,1,I6,1,1)
0040      CALL VDIV(I6,0,NTRAN,1,NTRAN,1,ILENGTH)
      C
      C APPLY FILTER
      C
0041      20 CALL VCLR(I6,1,NFCLR)
0042      CALL RFFT(NTRAN,NTRAN,1)
0043      CALL VMUL(0,1,NTRAN,1,0,1,2)
0044      CALL CVMUL(2,2,NTRAN,2,2,2,2,NM1,1)
0045      CALL RFFTSC(0,NTRAN,0,-1)
0046      CALL RFFT(0,NTRAN,-1)
      C
      C DO SCALING IF REQD
      C
0047      IF(IFLAG.NE.2)GOTO 30
0049      CALL SVESQ(0,1,8191,NSAMP)
0050      CALL VSQRT(8191,1,8191,1,1)
0051      CALL APWR
0052      CALL APPUT(EN,8190,1,2)
0053      CALL APWD
0054      CALL VDIV(8191,1,8190,1,8190,1,1)
0055      CALL VSMUL(0,1,8190,0,1,NSAMP)
0056      30 CALL APWR
0057      RETURN
0058      END

```

```
0001      SUBROUTINE PRDICT(NSAMP,NSAMP2,ILENTH,WHITE,IFLAG,NLAG)
C
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
C
0002      NTRAN=2*NSAMP2
0003      NCLR=NTRAN-NSAMP
0004      NM1=NSAMP2-1
0005      NTRAN2=NTRAN+2
0006      NLG=NTRAN+NLAG
0007      NFILT=ILENTH+NLAG
0008      I7=NLG+ILENTH
0009      I8=I7+ILENTH
C
C GET INPUT TRACE ENERGY
C
0010      IF(IFLAG.NE.2)GOTO 10
0011      CALL SVESQ(0,1,8191,NSAMP)
0012      CALL VSQRT(8191,1,8191,1,1)
0013      CALL APGET(EN,8191,1,2)
0014      CALL APWD
C
C GET AUTOCORRELATION FUNCTION
C
0015      10 CALL VCLR(NSAMP,1,NCLR)
0016      CALL RFFT(0,NTRAN,1)
0017      CALL VMOV(0,1,NTRAN,1,NTRAN)
0018      CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
0019      CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
0020      CALL VCLR(NTRAN+3,2,NM1)
0021      CALL RFFTSC(NTRAN,NTRAN,-1,-1)
0022      CALL RFFT(NTRAN,NTRAN,-1)
C
C NOW WHITEN IT
C
0023      CALL APWR
0024      CALL APPUT(WHITE,8191,1,2)
0025      CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)
C
C NOW SOLVE EQNS
C
0026      CALL WIENER(ILENTH,NTRAN,NLG,I7,I8,1)
0027      CALL APCHK(IER)
0028      IF(IER.NE.0)STOP 'LEVINSON FAILURE'
0029      IF(IFLAG.NE.1)GOTO 20
0030      CALL SVESQ(I7,1,I8,ILENTH)
0031
```

```
0034      CALL VSQRT(I8,1,I8,1,1)
0035      CALL VDIV(I8,0,I7,1,I7,1,ILENTH)
C
C APPLY FILTER
C
0036      20 CALL VCLR(NTRAN,1,NFILT)
0037      CALL VTSADD(NTRAN,1,2049,NTRAN,1,1)
0038      CALL VSUB(I7,1,NLG,1,NLG,1,ILENTH)
0039      CALL VCLR(I7,1,NTRAN-NFILT)
0040      CALL RFFT(NTRAN,NTRAN,1)
0041      CALL VMUL(0,1,NTRAN,1,0,1,2)
0042      CALL CVMUL(2,2,NTRAN,2,2,2,2,NM1,1)
0043      CALL RFFTSC(0,NTRAN,0,-1)
0044      CALL RFFT(0,NTRAN,-1)
C
C DO SCALING
C
0045      IF(IFLAG.NE.2)GOTO 30
0047      CALL SVESQ(0,1,8191,NSAMP)
0048      CALL VSQRT(8191,1,8191,1,1)
0049      CALL APWR
0050      CALL APPUT(EN,8190,1,2)
0051      CALL APWD
0052      CALL VDIV(8191,1,8190,1,8190,1,1)
0053      CALL VSMUL(0,1,8190,0,1,NSAMP)
0054      30 CALL APWR
0055      RETURN
0056      END
```

```
0001      SUBROUTINE TAPSUB(ICOM,IDRV,ISTAT,ILEN,IFNUM,BUF,NBYT,IEOT)
C
C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ,0 IS A WIND,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
0002      INTEGER*2 MASK(8),ESTATI,BUF(1)
0003      LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ECOM(4),
        *IFLEN,ESTAT,ERRS(8)
0004      DATA MASK/"1","2","4","10","20","40","100","200"/
0005      DATA SDSCOM/"0","1","2","3","4","5","6","7"/
0006      DATA ERRS/"377","377","377","377","377","377","377","377"/
0007      ITRY=0
0008      IF(ICOM) 10,30,20
C
C SECTION CONTROLLING A READ
C
C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
0009      10 ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
0010      NBUF=NBYT
0011      CALL TREAD(BUF,NBUF,ISTAT,IDRV)
0012      IF(ISTAT.EQ.0)RETURN
C
C ERROR DETECTED ON READ
C
0014      ISTATI=ISTAT
0015      GOTO 40
C
C IF SHORT RECORD FOUND REREAD TAPE
C
0016      50 ITMP=ISTATI.AND.MASK(6)
0017      IF(ITMP.NE.0)GOTO 10
0018      ITMP=ISTATI.AND.MASK(2)
0019      IF(ITMP.EQ.0)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
0020      WRITE(2,2010)IFNUM
0021      2010 FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
0022      IF(ITRY.GE.2)GOTO 130
0023      ECOM(1)=SDSCOM(6)
0024      ECOM(2)=1
0025      ECOM(3)=IDRV
0026      ECOM(4)=0
0027      CALL SDS10(ECOM,ESTAT, , )
0028      GOTO 10
```

```
C
C WRITE SECTION
C
0030      20 ITRY=ITRY+1
0031      IF(ITRY.GT.3)GOTO 130
0032      NBUF=NBYT
0033      IFLEN=(ILEN+3)/4
0034      IF(IFLEN.LT.2)IFLEN=2
0035      IPAD=(IFLEN*2048)-NBUF
0036      CALL TWRT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
0037      IF(ISTAT.EQ.0)RETURN
C
C WRITE ERROR DETECTED
C
0040      ISTAT1=ISTAT
0041      GOTO 40
0042      70 ITMP=ISTAT1.AND.MASK(6)
0043      ITMPI=ISTAT1.AND.MASK(2)
0044      IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
C
C REPORT AND RETRY
C
0049      WRITE(2,2020)IFNUM
0050      2020 FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
0051      ECOM(1)=SDSCOM(6)
0052      ECOM(2)=2
0053      ECOM(3)=IDRV
0054      ECOM(4)=0
0055      CALL SDS10(ECOM,ESTAT, , )
0056      NBUF=8
0057      IPAD=(IFLEN*2048)-NBUF
0058      CALL TWRT(ERRS,NBUF,ESTAT,IPAD,IFLEN,IDRV)
0059      GOTO 20
C
C WIND FOWARD ONE FILE
C
0060      30 COM(1)=SDSCOM(5)
0061      COM(2)=1
0062      COM(3)=IDRV
0063      COM(4)=0
0064      CALL SDS10(COM,ISTAT, , )
C
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
C
0065      ISTAT=ISTAT.AND..NOT.MASK(6)
0066      IF(ISTAT.EQ.2)RETURN
0067      ISTAT1=ISTAT
0068      IF(ISTAT.NE.0)GOTO 40
C
C IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
C
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
C
0071      ECOM(1)=SDSCOM(6)
```

```

0072      ECOM(2)=1
0073      ECOM(3)=IDRV
0074      ECOM(4)=0
0075      CALL SDS10(ECOM,ESTAT, , )
0076      35 RETURN
C
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS:= TAPE BUSY,TAPE OFFLINE
C BOT,EOT
C
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
C
0077      40 WRITE(2,1010)ISTATI,IFNUM
0078      1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
0079      ISTATI=ISTATI.AND..NOT.MASK(4)
0080      ITMP=ISTATI.AND.MASK(5)
0081      IF(ITMP.EQ.0)GOTO 80
0082      90 ECOM(1)=SDSCOM(1)
0083      ECOM(2)=0
0084      ECOM(3)=IDRV
0085      ECOM(4)=0
0086      CALL SDS10(ECOM,ESTAT, , )
C
C HAVING EXAMINED STATUS IF TAPE STILL
C BUSY, LOOP AGAIN,IF NOT TRY COMMAND AGAIN
C
0088      ESTATI=ESTAT
0089      ITMP=ESTATI.AND.MASK(5)
0090      IF(ITMP.NE.0)GOTO 90
0091      IF(ICOM) 10,30,20
C
C TAPE OFFLINE
C
0093      80 ITMP=ISTATI.AND.MASK(1)
0094      IF(ITMP.EQ.0)GOTO 100
0095      TYPE 1001,IDRV
0096      1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
0098      110 ECOM(1)=SDSCOM(1)
0099      ECOM(2)=0
0100      ECOM(3)=IDRV
0101      ECOM(4)=0
0102      CALL SDS10(ECOM,ESTAT, , )
0103      ESTATI=ESTAT
0104      ITMP=ESTATI.AND.MASK(1)
0105      IF(ITMP.NE.0)GOTO 110
0106      IF(ICOM) 10,30,20
C
C EOT
C
0108      100 ITMP=ISTATI.AND.MASK(3)
0109      IF(ITMP.EQ.0)GOTO 120

```

```

0111      TYPE 1002,IDRV
0112      1002 FORMAT(' EOT ON DRIVE ',I1)
0113      IEOT=-1
0114      RETURN
0115      120 IF(ICOM) 50,35,70
C
C ERROR EXIT RETURN
C
0116      130 ISTATI=-1
0117      RETURN
0118      END

```

Post-Stack Trace Mix :- MPTMIX

Input file.....DK1:MPTMIX.DAT

Log file.....DK1:MPTMIX.LOG

Input ParametersREAD(1,1000)NFILES,NSAMP,TPDRR,TPDRW,INFLG,OUTFLG1000 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)FSPECR1100 FORMAT(3A4)

FSPECR...Input files, 1 to NFILES

IF(OUTFLG.NE.0)READ(1,1100)FSPECW

FSPECW...output files, 1 to NFILES

```
C
C M J POULTER OCT 80
C MPTMIX
C THIS IS A PROGRAM WHICH PRODUCES
C A WEIGHTED MIX OF THREE INPUT
C TRACES TO GIVE ONE OUTPUT TRACE
C
0001 REAL*8 FSPECR(200),FSPECW(200),FBUF
0002 REAL*4 SEIS(2048),FNBUF(3)
0003 INTEGER*2 BUF(4352),IHBLK(256),IHBLKS(256),OUTFLG
0004 EQUIVALENCE (BUF(1),IHBLK(1)),(BUF(257),SEIS(1))
0005 LOGICAL*1 ISTAT,TPDRR,TPDRW
0006 DATA DEV/3RRK /
C
C SET UP I/O PARAMETERS
C
0007 IEOTR=0
0008 IEOTW=0
0009 IRD=IGETC( )
0010 IWRT=IGETC( )
0011 IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
0012 CALL ASSIGN(1,'DK2:MPTMIX.DAT',14)
0013 CALL ASSIGN(2,'DK2:MPTMIX.LOG',14)
C
C GET INPUT DATA
C
0015 READ(1,1000) NFILES,NSAMP,TPDRR,TPDRW,INFLG,OUTFLG
0016 1000 FORMAT(6I5)
0017 IF(INFLG.EQ.0)GOTO 10
0018 DO 20 J=1,NFILES
0019 READ(1,1001)FNBUF
0020 1001 FORMAT(3A4)
0021 CALL IRAD50(12,FNBUF,FBUF)
0022 FSPECR(J)=FBUF
0023 20 CONTINUE
0024 10 IF(OUTFLG.EQ.0)GOTO 30
0025 DO 40 J=1,NFILES
0026 READ(1,1001)FNBUF
0027 CALL IRAD50(12,FNBUF,FBUF)
0028 FSPECW(J)=FBUF
0029 40 CONTINUE
0030 30 CONTINUE
C
C SET UP CONSTANTS AND INIT AP
C
0033 CALL APINIT
0034 CALL VCLR(0,1,3*NSAMP)
0035 NBLKTR=NSAMP/128
0036 NBLKW=NBLKTR+1
0037 NBYTR=NBLKW*512
0038 NWDR=NBYTR/2
0039 NOPS=NFILES+1
0040 IOUT=0
C
```

```
C SET UP AP ADDRESSES
C
0041      IA1=0
0042      IA2=NSAMP
0043      IA3=IA2+NSAMP
0044      IA4=IA3+NSAMP
0045      CALL APWR
C
C MAIN OPS LOOP
C
0046      DO 100 IFIL=1,NOPS
0047      IFNUM=IFIL
C
C SWITCH VECTOR POSITIONS IN AP
C
0048      CALL VMOV(IA2,1,IA1,1,NSAMP)
0049      CALL VMOV(IA3,1,IA2,1,NSAMP)
0050      CALL VCLR(IA3,1,NSAMP)
0051      IF(IFNUM.GT.NFILES)GOTO 110
0053      IF(INFLG.NE.0)GOTO 120
C
C TAPE READ INPUT
C
0055      ITRY=1
0056      160 IF(ITRY.GT.3)GOTO 165
0058      IF(IEOTR.GE.0)GOTO 170
0060      165 WRITE(7,1600)TPDRR,IFNUM
0061      1600 FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I5)
0062      WRITE(7,1605)
0063      1605 FORMAT(' ENTER NEW READ DRIVE NUMBER:',S)
0064      READ(5,1610)TPDRR
0065      1610 FORMAT(I1)
0066      IEOTR=0
0067      IF(TPDRR.GT.2)STOP' EOT TERMINATE'
C
C READ TO MEMORY
C
0069      170 CALL TAPSUB(-1,TPDRR,ISTAT,IFLEN,IFNUM,BUF,NBYTR,IEOTR)
0070      IF(ISTAT.LT.0)WRITE(2,1620)IFNUM
0072      1620 FORMAT(' RETRY FAILED ON READ FILE NO:',I5)
0073      IF(IEOTR.LT.0)GOTO 180
C
C WIND OVER EOF MARKER
C
0075      CALL TAPSUB(0,TPDRR,ISTAT, ,IFNUM, , ,IEOTR)
0076      180 ITRY=ITRY+1
0077      IF(IHBLK(1).EQ."177777)GOTO 160
0079      GOTO 130
C
C INPUT FROM DISC
C
0080      120 CONTINUE
0081      CALL CLOSEC(IRD)
0082      FBUF=FSPECR(IFNUM)
```

```

0083      IF(LOOKUP(IRD,FBUF).LT.0)STOP 'LOOKUP ERROR'
0085      IF(IREADW(NWDR,BUF,0,IRD).LT.0)STOP 'READ ERROR'
0087      130 CONTINUE
C
C  HEADER BLOCK MANAGEMENT
C
0088      IPOS=IHBLK(24)
0089      IHBLK(129+IPOS)=5
0090      IPOS=IPOS+1
0091      IHBLK(24)=IPOS
C
C  PUT DATA IN AP
C
0092      CALL APPUT(SEIS,IA3,NSAMP,2)
C
C  SAVE HEADER BLOCK AND REPLACE WITH PREVIOUS ONE
C
0093      110 DO 135 J=1,256
0094          ITMP=IHBLK(J)
0095          IHBLK(J)=IHBLKS(J)
0096          IHBLKS(J)=ITMP
0097      135 CONTINUE
0098          IF(IFNUM.LE.1)GOTO 100
C
C  DO WEIGHTED MIX IN AP
C
0100      CALL APWD
0101      CALL VTSMUL(IA1,1,2329,I1,1,NSAMP)
0102      CALL VTSMUL(IA2,1,2327,IA4,1,NSAMP)
0103      CALL VADD(IA1,1,IA4,1,IA1,1,NSAMP)
0104      CALL VTSMUL(IA3,1,2329,IA4,1,NSAMP)
0105      CALL VADD(IA1,1,IA4,1,IA1,1,NSAMP)
0106      CALL SVESQ(IA1,1,IA4,NSAMP)
0107      CALL VSQRT(IA4,1,IA4,1,1)
0108      CALL VDIV(IA4,0,IA1,1,IA1,1,NSAMP)
0109      CALL APWR
0110      CALL APGET(SEIS,IA1,NSAMP,2)
C
C  OUTPUT RESULTATNT TRACE
C
0111      IF(OUTFLG.NE.0)GOTO 140
0113      IFLEN=NBLKW
0114      CALL APWD
0115      CALL TAPSUB(1,TPDRW,ISTAT,IFLEN,IFNUM,BUF,NBYTR,IEOTW)
C
C  CHECK FOR ERRORS
C
0116      IF(IEOTW.GE.0)GOTO 150
0118      WRITE(7,1700)TPDRW,IFNUM
0119      1700 FORMAT(' EOT ON WRITE DRIVE:',I2,' FILE NO:',I5)
0120      WRITE(7,1710)
0121      1710 FORMAT(' ENTER NEW WRITE DRIVE NUMBER:',S)
0122      READ(5,1620)TPDRW
0123      IEOTW=0

```

```

0124      IF(TPDRW.GT.2)STOP ' EOT TERMINATE '
0126      150 IF(ISTAT.GE.0)GOTO 100
0128      WRITE(2,1720)IFNUM
0129      1720 FORMAT(' FATAL WRITE ERROR ON FILE NUMBER:',I5)
0130      STOP ' WRITE ERROR TERMINATION '
C
C  DISC OUTPUT
C
0131      140 IOUT=IOUT+1
0132      FBUF=FSPECW(IOUT)
0133      IF(IENTER(IWRT,FBUF,NBLKW).LT.0)STOP 'ENTER ERROR'
0135      CALL APWD
0136      IF(IWRTW(NWDR,BUF,0,IWRT).LT.0)STOP 'WRITE ERROR'
0138      CALL CLOSEC(IWRT)
0139      100 CONTINUE
0140      CALL CLOSEC(IRD)
0141      CALL CLOSEC(IWRT)
0142      STOP ' NORMAL TERMINATION '
0143      END

```

Trace sequential-Time slice :- MPSLIC

Input file.....DK2:MPSLIC.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

```
C
C M J POULTER DEC 80
C
C THIS PROGRAM TAKES TRACE SEQUENTIAL DATA
C AND TIME SLICES IT FOR INPUT TO
C THE FD MIGRATION PROGRAM MPFMIG
C
0001     VIRTUAL BUFFER(16384)
0002     REAL*4 BUFFER,INBUF(128),OUTBUF(128),FBUF(3)
0003     REAL*8 FSPECW,FSPECR
0004     INTEGER*2 IAD(128),IWRTB(128)
0005     DATA DEV/3RRK /
C
C INPUT SET UP
C
0006     IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0008     IRD=IGETC()
0009     IWRT=IGETC()
0010     CALL ASSIGN(1,'DK2:MPSLIC.DAT',14)
C
C READ IN DATA
C
0011     READ(1,1000)NCHAN,NSAMP
0012     1000  FORMAT(2I5)
0013     READ(1,1001)FBUF
0014     1001  FORMAT(3A4)
0015     CALL IRAD50(12,FBUF,FSPECR)
0016     READ(1,1001)FBUF
0017     CALL IRAD50(12,FBUF,FSPECW)
C
C SET UP CONSTANTS
C
0018     NCHANW=0
0019     10  NCHANW=NCHANW+128
0020     IF(NCHAN.GT.NCHANW)GOTO 10
0022     NBLANK=NCHANW-NCHAN
0023     NBST=NBLANK/2
0024     ITIM=NCHANW/128
0025     NBLKR=NSAMP/128
0026     NFILW=(NCHANW*NBLKR)+1
C
C SET UP BUFFER ADDRESSES
C
0027     IT=1
0028     DO 20 J=1,128
0029     IAD(J)=IT
0030     20  IT=IT+128
C
C CLEAR STORE BUFFER
C
0031     DO 30 J=1,16384
0032     30  BUFFER(J)=0.0
C
C SET UP I/O FILES
```

```

C
0033      IF(LOOKUP(IRD,FSPECR).LT.0)STOP 'LOOKUP ERROR'
0035      IF(IENTER(IWRT,FSPECW,NFILW).LT.0)STOP 'ENTER ERROR'
C
C START OF TRANSFER LOOP
C
0037      DO 100 J=1,NBLKR
0038      IBLKR=J
0039      IST=NBST
C
C SET UP OUTPUT DISC ADDRESSES
C
0040      DO 110 JJ=1,128
0041      110 IWRTB(JJ)=((J-1)*(128*ITIM))+((JJ-1)*ITIM)+1
C
C SORT CODE
C
0042      DO 200 L=1,NCHAN
0043      IF(IREADW(256,INBUF,IBLKR,IRD).LT.0)STOP 'READ ERROR'
0045      IBLKR=IBLKR+NBLKR
C
C PUT DATA IN INT STORE
C
0046      DO 300 ISW=1,128
0047      IPOS=IAD(ISW)+IST
0048      BUFFER(IPOS)=INBUF(ISW)
0049      300 CONTINUE
0050      IST=IST+1
0051      IF(L.EQ.NCHAN)GOTO 210
0053      IF(IST.LT.128)GOTO 200
C
C OUTPUT CODE
C
0055      210 CONTINUE
0056      IST=0
0057      DO 220 LL=1,128
0058      IPOS=IAD(LL)
0059      IBLKW=IWRTB(LL)
0060      DO 230 LS=1,128
0061      OUTBUF(LS)=BUFFER(IPOS)
0062      BUFFER(IPOS)=0.0
0063      IPOS=IPOS+1
0064      230 CONTINUE
C
C WRITE OUT
C
0065      IF(IWRITE(256,OUTBUF,IBLKW,IWRT).LT.0)STOP 'WRITE ERROR'
0067      IWRTB(LL)=IBLKW+1
0068      220 CONTINUE
0069      200 CONTINUE
0070      100 CONTINUE
0071      CALL CLOSEC(IRD)
0072      CALL CLOSEC(IWRT)
0073      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
C M J POULTER DEC 80
C
C THIS PROGRAM TAKES THE OUTPUT FROM FD MIGRATION
C AND PUTS IT BACK INTO TIME SEQUENTIAL DATA
C
C
0001     VIRTUAL BUFFER(16384)
0002     REAL*4 BUFFER,INBUF(128),OUTBUF(128),FBUF(3)
0003     REAL*8 FSPECW,FSPECR
0004     INTEGER*2 IAD(128),IWRTB(128)
0005     DATA DEV/3RRK /
C
C INPUT SET UP
C
0006     IF(IFETCH(DEV).NE.0)STOP 'FETCH ERROR'
0008     IRD=IGETC()
0009     IWRT=IGETC()
0010     CALL ASSIGN(1,'DK2:MPUSLC.DAT',14)
C
C READ IN DATA
C
0011     READ(1,1000)NCHAN,NSAMP
0012     1000  FORMAT(2I5)
0013     READ(1,1001)FBUF
0014     1001  FORMAT(3A4)
0015     CALL IRAD50(12,FBUF,FSPECR)
0016     READ(1,1001)FBUF
0017     CALL IRAD50(12,FBUF,FSPECW)
C
C SET UP CONSTANTS
C
0018     NCHANW=0
0019     10  NCHANW=NCHANW+128
0020     IF(NCHAN.GT.NCHANW)GOTO 10
0022     NBLANK=NCHANW-NCHAN
0023     NBST=NBLANK/2
0024     ITIM=NCHANW/128
0025     NBLKW=NSAMP/128
0026     NFILW=(NCHANW*NBLKW)+1
C
C SET UP BUFFER ADDRESSES
C
0027     IT=1
0028     DO 20 J=1,128
0029     IAD(J)=IT
0030     20  IT=IT+128
C
C CLEAR STORE BUFFER
C
0031     DO 30 J=1,16384
0032     30  BUFFER(J)=0.0
C
C SET UP I/O FILES

```

```

C
0033      IF(LOOKUP(IRD,FSPECR).LT.0)STOP 'LOOKUP ERROR'
0035      IF(IENTER(IWRT,FSPECW,NFILW).LT.0)STOP 'ENTER ERROR'
C
C START OF TRANSFER LOOP
C
0037      INST=NBST+1
0038      NCHLST=0
0039      NCHANO=128-NBST
0040      IST=0
0041      NLEFT=NCHAN
0042      IF(NLEFT.LT.128)NCHANO=NLEFT
0044      NLEFT=NLEFT-NCHANO
C
C LOOP ON CHANNELS
C
0045      DO 100 JL=1,ITIM
0046      IBLKR=JL
0047      DO 110 JJ=1,NCHANO
0048      110 IWRTB(JJ)=((JJ-1)*NBLKW)+(NCHLST*NBLKW)+1
C
C LOOP ON SAMPLES
C
0049      DO 200 L=1,NSAMP
0050      IF(IREADW(256,INBUF,IBLKR,IRD).LT.0)STOP 'READ ERROR'
0052      IBLKR=IBLKR+ITIM
0053      IOUT=INST
C
C SORT DATA
C
0054      DO 300 LL=1,NCHANO
0055      IPOS=IAD(LL)+IST
0056      BUFFER(IPOS)=INBUF(IOUT)
0057      IOUT=IOUT+1
0058      300 CONTINUE
0059      IST=IST+1
0060      IF(IST.LT.128)GOTO 200
C
C WRITE OUT CODE
C
0062      DO 210 JS=1,NCHANO
0063      IPOS=IAD(JS)
0064      IBLKW=IWRTB(JS)
0065      DO 220 LS=1,128
0066      OUTBUF(LS)=BUFFER(IPOS)
0067      BUFFER(IPOS)=0.0
0068      IPOS=IPOS+1
0069      220 CONTINUE
0070      IF(IWRITW(256,OUTBUF,IBLKW,IWRT).LT.0)STOP 'WRITE ERROR'
0072      IWRTB(JS)=IBLKW+1
0073      210 CONTINUE
0074      IST=0
0075      200 CONTINUE
0076      INST=1

```

```

0077      NCHLST=NCHANO+NCHLST
0078      NCHANO=128
0079      IF(NLEFT.LT.128)NCHANO=NLEFT
0081      NLEFT=NLEFT-NCHANO
0082      100 CONTINUE
0083      CALL CLOSEC(IRD)
0084      CALL CLOSEC(IWRT)
0085      STOP 'NORMAL TERMINATION'
0086      END

```

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

1002 FORMAT(3A4)

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)

1003 FORMAT(2F10.0)

TLYR.....Two-way travel time

VLYR.....RMS velocity to this point

```

C
C M J POULTER NOV 80
C
C THIS IS A PROGRAM FOR
C 15 DEGREE FINITE DIFFERENCE MIGRATION
C
0001     VIRTUAL ASAVE(1024),BSAVE(1024),CSAVE(1024),
        #RHS(6144),APSAVE(2048),IBLK(2048),VSAVE(2048),
        #XSTOR(4096)
0002     REAL*8 FSPECR,FSVEL
0003     REAL*4 VINT(100),ADXST(4),V(100),VRMS(20),FBUF(3),
        #ADRHS(6),ADAPSV(2),ADXDIS(4)
0004     INTEGER*2 IV(100),IST(6),A0,A1,A2,A3,A4,A5,A6,A7,
        #IT(20)
0005     DATA A0/0/,A1/1024/,A2/2048/,A3/3072/,A4/4096/,
        #A5/5120/,A6/6144/,A7/7168/
0006     DATA DEV/3RRK /,FSVEL/12RDK4MPVTMPDAT/
C
C SET UP CONSTANTS AND READ IN DATA
C
0007     CALL APINIT
0008     CALL ASSIGN(1,'DK2:MPFMIG.DAT',14)
0009     IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0011     IVRT=IGETC()
0012     IRD=IGETC()
C
C READ IN DATA
C
0013     READ(1,1000) NSAMP,MSAMP,NTRACE,NV,NVELS
0014     1000  FORMAT(10I5)
0015     READ(1,1001)DX,DT,DTOR
0016     1001  FORMAT(3F10.0)
0017     READ(1,1000)(IV(I),I=1,NV)
0018     READ(1,1002)FBUF
0019     1002  FORMAT(3A4)
0020     CALL IRAD50(12,FBUF,FSPECR)
C
C SET UP VM ADDRESSES
C
0021     IPOS=1
0022     DO 10 J=1,6
0023     ADRHS(J)=APGAD(RHS(IPOS))
0024     IPOS=IPOS+1024
0025     10  CONTINUE
0026     ADVEL=ADGET(VSAVE(1))
0027     ADAPSV(1)=APGAD(APSAVE(1))
0028     ADAPSV(2)=APGAD(APSAVE(1025))
0029     ADASV=APGAD(ASAVE(1))
0030     ADBSV=APGAD(BSAVE(1))
0031     ADCSV=APGAD(CSAVE(1))
0032     IB=1
0033     DO 15 I=1,4
0034     ADXST(I)=APGAD(XSTOR(IB))
0035     ADXDIS(I)=ADGET(XSTOR(IB))

```

```
0036      IB=IB+1024
0037      15 CONTINUE
C
C SET UP CONSTANTS
C
0038      NCHANW=0
0039      20 NCHANW=NCHANW+128
0040      IF(NTRACE.GT.NCHANW)GOTO 20
0042      IF(NCHANW.GT.1024)STOP ' TOO MANY TRACES TO MIGRATE '
0044      ITIM=NCHANW/128
0045      NBST=(NCHANW-NTRACE)/2
0046      NEND=NCHANW-NTRACE-NBST
C
C SET UP CONSTANTS FOR LOOPS
C
0047      ITOR=DTOR/DT
0048      ITORLM=MSAMP/ITOR
0049      LIMIT=NSAMP
0050      NVM=NV-1
0051      ACOF=(0.518*DTOR*DT)/(16.0*DX*DX)
0052      IBLK(1)=1
0053      DO 30 L=2,NSAMP
0054      30 IBLK(L)=IBLK(L-1)+ITIM
0055      NVSIZ=0
0056      40 NVSIZ=NVSIZ+128
0057      IF(ITORLM.GT.NVSIZ)GOTO 40
0059      NVSIZ=NVSIZ/128
0060      NCHWM1=NCHANW-1
0061      NCHWR2=NCHANW*2
0062      ITRLM2=ITORLM*2
0063      IA01=A0+1
0064      IA31=A3+1
0065      IA41=A4+1
C
C SET UP FILES
C
0066      IF(LOOKUP(IRD,FSPECR).LT.0)STOP 'LOOKUP ERROR '
0068      IFILV=NV*NVSIZ
0069      IF(ENTER(IVRT,FSVEL,IFILV).LT.0)STOP 'ENTER ERROR '
C
C SET UP I AND I/12
C
0071      CALL VCLR(A4,1,NCHANW)
0072      CALL VTSADD(A4,1,2049,A4,1,NCHANW)
0073      CALL VMOV(A4,1,A5,1,NCHANW)
0074      CALL VTSMUL(A4,1,2331,A4,1,NCHANW)
0075      CALL VTSMUL(A4,1,"4427",A4,1,NCHANW)
0076      CALL APWR
0077      CALL APGETA(A4,NCHANW,2,ADAPSV(1))
0078      CALL APGETA(A5,NCHANW,2,ADAPSV(2))
C
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
C
0079      IBLKV=0
0080      DO 50 LV=1,NV
0081      IV(LV)=IV(LV)+NBST
C
C READ IN VELS
C
0082      DO 60 LL=1,NVELS
0083      READ(1,1003)T,VRMS(LL)
0084      1003  FORMAT(2F10.0)
0085      IT(LL)=T/DTOR+1
0086      60  CONTINUE
C
C DO LINEAR INTERP ON RMS VELS
C
0087      N1=1
0088      N2=IT(1)
0089      DO 55 LI=N1,N2
0090      55  VSAVE(LI)=VRMS(1)
0091      IF(NVELS.EQ.1)GOTO 65
0093      DO 70 LJ=2,NVELS
0094      N1=N2+1
0095      N2=IT(LJ)
0096      VT=VRMS(LJ-1)
0097      DELV=(VRMS(LJ)-VT)/(N2-N1-1)
0098      DO 75 LT=N1,N2
0099      VSAVE(LT)=VT
0100      VT=VT+DELV
0101      75  CONTINUE
0102      70  CONTINUE
0103      65  CONTINUE
0104      N1=N2+1
0105      N2=ITORLM+1
0106      DO 80 LL=N1,N2
0107      80  VSAVE(LL)=VRMS(NVELS)
C
C CHANGE INTO INT VELS
C
0108      VTP1=0.0
0109      VTP2=VSAVE(1)*VSAVE(1)
0110      DO 90 LINT=1,ITORLM
0111      VTP1=VTP2
0112      VTP2=VSAVE(LINT+1)*VSAVE(LINT+1)*(LINT+1)
0113      VSAVE(LINT)=SQRT(VTP2-VTP1)
0114      90  CONTINUE
0115      IF(IWRITA(IVRT,ITRLM2,IBLKV,ADVEL).LT.0)STOP 'WRITV ERROR'
0117      IBLKV=IBLKV+NVSIZ
0118      CALL IWAIT(IVRT)
0119      50  CONTINUE
C
C START MAIN LOOP
C

```

```

0120      DO 100 I TORCT=1,ITORLM
0121      LIMIT=LIMIT-ITOR
      C
      C ZERO X ARRAYS
      C
0122      CALL VCLR(A0,1,NCHANW)
0123      CALL APWR
0124      DO 110 I=1,4
0125      IST(I)=I
0126      FAD=ADXST(I)
0127      CALL APGETA(A0,NCHANW,2,FAD)
0128      110 CONTINUE
      C
      C READ IN VELOCITIES
      C
0129      IBLKV=0
0130      DO 120 L=1,NV
0131      IF (IREADA(IVRT,ITRLM2,IBLKV,ADVEL).LT.0)STOP 'READV ERROR'
0133      IBLKV=IBLKV+NVSIZ
0134      CALL IWAIT(IVRT)
0135      V(L)=VSAVE(ITORCT)
0136      120 CONTINUE
      C
      C GEN V SLICE
      C
0137      CALL APPUT(V,A0,NV,2)
0138      DO 130 I=1,NVM
0139      130 VINT(I)=(V(I+1)-V(I))/(IV(I+1)-IV(I))
0140      CALL APPUT(VINT,A1,NVM,2)
0141      CALL APWD
      C
0142      DO 140 I=1,NVM
0143      NVD=IV(I+1)-IV(I)+1
0144      CALL VRAMP(A0+I-1,A1+I-1,A2+IV(I)-1,1,NVD)
0145      140 CONTINUE
0146      CALL VFILL(A2+IV(1)-1,A2,1,NBST)
0147      CALL VFILL(A2+IV(NV)-1,A2+IV(NV),1,NEND)
0148      CALL VSQ(A2,1,A2,1,NCHANW)
      C
      C GEN A
      C
0149      CALL APPUT(ACOF,A0,1,2)
0150      CALL APWD
0151      CALL VSMUL(A2,1,A0,A0,1,NCHANW)
      C
      C SET UP NECESSARY COEFFS FROM SAVE
      C
0152      CALL APWR
0153      CALL APPUTA(A1,NCHANW,2,ADAPSV(1))
0154      CALL APPUTA(A2,NCHANW,2,ADAPSV(2))
0155      CALL APWD
      C
      C GEN COEFFS
      C

```

```
C
C (I+(B-A)T)
C
0156 CALL VSUB(A0,1,A1,1,A3,1,NCHANW)
0157 CALL VTSMUL(A3,1,2050,A4,1,NCHANW)
0158 CALL VNEG(A4,1,A4,1,NCHANW)
0159 CALL VADD(A2,1,A4,1,A4,1,NCHANW)
C
C (I+(A+B)T)
C
0160 CALL VADD(A0,1,A1,1,A5,1,NCHANW)
0161 CALL VTSMUL(A5,1,2050,A6,1,NCHANW)
0162 CALL VNEG(A6,1,A6,1,NCHANW)
0163 CALL VADD(A2,1,A6,1,A6,1,NCHANW)
0164 CALL APWR
0165 CALL APGETA(A5,NCHANW,2,ADRHS(3))
0166 CALL APGETA(A6,NCHANW,2,ADRHS(4))
0167 CALL APGETA(A5,NCHANW,2,ADRHS(5))
0168 CALL APGETA(A6,NCHANW,2,ADRHS(6))
0169 CALL APWD
0170 CALL VNEG(A3,1,A5,1,NCHANW)
0171 CALL VNEG(A4,1,A6,1,NCHANW)
0172 CALL APWR
0173 CALL APGETA(A5,NCHANW,2,ADRHS(1))
0174 CALL APGETA(A6,NCHANW,2,ADRHS(2))
0175 CALL APGETA(A3,NCHANW,2,ADCSV)
0176 CALL APWD
0177 CALL VMOV(A3,1,A5,1,NCHANW)
C
C PARTIALLY SOLVE AND SAVE RES
C
0178 CALL FACTOR(A3,1,A4,1,A5,1,A6,1,A7,1,NCHANW)
0179 CALL APWR
0180 CALL APGSP(15,IER)
0181 IF(IER.NE.0)STOP 'FACTOR ERROR'
0183 CALL APGETA(A6,NCHANW,2,ADASV)
0184 CALL APGETA(A7,NCHANW,2,ADBSV)
0185 CALL APWD
C
C START ON RHS
C
C
C LOOP ON SAMPLES
C
0186 ISTB=NSAMP
0187 DO 200 ITR=1,LIMIT
0188 IBLKR=IBLK(ISTB)
0189 FAD=ADXDIS(IST(3))
0190 CALL IWAIT(IRD)
0191 IF(IREADA(IRD,NCHWR2,IBLKR,FAD).LT.0)STOP 'READ ERROR'
0193 CALL VCLR(A7,1,NCHANW)
0194 IA1=-1
0195 IA2=0
0196 CALL IWAIT(IRD)
```

```
0197      DO 210 I=1,3
0198      IA1=IA2+1
0199      IA2=IA1+1
0200      CALL APWR
0201      CALL APPUTA(A0,NCHANW,2,ADRHS(IA1))
0202      CALL APPUTA(A1,NCHANW,2,ADRHS(IA2))
0203      FST=ADXST(IST(I))
0204      CALL APPUTA(A3,NCHANW,2,FST)
0205      CALL APWD
0206      CALL VMUL(A1,1,A3,1,A4,1,NCHANW)
0207      CALL VMA(IA01,1,A3,1,IA41,1,IA41,1,NCHWM1)
0208      CALL VMA(A0,1,IA31,1,A4,1,A4,1,NCHWM1)
0209      CALL VADD(A4,1,A7,1,A7,1,NCHANW)
0210      CONTINUE
C
C DO SOLUTION TO EQN
C
0211      CALL VCLR(A3,1,NCHANW)
0212      CALL VCLR(A4,1,NCHANW)
0213      CALL APWR
0214      CALL APPUTA(A0,NCHANW,2,ADASV)
0215      CALL APPUTA(A1,NCHANW,2,ADBSV)
0216      CALL APPUTA(A2,NCHANW,2,ADCSV)
0217      CALL APWD
0218      CALL SOLVE(A0,1,A1,1,A7,1,A2,1,A3,1,A4,1,NCHANW)
0219      CALL APWR
0220      CALL APGSP(IER,15)
0221      IF(IER.NE.0)STOP 'SOLVE FAILURE'
0222      FST=ADXST(IST(4))
0223      CALL APGETA(A4,NCHANW,2,FST)
0224      FAD=ADXDIS(IST(4))
0225      CALL APWD
0226      IF(IWRITA(IRD,NCHWR2,IBLKR,FAD).LT.0)STOP 'WRITE ERROR'
C
C TURN AROUND VECTORS
C
0229      IST(5)=IST(1)
0230      IST(6)=IST(2)
0231      DO 220 I=1,4
0232      220 IST(I)=IST(I+2)
0233      ISTB=ISTB-1
C
C END OF LOOP
C
0234      200 CONTINUE
0235      CALL IWAIT(IRD)
0236      100 CONTINUE
0237      CALL CLOSEC(IRD)
0238      STOP 'NORMAL TERMINATION'
0239      END
```

```

C
C M J POULTER NOV 80
C
C THIS IS A PROGRAM FOR
C 45 DEGREE FINITE DIFFERENCE MIGRATION
C
0001   VIRTUAL ASAVE(1024),BSAVE(1024),CSAVE(1024),
      #RHS(10240),APSAVE(2048),IBLK(2048),VSAVE(2048),
      #XSTOR(6144)
0002   REAL*8 FSPECR,FSVEL
0003   REAL*4 VINT(100),ADXST(6),V(100),VRMS(20),FBUF(3),
      #ADRHS(10),ADAPSV(2),ADXDIS(6)
0004   INTEGER*2 IV(100),IST(8),A0,A1,A2,A3,A4,A5,A6,A7,
      #IT(20)
0005   DATA A0/0/,A1/1024/,A2/2048/,A3/3072/,A4/4096/,
      #A5/5120/,A6/6144/,A7/7168/
0006   DATA DEV/3RRK /,FSVEL/12RDK4MPVTMPDAT/
C
C SET UP CONSTANTS AND READ IN DATA
C
0007   CALL APINIT
0008   CALL ASSIGN(1,'DK2:MPFMIG.DAT',14)
0009   IF(IFETCH(DEV).NE.0)STOP 'FETCH ERROR'
0010   IVRT=IGETC()
0011   IRD=IGETC()
C
C READ IN DATA
C
0013   READ(1,1000) NSAMP,MSAMP,NTRACE,NV,NVELS
0014   1000 FORMAT(10I5)
0015   READ(1,1001)DX,DT,DTOR
0016   1001 FORMAT(3F10.0)
0017   READ(1,1000)((IV(I),I=1,NV)
0018   READ(1,1002)FBUF
0019   1002 FORMAT(3A4)
0020   CALL IRAD50(12,FBUF,FSPECR)
C
C SET UP VM ADDRESSES
C
0021   IPOS=1
0022   DO 10 J=1,10
0023   ADRHS(J)=APGAD(RHS(IPOS))
0024   IPOS=IPOS+1024
0025   10 CONTINUE
0026   ADVEL=ADGET(VSAVE(1))
0027   ADAPSV(1)=APGAD(APSAVE(1))
0028   ADAPSV(2)=APGAD(APSAVE(1025))
0029   ADASV=APGAD(ASAVE(1))
0030   ADBSV=APGAD(BSAVE(1))
0031   ADCSV=APGAD(CSAVE(1))
0032   IB=1
0033   DO 15 I=1,6
0034   ADXST(I)=APGAD(XSTOR(IB))
0035   ADXDIS(I)=ADGET(XSTOR(IB))

```

```
0036      IB=IB+1024
0037      15 CONTINUE
      C
      C SET UP CONSTANTS
      C
0038      NCHANW=0
0039      20 NCHANW=NCHANW+128
0040      IF(NTRACE.GT.NCHANW)GOTO 20
0042      IF(NCHANW.GT.1024)STOP ' TOO MANY TRACES TO MIGRATE '
0044      ITIM=NCHANW/128
0045      NBST=(NCHANW-NTRACE)/2
0046      NEND=NCHANW-NTRACE-NBST
      C
      C SET UP CONSTANTS FOR LOOPS
      C
0047      ITOR=DTOR/DT
0048      ITORLM=MSAMP/ITOR
0049      LIMIT=NSAMP
0050      NVM=NV-1
0051      ACOF=(0.279*DT*DT)/(4.0*DX*DX)
0052      BCOF=(DTOR*DT)/(32.0*DX*DX)
0053      IBLK(1)=1
0054      DO 30 L=2,NSAMP
0055      30 IBLK(L)=IBLK(L-1)+ITIM
0056      NVSIZ=0
0057      40 NVSIZ=NVSIZ+128
0058      IF(ITORLM.GT.NVSIZ)GOTO 40
0060      NVSIZ=NVSIZ/128
0061      NCHWM1=NCHANW-1
0062      NCHWR2=NCHANW*2
0063      ITRLM2=ITORLM*2
0064      IA01=A0+1
0065      IA31=A3+1
0066      IA41=A4+1
      C
      C SET UP FILES
      C
0067      IF(LOOKUP(IRD,FSPECR).LT.0)STOP 'LOOKUP ERROR '
0069      IFILV=NV*NVSIZ
0070      IF(IENTER(IVRT,FSVEL,IFILV).LT.0)STOP 'ENTER ERROR '
      C
      C SET UP I AND I/12
      C
0072      CALL VCLR(A4,1,NCHANW)
0073      CALL VTSADD(A4,1,2049,A4,1,NCHANW)
0074      CALL VMOV(A4,1,A5,1,NCHANW)
0075      CALL VTSMUL(A4,1,2331,A4,1,NCHANW)
0076      CALL VTSMUL(A4,1,"4427",A4,1,NCHANW)
0077      CALL APWR
0078      CALL APGETA(A4,NCHANW,2,ADAPSV(1))
0079      CALL APGETA(A5,NCHANW,2,ADAPSV(2))
      C
      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
C
0080     IBLKV=0
0081     DO 50 LV=1,NV
0082     IV(LV)=IV(LV)+NBST
C
C READ IN VELS
C
0083     DO 60 LL=1,NVELS
0084     READ(1,1003)T,VRMS(LL)
0085     1003  FORMAT(2F10.0)
0086     IT(LL)=T/DTOR+1
0087     60  CONTINUE
C
C DO LINEAR INTERP ON RMS VELS
C
0088     N1=1
0089     N2=IT(1)
0090     DO 55 LI=N1,N2
0091     55  VSAVE(LI)=VRMS(LI)
0092     IF(NVELS.EQ.1)GOTO 65
0093     DO 70 LJ=2,NVELS
0094     N1=N2+1
0095     N2=IT(LJ)
0096     VT=VRMS(LJ-1)
0097     DELV=(VRMS(LJ)-VT)/(N2-N1-1)
0098     DO 75 LT=N1,N2
0099     VSAVE(LT)=VT
0100     VT=VT+DELV
0101     75  CONTINUE
0102     70  CONTINUE
0103     65  CONTINUE
0104     N1=N2+1
0105     N2=ITORM+1
0106     DO 80 LL=N1,N2
0107     80  VSAVE(LL)=VRMS(NVELS)
C
C CHANGE INTO INT VELS
C
0109     VTP1=0.0
0110     VTP2=VSAVE(1)*VSAVE(1)
0111     DO 90 LINT=1,ITORLM
0112     VTP1=VTP2
0113     VTP2=VSAVE(LINT+1)*VSAVE(LINT+1)*(LINT+1)
0114     VSAVE(LINT)=SQRT(VTP2-VTP1)
0115     90  CONTINUE
0116     IF(IWRITA(IVRT,ITRLM2,IBLKV,ADVEL).LT.0)STOP 'WRITV ERROR'
0117     IBLKV=IBLKV+NVSIZ
0118     CALL IWAIT(IVRT)
0119     50  CONTINUE
C
C START MAIN LOOP
```

```

C
0121      DO 100 I TORCT=1,ITORLM
0122      LIMIT=LIMIT-ITOR
C
C ZERO X ARRAYS
C
0123      CALL VCLR(A0,1,NCHANW)
0124      CALL APWR
0125      DO 110 I=1,6
0126      IST(I)=1
0127      FAD=ADXST(I)
0128      CALL APGETA(A0,NCHANW,2,FAD)
0129      110 CONTINUE
C
C READ IN VELOCITIES
C
0130      IBLKV=0
0131      DO 120 L=1,NV
0132      IF(IREADA(IVRT,ITRLM2,IBLKV,ADVEL).LT.0)STOP 'READV ERROR'
0133      IBLKV=IBLKV+NVSIZ
0134      CALL IWAIT(IVRT)
0135      V(L)=VSAVE(ITORCT)
0136      120 CONTINUE
C
C GEN V SLICE
C
0138      CALL APPUT(V,A0,NV,2)
0139      DO 130 I=1,NVM
0140      130 VINT(I)=(V(I+1)-V(I))/(IV(I+1)-IV(I))
0141      CALL APPUT(VINT,A1,NVM,2)
0142      CALL APWD
C
0143      DO 140 I=1,NVM
0144      NVD=IV(I+1)-IV(I)+1
0145      CALL VRAMP(A0+I-1,A1+I-1,A2+IV(I)-1,1,NVD)
0146      140 CONTINUE
0147      CALL VFILL(A2+IV(1)-1,A2,1,NBST)
0148      CALL VFILL(A2+IV(NV)-1,A2+IV(NV),1,NEND)
0149      CALL VSQ(A2,1,A2,1,NCHANW)
C
C GEN A,B
C
0150      CALL APPUT(ACOF,A0,1,2)
0151      CALL APPUT(BCOF,A1,1,2)
0152      CALL APWD
0153      CALL VSMUL(A2,1,A0,A0,1,NCHANW)
0154      CALL VSMUL(A2,1,A1,A1,1,NCHANW)
C
C SET UP NECESSARY COEFFS FROM SAVE
C
0155      CALL APWR
0156      CALL APPUTA(A2,NCHANW,2,ADAPSV(1))
0157      CALL APPUTA(A3,NCHANW,2,ADAPSV(2))
0158      CALL APWD

```

```
C
C GEN COEFFS
C
C
C (2+(2B+ACOF)T)
C
0159 CALL VTSMUL(A2,1,2050,A4,1,NCHANW)
0160 CALL VADD(A0,1,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 VTSMUL(A5,1,2050,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 VTSMUL(A7,1,2050,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 VTSMUL(A5,1,2050,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
0193      CALL VMOV(A4,1,A6,1,NCHANW)
0194      CALL FACTOR(A4,1,A5,1,A6,1,A3,1,A7,1,NCHANW)
0195      CALL APWR
0196      CALL APGSP(15,IER)
0197      IF( IER.NE.0)STOP 'FACTOR ERROR'
0199      CALL APGETA(A3,NCHANW,2,ADASV)
0200      CALL APGETA(A7,NCHANW,2,ADBSV)
0201      CALL APWD
      C
      C START ON RHS
      C
      C
      C LOOP ON SAMPLES
      C
0202      ISTB=NSAMP
0203      DO 200 ITR=1,LIMIT
0204      IBLKR=IBLK(ISTB)
0205      FAD=ADXDIS(IST(5))
0206      CALL IWAIT(IRD)
0207      IF( IREADA(IRD,NCHWR2,IBLKR,FAD).LT.0)STOP 'READ ERROR'
0209      CALL VCLR(A7,1,NCHANW)
0210      IA1=-1
0211      IA2=0
0212      CALL IWAIT(IRD)
0213      DO 210 I=1,5
0214      IA1=IA2+1
0215      IA2=IA1+1
0216      CALL APWR
0217      CALL APPUTA(A0,NCHANW,2,ADRHS(IA1))
0218      CALL APPUTA(A1,NCHANW,2,ADRHS(IA2))
0219      FST=ADXST(IST(I))
0220      CALL APPUTA(A3,NCHANW,2,FST)
0221      CALL APWD
0222      CALL VMUL(A1,1,A3,1,A4,1,NCHANW)
0223      CALL VMA(IA01,1,A3,1,IA41,1,IA41,1,NCHWM1)
0224      CALL VMA(A0,1,IA31,1,A4,1,A4,1,NCHWM1)
0225      CALL VADD(A4,1,A7,1,A7,1,NCHANW)
0226      210 CONTINUE
      C
      C DO SOLUTION TO EQN
      C
0227      CALL VCLR(A3,1,NCHANW)
0228      CALL VCLR(A4,1,NCHANW)
0229      CALL APWR
0230      CALL APPUTA(A0,NCHANW,2,ADASV)
0231      CALL APPUTA(A1,NCHANW,2,ADBSV)
0232      CALL APPUTA(A2,NCHANW,2,ADCSV)
0233      CALL APWD
0234      CALL SOLVE(A0,1,A1,1,A7,1,A2,1,A3,1,A4,1,NCHANW)
0235      CALL APWR
0236      CALL APGSP( IER,15)
0237      IF( IER.NE.0)STOP 'SOLVE FAILURE'
0239      FST=ADXST(IST(6))
```

```
0240      CALL APGETA(A4,NCHANW,2,FST)
0241      FAD=ADXDIS(IST(6))
0242      CALL APWD
0243      IF(IWRITA(IRD,NCHWR2,IBLKR,FAD).LT.0)STOP'WRITE ERROR'
C
C TURN AROUND VECTORS
C
0245      IST(7)=IST(1)
0246      IST(8)=IST(2)
0247      DO 220 I=1,6
0248      220 IST(I)=IST(I+2)
0249      ISTB=ISTB-1
C
C END OF LOOP
C
0250      200 CONTINUE
0251      CALL IWAIT(IRD)
0252      100 CONTINUE
0253      CALL CLOSEC(IRD)
0254      STOP'NORMAL TERMINATION'
0255      END
```

Kirchhoff Migration Operator Design :- MPOGEN

Input file.....DK2:MPOGEN.DAT

Input ParametersREAD(1,1000)TSTEP,XSTEP1000 FORMAT(2F10.0)

TSTEP....Sample interval in seconds

XSTEP....Trace spacing in kms

READ(1,1001)NHLFWD,NINC,NSAMP,NSTEP,IFRNGE,NVEL1001 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(1,1002)FSPECO1002 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.

```
C
C M J POULTER DEC 80
C
C THIS PROGRAM CALCULATES A SET OF
C MIGRATION OPERATORS FOR A GIVEN
C EARTH MODEL
C THIS DATA IS THEN USED AS INPUT
C TO MPKMIG
C
0001 REAL*8 FSPEC
0002 REAL*4 OPVAL(2560),FBUF(3),VEL(20),DZ(20)
0003 INTEGER*2 A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,
      #A11,A12,A13,A14,A15,AIOP,A(5),NRANGE(512),NLEAD(512),
      #IT(20),IOP(20),AC(5),AS,ACON
0004 DATA DEV/3RRK /
0005 DATA A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,
      #A15/0,512,1024,1536,2048,2560,3072,3584,4096,4608,
      #5120,5632,6144,6656,7168,7680/
0006 DATA A/5632,6144,6656,7168,7680/
0007 DATA AC/"4001","4002","4441","4442","4443/
C
C SET UP SYSTEM I/O
C
0008 IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0010 IRD=IGETC()
0011 CALL ASSIGN(1,'DK2:MPOGEN.DAT',14)
C
C READ IN DATA
C
0012 READ(1,1000)TSTEP,XSTEP
0013 1000 FORMAT(2F10.0)
0014 READ(1,1001)NHLFWD,NINC,NSAMP,NSTEP,IFRNGE,NVEL
0015 1001 FORMAT(6I5)
0016 READ(1,1002)FBUF
0017 1002 FORMAT(3A4)
0018 CALL IRAD50(12,FBUF,FSPEC)
0019 READ(1,1000)TMIN,TMAX
0020 ITMIN=TMIN/TSTEP
0021 ITMAX=TMAX/TSTEP
0022 DO 10 J=1,NVEL
0023 READ(1,1000)TPOS,VEL(J)
0024 IT(J)=TPOS/TSTEP
0025 VEL(J)=VEL(J)/2.0
0026 10 CONTINUE
C
C SET UP CONSTANTS
C
0027 PI=3.141592
0028 CONST=-1.0/(PI*TSTEP)
0029 NBUF=NHLFWD*5
0030 NBUF2=2*NBUF
C
C SET UP RANGE OF EACH OPERATOR +NO
C OF OPERATORS NECESSARY
```

```
      C
0031     IST=0
0032     5  IST=IST+1
0033     IF(ITMIN.GT.IT(IST))GOTO 5
0035     IA=IST
0036     ITMI=0
0037     ITMA=ITMIN
0038     NOPTOT=0
0039     IPNOP=1
0040     IVS=0
0041     15  ITMI=ITMA
0042     ITMA=IT(IA)
0043     IF(ITMA.GT.ITMAX)ITMA=ITMAX
0045     CALL OPCALC(ITMI,ITMA,NRANGE,IPNOP,NSTEP,IFRNGE,NOP)
0046     NOPTOT=NOPTOT+NOP
0047     IVS=IVS+1
0048     IOP(IVS)=NOP
0049     DZ(IVS)=VEL(IA)*TSTEP
0050     IA=IA+1
0051     IF(ITMA.LT.ITMAX)GOTO 15
0053     NRANGE(IPNOP-1)=ITMAX
0054     NOUT=(NOPTOT+1)
      C
      C SET UP I/O CONSTANTS
      C
0055     IBNOP=0
0056     NBRNG=(NOUT+255)/256
0057     NINLD=(NHLFWD+255)/256
0058     IOPINC=(NBUF+127)/128
0059     NBLD=NOPTOT*NINLD
0060     NBOPS=NOPTOT*IOPINC
0061     NFILS=NBOPS+NBLD+NBRNG+1
0062     INRNGS=1
0063     INLDST=1+NBRNG
0064     INOPST=INLDST+NBLD
      C
      C OPEN OUTPUT FILE
      C
0065     IF(IENTER(IRD,FSPEC,NFILS).LT.0)STOP'ENTER ERROR'
      C
      C WRITE OUT NOPTOT
      C
0067     IF(IWRITW(1,NOPTOT,0,IRD).LT.0)STOP'NOPWITW ERROR'
      C
      C WRITE OUT NRANGE
      C
0069     IF(IWRITW(NOUT,NRANGE,1,IRD).LT.0)STOP'WRITW ERR'
      C
      C CALC XSQ
      C
0071     CALL APINIT
0072     CALL VCLR(A0,1,NHLFWD)
0073     CALL APWR
0074     CALL APPUT(XSTEP,A0+1,1,2)
```

```
0075      CALL APWD
0076      CALL VRAMP(A0,A0+1,A0,1,NHLFWD)
0077      CALL VSQ(A0,1,A0,1,NHLFWD)
0078      CALL APWR
C
C GET IT VECTOR
C
0079      CALL APPUT(NRANGE,A8,NOPTOT+1,1)
0080      CALL APWD
0081      CALL VFLT(A8,1,A8,1,NOPTOT+1)
0082      CALL VADD(A8,1,A8+1,1,A7,1,NOPTOT)
0083      CALL VTSMUL(A7,1,2327,A6,1,NOPTOT)
0084      CALL APWR
C
C LOOP ON DIFFERENT VELOCITIES
C
0085      IADD=-1
0086      IBLKNL=INLDST
0087      IBLKOP=INOPST
0088      DO 20 IV=1,IVS
0089      IAOP=-1
0090      NOP=IOP(IV)
0091      DZINV=1.0/DZ(IV)
C
C GET (IT*DZ)**2
C
0092      CALL APPUT(DZ(IV),A7,1,2)
0093      CALL APWD
0094      CALL VSMUL(A6,1,A7,A7,1,NOP)
0095      CALL VSQ(A7,1,A7,1,NOP)
C
C LOOP ON OPERATORS
C
0096      DO 30 IL=1,NOP
0097      IADD=IADD+1
0098      IAOP=IAOP+1
C
C CALC OTHER INTERMEDIATE FACTORS
C USED IN OPERATOR CALCULATION
C
0099      CALL VSADD(A0,1,A7+IAOP,A1,1,NHLFWD)
0100      CALL VSQRT(A1,1,A1,1,NHLFWD)
0101      CALL APWR
0102      CALL APPUT(CONST,A15,1,2)
0103      CALL APWD
0104      CALL VSMUL(A6+IADD,1,A15,A3,1,1)
C
C FAC1
C
0105      CALL VFILL(A3,A3+1,1,NHLFWD-1)
0106      CALL VDIV(A1,1,A3,1,A2,1,NHLFWD)
C
C FAC2
C
```

```

0107      CALL APWR
0108      CALL APPUT(DZINV,A15,1,2)
0109      CALL APWD
0110      CALL VSMUL(A1,1,A15,A3,1,NHLFWD)
      C
      C K
      C
0111      CALL VINT(A3,1,A5,1,NHLFWD)
      C
      C DELTA
      C
0112      CALL VSUB(A3,1,A5,1,A4,1,NHLFWD)
      C
      C NLEAD
      C
0113      CALL VSADD(A5,1,A8+IADD,A5,1,NHLFWD)
0114      CALL VSUB(A6+IADD,0,A5,1,A5,1,NHLFWD)
0115      CALL VTSADD(A5,1,2049,A5,1,NHLFWD)
0116      CALL VFIX(A5,1,A5,1,NHLFWD)
0117      CALL APWR
0118      CALL APGET(NLEAD,A5,NHLFWD,1)
0119      CALL APWD
0120      IF(IWRITW(NHLFWD,NLEAD,IBLKNL,IRD).LT.0)STOP'NLWRIT ERR'
0122      IBLKNL=IBLKNL+NINLD
      C
      C GEN OPERATOR FROM INTERMEDIATE VALUES.
      C
0123      DO 40 I=1,5
0124      AS=A(I)
0125      ACON=AC(I)
0126      CALL VTSADD(A4,1,ACON,AS,1,NHLFWD)
0127      CALL VDIV(A3,1,AS,1,AS,1,NHLFWD)
0128      CALL VTSADD(AS,1,2050,A9,1,NHLFWD)
0129      CALL VMUL(AS,1,A9,1,AS,1,NHLFWD)
0130      CALL VSQRT(AS,1,AS,1,NHLFWD)
0131      CALL VMUL(AS,1,A2,1,AS,1,NHLFWD)
0132      40 CONTINUE
      C
      C OP1
      C
0133      CALL VNEG(A11,1,A2,1,NHLFWD)
      C
      C OP2
      C
0134      CALL VTSMUL(A11,1,2050,A3,1,NHLFWD)
0135      CALL VSUB(A12,1,A3,1,A3,1,NHLFWD)
      C
      C OP3
      C
0136      CALL VTSMUL(A12,1,2050,A9,1,NHLFWD)
0137      CALL VSUB(A11,1,A9,1,A9,1,NHLFWD)
0138      CALL VSUB(A13,1,A9,1,A9,1,NHLFWD)
      C
      C OP4

```

```

C
0139 CALL VTSMUL(A13,1,2050,A10,1,NHLFWD)
0140 CALL VSUB(A12,1,A10,1,A10,1,NHLFWD)
0141 CALL VSUB(A14,1,A10,1,A10,1,NHLFWD)
C
C OP5
C
0142 CALL VTSMUL(A14,1,2050,A11,1,NHLFWD)
0143 CALL VSUB(A13,1,A11,1,A11,1,NHLFWD)
0144 CALL VSUB(A15,1,A11,1,A11,1,NHLFWD)
C
C WRITE OUT OPERATOR
C
0145 CALL APWR
0146 IAD=1
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 APWD
0157 IF(IWRITW(NBUF2,OPVAL,IBLKOP,IRD).LT.0)STOP'OPWRIT ERR'
0159 IBLKOP=IBLKOP+IOPINC
0160 30 CONTINUE
0161 20 CONTINUE
0162 CALL CLOSEC(IRD)
0163 STOP'NORMAL TERMINATION'
0164 END

```

```

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=% ERROR
C
0002 INTEGER*2 NRANGE(512)
0003 DATA IN/0/
0004 IN=IN+1
0005 IF(IFLG.LE.0.AND.IN.EQ.1)READ(1,1000)D
0007 1000 FORMAT(F10.0)
0008 NOP=0
0009 NRANGE(IPOS)=ITMIN
0010 IPOS=IPOS+1
0011 I=ITMIN-1
0012 IF(IFLG.LE.0)N=I*(D/50.0)
0014 10 NOP=NOP+1
0015 IF(IFLG.GT.0)GOTO 20
0017 N=(I+N)*(D/50.0)
0018 IF(N.LE.0)N=1
0020 I=I+2*N-1
0021 GOTO 30
0022 20 I=I+NST
0023 30 NRANGE(IPOS)=I
0024 IPOS=IPOS+1
0025 IF(IPOS.GT.512)STOP'TOO MANY OPERATORS'
0027 IF(I.LT.ITMAX)GOTO 10
0029 RETURN
0030 END

```

Kirchhoff Migration :- MPKMIG

Input File.....DK2:MPKMIG.DAT

Input ParametersREAD(1,1000)NTRACE,NO,NSAMP,NHLFWD1000 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,TSTEP1001 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

```
C
C M J POULTER DEC 80
C
C MPKMIG-THIS IS A KIRCHHOFF MIGRATION
C PROGRAM WHICH USES THE OPERATOR
C DESIGNED IN MPOGEN TO PERFORM
C CONVOLUTIONAL MIGRATION
C
0001 REAL*8 FSPECR,FSPECW,FSPECO
0002 INTEGER*2 NLEAD(512),NRANGE(512),A0,A1,A2,
      #A0ST,ATRI,AOP,ASUM,NLEADI(512)
0003 REAL*4 FBUF(3),OPBUF(2560),TRACE(2560)
0004 DATA DEV/3RRK /
0005 DATA A0,A1,A2,ASUM/0,2048,4096,6704/
C
C SET I/O CALLS
C
0006 IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0008 IRD=IGETC()
0009 IWRT=IGETC()
0010 IROP=IGETC()
0011 CALL APINIT
0012 CALL ASSIGN(1,'DK2:MPKMIG.DAT',14)
C
C READ IN DATA
C
0013 READ(1,1000)NTRACE,N0,NSAMP,NHLFWD
0014 1000 FORMAT(4I5)
0015 READ(1,1000)ISTART,NSTART,NSTOP,NINC
0016 READ(1,1001)XSTEP,TSTEP
0017 1001 FORMAT(2F10.0)
0018 READ(1,1002)FBUF
0019 1002 FORMAT(3A4)
0020 CALL IRAD50(12,FBUF,FSPECO)
0021 READ(1,1002)FBUF
0022 CALL IRAD50(12,FBUF,FSPECR)
0023 READ(1,1002)FBUF
0024 CALL IRAD50(12,FBUF,FSPECW)
C
C SET UP CONSTANTS
C
0025 NBLKR=NSAMP/128
0026 NFIL0=NBLKR*NTRACE+1
0027 NW=NHLFWD-1
0028 NW2=NW+2
0029 NA=NTRACE+NHLFWD
0030 NWIDTH=2*NHLFWD-1
C
C OPEN UP I/O FILES
C
0031 IF(LOOKUP(IROP,FSPECO).LT.0)STOP'LOKKUP ERR'
0033 IF(LOOKUP(IRD,FSPECR).LT.0)STOP'LOOKUP ERROR'
0035 IF(IENTER(IWRT,FSPECW,NFIL0).LT.0)STOP'ENTER ERROR'
```

```
C
C GET OPERATOR CONSTANTS
```

```
C
```

```
0037 IF(IREADW(1,NOP,0,IROP).LT.0)STOP'READ ERROR'
0039 IF(IREADW(NOP+1,NRANGE,1,IROP).LT.0)STOP'READ ERR'
0041 NBRNG=(NOP+255)/256
0042 NBUF=NHLFWD*5
0043 NBUF2=2*NBUF
0044 NBUFAP=5*NOP
0045 IOPINC=(NBUF+127)/128
0046 NINLD=(NHLFWD+255)/256
0047 NBLD=NOP*NINLD
0048 INLDST=1+NBRNG
0049 INOPST=INLDST+NBLD
0050 IBLKO=1
0051 ISTOP=NRANGE(NOP+1)
```

```
C
```

```
C START MAIN LOOP
```

```
C
```

```
0052 DO 10 ITR=NSTART,NSTOP,NINC
0053 IBLK=((ITR-1)*NBLKR)+1
```

```
C
```

```
C READ IN OUTPUT TRACE
```

```
C
```

```
0054 IF(IREADW(2*NSAMP,TRACE,IBLK,IRD).LT.0)STOP'READ ERR2'
0056 CALL APPUT(TRACE,A0,NSAMP,2)
0057 CALL APWD
0058 CALL VCLR(A0+ISTART,1,ISTOP-ISTART)
0059 IMIN=MAX0(NW2-ITR,1)
0060 IMAX=MIN0(NA-ITR,NWIDTH)
0061 IBLKR=((ITR-NW+IMIN-2)*NBLKR)+1
```

```
C
```

```
C LOOP ON OPERATOR APPLICATION
```

```
C
```

```
0062 DO 20 IT=IMIN,IMAX
0063 IOP=IT-NW
0064 IF(IOP.LT.1)IOP=2-IOP
0066 A0ST=A0+ISTART
0067 IF(IREADW(2*NSAMP,TRACE,IBLKR,IRD).LT.0)STOP'READ ERR3'
0069 IBLKR=IBLKR+NBLKR
0070 CALL APPUT(TRACE,A1,NSAMP,2)
0071 LSTOR=1
0072 NSTOR=1
```

```
C
```

```
C GET OPERATOR AND LEADIN VALUES FROM FILE
```

```
C
```

```
0073 IBLKLD=INLDST
0074 IBLKOP=INOPST
0075 DO 40 J=1,NOP
0076 IF(IREADW(NHLFWD,NLEADI,IBLKLD,IROP).LT.0)STOP'READ ERR4'
0078 IF(IREADW(NBUF2,TRACE,IBLKOP,IROP).LT.0)STOP'READ ERR4'
0080 IBLKLD=IBLKLD+NINLD
0081 IBLKOP=IBLKOP+IOPINC
0082 IOFF=IOP
```

```
0083     ILD=IOP
0084     DO 50 JJ=1,5
0085     OPBUF(LSTOR)=TRACE(IOFF)
0086     LSTOR=LSTOR+1
0087     IOFF=IOFF+NHLFWD
0088     50 CONTINUE
0089     NLEAD(NSTOR)=NLEADI(ILD)
0090     NSTOR=NSTOR+1
0091     40 CONTINUE
0092     CALL APPUT(OPBUF,A2,NBUFAP,2)
0093     CALL APWD
C
C DO CONVOLUTIONAL MIGRATION
C
0094     DO 60 ILP=1,NOP
0095     ATRI=NLEAD(ILP)+A1
0096     LIMIT=NRANGE(ILP+1)
0097     AOP=A2+(5*(ILP-1))
0098     NRES=LIMIT-A0ST+1
C
C DO CONVOLVE
C
0099     CALL CONV(ATRI,1,AOP,1,ASUM,1,NRES,5)
0100     CALL VADD(A0ST,1,ASUM,1,A0ST,1,NRES)
0101     A0ST=LIMIT+1
0102     60 CONTINUE
0103     20 CONTINUE
C
C GET MIGRATED TRACE OUT OF AP
C
0104     CALL APWR
0105     CALL APGET(TRACE,A0,NSAMP,2)
0106     CALL APWD
C
C WRITE MIGRATED TRACE TO DISK
C
0107     IF(IWRTW(2*NSAMP,TRACE,IBLKO,IWRT).LT.0)STOP'WRIT WRR'
0108     IBLKO=IBLKO+NBLKR
0109     10 CONTINUE
0110     CALL CLOSEC(IWRT)
0111     STOP'NORMAL TERMINATION'
0112     END
0113
```

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

```

0001 VIRTUAL S0(23552),S1(23552)
0002 DIMENSION DUM(2048)
0003 DIMENSION FBUFR(3),FBUFW(3),FSTAP(23)
0004 REAL*8 FSPECR,FSPECW
0005 DATA K1M,K1,K2,KTOP,IA,IB,IB1,IB2,IC,IC1,IDI
1/-1.1,2,8191,0,2048,2049,2050,4096,4097,6145/
0006 DATA DEV/3RDK /
C
0007 CALL ASSIGN(1,'DK2:ANINV.DAT',13)
0008 IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0012 IDCHR=IGETC()
0013 IDCHW=IGETC()
C
0014 C READ IN REQUIRED INPUT PARAMETERS
0015 READ(1,1000) L,NCHAN,M,NTAP,INDEN
0016 1000 FORMAT(2I5,5X,3I5)
0017 READ(1,1100) FBUFR,FBUFW
0018 1100 FORMAT(2(3A4))
C VALIDATE DATA
0019 C NREAD IS THE NUMBER OF BLOCKS READ IN/TRACE
0020 NREAD=L/128
0021 C STOP IF L IS NOT AN INTEGER NUMBER OF BLOCKS
0022 IF(L.NE.128*NREAD)STOP'L NOT INTEGER NO. OF BLOCKS'
0023 C L2 IS NUMBER OF WORDS READ/TRACE
0024 L2=2*L
0025 C IAL IS POINTER NEEDED FOR COSINE TAPERING
0026 IAL=IA+L-1
0027 C FIND L2I, INTEGER POWER OF TWO G.E. THAN L.
0028 L2I=2
0029 DO 5 I=1,1000
0030 IF(L2I.GE.L) GOTO 10
0031 L2I=2*L2I
0032 5 CONTINUE
0033 C STOP IF L2I IS GREATER THAN 2048
0034 10 IF(L2I.GT.2048)STOP'L2I.GT.2048'
0035 L2I21=L2I/2-1
0036 C STOP IF M.GT.23
0037 IF(M.GT.23)STOP'M.GT.23'
0038 C LM IS NUMBER OF INTERPOLATED SAMPLES/TRACE
0039 LM=L*M
0040 C STOP IF LM.GT.23552
0041 IF(LM.GT.23552)STOP'LM.GT.23552'
0042 C LM2 IS THE TOTAL NUMBER OF WORDS WRITTEN OUT/INTERPOLATED TRACE
0043 LM2=2*LM
0044 C NWRITE IS THE NUMBER OF BLOCKS WRITTEN/TRACE
0045 NWRITE=LM/128
0046 C FST0 IS ADDRESS NEEDED FOR DISK READ INTO S0(1)
0047 FST0=ADGET(S0(1))
0048 C FST1 IS ADDRESS NEEDED FOR DISK WRITE FROM S1
0049 FST1=ADGET(S1(1))
0050 C FSTAP IS ARRAY OF ADDRESSES NEEDED FOR AP TRANSFERS
0051 KL=1
0052 DO 20 K=1,M
0053 FSTAP(K)=APGAD(S0(KL))

```

```

0042      KL=KL+L
0043  20   CONTINUE
C OPEN READ FILE
0044      CALL IRAD50(12,FBUFR,FSPECR)
0045      IF(LOOKUP(IDCHR,FSPECR).LT.0)STOP 'LOOKUP ERROR'
C OPEN WRITE FILE
0047      CALL IRAD50(12,FBUFW,FSPECW)
0048      IF(IENTER(IDCHW,FSPECW,NWRITE*NCHAN+1).LT.0)STOP 'ENTER ERROR'

```

C
C=====

C PART

C=====

C-----
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
0050      CALL APINIT
C PUT 1.0/FLOAT(M*L2I) IN IC AND 1.0/FLOAT(NTAP) IN KTOP
0051      CALL APWR
0052      CALL APPUT(1.0/FLOAT(M*L2I),IC,K1,K2)
0053      CALL APPUT(1.0/FLOAT(NTAP),KTOP,K1,K2)
0054      CALL APWD
C MULTIPLY BY 2*PI FROM TM
0055      CALL VTSMUL(IC,K1,2317,IC,K1,K1)
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.
0056      CALL VRAMP(IC,IC,IA,K1,L2I21)
0057      CALL CVEXP(IA,K1,IC1,K2,L2I21)
C FORM COSINE TAPER NTAP LONG STARTING AT ID1
0058      CALL VCLR(ID1,K1,K1)
0059      CALL VTSADD(ID1,K1,2306,ID1,K1,K1)
0060      CALL VTSMUL(KTOP,K1,2306,KTOP,K1,K1)
0061      CALL VRAMP(ID1,KTOP,ID1,K1,NTAP)
0062      CALL VCOS(ID1,K1,ID1,K1,NTAP)
0063      CALL VTSADD(ID1,K1,2049,ID1,K1,NTAP)
0064      CALL VTSMUL(ID1,K1,2327,ID1,K1,NTAP)

```

C=====

C PART

C=====

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 LM.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
0065      JBLKW=1
0066      JBLKR=1
C

```

```

C NOW ITERATE THROUGH CHANNELS INTERPOLATING THE TRACES.
0067 DO 800 JCHAN=1,NCHAN
C READ IN TRACE FROM UNIT 2
0068 IF(IREADA(IDCHR,L2,JBLKR,FST0).LT.0)STOP'READA ERROR'
0070 JBLKR=JBLKR+NREAD
C CLEAR A AND TRANSFER TRACE INTO IT
0071 CALL VCLR(IA,K1,L2I)
0072 CALL APWR
0073 CALL IWAIT(IDCHR)
0074 CALL APPUTA(IA,L,K2,FSTAP(1))
0075 CALL APWD
C FIND MEAN VALUE OF TRACE AND PLACE IN AP(KTOP)
0076 CALL MEANV(IA,K1,KTOP,L)
C SUBTRACT MEAN FROM TRACE
0077 CALL VNEG(KTOP,K1,KTOP,K1,K1)
0078 CALL VSADD(IA,K1,KTOP,IA,K1,L)
C MULTIPLY ENDS OF DATA BY COSINE TAPER
0079 CALL VMUL(IA,K1,ID1,K1,IA,K1,NTAP)
0080 CALL VMUL(IAL,K1M,ID1,K1,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)
0081 IF(INDEN.LT.0)GOTO 350
0083 CALL RMSQV(IA,K1,KTOP,L)
0084 IF(INDEN.GT.0)CALL VSQ(KTOP,K1,KTOP,K1,K1)
C PLACE 1.0 FROM TM IN IC
0085 CALL VCLR(IC,K1,K1)
0087 CALL VTSADD(IC,K1,2049,IC,K1,K1)
C USE THIS VALUE TO CREATE RECIPROCAL IN KTOP
0088 CALL VDIV(KTOP,K1,IC,K1,KTOP,K1,K1)
C MULTIPLY TRACE BY RECIPROCAL
0089 CALL VSMUL(IA,K1,KTOP,IA,K1,L)
C TRANSFER MODIFIED TRACE BACK TO S0
0090 350 CALL APWR
0091 CALL APGETA(IA,L,K2,FSTAP(1))
0092 CALL APWD
0093 IF(M.EQ.1)GOTO 500
C TAKE TRANSFORM OF TRACE AND PUT IN B
0094 CALL RFFT(B,IA,IB,L2I,K1)
0095 CALL RFFTSC(IB,L2I,K2,K1)
C ITERATE FROM 2 TO M
0097 DO 400 K=2,M
C MULTIPLY TRANSFORM BY COMPLEX EXPONENTIAL VECTOR
0098 CALL CVMUL(IB2,K2,IC1,K2,IB2,K2,L2I2I)
C MOVE B TO A AND TAKE IN PLACE IFFT
0099 CALL VMOV(IA,K1,IB,K1,L2I)
0100 CALL RFFT(IA,L2I,K1M)
C TRANSFER SHIFTED TRACE BACK TO S0
0101 CALL APWR
0102 CALL APGETA(IA,L,K2,FSTAP(K))
0103 CALL APWD
0104 400 CONTINUE
C INTERPOLATED TRACE IS NOW IN S0 IN SCRAMBLED ORDER.
C UNSCRAMBLE INTO S1 SO THAT RECORD IS IN CORRECT TIME SEQUENCE.

```

```

0105 500 J1=1
0106 DO 700 JL=1,L
0107 J0=JL
0108 DO 600 JM=1,M
0109 S1(J1)=S0(J0)
0110 J0=J0+L
0111 J1=J1+1
0112 600 CONTINUE
0113 700 CONTINUE
C NOW WRITE OUT INTERPOLATED TRACE ONTO DISK
0114 IF(IWRITA(IDCHW,LM2,JBLKW,FST1).LT.0)STOP'WRITA ERROR'
0116 JBLKW=JBLKW+NWRITE
0117 CALL IWAIT(IDCHW)
0118 800 CONTINUE
0119 CALL CLOSEC(IDCHR)
0120 CALL CLOSEC(IDCHW)
0121 STOP
0122 END

```

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

```
C
C THIS IS A PROGRAM WHICH INTERROGATES
C A DATA FILE TO GIVE ALL OF ITS
C PROGRAMMING PARAMETERS TO THE USER
C
0001 REAL*8 FSPECR,RUNITS(2),SOURCE(4)
0002 REAL*4 FNAMR(3)
0003 INTEGER*2 HBLK(256)
0004 LOGICAL*1 HSBLK(512),NOCHAN,IGCODE,IUNITS,ISAMP
0005 EQUIVALENCE (SROFF,HSBLK(21)),(SLSPAC,HSBLK(29)),
%(STSPAC,HSBLK(33)),(SDEPT,HSBLK(37)),(RDEPT,HSBLK(41)),
%(RSPAC,HSBLK(25)),(NOCHAN,HSBLK(12)),(IGCODE,HSBLK(19)),
%(IUNITS,HSBLK(20)),(ISCODE,HSBLK(45)),(IBFREE,HSBLK(47)),
%(NCHAN,HSBLK(13)),(NBEG,HSBLK(15)),(NFIN,HSBLK(17)),
%(ISAMP,HSBLK(9)),(HBLK(1),HSBLK(1))
0006 DATA DEV/3RRK /,RUNITS(1)/'METRES '/,RUNITS(2)/' FEET '/
0007 DATA SOURCE(1)/'AIRGUNS '/,SOURCE(2)/'EXPLOSIV'/,
%SOURCE(3)/'VIBROSEI'/,SOURCE(4)/'WEIGHTS '/
C
C GET NAME OF INPUT FILE
C
0008 WRITE(7,5)
0009 5 FORMAT(' ENTER NAME OF FILE TO BE EXAMINED:',5)
0010 READ(5,10)FNAMR
0011 10 FORMAT(3A4)
0012 CALL IRAD50(12,FNAMR,FSPECR)
C
C SET UP I/O PARAMETERS
C
0013 ICHR=IGETC()
0014 IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
0016 ILEN=LOOKUP(ICHR,FSPECR)
0017 IF(ILEN.LT.0)STOP'LOOKUP ERR'
0019 IF(IREADW(256,HSBLK,0,ICHR).LT.0)STOP'READ ERROR'
C
C DECODE HEADER AND PRINT OUT
C
0021 WRITE(7,20)FNAMR,ILEN
0022 20 FORMAT(' FILE ',3A4,' TO BE ANALYSED, LENGTH=',I5,' BLOCKS')
0023 WRITE(7,30)(HSBLK(I),I=1,8)
0024 30 FORMAT(' FILE NO:',3A1,' TAPE NO:',5A1)
0025 WRITE(7,40) HSBLK(9),HSBLK(11),HSBLK(12)
0026 40 FORMAT(' SAMPLING INTERVAL SET AT ',I1,' MSEC',/,
%' AND RECORDING LASTED ',I2,' SECS',/,
%' ON ',I2,' ACTIVE CHANNELS')
0027 GOTO (50,60,60,80)IGCODE
0028 50 WRITE(7,110)
0029 GOTO 100
0030 60 WRITE(7,120)
0031 GOTO 100
0032 70 WRITE(7,130)
0033 GOTO 100
0034 80 WRITE(7,140)
0035 100 CONTINUE
```

```

0036 110 FORMAT(' THE FILE CONTAINS A SHOT POINT GATHER')
0037 120 FORMAT(' THE FILE CONTAINS A COMMON MIDPOINT GATHER')
0038 130 FORMAT(' THE FILE CONTAINS A STACKED TRACE')
0039 140 FORMAT(' THE FILE CONTAINS AN UNSTACKED TRACE')
0040 WRITE(7,90)NCHAN,NBEG,NFIN
0041 90 FORMAT(' CONSISTING ',I2,' CHANNELS,STARTING AT SAMPLE NO:'
      % ,I5,/, ' AND ENDING WITH SAMPLE NO:'I5)
0042 WRITE(7,150)SOURCE(ISCODE),RUNITS(IUNITS)
0043 150 FORMAT(' TYPE OF SOURCE= ',A8,/,
      % ' AND THE UNITS OF LENGTH USED= ',A8)
0044 WRITE(7,160)RDEPT,SDEPT,SROFF,RSPEC,SLSPAC,STSPAC
0045 160 FORMAT(' RECIEVER DEPTH=',F10.2,/,
      % ' SOURCE DEPTH=',F10.2,/,
      % ' SOURCE-RECIEVER OFFSET=',F10.2,/,
      % ' RECIEVER SPACING=',F10.2,/,
      % ' SHOT SPACING=',F10.2,/,
      % ' SHOT REPITITION RATE=',F10.2,' SECS')
0046 WRITE(7,170)(HBLK(I),I=51,254)
0047 170 FORMAT(' USER INFO PUT INTO HEADER BLOCK',/,
      % ' IS GIVEN BELOW',/,3(1X,80A1,/,))
0048 IF(IBFREE.EQ.0)GOTO 999
C
C PROCESS DECODE AREA
C
0050 WRITE(7,180)
0051 180 FORMAT(/,/, ' THE FOLLOWING PROCESSES HAVE BEEN
      % PERFORMED ON THE DATA',/)
0052 IPOS=0
0053 190 CONTINUE
0054 GOTO(200,210,220,230,240)HBLK(129+IPOS)
C
C PRE STACK DECODE
C
0055 200 IPOS=IPOS+1
0056 ICNT=HBLK(129+IPOS)
0057 IPOS=IPOS+1
0058 WRITE(7,300)ICNT
0059 300 FORMAT(' PRE-STACK PROCESSING CONSISTING OF',/,
      % I3,' OPERATIONS IN THE FOLLOWING ORDER')
0060 DO 310 J=1,ICNT
0061 ICD=HBLK(129+IPOS)
0062 IPOS=IPOS+1
0063 GOTO(311,312,313,314,315,316,317,318,319,320,321)ICD
0064 311 WRITE(7,400)
0065 400 FORMAT(' TRACE EDITING')
0066 GOTO 310
0067 312 WRITE(7,401)
0068 401 FORMAT(' POLARITY REVERSAL')
0069 GOTO 310
0070 313 WRITE(7,402)
0071 402 FORMAT(' EXP(0.2T) AMP RECOVERY')
0072 GOTO 310
0073 314 WRITE(7,403)
0074 403 FORMAT(' T*EXP(0.2T) AMP RECOVERY')

```

```

0076      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      310 CONTINUE
0097      IF(IPOS.LT.IBFREE)GOTO 190
0098      GOTO 999

```

C
C STACKING DECODE

```

0100      210 WRITE(7,420)
0101      420 FORMAT(/,' NMO CORRECTION AND CDP STACK')
0102      IPOS=IPOS+1
0103      NCHST=HBLK(129+IPOS)
0104      IPOS=IPOS+1
0105      NVST=HBLK(129+IPOS)
0106      IPOS=IPOS+1
0107      NLYRST=HBLK(129+IPOS)
0108      IPOS=IPOS+1
0109      MST=HBLK(129+IPOS)
0110      IPOS=IPOS+1
0111      INTSW=HBLK(129+IPOS)
0112      IPOS=IPOS+1
0113      WRITE(7,421)NCHST,MST
0114      421 FORMAT(' WITH',13,' CHANNELS AT A LEVEL OF',/,
#14,' TIMES INTERPOLATION')
0115      WRITE(7,422)NVST,NLYRST,INTSW
0116      422 FORMAT(I5,' VELOCITY ANALYSES WERE USED EACH WITH',/,
#13,' LAYERS AND THE INTERPOLATION SWITCH=',11)
0117      IF(IPOS.LT.IBFREE)GOTO 190
0118      GOTO 999

```

C
C POST STACK DECODE

```

0120      220 IPOS=IPOS+1
0121      ICNT=HBLK(129+IPOS)
0122      IPOS=IPOS+1

```

```
0120 WRITE(7,430)ICNT
0124 430 FORMAT(' POST STACK PROCESSING CONSISTING OF ',/,
             %13,' OPERATIONS IN THE FOLLOWING ORDER')
0125 DO 330 J=1,ICNT
0126 ICD=HBLK(129+IPOS)
0127 IPOS=IPOS+1
0128 GOTO(331,332,333,334,335,336,337,338,339,340)ICD
0129 331 WRITE(7,400)
0132 GOTO 330
0131 332 WRITE(7,402)
0132 GOTO 330
0133 333 WRITE(7,403)
0134 GOTO 330
0135 334 WRITE(7,404)
0136 GOTO 330
0137 335 WRITE(7,405)
0138 GOTO 330
0139 336 WRITE(7,406)
0140 GOTO 330
0141 337 WRITE(7,407)
0142 GOTO 330
0143 338 WRITE(7,408)
0144 GOTO 330
0145 339 WRITE(7,409)
0146 GOTO 330
0147 340 WRITE(7,410)
0148 330 CONTINUE
0149 IF(IPOS.LT.IBFREE)GOTO 190
0151 GOTO 999
```

```
C
C MIGRATION
C
```

```
0152 230 WRITE(7,450)
0153 450 FORMAT(' MIGRATION')
0154 IPOS=IPOS+1
0155 IF(IPOS.LT.IBFREE)GOTO 190
0157 GOTO 999
```

```
C
C THREE TRACE MIX
C
```

```
0158 240 WRITE(7,460)
0159 460 FORMAT(' THREE TRACE MIX')
0160 IPOS=IPOS+1
0161 IF(IPOS.LT.IBFREE)GOTO 190
```

```
C
0163 999 STOP ' NORMAL TERMINATION'
0164 END
```

Synthetic CMP Gathers :- ANSEI

Input file.....DK1:ANINS.DAT

Output file.....DK2:ANOOTS.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)

1200 FORMAT(2F10.0)

TOLYR....Two-way travel time of reflection on zero offset
trace

VLYR.....RMS velocity down to the event

```

0001 REAL*8 FSPEC
0002 DIMENSION SEISM(2048),T0LYR(99),VLYR(99),WAVE(500),CONST(4)
0003 DATA K1N,K0,K1,K2,K4,KAMP,KSN,KTWOPI,KSEED,KU/-1,0,1,2,4,8187,
18188,8189,8190,8191/
0004 DATA NWDBLK,DEV,FSPEC/256,3RDK,12RDK2ANOUTSDAT/
0005 CALL ASSIGN(1,'DK1:ANINS.DAT',13)
0006 IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0007 IDCH=IGETC()
0008 READ(1,1000) L,NCHAN,NSTART,NLYR
0009 1000 FORMAT(4I5)
0010 READ(1,1100) XSTART,XSTEP,FSAMP,FRICK,FNOISE,SN
0011 1100 FORMAT(6F10.0)
0012 READ(1,1200) (T0LYR(I),VLYR(I),I=1,NLYR)
0013 1200 FORMAT(2F10.0)
0014 NBLTR=IFIX(2.0*FLOAT(L)/FLOAT(NWDBLK)+0.5)
0015 L=NWDBLK*NBLTR/2
0016 NBLTOT=NBLTR*NCHAN
0017 IF(IENTER(IDCH,FSPEC,NBLTOT+1).LT.0)STOP'ENTER ERROR'
0018 JBLOCK=1
0019 CALL RICKER(FSAMP,FRICK,NBEGIN,NWAVE,WAVE)
0020 L2INT=2
0021 DO 5 K=1,11
0022 IF(L2INT.GE.L) GOTO 10
0023 L2INT=2*L2INT
0024 5 CONTINUE
0025 CONST(1)=1.0/SN
0026 10 CONST(2)=8.0*ATAN(1.0)
0027 CONST(3)=0.2510638
0028 CONST(4)=1.0
0029 CALL APINIT(IDUM,IDUM,IDUM)
0030 CALL APPUT(CONST,KSN,K4,K2)
0031 CALL APWD
0032 ISEISM=1
0033 IWAVE=ISEISM+L2INT+NWAVE-1
0034 IWAVER=IWAVE+NWAVE-1
0035 IRAND=IWAVE+NWAVE
0036 NRAND=L2INT*FNOISE/FSAMP
0037 INOISE=IRAND+NRAND
0038 INOIS2=INOISE+2
0039 INOIS3=INOISE+3
0040 MRAND2=2*NRAND
0041 CALL APPUT(WAVE,IWAVE,NWAVE,K2)
0042 CALL APWD
0043 XJCHAN=XSTART
0044 NOFFS=NSTART-NBEGIN
0045 DO 40 JCHAN=1,NCHAN
0046 XSQ=XJCHAN**2
0047 XJCHAN=XJCHAN+XSTEP
0048 CALL VCLR(K0,K1,IWAVE)
0049 DO 30 JLYR=1,NLYR
0050 NT=IFIX(FSAMP*SQRT(T0LYR(JLYR)**2+XSQ
0051 *VLYR(JLYR)**2)+0.5)-NOFFS
0052 CALL VADD(NT,K1,KU,K1,NT,K1,K1)
0053 30 CONTINUE
0054
0055

```

```

56 CALL CONV(ISEISM,K1,IWAVER,K1N,ISEISM,K1,L2INT,NWAVE)
57 CALL VRAND(KSEED,IRAND,K1,NRAND)
58 CALL VSMUL(IRAND,K1,KTWOPI,IRAND,K1,NRAND)
59 CALL VCLR(INOISE,K1,L2INT)
60 CALL VCOS(IRAND,K1,INOIS2,K2,NRAND)
61 CALL VSIN(IRAND,K1,INOIS3,K2,NRAND)
62 CALL RFFT(INOISE,L2INT,K1N)
63 CALL RMSQV(INOISE,K1,KAMP,L2INT)
64 CALL VDIV(KAMP,K1,KSN,K1,KAMP,K1,K1)
65 CALL VSMUL(INOISE,K1,KAMP,INOISE,K1,L2INT)
66 CALL VADD(ISEISM,K1,INOISE,K1,ISEISM,K1,L2INT)
67 CALL APWR
68 CALL IWAIT(IDCH)
69 CALL APGET(SEISM(1),ISEISM,L,K2)
70 CALL APWD
71 IF(IWRITE(2*L,SEISM,JBLOCK,IDCH).LT.0)STOP'WRITE ERROR'
72 JBLOCK=JBLOCK+NBLTR
73
74 CONTINUE
75 CALL CLOSEC(IDCH)
76 STOP
77 END

```

```

001 SUBROUTINE RICKER(FSAMP,FRICK,NBEGIN,NWAVE,WAVE)
002 IMPLICIT INTEGER*2(I-N)
003 DIMENSION WAVE(500)
004 N=IFIX(FSAMP/FRICK)
005 NWAVE=2*N+1
006 NBEGIN=N+1
007 WAVE(NBEGIN)=1.
008 IF(N.EQ.0) RETURN
009 IPLUS=NBEGIN
010 IMINUS=NBEGIN
011 X=0.
012 DELX=3.14159265*FRICK/FSAMP
013 DO 10 I=1,N
014 IPLUS=IPLUS+1
015 IMINUS=IMINUS-1
016 X=X+DELX
017 WAVE(IPLUS)=0.5*(1.+COS(X))*(1.-2.*X**2)*EXP(-X**2)
018 WAVE(IMINUS)=WAVE(IPLUS)
019 CONTINUE
020 RETURN
021 END
022

```

Synthetic Sections :- MPSYNS

Input file.....DK2:MPSYND.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

```

C
C PROGRAM SYNTH...MODIFIED FROM
C A PROGRAM BY C GODBOLD BY M J POULTER
C NOV 80
C
0001 REAL*8 FSPECW
0002 DIMENSION X0(5),X1(5),X2(5),DIP(5),SINDIP(5),COSDIP(5),
$XP(10),ZP(10),FBUF(3)
0003 COMMON/AA/TRACE(2048),TSTEP,NSAMP,IFLAG1
0004 COMMON/BB/AUX(2048)
0005 COMMON/CC/VEL(11),THICKN(10),NLAYER
0006 EQUIVALENCE(DIP(1),COSDIP(1))
0007 DATA KN1,K0,K1,K2,PI/-1,0,1,2,3.141593/
0008 DATA DEV/3RRK /
0009 CALL ASSIGN(1,'DK2:MPSYND.DAT',14)
0010 IWRT=IGETC()
0011 IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
0012 IBLK=1
C
C READ IN PARAMETERS
C
0014 READ(1,1000)TSTEP,XSTEP,FRICK,FNOISE,AMP
0015 1000 FORMAT(10F10.0)
0016 READ(1,1001)NTRACE,NSTART,NSTOP,NINC,N0,NSAMP,IFLAG1
0017 1001 FORMAT(10I5)
0018 READ(1,1001)NPLANE,NPOINT,NLAYER
0019 IF(NLAYER.LE.0)GOTO 10
0021 READ(1,1002)(VEL(I),THICKN(I),I=1,NLAYER)
0022 10 READ(1,1002)VEL(NLAYER+1)
0023 1002 FORMAT(2F10.0)
0024 IF(NPLANE.LE.0)GOTO 20
0026 READ(1,1003)(X0(I),X1(I),X2(I),DIP(I),I=1,NPLANE)
0027 1003 FORMAT(4F10.0)
0028 DO 30 J=1,NPLANE
0029 SINDIP(J)=SIN(DIP(I)*PI/180.0)
0030 30 COSDIP(J)=SQRT(1.0-SINDIP(I)**2)
0031 20 IF(NPOINT.LE.0)GOTO 40
0033 READ(1,1002)(XP(I),ZP(I),I=1,NPOINT)
0034 40 NVEL=NLAYER+1
0035 READ(1,1004)FBUF
0036 1004 FORMAT(3A4)
0037 CALL IRAD50(12,FBUF,FSPECW)
0038 NBLKR=NSAMP/128
0039 NBLKF=(NTRACE*NBLKR)+1
0040 IF(IENTER(IWRT,FSPECW,NBLKF).LT.0)STOP'ENTER ERROR'
C
C CONVERT TSTEP TO MSEC
C
0042 TSTEP=INT(TSTEP*1000.0+0.5)
C
C EVALUATE RICKER WAVELET
C
0043 CALL RICKER(TSTEP,FRICK,N)
C

```

C SET UP AP AND GET CONSTANTS

C

```

0044 CALL APINIT
0045 KNSAMP=NSAMP
0046 KNOISE=2
0047 IF(AMP.LT.0.0)GOTO 50
0049 60 KNOISE=KNOISE*2
0050 IF(KNOISE.LT.KNSAMP)GOTO 60
0052 KRAND=FNOISE*TSTEP*KNOISE/2000.0
0053 50 KWAVE=2*N+1
0054 KWAVE3=KWAVE+3
0055 KB=N
0056 KA=8192-KWAVE3
0057 KC=KB+KNSAMP
0058 KD=8189
0059 AUX(KWAVE+1)=2.0*PI
0060 AUX(KWAVE+2)=0.2510638
0061 AUX(KWAVE3)=AMP
0062 CALL APPUT(AUX,KA,KWAVE3,2)

```

C

C START TRACE GENERATION LOOP

C

```

0063 DO 100 L=1,NTRACE
0064 X=(L-1)*XSTEP
0065 CALL VCLR(0,1,KNSAMP)
0066 CALL APWR
0067 CALL APGET(TRACE,0,KNSAMP,2)
0068 CALL APWD

```

C

C EVALUATE ARRIVAL FOR EACH REFLECTOR

C

```

0069 IF(NPLANE.LE.0)GOTO 110
0071 DO 200 J=1,NPLANE
0072 IF(X.LT.X1(J).OR.X.GT.X2(J))GOTO 200
0074 XA=ABS(X-X0(J))*SINDIP(J)**2
0075 ZA=ABS(X-X0(J))*SINDIP(J)*COSDIP(J)
0076 IF(ABS(SINDIP(J)).LT.0.001)ZA=X0(J)
0078 CALL NEWTON(XA,ZA,T0)
0079 CALL IMPULS(T0)
0080 200 CONTINUE

```

C

C EVALUATE FOR EACH POINT REFLECTOR

C

```

0081 110 IF(NPOINT.LE.0)GOTO 120
0083 DO 210 J=1,NPOINT
0084 CALL NEWTON(ABS(X-XP(J)),ZP(J),T0)
0085 CALL IMPULS(T0)
0086 210 CONTINUE

```

C

C CONVOLVE SERIES WITH WAVEFORM

C

```

0087 120 CALL VCLR(0,1,KB)
0088 CALL VCLR(KC,1,KB)
0089 CALL APPUT(TRACE,KB,KNSAMP,2)

```

```

0095 CALL APWD
0091 CALL CONV(0,1,KA,1,0,1,KNSAMP,KWAVE)
0092 CALL APWR
0093 IF(AMP.LE.0.0)GOTO 130
C
C GENERATE NOISE
C
0095 CALL VRAND(KD+1,2049,2,KRAND)
0096 CALL VSMUL(2049,2,KD,2049,2,KRAND)
0097 CALL CVEXP(2049,2,2048,2,KRAND)
0098 CALL VCLR(2048+(2*KRAND),1,KNOISE-(2*KRAND))
0099 CALL RFFT(2048,KNOISE,-1)
0100 CALL VSMUL(2048,1,KD+2,2048,1,KNSAMP)
0101 CALL VADD(0+N0,1,2048+N0,1,0+N0,1,NSAMP-N0+1)
0102 130 CALL APWR
0103 CALL APGET(TRACE,0,NSAMP,2)
0104 CALL APWD
0105 IF(IWRTW(2*NSAMP,TRACE,IBLK,IWRT).LT.0)STOP'WRITE ERROR'
0107 IBLK=IBLK+NBLKR
0108 100 CONTINUE
0109 CALL CLOSEC(IWRT)
0110 STOP'NORMAL TERMINATION'
0111 END

```

```

0001 SUBROUTINE RICKER(TSTEP,FRICK,N)
C
C SUBROUTINE TO EVALUATE A RICKER WAVELET
C DUE TO A G NUNNS 1979--- WITH MODS
C
0002 COMMON/BB/AUX(2048)
0003 N=IFIX(1000.0/(FRICK*TSTEP))
0004 IPLUS=N+1
0005 AUX(IPLUS)=1.0
0006 IF(N.EQ.0)RETURN
0008 IMINUS=IPLUS
0009 X=0.0
0010 DELX=3.14159265*FRICK*TSTEP/1000.0
0011 DO 100 I=1,N
0012 IPLUS=IPLUS+1
0013 IMINUS=IMINUS-1
0014 X=X+DELX
0015 AUX(IPLUS)=(1.0+COS(X))*(0.5-X**2)*EXP(-X**2)
0016 AUX(IMINUS)=AUX(IPLUS)
0017 100 CONTINUE
0018 RETURN
0019 END

```

```
0001      SUBROUTINE IMPULS(T0)
      C
      C SUB TO CALC BANDLIMITED IMPULSE RESPONSE
      C
0002      COMMON/AA/TRACE(2048),TSTEP,NSAMP,IFLAG
0003      DATA FRAC1/1.0/,FRAC2/10.0/
0004      TP=FRAC1*TSTEP
0005      I=(T0-TP)/TSTEP
0006      T=I*TSTEP
0007      I=I+1
0008      IF(IFLAG.GE.0)GOTO 120
0010 100 I=I+1
0011      IF(I.GT.NSAMP)RETURN
0013      T=T+TSTEP
0014      IF(I.LT.1)GOTO 100
0016      IF(T-T0-TP.GE.0.0)RETURN
0018      IF(T-T0+TP.LE.0.0)STOP 'ERROR 1'
0020      TRACE(I)=TRACE(I)+1000.0/T0
0021      GOTO 100
0022 120 VALMIN=FRAC2/SQRT(T0*TP)
0023      C1=SQRT(T0/2.0)
0024      C2=500.0/(C1*TP)
0025 140 I=I+1
0026      IF(I.GT.NSAMP)RETURN
0028      T=T+TSTEP
0029      IF(I.LT.1)GOTO 140
0031      IF(T-T0-TP.GT.0.0)GOTO 180
0033      IF(T-T0+TP.LT.0.0)STOP 'EROR 2'
0035      TRACE(I)=TRACE(I)+C2*SQRT(T-T0+TP)
0036      GOTO 140
0037 160 TRACE(I)=TRACE(I)+VAL
0038      I=I+1
0039      IF(I.GT.NSAMP)RETURN
0041      T=T+TSTEP
0042 180 VAL=C1/(SQRT(T-T0+TP)+SQRT(T-T0-TP))
0043      IF(VAL.GT.VALMIN)GOTO 160
0045      RETURN
0046      END
```

```

0001      SUBROUTINE NEWTON(X,Z,T)
C
C SUBROUTINE TO EVALUATE TRAVEL TIMES THROUGH
C A SERIES OF CONSTANT VELOCITY LAYERS
C USES NEWTON - RAPHSON (N-R) TECHINQUE
C
0002      COMMON/CC/VEL(11),THICKN(10),NLAYR
0003      COMMON/DD/N,D(11)
0004      IF(NLAYR.LE.0)GOTO 160
C
C FIND N
C
0005      TH=0.0
0007      DO 100 I=1,NLAYR
0008      N=I
0009      IF(Z.LE.TH+THICKN(I))GOTO 110
0011      100 TH=TH+THICKN(I)
0012      N=N+1
0013      110 IF(N.EQ.1)GOTO 160
C
C SET VALUES OF D(I)
C
0015      I1=N-1
0016      DO 120 I=1,I1
0017      120 D(I)=THICKN(I)
0018      D(N)=Z-TH
0019      IF(X.LE.0.0)GOTO 170
C
C FIND N-R STARTING VALUE
C
0021      C0=3.0*VEL(N)
0022      130 C0=(C0+VEL(N))*0.5
0023      IF(A(C0,1).LE.X)GOTO 130
C
C FIND C BY N-R
C
0025      C1=C0
0026      140 C0=C1
0027      C1=C0+(A(C0,1)-X)/(C0*A(C0,3))
0028      IF(ABS((C1-C0)/C1).GT.0.1E-03)GOTO 140
C
C EVALUATE T
C
0030      T=X
0031      DO 150 I=1,N
0032      150 T=T+D(I)*SQRT((C1/VEL(I))**2-1.0)
0033      T=T*1000.0/C1
0034      RETURN
C
C NO VELOCITY INTERFACE BETWEEN SOURCE AND RECIEVER
C
0035      160 T=SQRT(X**2+Z**2)*1000.0/VEL(1)
0036      RETURN
C

```

C VERTICAL RAYPATH

```
C
0037 170 T=0.0
0038 DO 180 I=1,N
0039 180 T=T+D(I)/VEL(I)
0040 T=T+1000.0
0041 RETURN
0042 END
```

0001 FUNCTION A(C,M)

```
C
C ANCILLARY FUNCTION FOR NEWTON
C
```

```
0002 COMMON/CC/VEL(11),THICKN(10),NLAYER
0003 COMMON/DD/N,D(11)
0004 A=0.0
0005 B=1.0
0006 DO 100 I=1,N
0007 IF(M.EQ.1)GOTO 100
0008 IF(M.LT.1)STOP 'ERROR 3'
0009 B=VEL(I)**(M-1)
0010 100 A=A+D(I)/(B*SQRT((C/VEL(I))**2-1.0)**M)
0011 RETURN
0012 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
C

```

```

VIRTUAL Z(120,256)
REAL*4 X(256),Y(120),CZ(20),Z
INTEGER*2 PTR(20),NMAX(256)
LOGICAL*1 SWCHES(5)
COMMON /PPEP1/ IX1,IY1,IX2,IY2,ISCAN,NSCAN,NBAND,NIPS,NIP0,
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,NBYTM1,MSK,LMSK
DATA CZ/1.0,0.95,0.9,0.85,0.8,0.75,0.7,0.65,0.6,0.55,0.5,0.45,
%0.4,0.35,0.3,0.25,0.2,0.15,0.1,0.05/
DATA SWCHES/.TRUE.,.TRUE.,.TRUE.,.TRUE.,.FALSE./
DATA NC/18/

```

```

C
C ZERO CONTOUR ARRAY
C

```

```

DO 1 I=1,256
DO 2 J=1,120
2 Z(J,I)=0.0
1 CONTINUE
CALL PLOTS(0,0,0)

```

```

C
C READ IN CONTROL DATA
C

```

```

CALL ASSIGN(2,'DK2:ANOUTV.DAT',14)
READ(2) LNUM,NCHAN,NSTART,M
READ(2) XSTART,XSTEP,FSAMP
READ(2) T01,T02,T0STEP,TGATE
READ(2) VSTEP,V1,V2MIN,V2MAX
READ(2) NT0
XSTEP=10.0/FLOAT(NT0-1)
IXPOS=NT0+1

```

```

C
C READ IN THE SEMBLANCE DATA
C

```

```

DO 10 JT0=1,NT0
IXPOS=IXPOS-1
READ(2) NPT,NMAX(JT0),(Z(J,IXPOS),J=1,NPT)
IF(NPT.GT.MYPT)MYPT=NPT
10 CONTINUE
VSTEP=4.0/FLOAT(MYPT-1)

```

```

C
C SET UP COORDINATES OF CONTOUR ARRAY
C

```

```

Y(1)=0.3
DO 20 L=1,MYPT
20 Y(L)=Y(L-1)+VSTEP
X(1)=0.0
DO 30 LL=2,NT0
30 X(LL)=X(LL-1)+XSTEP

```

```

C
C USE CONSYS CONTOUR PACKAGE
C

```

```

CALL CONTUR(Y,MYPT,X,NT0,Z,120,CZ,NC,PTR,SWCHES,
% , , , , )

```

```

C
C PLOT TIME GRID
C

```

```

T0LEN=T02-T01
TUNIT=1000.0*(X(NT0)-X(1))/T0LEN

```

```
TTUNIT=TUNIT/10.0
T0BEG=FLOAT(IFIX(T01+1000.0)/1000)
XOFF=(T0BEG-(T01/1000.0))*TUNIT
```

```
C
C PLOT SECOND GRID
```

```
C
TNUM=T0BEG
XPT=X(NT0)-XOFF
50 CALL PLOT(Y(1),XPT,+3)
CALL PLOT(Y(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 50
```

```
C
C PLOT TENTHS OF SECONDS GRID
```

```
C
LMSK="1403
MSK=+1
XPT=X(NT0)-XOFF
60 CALL PLOT(Y(1),XPT,+3)
CALL PLOT(Y(MYPT),XPT,+2)
XPT=XPT-TTUNIT
IF(XPT.GT.X(1))GOTO 60
XPT=X(NT0)-XOFF
70 CALL PLOT(Y(1),XPT,+3)
CALL PLOT(Y(MYPT),XPT,+2)
XPT=XPT+TTUNIT
IF(XPT.LT.X(NT0))GOTO 70
```

```
C
C VELOCITY GRID
```

```
C
VLEN=V2MAX-V1
VUNIT=(Y(MYPT)-Y(1))/VLEN
VTUNIT=VUNIT/10.0
VBEG=FLOAT(IFIX(V1+1.0))
VOFF=(VBEG-V1)*VUNIT
```

```
C
C PLOT KM/S GRID
```

```
C
LMSK=-1
MSK=0
YPT=Y(1)+VOFF
VNUM=VBEG
80 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+VUNIT
VNUM=VNUM+1.0
IF(YPT.LT.Y(MYPT))GOTO 80
```

```
C
C PLOT TENTHS GRID
```

```
C
LMSK="1403
MSK=-1
YPT=Y(1)+VOFF
90 CALL PLOT(YPT,X(NT0),+3)
CALL PLOT(YPT,X(1),+2)
YPT=YPT-VTUNIT
IF(YPT.GT.Y(1))GOTO 90
YPT=Y(1)+VOFF
100 CALL PLOT(YPT,X(NT0),+3)
CALL PLOT(YPT,X(1),+2)
YPT=YPT+VTUNIT
```

```

      IF(YPT.LT.V(MYPT))GOTO 100
C
C PLOT POSITION OF MAX PTS
C
      LMSK=-1
      MSK=0
      IXMP=NT0+1
      DO 40 MP=1,NT0
      IXMP=IXMP-1
      XP=X(IXMP)
      YP=Y(NMAX(MP))
      CALL SYMBOL(YP,XP,0.02,0.0,-1)
40 CONTINUE
C
C DO ANOTATION
C
      CALL SYMBOL(1.0,10.1,0.1,'VELOCITY ANALYSIS CONTOURS',0.0,26)
      CALL SYMBOL(4.8,10.3,0.1,'PROCESSING PARAMETERS',0.0,21)
      CALL PLOT(4.8,10.3,+3)
      CALL PLOT(6.9,10.3,+2)
      CALL SYMBOL(5.0,10.0,0.1,'NO. OF CHANNELS = ',0.0,18)
      FNUM=NCHAN
      CALL NUMBER(6.8,10.0,0.1,FNUM,0.0,-1)
      CALL SYMBOL(5.0,9.8,0.1,'SAMPLES PER CHANNEL = ',0.0,22)
      FNUM=LNUM
      CALL NUMBER(7.2,9.8,0.1,FNUM,0.0,-1)
      CALL SYMBOL(5.0,9.6,0.1,'SAMPLE DELAY = ',0.0,15)
      FNUM=NSTART
      CALL NUMBER(6.5,9.6,0.1,FNUM,0.0,-1)
      CALL SYMBOL(5.0,9.4,0.1,'LEVEL OF INTERPOLATION = ',0.0,25)
      FNUM=M
      CALL NUMBER(7.5,9.4,0.1,FNUM,0.0,-1)
      CALL SYMBOL(5.0,9.2,0.1,'CHANNEL 1 OFFSET = ',0.0,19)
      CALL NUMBER(6.9,9.2,0.1,XSTART,0.0,1)
      CALL SYMBOL(5.0,9.0,0.1,'CHANNEL SPACING = ',0.0,18)
      CALL NUMBER(6.8,9.0,0.1,XSTEPR,0.0,1)
      CALL SYMBOL(5.0,8.8,0.1,'SAMPLING INTERVAL MS = ',0.0,23)
      FNUM=1.024/FSAMP
      CALL NUMBER(7.3,8.8,0.1,FNUM,0.0,-1)
      CALL SYMBOL(5.0,8.6,0.1,'START OF ANALYSIS MS = ',0.0,23)
      CALL NUMBER(7.3,8.6,0.1,T01,0.0,-1)
      CALL SYMBOL(5.0,8.4,0.1,'END OF ANALYSIS MS = ',0.0,21)
      CALL NUMBER(7.1,8.4,0.1,T02,0.0,-1)
      CALL SYMBOL(5.0,8.2,0.1,'TIME STEP MS = ',0.0,15)
      CALL NUMBER(6.5,8.2,0.1,T0STEP,0.1,-1)
      CALL SYMBOL(5.0,8.0,0.1,'OPERATOR GATEWIDTH MS = ',0.0,23)
      CALL NUMBER(7.3,8.0,0.1,TGATE,0.0,-1)
      CALL SYMBOL(5.0,7.8,0.1,'START VELOCITY KM/S = ',0.0,22)
      CALL NUMBER(7.2,7.8,0.1,V1,0.0,2)
      CALL SYMBOL(5.0,7.6,0.1,'END VELOCITY KM/S = ',0.0,20)
      CALL NUMBER(7.0,7.6,0.1,V2MAX,0.0,2)
      CALL SYMBOL(5.0,7.4,0.1,'VELOCITY STEP KM/S = ',0.0,21)
      CALL NUMBER(7.1,7.4,0.1,VSTEP,0.0,2)
      CALL PLOT(0.0,0.0,999)
      STOP
      END

```

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

```

C
C   M J POULTER SEPT 79
C   PLOTTING PROGRAM FOR SEISMIC SECTION DATA
C   THIS PROGRAM TAKES IN SEISMIC TRACES AND
C   DISPLAYS THEM IN A NIB IMAGE FORM IN AN OUTPUT
C   FILE IN A FORM READY FOR POST PROCESSING.
C
0001   VIRTUAL IPBUF(256,80)
0002   REAL*8 FSPECR,FSPECW
0003   REAL*4 XBUF(2048),FBUF(3)
0004   INTEGER*2 OUTFLG,MASK(16),IDOT(4096),IHBLK(256),
      % BUF(5376),JROW(80)
0005   LOGICAL*1 TPD RR,TPDRW1,TPDRW2,ISTAT
0006   EQUIVALENCE (BUF(1),IHBLK(1)),(BUF(257),XBUF(1)),
      %(BUF(257),IDOT(1))
0007   COMMON /IO/NBLKBF,NWDBF,NBYTBF,IWRT
0008   DATA DEV/3RRK /
0009   DATA MASK/"200","100","40","20","10","4","2","1",
      %"100000","40000","20000","10000","4000","2000","1000","400/
0010   IEOTR=0
0011   IRD=IGETC()
0012   IWRT=IGETC()
0013   IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
0014   CALL ASSIGN(1,'DK1:MPPLTD.DAT',14)
0015   CALL ASSIGN(2,'DK1:MPPLTD.LOG',14)
C
C   DATA READ IN SECTION
C
0017   READ(1,1000) NTR,NPT,NINT,NDSTEP,ISBEG,ISFIN
0018   1000 FORMAT(6I5)
0019   READ(1,1000)TPDRR,TPDRW1,TPDRW2,INFLG,OUTFLG
0020   READ(1,1001) XSF
0021   1001 FORMAT(F10.0)
0022   1002 FORMAT(3A4)
C
C   SET UP CONSTANTS AND FILE ACCESS
C
0023   NPTS=ISFIN-ISBEG+1
0024   NPINT=(NPTS*NINT)-1
0025   NTIM=1
0026   IF(NPINT.GT.2048)NTIM=2
0027   IF(NPINT.GT.4096) STOP' ERROR IN INTERPOLATION SPECS'
0028   NBLKR=NPT/128
0029   NWDR=NPT*2
0030   NBYTR=NWDR*2+512
0031   NBLKBF=NDSTEP/2
0032   NBLKW=(NTR+24)*NBLKBF*NTIM
0033   NWDBF=NBLKBF*256
0034   NBYTBF=NWDBF*2
0035   IOFF=2*NDSTEP
0036   IBLKR=1
0037   IBLKW1=1
0038   IBLKW2=NBLKW/NTIM
0039   IA0=0

```

```
0042      IA1=4096
0043      IC1=IA1+NPTS
0044      IC2=IC1+1
0045      IC3=IC2+1
0046      IC4=IC3+1
      C
      C SET UP INPUT FILES
      C
0047      IF(INFLG.EQ.0)GOTO 10
0049      READ(1,1002)FBUF
0050      CALL IRAD50(12,FBUF,FSPECR)
0051      IF(LOOKUP(IRD,FSPECR).LT.0)STOP 'LOOKUP ERROR'
      C
      C SET UP OUTPUT FILES
      C
0053      10 IF(OUTFLG.EQ.0)GOTO 20
0055      READ(1,1002)FBUF
0056      CALL IRAD50(12,FBUF,FSPECW)
0057      IF(IENTER(IWRT,FSPECW,NBLKW).LT.0)STOP 'ENTER ERROR'
      C
      C SET UP ROW COUNTER AND CLEAR PLOT BUFFER
      C
0059      20 CONTINUE
0060      DO 15 I=1,80
0061      JROW(I)=I
0062      DO 15 J=1,256
0063      IPBUF(J,I)=0
0064      15 CONTINUE
      C
      C SET UP AP
      C
0065      CALL APINIT
0066      CALL VCLR(0,1,8192)
0067      CALL APWR
      C
      C MAIN LOOP FOR PLOTTING DIFFERENT TRACES
      C
0068      DO 100 I=1,NTR
0069      IFIL=I
      C
      C READ IN DATA FROM DISC
      C
0070      IF(INFLG.EQ.0)GOTO 110
0072      IF(IREADW(NWDR,XBUF,IBLKR,IRD).LT.0)STOP 'READW ERR'
0074      IBLKR=IBLKR+NBLKR
0075      GOTO 120
      C
      C TAPE READ ROUTINE
      C
0076      110 ITRY=1
0077      300 IF(ITRY.GT.3)GOTO 310
0079      IF(IEOTR.GE.0)GOTO 320
0081      310 WRITE(7,1100)TPDRR,IFIL
0082      1100 FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I5)
```

```

0083      WRITE(7,1101)
0084 1101  FORMAT(' ENTER NEW READ DRIVE NUMBER:',5)
0085      READ(5,1102)TPDRR
0086 1102  FORMAT(I1)
0087      IEOTR=0
0088      IF(TPDRR.GT.2)STOP'EOT TERMINATE'
C
C DO READ
C
0090 320  CALL TAPSUB(-1,TPDRR,ISTAT,IFLEN,IFIL,BUF,NBYTR,IEOTR)
0091      IF(ISTAT.LT.0)WRITE(2,1200)IFIL
0092 1200  FORMAT(' WARNING READ RETRY FAILED ON FILE NO:',15)
0093      IF(IEOTR.LT.0)GOTO 330
C
C WIND OVER EOF MARKER
C
0096      CALL TAPSUB(0,TPDRR,ISTAT, ,IFIL, , ,IEOTR)
0097 330  ITRY=ITRY+1
0098      IF(IHBLK(1).EQ."177777")GOTO 300
0100 120  CONTINUE
C
C SET UP PROCESSING CONSTANTS
C
0101      XBUF(ISFIN+1)=1.0/FLOAT(NINT)
0102      XBUF(ISFIN+2)=-FLOAT(IOFF-1)
0103      XBUF(ISFIN+3)=FLOAT(IOFF)
0104      XBUF(ISFIN+4)=XSF
C
C SECTION WHICH DEALS WITH INTERPLOATION
C AND SCALING OF DATA BEFORE PLOTTING
C
0105      CALL APPUT(XBUF,IA1,NPTS+4,2)
0106      CALL APWD
0107      CALL VMOV(IA1,1,IA0,NINT,NPTS)
0108      IF(NINT.EQ.1)GOTO 125
0109      CALL VSUB(IA1,1,IA1+1,1,IA1,1,NPTS-1)
0110      CALL VSMUL(IA1,1,IC1,IA1,1,NPTS-1)
0111      IST=IA0
0112      DO 130 J=2,NINT
0113      CALL VADD(IST,NINT,IA1,1,IST+1,NINT,NPTS-1)
0114      IST=IST+1
0115 130  CONTINUE
0116 125  CALL VSMUL(IA0,1,IC4,IA0,1,NPINT)
0117      CALL VTSADD(IA0,1,"4427,IA0,1,NPINT)
0118      CALL VCLIP(IA0,1,IC2,IC3,IA0,1,NPINT)
0119      CALL VINT(IA0,1,IA0,1,NPINT)
0120      CALL VFIX(IA0,1,IA0,1,NPINT)
0121      CALL APWR
0122      CALL APGET(IDOT,0,NPINT,1)
0123      IBIT=16
0124      IWORD=256
0125      CALL APWD
0126      IDOT(1)=IDOT(1)+IOFF
0127      DO 30 IP=2,NPINT

```

```
0129      IBITL=IBIT
0130      IWORDL=IWORD
0131      IBIT=IBIT-1
0132      IF (IBIT.GT.0) GOTO 40
0134      IWORD=IWORD-1
0135      IF (IWORD.EQ.0)GOTO 97
0137      IBIT=16
C
C SECTION WHERE POSITIVE (SHADED) LOBES ARE PLOTTED
C
0138      40 IF (IDOT(IP).LT.0)GOTO 50
0140      IDOT(IP)=IDOT(IP)+IOFF
0141      IDT=IDOT(IP)
0142      IF (IDOT(IP-1).GE.IOFF)GOTO 45
0144      IROW=IDOT(IP-1)
0145      55 IPBUF(IWORDL,JROW(IROW))=IPBUF(IWORDL,JROW(IROW)).OR.MASK(IBITL)
0146      IROW=IROW+1
0147      IF (IROW.LT.IOFF)GOTO 55
0149      45 IROW=IOFF
0150      60 IPBUF(IWORD,JROW(IROW))=IPBUF(IWORD,JROW(IROW)).OR.MASK(IBIT)
0151      IROW=IROW+1
0152      IF (IROW.LT.IDT)GOTO 60
0154      GOTO 30
C
C SECTION WHERE NEGATIVE (UNSHADED) LOBES ARE PLOTTED
C
0155      50 IDOT(IP)=IDOT(IP)+IOFF
0156      IF (IDOT(IP).LT.1.OR.IDOT(IP-1).LT.1)GOTO 30
0158      IF (IDOT(IP).GT.IDOT(IP-1))GOTO 70
0160      IDTB=IDOT(IP)
0161      IDTE=IDOT(IP-1)
0162      MSKB=MASK(IBIT)
0163      MSKE=MASK(IBITL)
0164      IWORDB=IWORD
0165      IWORDE=IWORDL
0166      GOTO 80
0167      70 IDTB=IDOT(IP-1)
0168      IDTE=IDOT(IP)
0169      MSKB=MASK(IBITL)
0170      MSKE=MASK(IBIT)
0171      IWORDB=IWORDL
0172      IWORDE=IWORD
0173      80 IDTM=(IDTB+IDTE)/2
0174      IROW=IDTB
0175      90 IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
0176      IROW=IROW+1
0177      IF (IROW.LE.IDTM)GOTO 90
0179      IROW=IDTM
0180      95 IPBUF(IWORDE,JROW(IROW))=IPBUF(IWORDE,JROW(IROW)).OR.MSKE
0181      IROW=IROW+1
0182      IF (IROW.LE.IDTE)GOTO 95
0184      30 CONTINUE
0185      97 JST=JROW(1)
0186      JFIN=JROW(NDSTEP)
```

```
0187      LIN=1
C
C FILL OUTPUT BUFFER
C
0188      DO 150 L=JST,JFIN
0189      DO 150 J=129,256
0190      IDOT(LIN)=IPBUF(J,L)
0191      IPBUF(J,L)=0
0192      LIN=LIN+1
0193      150 CONTINUE
0194      CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLKW1)
C
C DO SECOND SLICE IF NECESSARY
C
0195      IF(NTIM.LT.2)GOTO 170
0197      LIN=1
0198      DO 160 L=JST,JFIN
0199      DO 160 J=1,128
0200      IDOT(LIN)=IPBUF(J,L)
0201      IPBUF(J,L)=0
0202      LIN=LIN+1
0203      160 CONTINUE
0204      CALL BUFOUT(BUF,OUTFLG,TPDRW2,IBLKW2)
C
C SORT JROW ARRAY
C
0205      170 CALL APPUT(JROW,0,80,1)
0206      CALL APGET(JROW,NDSTEP,80-NDSTEP,1)
0207      CALL APGET(JROW(81-NDSTEP),0,NDSTEP,1)
0208      CALL APWD
0209      180 CONTINUE
C
C FLUSH BUFFER AT END OF A STRIP
C
0210      IRBEG=1
0211      IRFIN=NDSTEP
0212      DO 190 IF=1,3
0213      JST=JROW(IRBEG)
0214      JFIN=JROW(IRFIN)
C
C DO FIRST BUFFER
C
0215      LIN=1
0216      DO 200 L=JST,JFIN
0217      DO 200 J=129,256
0218      IDOT(LIN)=IPBUF(J,L)
0219      LIN=LIN+1
0220      200 CONTINUE
0221      CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLKW1)
C
C DO SECOND BUFFER IF NECESSARY
C
0222      IF(NTIM.LT.2)GOTO 195
0224      LIN=1
```

```

0225 DO 210 L=JST,JFIN
0226 DO 210 J=1,128
0227 IDOT(LIN)=IPBUF(J,L)
0228 LIN=LIN+1
0229 210 CONTINUE
0230 CALL BUFOUT(BUF,OUTFLG,TPDRW2,IBLKW2)
0231 195 IRBEG=IRBEG+NDSTEP
0232 IRFIN=IRFIN+NDSTEP
0233 190 CONTINUE
C
C PUT OUT EXTRA LINES TO HELP WITH TONER PROBLEM
C
0234 DO 220 ICL=1,NWDBF
0235 220 IDOT(ICL)=0
C
C SEND OUT EXTRA LINES
C
0236 DO 230 ISEN=1,20
0237 CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLKW1)
0238 IF(NTIM.GE.2)CALL BUFOUT(BUF,OUTFLG,TPDRW2,IBLKW2)
0240 230 CONTINUE
0241 CALL CLOSEC(IWRT)
0242 STOP 'NORMAL TERMINATION'
0243 END
    
```

```

0001 SUBROUTINE BUFOUT(IBUF,OUTFLG,TPDRW,IBLK)
C
C THIS A ROUTINE FOR PUTTING OUT THE BUFFERS FROM
C THE PLOTTING PROGRAM MPSPLT
C
0002 INTEGER*2 IBUF(1),OUTFLG
0003 LOGICAL*1 TPDRW,ISTAT
0004 COMMON /IO/ NBLKBF,NWDBF,NBYTBF,ICHAN
0005 DATA ICOUNT/0/
C
C KEEP A COUNT OF NUMBER OF BUFFERS WRITTEN
C
0006 ICOUNT=ICOUNT+1
0007 IF(OUTFLG.EQ.0)GOTO 10
0009 IF(IWRITW(NWDBF,IBUF(257),IBLK,ICHAN).LT.0)STOP 'WRITE ERROR'
0011 IBLK=IBLK+NBLKBF
0012 RETURN
C
C DO TAPE OUTPUT
C
0013 10 NBYT=NBYTBF+512
0014 IFLEN=NBLKBF+1
0015 IEOTW=0
0016 CALL TAPSUB(1,TPDRW,ISTAT,IFLEN,ICOUNT,IBUF,NBYT,IEOTW)
C
C CHECK FOR ERRORS
C
0017 IF(IEOTW.GE.0)GOTO 20
0019 WRITE(7,1000)TPDRW,ICOUNT
0020 1000 FORMAT(' EOT ON WRITE DRIVE:',I2,' BUFFER NUMBER:',I5)
0021 WRITE(7,1100)
0022 1100 FORMAT(' ENTER NEW WRITE DRIVE NUMBER:',S)
0023 READ (5,1200)TPDRW
0024 1200 FORMAT(I1)
0025 IEOTW=0
0026 IF(TPDRW.GT.2)STOP ' EOT TERMINATE '
0028 20 IF(ISTAT.GE.0)GOTO 30
0030 WRITE(7,1300)ICOUNT
0031 1300 FORMAT(' FATAL WRITE ERROR ON BUFFER:',I5)
0032 STOP ' WRITE ERROR TERMINATION '
0033 30 RETURN
0034 END
    
```

Section Plot Background :- MPPLBK

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
C PLOTTING PROGRAM TO GENERATE THE TIME AND
C POSITION BACKGROUND GRID FOR SECTION PLOTS
C

```

```

0001     DIMENSION IVEL(200),ATIME1(20),AVINT1(20),AVRMS1(20)
0002     COMMON /PPEP1/IX1,IY1,IX2,IY2,ISCCN,NSCCN,NBAND,NPIS,NIP0,
>NIPM1,LYNES,NIBSX,MSGLVL,XDOTS,YDOTS,PREF(2),
>RORG(2),PORT(2,2),INND(4),ALMT,FACT,JPEN,XOFF,
>XFAC,YOFF,NBITS,NBITM1,NBYTES,NBYTM1,MSK,LMSK
0003     DATA LMASK1/'104210'/,LMASK2/'177777'/
0004     CALL ASSIGN(1,'DK2:MPVDAT.DAT',14)
0005     CALL PLOTS(0,0,0)

```

```

C
C DATA INPUT
C

```

```

0006     WRITE(7,1000)
0007     1000 FORMAT(' ENTER TRACE LENGTH IN SECS(F10.0):',S)
0008     READ(5,1001) TSEC
0009     1001 FORMAT(F10.0)
0010     WRITE(7,1011)
0011     1011 FORMAT(' ENTER TIME DELAY TO START OF TRACES(F10.0):',S)
0012     READ(5,1001) TDELAY
0013     WRITE(7,1002)
0014     1002 FORMAT(' ENTER TRACE INTERVAL IN INCHES(F10.0):',S)
0015     READ(5,1001) TSPACE
0016     WRITE(7,1003)
0017     1003 FORMAT(' ENTER START TRACE NO(I5):',S)
0018     READ(5,1004) ISTART
0019     1004 FORMAT(I5)
0020     WRITE(7,1005)
0021     1005 FORMAT(' ENTER LAST TRACE NUMBER(I5):',S)
0022     READ(5,1004) IEND
0023     WRITE(7,1006)
0024     1006 FORMAT(' ENTER THE SCAN AMPLIFICATION FACTOR(I1):',S)
0025     READ(5,1009) ISCAN
0026     1009 FORMAT(I1)
0027     WRITE(7,1010)
0028     1010 FORMAT(' ENTER 1 FOR GRID PLOTTING 0 IF NOT(I1):',S)
0029     READ(5,1009) IBKFLG
0030     WRITE(7,1007)
0031     1007 FORMAT(' ENTER 1 IF WANT DOCUMENTATION 0 IF NOT(I1):',S)
0032     READ(5,1009) IDSCAN
0033     WRITE(7,1008)
0034     1008 FORMAT(' ENTER 1 IF VELOCITIES TO BE PLOTTED 0 IF NOT(I1):',S)
0035     READ(5,1009) IVSCAN

```

```

C
C BASIC PROCESSING PARAMETERS
C

```

```

0036     TPAPER=10.56
0037     TMAX=8.192
0038     TLENTH=10.24
0039     DSHIFT=1.0
0040     TSHIFT=1.6
0041     DSIZE=7.25

```

0042 VMAX=6.0
0043 VBOX=1.0

C
C EXPLANATION OF VARIABLES:
C TPAPER=BASIC PAPER WIDTH
C TSPACE=TRACE INTERVAL IN INCHES
C TSEC=TRACE LENGTH IN SECONDS
C TMAX=MAX SCAN IN SECONDS
C TLENTH=DISPLAY WIDTH IN INCHES FOR TRACES
C ISTART=FIRST TRACE NUMBER
C IEND=LAST TRACE NUMBER
C ISCAN=1,2,4,8-TRACE AMPLIFICATION
C IDSCAN=COSMETICS FLAG,1=ON,0=OFF
C DSHIFT=X SHIFT FORM ORIGIN
C TSHIFT=SEPARATION OF T SLICES
C DSIZE=DECLARATION SIZE
C IVSCAN=0-NO VELOCITIES
C 1-VELOCITY DISPLAY
C VMAX=MAX VELOCITY
C VBOX=VELOCITY BOX SIZE
C
C
C
C FORM DISPLAY CONSTANTS AND FIND NO OF TIME SLICES

0044 IPAD=1
0045 TSTART=TMAX
0046 TEQ=TSEC*ISCAN
0047 10 IF(TEQ.LE.TSTART) GOTO 11
0048 IPAD=IPAD+1
0049 TSTART=TSTART+TMAX
0050 GOTO 10
0051 IPAD=IPAD*ISCAN
0052 11 TSTART=TMAX/FLOAT(ISCAN)

C
C IPAD=NO OF TIME PADS
C
C TSTART=NO OF TRUE SECONDS PER PAD
C

0053 DITIM=-TLENTH/TSTART
0054 DITIM=DITIM*0.1

C
C STEPS FOR 1 AND 1/10 SEC INTERVALS
C

0055 ITRACE=IEND-ISTART+1
0056 ITR10=ITRACE/10

C
C NO OF TRACES AND DECADE POINTS
C

0057 IT1=ISTART/10
0058 IT2=IT1*10
0059 IST1=IT2+10-ISTART
0060 IF(ISTART.EQ.IT2) ITR10=ITR10-1

C
C GET TRACE AND DECADE STARTING POSITIONS

```

C AND REDUCE NO OF DECADES BY 1 IF STARTING ON A DECADE
C
0063 XPOPST=TSPACE*.5
0064 XPOP10=XPOPST+TSPACE*FLOAT(IST1)
C
C STARTING POSITIONS FOR TRACES AND DECADES
C
0065 TWIDTH=FLOAT(ITRACE)*TSPACE
0066 T10SPA=TSPACE*10.0
C
C WIDTH OF DIPLAY AND DECADE TRACE INTERVAL
C
0067 ITNUM=IT2+10
C
C STARTING DECADE NUMBER
C
0068 TUPPR=TPAPER-0.01
C
C UPPER DISPLAY LIMIT
C
0069 TSLICE=0.3
0070 TSIZE=TSLICE*.5
0071 PSIZE=(TUPPR-TLENTH)*0.25
C
C ANNOTATION PARAMETERS
C
0072 TOTAL=DSHIFT+FLOAT(IPAD)*(TWIDTH+TSHIFT)+
>FLOAT(IVSCAN)*(TWIDTH+TSHIFT)+DSIZE
0073 PORT(1,1)=TOTAL
C
C TOTAL DISPLAY SIZE
C
0074 XSTART=DSHIFT
0075 YSTART=0.0
0076 IF(IBKFLG.EQ.0)GOTO 201
C
C
C BEGIN MAIN DISPLAY LOOP
C SWITCHING OUT VELOCITY PLOTS WHEN
C NOT REQUIRED
C
0078 DO 200 ISCNT=1,IPAD
0079 CALL PLOT(XSTART,YSTART,-3)
0080 X1=-TSLICE
0081 X2=TWIDTH+TSLICE
0082 IF(ISCNT.NE.1)GOTO 101
C
C DRAW BORDER FOR FIRST FRAME
C
0084 CALL NEWPEN(3)
0085 CALL PLOT(X1,0.0,3)
0086 CALL PLOT(X1,TUPPR,2)
0087 CALL PLOT(X2,TUPPR,2)
0088 CALL PLOT(X2,0.0,2)

```

```

0089      IF(IPAD.EQ.1)CALL PLOT(X1,0.0,2)
C
C DRAW INSIDE MARGIN
C
0091      CALL NEWPEN(1)
0092      CALL PLOT(0.0,0.0,3)
0093      CALL PLOT(0.0,TUPPR,2)
0094      CALL PLOT(X1,TLENT,3)
0095      CALL PLOT(X2,TLENT,2)
0096      CALL PLOT(TWIDTH,TUPPR,3)
0097      CALL PLOT(TWIDTH,0.0,2)
C
C ANNOTATE MARGIN
C
0098      YA=TLENT+PSIZE*0.8
0099      X=-TSLICE*0.33
0100      CALL SYMBOL(X,YA,PSIZE,3HSEC,90.0,3)
0101      X=X+X2
0102      CALL SYMBOL(X,YA,PSIZE,3HSEC,90.0,3)
C
C ALL SURROUND PLOTTED NOW
C INSERT SHOT POINT LOCATIONS
C
0103      X=XPOP10
0104      Y=TLENT+0.33*(TUPPR-TLENT)
0105      COUNT=FLOAT(ITNUM)
0106      JLIM=ITR10-1
0107      DO 100 I=1,ITR10
0108      CALL NUMBER(X,Y,PSIZE,COUNT,0.0,-1)
0109      X=X+T10SPA
0110      100 COUNT=COUNT+10.0
C
C VELOCITY ANALYSIS POSITIONING IF REQD
C
0111      IF(IVSCAN.EQ.0)GOTO 31
0112      READ(1,25)IVCNT
0114      25 FORMAT(12I5)
0115      IF(IVCNT.LE.0)GOTO 31
C
C IVCNT=NO OF VELOCITY SCANS
C
0117      READ(1,25)(IVEL(I),I=1,IVCNT)
C
C READ IN VEL POSITIONS
C
0118      XBASE=0.5*TSPACE
0119      YBASE=TLENT
0120      DO 29 I=1,IVCNT
0121      ISHIFT=IVEL(I)-ISTART
0122      SHIFT=FLOAT(ISHIFT)*TSPACE+XBASE-0.5*PSIZE
0123      29 CALL SYMBOL(SHIFT,YBASE,PSIZE,14,0.0,-1)
0124      GOTO 31
C
C BORDER FOR OTHER DATA PADS THAN FIRST

```

```

C
0125 101 CALL NEWPEN(3)
0126 CALL PLOT(X1,0.0,3)
0127 CALL PLOT(X1,TUPPR,2)
0128 CALL NEWPEN(1)
0129 CALL PLOT(0.0,TUPPR,2)
0130 CALL PLOT(0.0,0.0,2)
0131 CALL PLOT(TWIDTH,0.0,3)
0132 CALL PLOT(TWIDTH,TUPPR,2)
0133 CALL NEWPEN(3)
0134 CALL PLOT(X2,TUPPR,3)
0135 CALL PLOT(X2,0.0,2)
0136 IF(ISCNT.EQ.IPAD)CALL PLOT(X1,0.0,2)

C
C INSERT SHOT POSITION LINES
C
0138 31 ITR=ITRACE-1
0139 CALL GRID(XPOPST,0.0,ITR,TSPACE,-1,TLENTH,LMASK1)
0140 ITR=ITR10-1
0141 CALL GRID(XPOP10,0.0,ITR,T10SPA,-1,TLENTH,LMASK2)

C
C INSERT TIMING LINES
C
0142 TBASE=(FLOAT(ISCNT-1)*TSTART*10.0)+(TDELAY*10.0)
0143 IT1=IFIX(TBASE)
0144 T1=FLOAT(IT1)
0145 TERR=TBASE-T1
0146 TBASE=TBASE*0.1
0147 IT1=IFIX(TBASE)
0148 T2=FLOAT(IT1)
0149 TERR10=TBASE-T2
0150 IF(TERR10.GE.1.0)TERR10=TERR10-1.0
0151 T10ST=TLENTH+D10TIM*(1.0-TERR)
0152 T1ST=TLENTH+D1TIM*(1.0-TERR10)
0153 TINT=(TSTART+TERR)*10.0
0154 ITR=IFIX(TINT)
0155 CALL GRID(0.0,T10ST,-1,TWIDTH,ITR,D10TIM,LMASK1)
0156 I1=IFIX(TSTART)
0157 ITR=I1-1
0158 CALL GRID(0.0,T1ST,-1,TWIDTH,ITR,D1TIM,LMASK2)

C
C INSERT ANNOTATION ON SECONDS
C
0160 X=-0.66*TSlice
0161 X1=TWIDTH+0.33*TSlice
0162 Y=T1ST
0163 TIMT=IFIX(0.1*T1+1.0)
0164 TIM=FLOAT(TIMT)
0165 CALL NEWPEN(1)
0166 DO 110 I=1,I1
0167 CALL NUMBER(X,Y,TSIZE,TIM,0.0,-1)
0168 CALL NUMBER(X1,Y,TSIZE,TIM,0.0,-1)
0169 TIM=TIM+1.0
0170 110 Y=Y+D1TIM

```

```
0171      XSTART=TWIDTH+TSHIFT
0172      Y=0.0
0173      200 CONTINUE
C
C VELOCITY ANALYSIS PLOT PROGRAM
C
0174      201 IF(IVSCAN.EQ.0)GOTO 641
0175      CALL PLOT(XSTART,YSTART,-3)
0176      DO 64 I=1,IVCNT
0177      READ(1,25)NVEL
0178      READ(1,26)(ATIME1(J),AVINT1(J),AVRMS1(J),J=1,NVEL)
0179      26 FORMAT(3F10.0)
0180      IVEL1=IVEL(I)
0181      VBSIZE=0.22*VBOX
0182      VBLINE=0.11*VBOX
0183      VBSTX=VPLT1-0.5*VBSIZE*3.0
0184      VBSTY=TLENTH*(0.5-AMARK*0.25)-23.0*VBLINE
C
C DRAW BOXES FOR VELOCITY PICKS
C
0186      CALL PLOT(VBSTX,VBSTY,3)
0187      CALL GRID(VBSTX,VBSTY,3,VBSIZE,22,VBLINE,LMASK2)
0188      SIZE=0.04*VBOX
0189      COFFST=0.025*VBOX
0190      Y=TLENTH*(0.5-AMARK*0.25)-VBSIZE+COFFST
0191      X=VPLT1-2.0*SIZE
0192      AV=FLOAT(IVEL1)
0193      CALL NUMBER(X,Y,SIZE,AV,0.0,-1)
0194      Y=Y-VBLINE
0195      X=VBSTX+0.1*VBSIZE
0196      XST=X
C
C DO TITLES
C
0197      CALL SYMBOL(X,Y,SIZE,4HTIME,0.0,4)
0198      X=X+VBSIZE
0199      CALL SYMBOL(X,Y,SIZE,4HVINT,0.0,4)
0200      X=X+VBSIZE
0201      CALL SYMBOL(X,Y,SIZE,4HVRMS,0.0,4)
0202      Y=Y-VBLINE
0203      X=XST
C
C PUT IN THE NUMBERS
C
0204      DO 63 JJ=1,NVEL
0205      CALL NUMBER(X,Y,SIZE,ATIME1(JJ),0.0,3)
0206      X=X+VBSIZE
0207      CALL NUMBER(X,Y,SIZE,AVINT1(JJ),0.0,3)
0208      X=X+VBSIZE
0209      CALL NUMBER(X,Y,SIZE,AVRMS1(JJ),0.0,3)
0210      X=XST
0211      63 Y=Y-VBLINE
0212      IF(AMARK.EQ.0.0)GOTO 632
0213      AMARK=0.0
```

```
0215      GOTO 64
0216      632 AMARK=1.0
0217      64 CONTINUE
0218      641 CONTINUE
C
C PLOT COSMETICS
C
0219      IF(IDSCAN.EQ.0)GOTO 73
0220      XSTART=TWIDTH+TSHIFT
0221      YSTART=0.0
0222      CALL PLOT(XSTART,YSTART,-3)
C
C REORIGIN
C
0224      X2=DSIZE
0225      Y2=TLENTH
0226      CALL NEWPEN(4)
0227      CALL PLOT(0.0,Y2,2)
0228      CALL PLOT(X2,Y2,2)
0229      CALL PLOT(X2,0.0,2)
0230      CALL PLOT(0.0,0.0,2)
0231      X1=0.05
0232      X2=X2-X1
0233      Y1=X1
0234      Y2=Y2-Y1
0235      CALL NEWPEN(1)
0236      CALL PLOT(X1,Y1,3)
0237      CALL PLOT(X1,Y2,2)
0238      CALL PLOT(X2,Y2,2)
0239      CALL PLOT(X2,Y1,2)
0240      CALL PLOT(X1,Y1,2)
0241      CALL PLOT(0.0,0.0,3)
C
C SURROUND DONE
C
0242      SIZE1=0.4
0243      SIZE2=0.3
0244      SIZE3=0.2
0245      SIZE4=0.1
0246      XMID=0.5*DSIZE
0247      Y=TLENTH
0248      X=XMID-8.0*SIZE1
0249      Y=Y-1.5*SIZE1
C
C DRAW IN THE HEADER
C
0250      CALL NEWPEN(3)
0251      CALL SYMBOL(X,Y,SIZE1,17HDURHAM UNIVERSITY,0.0,17)
0252      X=XMID-9.0*SIZE2
0253      Y=Y-1.5*SIZE2
0254      CALL NEWPEN(1)
0255      CALL SYMBOL(X,Y,SIZE2,18HSEISMIC PROCESSING,0.0,18)
0256      CALL NEWPEN(1)
0257      X=DSIZE*0.1
```

```
0259      Y=Y-1.5*SIZE3
C
C READ IN TITLE AND PLOT
C
0260      READ(1,70)(IVEL(I),I=1,40)
0261      70 FORMAT(40A2)
0262      CALL SYMBOL(X,Y,SIZE3,IVEL,0.0,80)
0263      Y=Y-0.2*SIZE3
0264      X=0.0
0265      CALL PLOT(X,Y,3)
0266      CALL PLOT(DSIZE,V,2)
0267      Y=Y-1.5*SIZE3
0268      X=XMID-8.0*SIZE3
0269      CALL SYMBOL(X,Y,SIZE3,17HSYSTEM PARAMETERS,0.0,17)
0270      X=0.1*DSIZE
0271      Y=Y-1.5*SIZE3
0272      READ(1,25)ILINES
C
C FILL IN PROCESSING PARAMETERS
C
0273      DO 72 I=1,ILINES
0274      READ(1,70)(IVEL(J),J=1,40)
0275      CALL SYMBOL(X,Y,SIZE4,IVEL,0.0,80)
0276      72 Y=Y-1.5*SIZE4
0277      73 CONTINUE
0278      CALL PLOT(0.0,0.0,999)
0279      STOP
0280      END
```

Section Plotting Merge and Post-Process :- MPMERG

Input file.....DK2:MPMERG.DAT

Log file.....DK1:MPMERG.LOG

Input ParametersREAD(1,1000)LPLT,NSTRIP,NDSTEP,IBKFLG,IPLFLG1000 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)FBUFP

FBUFP....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

```
C      M.J.POULTER OCT 80
C      POST PROCESS AND MERGE PROG FOR
C      SEISMIC PLOT SYSTEM
0001      INTEGER*2 IPBUF(2112),LBUF(132)
0002      REAL*4 FBUF(3)
0003      REAL*8 FSPECP,FSPECB
0004      LOGICAL*1 ITPDR1,ITPDR2,ITPDRP,ITPDRB
0005      DATA DEV/3RRK /
C
C      SET UP AP TO USE FOR ZEROING ARRAYS
C
0006      CALL APINIT
0007      CALL VCLR(0,1,2112)
C
C      SET UP I/O DEFINITIONS
C
0008      IF(IFETCH(DEV).NE.0)STOP 'FETCH ERR'
0010      CALL ASSIGN(1,'DK1:MPMERG.DAT',14)
0011      CALL ASSIGN(2,'DK1:MPMERG.LOG',14)
C
C      GET INPUT PARAMETERS
C
0012      READ(1,1000)LPLT,NSTRIP,NDSTEP,IBKFLG,IPLFLG
0013      1000  FORMAT(5I5)
0014      IF(IPLFLG.EQ.0)GOTO 10
0016      READ(1,1000)LSTP,LFINP,IOFLGP,ITPDR1,ITPDR2
0017      IF(IOFLGP.EQ.0)GOTO 10
0019      READ(1,1001)FBUF
0020      1001  FORMAT(3A4)
0021      CALL IRAD50(12,FBUF,FSPECP)
0022      ICHP=IGETC()
0023      IF(LOOKUP(ICHP,FSPECP).LT.0)STOP 'LOOKUP ERROR'
0025      10  IF(IBKFLG.EQ.0)GOTO 20
0027      READ(1,1000)LSTB,LFINB,IOFLGB,ITPDRB
0028      IF(IOFLGB.EQ.0)GOTO 20
0030      READ(1,1001)FBUF
0031      CALL IRAD50(12,FBUF,FSPECB)
0032      ICHB=IGETC()
0033      IF(LOOKUP(ICHB,FSPECB).LT.0)STOP 'LOOKUP ERROR'
0035      20  CONTINUE
C
C      SET UP MATRIX FOR MAIN LOOP
C
0036      CALL MTXSET
0037      CALL MTX(IPBUF,0,1)
0038      LNUM=0
0039      NTIM=0
0040      IST=1057
0041      NBYTP=NDSTEP*256+512
0042      NBYTB=4608
0043      LIMP=4608
0044      NWDP=NDSTEP/2*256
0045      IF(IOFLGP.EQ.0)LIMP=2560+NWDP
0047      LIMB=2304
```

```
0048      ITPDRP=ITPDR1
0049      IEOF=0
0050      IEOFB=0
      C
      C ZERO THE PLOT ARRAY
      C
0051      40 IST=1056-IST+2
0052      CALL APGET(IPBUF(IST),0,1056,1)
0053      IPSV=IST
0054      CALL APWD
      C
      C START OF MAIN LOOP
      C
0055      DO 50 I =1,8
0056      LNUM=LNUM+1
0057      EOFLG=1
      C
      C THIS LOOP FILLS ONE LINE OF PLOT BUFFER
      C WITH INPUT FROM EACH PLOT MASK
      C
      C DO SEISMIC SECTION FIRST
      C
0058      IF(IPLFLG.EQ.0)GOTO 60
0059      IF(LNUM.GT.LFINP)GOTO 60
0060      IF(IEOF.NE.0)GOTO 60
0061      EOFLG=0
0062      IF(LNUM.LT.LSTP)GOTO 60
0063      IPOS=IPSV
0064      ICD=1
      C
      C FILL LINE BUFFER
      C
0065      CALL LINFIL(LBUF,ICD,IEOF,ITPDRP,IOFLGP,NBYTP,LNUM,ICHP,LIMP)
0066      DO 70 J=1,132
0067      IPBUF(IPOS)=IPBUF(IPOS).OR.LBUF(J)
0068      70 IPOS=IPOS+1
      C
      C DO BACKGROUND
      C
0069      60 IF(IBKFLG.EQ.0)GOTO 80
0070      IF(LNUM.GT.LFINB)GOTO 80
0071      IF(IEOFB.NE.0)GOTO 80
0072      EOFLG=0
0073      IF(LNUM.LT.LSTB)GOTO 80
0074      IPOS=IPSV
0075      ICD=2
0076      CALL LINFIL(LBUF,ICD,IEOFB,ITPDRB,IOFLGB,NBYTB,LNUM,ICHB,LIMB)
0077      DO 90 J=1,132
0078      IPBUF(IPOS)=IPBUF(IPOS).OR.LBUF(J)
0079      90 IPOS=IPOS+1
0080      80 IPSV=IPOS
0081      50 CONTINUE
      C
```

C WHEN A PLOT BUFFER IS FULL COME HERE TO EMPTY IT
C

```
0090 CALL MWAIT
0091 CALL MTX(IPBUF(IST),1056,2)
0092 IF(EOFFLG.NE.0)GOTO 100
0094 IF(LNUM.LT.LPLT)GOTO 40
0096 100 CALL MWAIT
0097 CALL APGET(IPBUF,0,2112,1)
0098 DO 110 J=1,20
0099 CALL MTX(IPBUF,2112,2)
0100 CALL MWAIT
0101 110 CONTINUE
0102 CALL MTX(IPBUF,0,1)
0103 CALL MWAIT
0104 NTIM=NTIM+1
0105 LNUM=0
0106 ITPDRP=ITPDR2
0107 IF(NTIM.LT.NSTRIP)GOTO 40
0109 STOP 'NORMAL TERMINATION'
0110 END
```

```

0001      SUBROUTINE LINFIL(LBUF,ICD,IEOF,IDRV,IOFLG,NBYTR,IFIL,ICHAN,NLIM)
0002      INTEGER*2 LBUF(1),INBUF(5120),ITPST(2),IST(2),NPTS(2),
        %IBLKS(2),IPOSS(2)
0003      LOGICAL*1 IDRV,ISTAT
0004      DATA ITPST(1)/2305/,ITPST(2)/1/,IST(1)/2561/,
        %IST(2)/257/,NPTS(1)/128/,NPTS(2)/132/,IPOSS(1)/5120/,
        %IPOSS(2)/5120/,IBLKS(1)/1/,IBLKS(2)/1/,IEOTR/0/
C
C CLEAR LINE BUFFER
C
0005      CALL APGET(LBUF,0,132,1)
0006      NPT=NPTS(ICD)
0007      IPOS=IPOSS(ICD)
0008      IBLK=IBLKS(ICD)
0009      ITST=ITPST(ICD)
0010      IBEG=IST(ICD)
0011      CALL APWD
C
C FILL BUFFER
C
0012      DO 10 J=1,NPT
0013      IPOS=IPOS+1
0014      IF(IPOS.LE.NLIM)GOTO 30
0016      IF(IOFLG.EQ.0)GOTO 40
C
C DISC INPUT
C
0018      IN=IREADW(2048,INBUF(IBEG),IBLK,ICHAN)
0019      IBLK=IBLK+8
0020      IPOS=IBEG
0021      IF(IN.EQ.2048)GOTO 30
0023      IEOF=1
0024      NLIM=IN+IPOS
0025      IF(IN.LT.-1)STOP 'READ ERROR'
0027      IF(IN.LE.0)RETURN
0029      GOTO 30
C
C TAPE INPUT
C
0030      40 ITRY=1
0031      50 IF(ITRY.GT.3)GOTO 60
0033      IF(IEOTR.GE.0)GOTO 70
0035      60 WRITE(7,1000)IDRV
0036      1000 FORMAT(' EOT ON READ DRIVE:',I2,/,
        %' ENTER NEW DRIVE NUMBER:',$)
0037      READ(5,1001)IDRV
0038      1001 FORMAT(I1)
0039      IEOTR=0
0040      IF(IDRV.GT.2)IEOF=1
0042      IF(IEOF.NE.0)RETURN
0044      70 CALL TAPSUB(-1,IDRV,ISTAT,IFLEN,IFIL,INBUF(ITST),NBYTR,IEOTR)
0045      IF(ISTAT.LT.0)WRITE(2,1002)IFIL
0047      1002 FORMAT(' WARNING RETRIES FAILED ONREAD FILE NO:'I5)
0048      IF(IEOTR.LT.0)GOTO 80

```

```

0050      CALL TAPSUB(0,IDRV,ISTAT,IFIL,,IEOTR)
0051      80 ITRY=ITRY+1
0052      IF(INBUF(ITST).EQ."17777")GOTO 50
0054      IPOS=IBEG
0055      30 CONTINUE
0056      LBUF(J)=INBUF(IPOS)
0057      10 CONTINUE
0058      IBLKS(ICD)=IBLK
0059      IPOSS(ICD)=IPOS
0060      RETURN
0061      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

```

C
C   M J POULTER SEPT 79
C   PLOTTING PROGRAM FOR SEISMIC SECTION DATA
C   THIS PROGRAM TAKES IN SEISMIC TRACES AND
C   DISPLAYS THEM IN A NIB IMAGE FORM IN AN OUTPUT
C   FILE IN A FORM READY FOR POST PROCESSING.
C
0001   DIMENSION IPBUF(128,80),JROW(80),FBUF(3),
0002   %IDOT(2048),XBUF(2052),MASK(16),NSAVE(20)
0003   REAL*8 FSPECR,FSPECW
0004   EQUIVALENCE (XBUF(1),IDOT(1))
0005   DATA DEV/3RRK /
0006   DATA MASK/"200","100","40","20","10","4","2","1",
0007   %"100000","40000","20000","10000","4000","2000","1000","400"/
C
C   DATA READ IN SECTION
C
0008   WRITE(7,1005)
0009   1005 FORMAT(' ENTER NO OF TRACES',/,
0010   %' NO OF POINTS PER TRACE',/,
0011   %' NO OF TIMES TO EXPAND',/,
0012   %' AND STEP SIZE (415)')
0013   READ(5,1000) NTR,NPT,NDPT,NDSTEP
0014   1000 FORMAT(4I5)
0015   WRITE(7,1006)
0016   1006 FORMAT(' ENTER SCALE FACTOR:',$)
0017   READ(5,1001) XSF
0018   1001 FORMAT(F10.0)
0019   WRITE(7,1007)
0020   1007 FORMAT(' ENTER INPUT FILE NAME:',$)
0021   READ(5,1002)FBUF
0022   1002 FORMAT(3A4)
0023   CALL IRAD50(12,FBUF,FSPECR)
0024   WRITE(7,1008)
0025   1008 FORMAT(' ENTER OUTPUT FILE NAME:',$)
0026   READ(5,1002)FBUF
0027   CALL IRAD50(12,FBUF,FSPECW)
C
C   SET UP CONSTANTS AND FILE ACCESS
C
0028   ICHR=IGETC()
0029   ICHW=IGETC()
0030   IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
0031   IER=LOOKUP(ICHR,FSPECR)
0032   IF(IER.LT.0)WRITE(7,*)IER
0033   IF(IER.LT.0)STOP'LOOKUP ERR'
0034   NFIN=NPT*NDPT-1
0035   NBLKR=NPT/128
0036   NWDR=2*NPT
0037   NBLKW=(NTR+4)*(NDSTEP/2)
0038   NBLKBF=NDSTEP/2
0039   NWBUF=NBLKBF*256
0040   IOFF=2*NDSTEP
0041   IBLKR=1

```

```

0040      IBLKW=1
0041      NTIM=1
0042      NSTRIP=0
0043      IBOFF=0
C
C SECTION WHICH SETS UP PARAMETERS TO ALLOW
C STRIPPING OF PLOT IF REQUIRED
C
0044      IF(NFIN.GT.2048)GOTO 1
0045      IF(IENTER(ICHW,FSPECW,NBLKW).LT.0)STOP 'ENTER ERR'
0046      NSTRIP=1
0047      GOTO 2
0048      1 NS=NFIN
0049      3 NSAVE(NTIM)=2047
0050      NS=NS-2048
0051      NTIM=NTIM+1
0052      IF(NTIM.GT.20)STOP 'TOO MANY STRIPS'
0053      IF(NS.GT.2048)GOTO 3
0054      NSAVE(NTIM)=NS
0055      NBLKW=NBLKW*NTIM
0056      IF(IENTER(ICHW,FSPECW,NBLKW).LT.0)STOP 'ENTER ERR'
C
C START OF LOOP IF STRIPPING NECESSARY
C
0057      4 NSTRIP=NSTRIP+1
0058      NFIN=NSAVE(NSTRIP)
0059      NPT=(NFIN+1)/NDPT
0060      NWDR=2*NPT
0061      IBLKR=1+IBOFF
0062      NBOF=NPT/128
0063      IBOFF=IBOFF+NBOF
C
C SET UP ROW COUNTER FOR BUFFER USAGE
C
0064      2 DO 5 I=1,80
0065      5 JROW(I)=I
C
C SET UP AP AND CLEAR PLOT BUFFER
C
0066      CALL APINIT
0067      CALL VCLR(0,1,5120)
0068      CALL APWR
0069      CALL APGET(IPBUF(1,1),0,5120,2)
C
C MAIN LOOP FOR PLOTTING DIFFERENT TRACES
C
0070      DO 10 I=1,NTR
0071      IF(IREADW(NWDR,XBUF,IBLKR,ICHR).LT.0)STOP 'READW ERR'
0072      IBLKR=IBLKR+NBLKR
0073      XBUF(NPT+1)=1.0/FLOAT(NDPT)
0074      XBUF(NPT+2)=-FLOAT(IOFF-1)
0075      XBUF(NPT+3)=FLOAT(IOFF)
0076      XBUF(NPT+4)=XSF
C

```

C SECTION WHICH DEALS WITH INTERPLOATION
 C AND SCALING OF DATA BEFORE PLOTTING
 C

```

0083 CALL APPUT(XBUF,2048,NPT+4,2)
0084 CALL APWD
0085 CALL VMOV(2048,1,0,NDPT,NPT)
0086 IF(NDPT.EQ.1)GOTO 25
0087 CALL VSUB(2048,1,2049,1,6144,1,NPT-1)
0088 CALL VSMUL(6144,1,2048+NPT,6144,1,NPT-1)
0089 IBEG=0
0090 IFIN=1
0091 DO 20 J=2,NDPT
0092 CALL VADD(IBEG,NDPT,6144.1,IFIN,NDPT,NPT-1)
0093 IBEG=IBEG+1
0094 IFIN=IFIN+1
0095 20 CONTINUE
0096 25 CALL VSMUL(0,1,2051+NPT,0,1,NFIN)
0097 CALL VTSADD(0,1,"4427,0,1,NFIN)
0098 CALL VCLIP(0,1,2049+NPT,2050+NPT,0,1,NFIN)
0099 CALL VINT(0,1,0,1,NFIN)
0100 CALL VFIX(0,1,0,1,NFIN)
0101 CALL APWR
0102 CALL APGET(IDOT,0,NFIN,1)
0103 IBIT=16
0104 IWORD=128
0105 CALL APWD
0106 IDOT(1)=IDOT(1)+IOFF
0107 DO 30 IP=2,NFIN
0108 IBITL=IBIT
0109 IWORDL=IWORD
0110 IBIT=IBIT-1
0111 IF(IBIT.GT.0) GOTO 40
0112 IWORD=IWORD-1
0113 IF(IWORD.EQ.0)GOTO 120
0114 IBIT=16
  
```

C
 C SECTION WHERE POSITIVE (SHADED) LOBES ARE PLOTTED
 C

```

0118 40 IF(IDOT(IP).LT.0)GOTO 50
0119 IDOT(IP)=IDOT(IP)+IOFF
0120 IDT=IDOT(IP)
0121 IF(IDOT(IP-1).GE.IOFF)GOTO 45
0122 IROW=IDOT(IP-1)
0123 55 IPBUF(IWORDL,JROW(IROW))=IPBUF(IWORDL,JROW(IROW)).OR.MASK(IBITL)
0124 IROW=IROW+1
0125 IF(IROW.LT.IOFF)GOTO 55
0126 45 IROW=IOFF
0127 60 IPBUF(IWORDL,JROW(IROW))=IPBUF(IWORDL,JROW(IROW)).OR.MASK(IBIT)
0128 IROW=IROW+1
0129 IF(IROW.LT.IDT)GOTO 60
0130 GOTO 30
  
```

C
 C SECTION WHERE NEGATIVE (UNSHADED) LOBES ARE PLOTTED
 C

```

0135 50 IDOT(IP)=IDOT(IP)+IOFF
0136 IF(IDOT(IP).LT.1.OR.IDOT(IP-1).LT.1)GOTO 30
0136 IF(IDOT(IP).GT.IDOT(IP-1))GOTO 70
0140 IDTB=IDOT(IP)
0141 IDTE=IDOT(IP-1)
0142 MSKB=MASK(IBIT)
0143 MSKE=MASK(IBITL)
0144 IWORDB=IWORD
0145 IWORDE=IWORDL
0146 GOTO 80
0147 70 IDTB=IDOT(IP-1)
0148 IDTE=IDOT(IP)
0149 MSKB=MASK(IBITL)
0150 MSKE=MASK(IBIT)
0151 IWORDB=IWORDL
0152 IWORDE=IWORD
0153 80 IDTM=(IDTB+IDTE)/2
0154 IROW=IDTB
0155 90 IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
0156 IROW=IROW+1
0157 IF(IROW.LE.IDTM)GOTO 90
0158 IROW=IDTM
0159 100 IPBUF(IWORDE,JROW(IROW))=IPBUF(IWORDE,JROW(IROW)).OR.MSKE
0160 IROW=IROW+1
0161 IF(IROW.LE.IDTE)GOTO 100
0162 30 CONTINUE
0163 120 JST=JROW(1)

```

C
C WRITE OUT PART OF BUFFER AND INITIALISE
C FOR A NEW TRACE

```

0166 IF(IWRITE(NWBUF,IPBUF(1,JST),IBLKW,ICHW).LT.0)STOP 'WRITE ERR'
0168 IBLKW=IBLKW+NBLKBF
0169 CALL APPUT(JROW,0,80,1)
0170 CALL APGET(JROW,NDSTEP,80-NDSTEP,1)
0171 CALL APGET(JROW(81-NDSTEP),0,NDSTEP,1)
0172 CALL APWD
0173 CALL VCLR(0,1,NWBUF)
0174 CALL APGET(IPBUF(1,JST),0,NWBUF,1)
0175 10 CONTINUE
0176 IROW=1

```

C
C FLUSH BUFFER AT END OF A STRIP

```

0177 DO 110 IF=1,3
0178 IF(IWRITE(NWBUF,IPBUF(1,JROW(IROW)),IBLKW,ICHW).LT.0)STOP 'WRT ER'
0180 IBLKW=IBLKW+NBLKBF
0181 IROW=IROW+NDSTEP
0182 110 CONTINUE

```

C
C CHECK IF MORE STRIPS TO BE DONE

```

0183 IF(NSTRIP.LT.NTIM)GOTO 4
0184 CALL CLOSEC(ICHW)

```

```

0185 STOP 'NORMAL TERMINATION'
0186 END

```

Gather Plotting(Large Separation) :- MPGPI

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
C M J POULTER SEPT 79
C PLOTTING PROGRAM FOR SEISMIC GATHER DATA
C THIS PROGRAM TAKES IN SEISMIC TRACES AND
C DISPLAYS THEM IN A NIB IMAGE FORM IN AN OUTPUT
C FILE IN A FORM READY FOR POST PROCESSING.
C
0001 DIMENSION IPBUF(128,80),JROW(80),FBUF(3),
0002 %IDOT(2048),XBUF(2052),MASK(16),NSAVE(20)
0003 REAL*8 FSPECR,FSPECW
0004 EQUIVALENCE (XBUF(1),IDOT(1))
0005 DATA DEV/3RRK /
DATA MASK/"200","100","40","20","10","4","2","1,
%"100000","40000","20000","10000","4000","2000","1000","400/
C
C DATA READ IN SECTION
C
0006 WRITE(7,1005)
0007 1005 FORMAT(' ENTER NO OF TRACES',/,
%' NO OF POINTS PER TRACE',/,
%' NO OF TIMES TO EXPAND')
0008 READ(5,1000) NTR,NPT,NDPT
0009 1000 FORMAT(4I5)
0010 WRITE(7,1006)
0011 1006 FORMAT(' ENTER SCALE FACTOR:',$)
0012 READ(5,1001) XSF
0013 1001 FORMAT(F10.0)
0014 WRITE(7,1007)
0015 1007 FORMAT(' ENTER INPUT FILE NAME:',$)
0016 READ(5,1002)FBUF
0017 1002 FORMAT(3A4)
0018 CALL IRAD50(12,FBUF,FSPECR)
0019 WRITE(7,1008)
0020 1008 FORMAT(' ENTER OUTPUT FILE NAME:',$)
0021 READ(5,1002)FBUF
0022 CALL IRAD50(12,FBUF,FSPECW)
C
C SET UP CONSTANTS AND FILE ACCESS
C
0023 ICHR=IGETC()
0024 ICHW=IGETC()
0025 IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
0026 IER=LOOKUP(ICHR,FSPECR)
0027 IF(IER.LT.0)WRITE(7,*)IER
0028 IF(IER.LT.0)STOP'LOOKUP ERR'
0029 NDSTEP=20
0030 NBUFSZ=80
0031 NBSPAC=NDSTEP/2
0032 NWSPAC=NBSPAC*256
0033 NFIN=NPT*NDPT-1
0034 NBLKR=NPT/128
0035 NWDR=2*NPT
0036 NBLKW=(NTR*NBUFSZ/2)+(NTR*10)
0037 NBLKBF=NBUFSZ/2

```

```

0041      NWSUF=NBLKBF*256
0042      IOFF=2*NDSTEP
0043      IBLKR=1
0044      IBLKW=1
0045      NTIM=1
0046      NSTRIP=0
0047      IBOFF=0
C
C SECTION WHICH SETS UP PARAMETERS TO ALLOW
C STRIPPING OF PLOT IF REQUIRED
C
0048      IF(NFIN.GT.2048)GOTO 1
0049      IF(IENTER(ICHW,FSPECW,NBLKW).LT.0)STOP 'ENTER ERR'
0050      NSTRIP=1
0051      GOTO 2
0052      1 NS=NFIN
0053      3 NSAVE(NTIM)=2047
0054      NS=NS-2048
0055      NTIM=NTIM+1
0056      IF(NTIM.GT.20)STOP 'TOO MANY STRIPS'
0057      IF(NS.GT.2048)GOTO 3
0058      NSAVE(NTIM)=NS
0059      NBLKW=NBLKW*NTIM
0060      IF(IENTER(ICHW,FSPECW,NBLKW).LT.0)STOP 'ENTER ERR'
C
C START OF LOOP IF STRIPPING NECESSARY
C
0061      4 NSTRIP=NSTRIP+1
0062      NFIN=NSAVE(NSTRIP)
0063      NPT=(NFIN+1)/NDPT
0064      NWDR=2*NPT
0065      IBLKR=1+IBOFF
0066      NBOF=NPT/128
0067      IBOFF=IBOFF+NBOF
C
C SET UP ROW COUNTER FOR BUFFER USAGE
C
0068      2 DO 5 I=1,80
0069      5 JROW(I)=I
C
C SET UP AP AND CLEAR PLOT BUFFER
C
0070      CALL APINIT
0071      CALL VCLR(0,1,5120)
0072      CALL APWR
0073      CALL APGET(IPBUF(1,1),0,5120,2)
C
C MAIN LOOP FOR PLOTTING DIFFERENT TRACES
C
0074      DO 10 I=1,NTR
0075      IF(IREADW(NWDR,XBUF,IBLKR,ICHR).LT.0)STOP 'READW ERR'
0076      IBLKR=IBLKR+NBLKR
0077      XBUF(NPT+1)=1.0/FLOAT(NDPT)
0078      XBUF(NPT+2)=-FLOAT(IOFF-1)

```

0085
0086

XBUF(NPT+3)=FLOAT(IOFF)
XBUF(NPT+4)=XSF

C
C SECTION WHICH DEALS WITH INTERPLOATION
C AND SCALING OF DATA BEFORE PLOTTING
C

0097
0098
0099
0100
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110
0111
0112
0113
0114
0115
0116
0117
0118
0119
0120

CALL APPUT(XBUF,2048,NPT+4,2)
CALL APWD
CALL VMOV(2048,1,0,NDPT,NPT)
IF(NDPT.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
25 CALL VSMUL(0,1,2051+NPT,0,1,NFIN)
CALL VTSADD(0,1,"4427,0,1,NFIN)
CALL VCLIP(0,1,2049+NPT,2050+NPT,0,1,NFIN)
CALL VINT(0,1,0,1,NFIN)
CALL VFIX(0,1,0,1,NFIN)
CALL APWR
CALL APGET(IDOT,0,NFIN,1)
IBIT=16
IWORD=128
CALL APWD
IDOT(1)=IDOT(1)+IOFF
DO 30 IP=2,NFIN
IBITL=IBIT
IWORDL=IWORD
IBIT=IBIT-1
IF(IBIT.GT.0) GOTO 40
IWORD=IWORD-1
IF(IWORD.EQ.0)GOTO 120
IBIT=16

C
C SECTION WHERE POSITIVE (SHADED) LOBES ARE PLOTTED
C

0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137

40 IF(IDOT(IP).LT.0)GOTO 50
IDOT(IP)=IDOT(IP)+IOFF
IDT=IDOT(IP)
IF(IDOT(IP-1).GE.IOFF)GOTO 45
IROW=IDOT(IP-1)
55 IPBUF(IWORDL,JROW(IROW))=IPBUF(IWORDL,JROW(IROW)).OR.MASK(IBITL)
IROW=IROW+1
IF(IROW.LT.IOFF)GOTO 55
45 IROW=IOFF
60 IPBUF(IWORD,JROW(IROW))=IPBUF(IWORD,JROW(IROW)).OR.MASK(IBIT)
IROW=IROW+1
IF(IROW.LT.IDT)GOTO 60
GOTO 30

C
C SECTION WHERE NEGATIVE (UNSHADED) LOBES ARE PLOTTED
C

```

0139 50 IDOT(IP)=IDOT(IP)+IOFF
0140 IF(IDOT(IP).LT.1.OR.IDOT(IP-1).LT.1)GOTO 30
0141 IF(IDOT(IP).GT.IDOT(IP-1))GOTO 70
0142 IDTB=IDOT(IP)
0143 IDTE=IDOT(IP-1)
0144 MSKB=MASK(IBIT)
0145 MSKE=MASK(IBITL)
0146 IWORDB=IWORD
0147 IWORDE=IWORDL
0148 GOTO 80
0149 70 IDTB=IDOT(IP-1)
0150 IDTE=IDOT(IP)
0151 MSKB=MASK(IBITL)
0152 MSKE=MASK(IBIT)
0153 IWORDB=IWORDL
0154 IWORDE=IWORD
0155 80 IDTM=(IDTB+IDTE)/2
0156 IROW=IDTB
0157 90 IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
0158 IROW=IROW+1
0159 IF(IROW.LE.IDTM)GOTO 90
0160 IROW=IDTM
0161 100 IPBUF(IWORDE,JROW(IROW))=IPBUF(IWORDE,JROW(IROW)).OR.MSKE
0162 IROW=IROW+1
0163 IF(IROW.LE.IDTE)GOTO 100
0164 30 CONTINUE
0165 120 JST=JROW(1)

```

C
C WRITE OUT PART OF BUFFER AND INITIALISE
C FOR A NEW TRACE
C

```

0171 IF(IWRITW(NWBUF,IPBUF(1,JST),IBLKW,ICHW).LT.0)STOP 'WRITE ERR'
0172 IBLKW=IBLKW+NBLKBF
0173 CALL APWR
0174 CALL VCLR(0,1,NWBUF)
0175 CALL APGET(IPBUF(1,JST),0,NWBUF,1)
0176 CALL APWD
0177 IF(IWRITW(NWSPAC,IPBUF(1,JST),IBLKW,ICHW).LT.0)STOP 'WRITE ERR'
0178 IBLKW=IBLKW+NBSPEC
0179 10 CONTINUE
0180 IROW=1

```

C
C CHECK IF MORE STRIPS TO BE DONE
C

```

0181 IF(NSTRIP.LT.NTIM)GOTO 4
0182 CALL CLOSEC(ICHW)
0183 STOP 'NORMAL TERMINATION'
0184 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.

```
0001 DIMENSION IBUF(132,8),IBUFC(1056),INBUF(1024),FBUF(3)
0002 REAL*8 FSPECR
0003 EQUIVALENCE( IBUF(1),IBUFC(1) )
0004 DATA DEV/3RRK /
0005 WRITE(7,1001)
0006 1001 FORMAT(' ENTER FILE NAME TO BE PROCESSED:',S)
0007 READ(5,1000)FBUF
0008 1000 FORMAT(3A4)
0009 CALL IRAD50(12,FBUF,FSPECR)
0010 IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
0011 IDCH=IGETC()
0012 IF(LOOKUP(IDCH,FSPECR).LT.0)STOP'LOOKUP ERR'
0013 CALL MTXSET
0014 CALL MTX( IBUF,0,1)
0015 IBLK=1
0016 10 IN=IREADW(1024,INBUF,IBLK,IDCH)
0017 IBLK=IBLK+4
0018 IF(IN.EQ.1024)GOTO 20
0019 IF(IN.LT.-1)STOP'READ ERR'
0020 IF(IN.LE.0)GOTO 40
0021 20 IPOS=1
0022 IROW=1
0023 NROW=IN/128
0024 IF((IN-(NROW*128)).NE.0)NROW=NROW+1
0025 NWORD=NROW*132
0026 CALL MWAIT
0027 DO 5 I=1,1056
0028 5 IBUFC(I)=0
0029 DO 30 I=1,IN
0030 IBUF(IPOS,IROW)=INBUF(I)
0031 IPOS=IPOS+1
0032 IF(IPOS.LE.128)GOTO 30
0033 IPOS=1
0034 IROW=IROW+1
0035 30 CONTINUE
0036 CALL MTX( IBUF,NWORD,2)
0037 IF(IN.EQ.1024)GOTO 10
0038 CALL MWAIT
0039 40 CALL MTX( IBUF,-1,1)
0040 CALL MWAIT
0041 STOP'NORMAL TERMINATE'
0042 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

```

C   M.J.POULTER SEPT 79
C   POST PROCESS AND MERGE PROG FOR
C   SEISMIC PLOT SYSTEM
0001  INTEGER*2 IPBUF(2112),INBUF(8192),LBUF(132),IOFF(8),
      %LST(8),LFIN(8),IBLK(8),NBUF(8),NPLIN(8),ISPOS(8),ICOM(8),
      %ILOG(8),ICHAN(8),EOF(8)
      REAL*4 FBUF(3)
      REAL*8 FSPEC(8)
0004  COMMON /LFIL/ INBUF,IOFF,LBUF,IBLK,NBUF,NPLIN,ISPOS,EOF,ICHAN,ICOM
0005  DATA IOFF/1,1025,2049,3073,4097,5721,6145,7169/
0006  DATA DEV/3RRK /
C
C   SET UP AP TO USE FOR ZEROING ARRAYS
C
0007  CALL APINIT
0008  CALL VCLR(0,1,1056)
C
C   SET UP I/O DEFINITIONS
C
0009  IF(IFETCH(DEV).NE.0)STOP 'FETCH ERR'
0011  IF(ICDFN(30).NE.0)STOP 'CHAN DEF ERR'
0013  CALL ASSIGN(1,'DK1:MPMERD.DAT',14)
C
C   GET INPUT PARAMETERS
C
0014  READ(1,10) NPLT,LPLT,NSTRIP
0015  10  FORMAT(3I5)
0016  IF(NPLT.GT.8)STOP 'TOO MANY MASKS'
0018  DO 20 I=1,NPLT
0019  READ(1,10)LST(I),LFIN(I)
0020  READ(1,10)ICOM(I),ILOG(I)
0021  READ(1,10)NPLIN(I)
0022  READ(1,30)FBUF
0023  30  FORMAT(3A4)
0024  WRITE(7,*) NPLT,LPLT,LST(I),LFIN(I),ICOM(I),ILOG(I)
0025  WRITE(7,*)NPLIN(I)
0026  WRITE(7,30)FBUF
0027  CALL IRAD50(12,FBUF,FSPEC(I))
0028  ICHAN(I)=20+I
0029  IBLK(I)=1
0030  EOF(I)=0
0031  NBUF(I)=1023+IOFF(I)
0032  ISPOS(I)=NBUF(I)
0033  IF(LOOKUP(ICHAN(I),FSPEC(I)).LT.0)STOP 'LOOKUP ERR'
0035  20  CONTINUE
C
C   SET UP MATRIX FOR MAIN LOOP
C
0036  CALL MTXSET
0037  CALL MTX(IPBUF,0,1)
      LNUM=0
0039  NTIM=0
      IST=1057
C

```

```
C ZERO THE PLOT ARRAY
C
0041      40 IST=1056-IST+2
0042      CALL APGET(IPBUF(IST),0,1056,1)
0043      IPSV=IST
0044      CALL APWD
C
C START OF MAIN LOOP
C
0045      DO 50 I =1,8
0046      LNUM=LNUM+1
0047      EOFFLG=1
C
C THIS LOOP FILLS ONE LINE OF PLOT BUFFER
C WITH INPUT FROM EACH PLOT MASK
C
0048      DO 60 J=1,NPLT
0049      IF(LNUM.GT.LFIN(J))GOTO 60
0051      IF(EOF(J).NE.0)GOTO 60
0053      EOFFLG=0
0054      IF(LNUM.LT.LST(J))GOTO 60
0056      IPOS=IPSV
0057      ID=J
0058      CALL LINFIL(ID)
0059      IF(ILOG(J).NE.0)GOTO 70
0061      DO 80 L=1,132
0062      IPBUF(IPOS)=IPBUF(IPOS).OR.LBUF(L)
0063      IPOS=IPOS+1
0064      GOTO 60
0065      70 DO 90 L=1,132
0066      IPBUF(IPOS)=IPBUF(IPOS).AND.LBUF(L)
0067      IPOS=IPOS+1
0068      60 CONTINUE
0069      IPSV=IPOS
0070      50 CONTINUE
C
C WHEN A PLOT BUFFER IS FULL COME HERE TO EMPTY IT
C
0071      CALL MWAIT
0072      CALL MTX(IPBUF(IST),1056,2)
0073      IF(EOFFLG.NE.0)GOTO 100
0075      IF(LNUM.LT.LPLT)GOTO 40
0077      100 CALL MWAIT
0078      CALL MTX(IPBUF,0,1)
0079      CALL MWAIT
0080      NTIM=NTIM+1
0081      LNUM=0
0083      IF(NTIM.LT.NSTRIP)GOTO 40
0084      STOP 'NORMAL TERMINATION'
0085      END
```

```

0001      SUBROUTINE LINFIL(J)
0002      INTEGER*2 INBUF(8192),IOFF(8),LBUF(132),IBLK(8),NBUF(8),
0003      %NPLIN(8),ISPOS(8),EOF(8),ICHAN(8),ICOM(8)
0004      COMMON /LFIL/ INBUF,IOFF,LBUF,IBLK,NBUF,NPLIN,ISPOS,EOF,ICHAN,ICOM
0005
0006      C
0007      C ZERO LINE ARRAY
0008      C
0009      CALL APGET(LBUF,0,132,1)
0010      CALL APWD
0011      C
0012      C FILL LINE BUFFER FROM INPUT BUFFER
0013      C
0014      NPT=NPLIN(J)
0015      IPOS=ISPOS(J)
0016      NLIM=NBUF(J)
0017      IF(ICOM(J).NE.0)GOTO 20
0018      DO 10 I=1,NPT
0019      IPOS=IPOS+1
0020      IF(IPOS.LE.NLIM)GOTO 30
0021
0022      C
0023      C REFILL BUFFER FROM FILE WHEN EMPTY
0024      C
0025      IN=IREADW(1024,INBUF(IOFF(J)),IBLK(J),ICHAN(J))
0026      IBLK(J)=IBLK(J)+4
0027      IPOS=IOFF(J)
0028      IF(IN.EQ.1024)GOTO 30
0029      EOF(J)=1
0030      NLIM=IN+IPOS
0031      IF(IN.LT.-1)STOP'READ ERR'
0032      IF(IN.LE.0)RETURN
0033      30 LBUF(I)=INBUF(IPOS)
0034      10 CONTINUE
0035      GOTO 90
0036      20 DO 40 I=1,NPT
0037      IPOS=IPOS+1
0038      IF(IPOS.LE.NLIM)GOTO 50
0039
0040      C
0041      C REFILL BUFFER FROM FILE WHEN EMPTY
0042      C
0043      IN=IREADW(1024,INBUF(IOFF(J)),IBLK(J),ICHAN(J))
0044      IBLK(J)=IBLK(J)+4
0045      IPOS=IOFF(J)
0046      IF(IN.EQ.1024)GOTO 50
0047      EOF(J)=1
0048      NLIM=IN+IPOS
0049      IF(IN.LT.-1)STOP'READ ERR'
0050      IF(IN.LE.0)RETURN
0051      50 LBUF(I)=.NOT.INBUF(IPOS)
0052      40 CONTINUE
0053      90 NBUF(J)=NLIM
0054      ISPOS(J)=IPOS
0055      RETURN
0056      END

```

Appendix 2

Contained 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 DescriptionsTREAD

```
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

```

TWRIT

```
CALL TWRIT(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

```

SDS10

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 forward

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, IEOT)

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
DR$CSR=164000
DRB=164002
DO$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
        MOV      @4(R5),R0        ;GET NO OF BYTES REQUIRED
        MOV      RDLST,ARGLST     ;SET UP SDSI0 COMMAND
        MOVB     @8.(R5),ARGLT2
        JSR      PC,MSG           ;SEND COMMAND
;MAIN TRANSFER LOOP
1S:     TSTB     @#DR$CSR         ;WAIT TILL 8 READY
        BPL      1$
        MOVB     @#DRB,(R1)+      ;MOVE DATA FROM BUFFER TO TARGET
        DEC      R0
        BEQ      2$              ;EXIT LOOP IF COUNT COMPLETE
        TST      @#DR$CSR        ;TEST FOR END OF DATA
        BMI      3$              ;FROM TAPE
        BIS      #1,@#DR$CSR     ;KEEP ENABLE BIT SET
        BR       1$              ;LOOP BACK
;EXIT FOR COUNT COMPLETE
2S:     MOV      R0,@4(R5)        ;RETURN ZERO COUNTER
        TST      @#DR$CSR        ;TEST IF 8 IS FINISHED ALSO
        BPL      4$              ;IF NOT GO TO TAKE REST OF DATA
        DEC      R1              ;IF YES GET STATUS OUT OF DATA ARRAY
        MOVB     (R1),@6(R5)
        BR       6$              ;GOTO EOF TRANSFER CODE
4S:     MOVB     (R1),R2          ;SET UP STATUS SEARCH
7S:     TSTB     @#DR$CSR        ;WAIT FOR 8 READY
        BPL      7$
        TST      @#DR$CSR        ;TEST FOR EOF
        BPL      5$              ;IF NOT TRANSFER DATA
        MOVB     R2,@6(R5)        ;IF YES MOVE SAVED STATUS
        MOVB     @#DRB,R2        ;AND CLEAR BUFFER
        BR       6$              ;BEFORE EXITING
5S:     MOVB     @#DRB,R2        ;TAKE BYTE FROM 8
        BIS      #1,@#DR$CSR
        BR       7$              ;GOBACK AND TAKE MORE DATA
;EXIT FOR EOF SIGNALLED
3S:     MOV      R0,@4(R5)        ;RETURN NO OF BYTES LEFT
        DEC      R1
        MOVB     (R1),@6(R5)     ;RETURN STATUS
;COMMON EXIT
6S:     TSTB     @#DR$CSR
        BPL      6$
        MOVB     @#DRB,R2        ;TAKE LAST CLEAN UP BYTE
        RTS      PC              ;AND EXIT
;
;ROUTINE TO GO FROM MEMORY TO TAPE
; CALL TWRIT(BUFF,NYTE,STATUS,IBYTEPAD,NBLK,IDRV)
;
TWRIT:  MOV      2(R5),R1          ;GET DATA ADDRESS
        MOV      @4(R5),R0        ;GET NUM OF BYTES
        MOVB     WRTLST,ARGLST   ;SET UP COMMAND
        MOVB     @10.(R5),ARGLT1
        MOVB     @12.(R5),ARGLT2
        JSR      PC,MSG
;SET UP CODE
10S:    TSTB     @#DO$CSR
        BPL      10$            ;WAIT FOR SYNCHRONISATION

```

```

        BIC      #400,@#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      R0             ;DEC COUNTER
        BNE      11$            ;GO BACK FOR MORE
;ZERO PASSING LOOP
        MOV      @8.(R5),R2
        BEQ      EOFT          ;IF COUNT OF ZEROS=0 EXIT
14$:    BIS      #1,@#DOSCSR
        MOV      #0,@#DOB      ;PASS ZEROS AS PADDING
15$:    TSTB     @#DOSCSR      ;WAIT TILL ACCEPTED
        BPL      15$
        DEC      R2            ;DEC COUNTER
        BNE      14$
EOFT:   BIS      #400,@#DOSCSR
        MOV      #0,@#DOB      ;SET EOF BIT
;STATUS REPLY LOOP
16$:    TSTB     @#DR$CSR      ;SEE IF REPLY READY
        BPL      16$
        MOV      @#DRB,@6(R5)  ;GET STATUS
        MOV      #2,R0
17$:    TSTB     @#DR$CSR
        BPL      17$
        MOV      @#DRB,R2      ;CLEAR SYNCH ZEROS
        DEC      R0
        BNE      17$
        RTS      PC
;
;MESSAGE SENDING SUBROUTINE
;
MSG:    BIC      #400,@#DOSCSR      ;CLEAR EOF BIT
        MOV      #ARGLST,R3      ;GET ARGUMENT ADDRESS
        MOV      #3,R4
MLOOP:  BIS      #1,@#DOSCSR      ;SET ENABLE BIT
        CLR      R2
        MOV      (R3)+,R2      ;GET COMMAND INTO R2
        MOV      R2,@#DOB      ;MOV TO BUFFER
9$:     TSTB     @#DOSCSR      ;WAIT TILL ACCEPTED
        BPL      9$
        DEC      R4
        BNE      MLOOP
        BIS      #400,@#DOSCSR      ;SET EOF BIT
        MOV      #0,@#DOB      ;RETURN
        RTS      PC
;STORAGE AREA
ARGLST: .BYTE   0
ARGLT1: .BYTE   0
ARGLT2: .BYTE   0
WRTLST: .BYTE   6
        .EVEN
RDLST:  .WORD   403
        .EVEN
        .END

```

```

.TITLE SDS10
.GLOBL SDS10
.MCALL .READW,.WRITW,.EXIT,.PRINT

```

```

DR$CSR==164000
DOSCSR==164010
DRB==164002
DOB==164012
ERRBYT=52

```

```

SDS10:
MOV 2(R5),R1 ;COMMON ENTRY POINT
MOV #ARGLIST,R2 ;MOV ADDRESS OF COMMAND->R1
MOV #4,R0 ;PUT ARGLIST IN R2
1$: MOV (R1)+,(R2)+ ;PUT COUNTER IN R0
DEC R0 ;MOV COMMANDS TO ARGLIST
BNE 1$
TSTB FLAG ;TEST TYPE OF COMMAND
BMI READ ;GOTO APPROPRIATE SECTION
BEQ WIND
JMP WRITE
ARGLIST:.BYTE 0
.BYTE 0
.BYTE 0
FLAG:.BYTE 0

;END OF COMMON ENTRY POINT
;NEXT SECTION IS TAPE FAST READ

READ: CLR BLKN ;CLEAR BLOCK COUNTER
.WRITW #AREA,#20.,#BUFF,#256.,BLKN
JSR PC,MSG ;SET UP DISC POINTER
;AND SEND MESSAGE
RESTRT: MOV #BUFF,R4 ;PUT ADDR BUFF IN R4
MOV #2048.,R3 ;PUT COUNTER IN R3
DRLOOP: TSTB @#DR$CSR ;TEST IF BYTE READY
BPL DRLOOP ;IF NOT GO BACK AND TRY AGAIN
MOV @#DRB,(R4)+ ;PUT BYTE FROM INTERFACE TO BUFFER
DEC R3 ;DEC THE COUNTER
BEQ DRDONE ;IF 0 BUFFER FULL
TST @#DR$CSR ;TEST FOR EOF
BMI DREOF
BIS #1,@#DR$CSR ;KEEP ENABLE BIT SET
BR DRLOOP ;IF GOT HERE GO BACK FOR MORE

DRDONE: .WRITW #AREA,#20.,#BUFF,#1024.,BLKN
BCS WERR ;WRITE OUT BUFFER AND TEST FOR ERRORS
ADD #4,BLKN ;BUMP BLOCK COUNTER
BR RESTRT ;GO BACK TO FILL ANOTHER BUFFER

DREOF: TSTB @#DR$CSR ;COME HERE ON EOF
BPL DREOF ;TEST IF BYTE READY
MOV @#DRB,R1 ;MOVE OVER LAST TWO BYTES
DEC R4
MOV -(R4),@4(R5) ;MOVE STATUS
MOV -(R4),@6(R5) ;AND TIME LENGTH TO RETURN ARGLIST
ADD #2.,R4 ;RESET R4 POINTER
MOV #0,(R4) ;AND ZERO THE LAST BYTE IN BUFFER
SUB #BUFF,R4 ;GET NO OF BYTES
ASR R4 ;GET NO OF WORDS
BCC 2$ ;IF C CLEAR NO EVEN WRITE OUT
INC R4 ;ODD ADD ONE ON
2$: .WRITW #AREA,#20.,#BUFF,R4,BLKN
BCS WERR

```

```

3$:   INC      BLKN      ;NOW BY A ROUND ABOUT METHOD
      SUB      #512.,R4  ;FIND THE NO OF BLOCKS
      BGT      3$
      MOV      BLKN,@8.(R5) ;WHEN FOUND RETURN TO CALLING PROGRAM
      RTS      PC        ;RETURN FROM SUB
WERR: .PRINT   WMSG
      .EXIT

```

```

;NEXT SECTION RESPONSIBLE FOR WINDING THE TAPE ON
;WHEN THERE IS A SHORT FILE TO JUMP OVER

```

```

WIND: JSR      PC,MSG    ;TELL B/E WHAT IS REQUIRED
REPLY: TSTB    @#DR$CSR  ;AND GO THROUGH PROC
      BPL      REPLY    ;FOR RECEIVING A REPLY
      MOVB    @#DRB,@4(R5) ;WHICH IS RETURNED TO CALLING PROG
      MOV     #2,R0     ;TAKE LAST TWO SYNCHRO BYTES
5$:   TSTB    @#DR$CSR
      BPL      5$
      MOVB    @#DRB,R1
      DEC     R0
      BNE     5$
      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
9$:   TSTB    @#DOS$CSR  ;TEST MESSAGE RECIEVED
      BPL      9$
      BIC     #400,@#DOS$CSR ;CLEAR EOF
      BR      DODONE     ;START TRANSFER

WRSTRT: MOV     #BUFF,R4  ;MOVE BUFFER ADDR->R4
      ASL     R0          ;FIND NO OF BYTES TO TRANSFER
      BEQ     DODONE     ;IF 0 FINISHED
DOLOOP: BIS     #1,@#DOS$CSR ;SET ENABLE BIT
      MOVB    (R4)+,@#DOB  ;MOV FROM BUFF->DOB
6$:   TSTB    @#DOS$CSR  ;TEST IF READY FOR NEXT BYTE
      BPL      6$
      DEC     R0          ;DEC THE COUNTER
      BNE     DOLOOP     ;IF 0 GOTO FINISH

DODONE: TST     EOFW      ;TEST INTERNAL EOF FLAG
      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.,R0  ;SEE IF GOT A FULL BUFFER
      BEQ     WRSTRT    ;YES THEN TRANSFER
      BIS     #1000000,EOFW ;NO THEN SET EOF FLAG
      MOV     #1024.,R1  ;SET UP COMPLETION COUNTER
      SUB     R0,R1
      ASL     R1          ;CONVERT WORD TO BYTE COUNT
      BR     WRSTRT     ;AND WRITE OUT AMOUNT OF BUFFER REQUIRED

DOEFLP: BIS     #1,@#DOS$CSR
      MOV     #0,@#DOB   ;MOVE ZEROS TO OUTPUT
8$:   TSTB    @#DOS$CSR  ;TEST IF OK FOR MORE
      BPL      8$
      DEC     R1          ;DEC THE COUNTER
      BEQ     DOEOF     ;IF 0 GOTO END
      BR     DOEFLP

```

```

DOEOF:  BIS      #400, @D0$CSR   ;SET INTERFACE EOF FLAG
        MOV      #0, @D0B      ;CLEAR INTERFACE BUFFER
        BR       REPLY         ;GET REPLY
;THIS SECTION IS THE MESSAGE SENDING
;SUBROUTINE MSG AND THE BUFFER AND ERROR MESSAGE BLOCKS

MSG:    BIC      #400, @D0$CSR   ;CLEAR OUTPUT EOF FLAG
        MOV      #ARGLIST, R1   ;MOV ARGLIST ADDR TO R1
        MOV      #3, R0        ;MOV COUNTER ->R0
MLOOP:  BIS      #1, @D0$CSR     ;SET ENABLE BIT
        CLR      R2            ;CLEAR INTER BUFFER
        MOVB     (R1)+, R2      ;MOV FROM ARGLIST TO R2
        MOV      R2, @D0B      ;MOV R2->INTERFACE
7$:     TSTB     @D0$CSR        ;SEE IF OK FOR NEXT BYTE
        BPL      7$           ;
        DEC     R0            ;DEC THE COUNTER
        BNE     MLOOP         ;AND GO BACK FOR MORE IF NON0
        BIS     #400, @D0$CSR  ;OR IF 0 SET EOF FLAG AND
        MOV     #0, @D0B      ;FLUSH THE BUFFER
        RTS     PC            ;RETURN

RERR:   TSTB     @ERRBYT       ;TEST FOR TYPE OF ERROR
        BEQ     8$           ;
        .PRINT  #RMSG
        .EXIT
8$:     BIS      #400, @D0$CSR  ;SET UP ERROR FINISH
        MOV     #0, @D0B
        JMP     REPLY
WMSG:   .ASCIZ   / WRITE ERROR/
RMSG:   .ASCIZ   / READ ERROR /

```

; STORAGE AREA

```

AREA:   .BLKW   10
BLKN:   .WORD   0
BUFF:   .BLKW   1024.
EOFW:   .WORD   0
        .END

```

```

0007      SUBROUTINE TAPRED(ICOM,IDRV,ISTAT,ITLEN,ILEN,IFNUM,IEOT)
0008
0009      C TAPE HANDLING SUBROUTINE
0010      C ICOM IS THE COMMAND SIGNAL
0011      C -1 IS A READ,0 IS A WIND,1 IS AWRITE
0012      C IDRV IS THE DRIVE BEING USED
0013      C ISTAT IS THE STATUS ON RETURN
0014      C ITLEN IS THE TIME LENGTH OF A FILE READ
0015      C ILEN IS THE BLOCK LENGTH OF A FILE READ OR WRITTEN
0016
0017      INTEGER*2 MASK(8),ESTATI
0018      LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
0019      %IFLEN,ESTAT,ERRS(8)
0020      DATA MASK/"1","2","4","10","20","40","100","200"/
0021      DATA SDSCOM/"0","1","2","3","4","5","6","7"/
0022      DATA ERRS/"377","377","377","377","377","377","377","377"/
0023      ITRY=0
0024      IF(ICOM) 10,30,20
0025
0026      C SECTION CONTROLLING A READ
0027
0028      C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
0029
0030      10 ITRY=ITRY+1
0031
0032      C SET UP COMMAND FOR READ
0033
0034      COM(1)=SDSCOM(4)
0035      COM(2)=1
0036      COM(3)=IDRV
0037      COM(4)=-1
0038      CALL SDS10(COM,ISTAT,ITLEN,ILEN)
0039      IF(ISTAT.EQ.0)RETURN
0040
0041      C ERROR DETECTED ON READ
0042
0043      ISTAT1=ISTAT
0044      GOTO 40
0045
0046      C IF SHORT RECORD FOUND REREAD TAPE
0047
0048      50 ITMP=ISTAT1.AND.MASK(6)
0049      IF(ITMP.NE.0)GOTO 10
0050      ITMP=ISTAT1.AND.MASK(2)
0051      IF(ITMP.EQ.0)RETURN
0052
0053      C IF CRC ERROR FOUND REWIND TAPE AND RETRY
0054
0055      WRITE(2,2010)IFNUM
0056      2010 FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
0057      IF(ITRY.GE.2)GOTO 130
0058      ECOM(1)=SDSCOM(6)
0059      ECOM(2)=1

```

```
0031      ECOM(3)=IDRV
0032      ECOM(4)=0
0033      CALL SDS10(ECOM,ESTAT, , )
0034      GOTO 10
C
C WRITE SECTION
C
0035      23 ITRY=ITRY+1
0036      IF(ITRY.GT.3)GOTO 130
0037      COM(1)=SDSCOM(7)
0038      IFLEN=(ILEN+3)/4
0039      COM(2)=IFLEN
0040      COM(3)=IDRV
0041      COM(4)=1
0042      CALL SDS10(COM,ISTAT, , )
0043      IF(ISTAT.EQ.0)RETURN
C
C WRITE ERROR DETECTED
C
0044      ISTAT=ISTAT
0045      GOTO 40
0046      73 ITMP=ISTAT.AND.MASK(6)
0047      ITMPI=ISTAT.AND.MASK(2)
0048      IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
C
C REPORT AND RETRY
C
0052      WRITE(2,2020)IFNUM
0053      2020 FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
0054      ECOM(1)=SDSCOM(6)
0055      ECOM(2)=2
0056      ECOM(3)=IDRV
0057      ECOM(4)=0
0058      CALL SDS10(ECOM,ESTAT, , )
0059      NBUF=8
0060      IPAD=32760
0061      IFLENE=16
0062      CALL TWRTT(ERRS,NBUF,ESTAT,IPAD,IFLENE,IDRV)
0063      GOTO 20
C
C WIND FOWARD ONE FILE
C
0064      33 COM(1)=SDSCOM(5)
0065      COM(2)=1
0066      COM(3)=IDRV
0067      COM(4)=0
0068      CALL SDS10(COM,ISTAT, , )
C
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
C
0070      ISTAT=ISTAT.AND..NOT.MASK(6)
0071      IF(ISTAT.EQ.2)RETURN
0072      ISTAT=ISTAT
0073      IF(ISTAT.NE.0)GOTO 40
```

```

C
C IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
0075      ECOM(1)=SDSCOM(6)
0076      ECOM(2)=1
0077      ECOM(3)=IDRV
0078      ECOM(4)=0
0079      CALL SDS10(ECOM,ESTAT, , )
0080      35 RETURN

C
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS:= TAPE BUSY,TAPE OFFLINE
C BOT,EOT

C
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
0081      40 WRITE(2,1010)ISTATI,IFNUM
0082      1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
0083      ISTATI=ISTATI.AND..NOT.MASK(4)
0084      ITMP=ISTATI.AND.MASK(5)
0085      IF(ITMP.EQ.0)GOTO 80
0086      90 ECOM(1)=SDSCOM(1)
0087      ECOM(2)=0
0088      ECOM(3)=IDRV
0089      ECOM(4)=0
0090      CALL SDS10(ECOM,ESTAT, , )
0091

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

0092      ESTATI=ESTAT
0093      ITMP=ESTATI.AND.MASK(5)
0094      IF(ITMP.NE.0)GOTO 90
0095      IF(ICOM) 10,30,20

C
C TAPE OFFLINE
0096      80 ITMP=ISTATI.AND.MASK(1)
0097      IF(ITMP.EQ.0)GOTO 100
0098      TYPE 1001,IDRV
0099      1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')

C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED

0100      110 ECOM(1)=SDSCOM(1)
0101      ECOM(2)=0
0102      ECOM(3)=IDRV
0103      ECOM(4)=0
0104      CALL SDS10(ECOM,ESTAT, , )
0105      ESTATI=ESTAT
0106      ITMP=ESTATI.AND.MASK(1)
0107      IF(ITMP.NE.0)GOTO 110
    
```

```

0111      IF(ICOM) 10,30,20

C
C EOT
0112      100 ITMP=ISTATI.AND.MASK(3)
0113      IF(ITMP.EQ.0)GOTO 120
0114      TYPE 1002,IDRV
0115      1002 FORMAT(' EOT ON DRIVE ',I1)
0116      IEOT=-1
0117      RETURN
0118      120 IF(ICOM) 50,35,70

C
C ERROR EXIT RETURN
0120      130 ISTATI=-1
0121      RETURN
0122      END
    
```

```

SUBROUTINE TAPSUB(ICOM,IDRV,ISTAT,ILEN,IFNUM,BUF,NBYT,IEOT)
C
C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C -1 IS A READ,0 IS A WIND,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","10","20","40","100","200"/
      DATA SDSCOM/"0","1","2","3","4","5","6","7"/
      DATA ERRS/"377","377","377","377","377","377","377"/
      ITRY=0
      IF(ICOM) 10,30,20
C
C SECTION CONTROLLING A READ
C
C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
      10 ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
      NBUF=NBYT
      CALL TREAD(BUF,NBUF,ISTAT,IDRV)
      IF(ISTAT.EQ.0)RETURN
C
C ERROR DETECTED ON READ
      ISTAT=ISTAT
      GOTO 40
C
C IF SHORT RECORD FOUND REREAD TAPE
      50 ITMP=ISTATI.AND.MASK(6)
      IF(ITMP.NE.0)GOTO 10
      ITMP=ISTATI.AND.MASK(2)
      IF(ITMP.EQ.0)RETURN
C
C 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
      ECOM(3)=IDRV
      ECOM(4)=0
      CALL SDS10(ECOM,ESTAT, , )
      GOTO 10
C
C WRITE SECTION
      20 ITRY=ITRY+1
      IF(ITRY.GT.3)GOTO 130
      NBUF=NBYT
      IFLEN=(ILEN+3)/4
      IF(IFLEN.LT.2)IFLEN=2
      IPAD=(IFLEN*2048)-NBUF

```

```

CALL TWRTIT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
IF(ISTAT.EQ.0)RETURN
C
C WRITE ERROR DETECTED
C
    ISTATI=ISTAT
    GOTO 40
70 ITMP=ISTATI.AND.MASK(6)
    ITMPI=ISTATI.AND.MASK(2)
    IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
C
C REPORT AND RETRY
C
    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 SDS10(ECOM,ESTAT, , )
    NBUF=8
    IPAD=(IFLEN*2048)-NBUF
    CALL TWRTIT(ERRS,NBUF,ESTAT,IPAD,IFLEN,IDRV)
    GOTO 20
C
C WIND FOWARD ONE FILE
C
    30 COM(1)=SDSCOM(5)
    COM(2)=1
    COM(3)=IDRV
    COM(4)=0
    CALL SDS10(COM,ISTAT, , )
C
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
C
    ISTAT=ISTAT.AND..NOT.MASK(6)
    IF(ISTAT.EQ.2)RETURN
    ISTATI=ISTAT
    IF(ISTAT.NE.0)GOTO 40
C
C IF ISTAT=0 REWIND AND SET UP FOR NEXT READ
C
C AS THIS WAS A DATA FILE NOT A SHORT RECORD
C
    ECOM(1)=SDSCOM(6)
    ECOM(2)=1
    ECOM(3)=IDRV
    ECOM(4)=0
    CALL SDS10(ECOM,ESTAT, , )
    35 RETURN
C
C IN THIS SECTION THE MAIN TAPE ERRORS ARE
C HANDLED SUCH AS:= TAPE BUSY,TAPE OFFLINE
C BOT,EOT
C
C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
C
    40 WRITE(2,1010)ISTATI,IFNUM
1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
    ISTATI=ISTATI.AND..NOT.MASK(4)
    ITMP=ISTATI.AND.MASK(5)
    IF(ITMP.EQ.0)GOTO 80
    90 ECOM(1)=SDSCOM(1)
    ECOM(2)=0

```

```

      ECOM(3)=IDRV -
      ECOM(4)=0
      CALL SDS10(ECOM,ESTAT, , )
C
C HAVING EXAMINED STATUS IF TAPE STILL
C BUSY, LOOP AGAIN,IF NOT TRY COMMAND AGAIN
C
      ESTATI=ESTAT
      ITMP=ESTATI.AND.MASK(5)
      IF(ITMP.NE.0)GOTO 90
      IF(ICOM) 10,30,20
C
C TAPE OFFLINE
C
      80 ITMP=ISTATI.AND.MASK(1)
      IF(ITMP.EQ.0)GOTO 100
      TYPE 1001,IDRV
1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
      110 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
      IF(ICOM) 10,30,20
C
C EOT
C
      100 ITMP=ISTATI.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
C ERROR EXIT RETURN
C
      130 ISTATI=-1
      RETURN
      END

```

Floating Point TransfersGETNO

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
DR$CSR=164000
DRB=164002
DO$CSR=164010
DOB=164012

GETNO:  MOV      (R5)+,R0
        MOV      (R5)+,R1
        MOV      R1,R2
        MOV      #4,R0
1$:     TSTB     @#$CSR
        BPL      1$
        MOVB     @#$RB,(R1)+
        BIS      #1,@#$CSR
        DEC      R0
        BNE      1$
        SWAB     (R2)+
        SWAB     (R2)+
        RTS      PC

GETDAT: MOV      (R5)+,R0
        MOV      (R5)+,R0
        MOV      (R5)+,R1
        MOV      R1,R2
        ASL      @R0
2$:     TSTB     @#$CSR
        BPL      2$
        MOVB     @#$RB,(R1)+
        BIS      #1,@#$CSR
3$:     TSTB     @#$CSR
        BPL      3$
        MOVB     @#$RB,(R1)+
        BIS      #1,@#$CSR
        SWAB     (R2)+
        DEC      @R0
        BNE      2$
        RTS      PC

SENDAT: MOV      (R5)+,R0
        MOV      (R5)+,R0
        MOV      (R5)+,R1
        MOV      (R5)+,R2
        MOV      #2,R3
        CLR      R4
        ASL      @R0
4$:     SWAB     (R1)
        BIS      #1,@#$CSR
        MOVB     (R1)+,R4
        MOV      R4,@$DOB
5$:     TSTB     @#$CSR
        BPL      5$
        BIS      #1,@#$CSR
        MOVB     (R1)+,R4
        MOV      R4,@$DOB
6$:     TSTB     @#$CSR
        BPL      6$
        DEC      R3
        BNE      4$
7$:     SWAB     (R2)
        BIS      #1,@#$CSR
        MOVB     (R2)+,R4
        MOV      R4,@$DOB
8$:     TSTB     @#$CSR

```

```
          BPL      8$
          BIS      #1, @D$CSR
          MOVB    (R2)+, R4
9$:      MOV      R4, @D$OB
          TSTB   @D$CSR
          BPL     9$
          DEC     @R0
          BNE     7$
          RTS     PC
          .END
```

Extended Memory Input/OutputAP 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 preceding call to AP^GAD.

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 preceding 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


```
DEC R0          ;SEE IF DONE??
BNE LDSP
```

```
;
;
;C
;C PUT THE STARTING ADDRESS INTO TMA, START BOOTSTRAP AT 4,
;C WITH BREAK ON PSA ENABLED AND BREAK IN SWR
```

```
;20 CALL APOUT(STRT,1)
; CALL APOUT(515,2)
; CALL RUNAP(4,0,BRKLOC,8448)
; RETURN
```

```
;
;
;SBRGO: 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 @#SWR             ;ZERO APSTAT, CLEAR PARITY ENANLE
MOV #518.,@#FN        ;DEP TO APSTAT
MOV @(R5)+,@#SWR      ;SET PSA BREAKPOINT
MOV #8448.,@#FN       ;AND GO
RETURN
```

```
;
;
ZERO: 0
;
; END
```

```
;C
; /***** ABORT = ABORT AP-EXECUTION = REL 2.0 , NOV 77 *****/
; /***** APRSET = RESET THE AP = REL 2.0 , NOV 77 *****/
```

```
;C
; SUBROUTINE ABORT
```

```
;C
; STOPS ANY TRANSFER, AND/OR RUN IN PROGRESS, RESETS INTERFACE AND
;C CLEANS UP ANY SOFTWARE STATE INDICATORS
```

```
;C
; ROUTINES USED: APOUT
```

- ;C 1. DO AN INTERFACE RESET (ORESET)
- ;C 2. CLEAR THE CONTROL REGISTER (CTRL(0))
- ;C 3. DO AN INTERNAL RESET (OFN(2048))

```
; CALL APOUT(0,10)
; CALL APOUT(0,7)
; CALL APOUT(2048,2)
; RETURN
```

```
;
;
; APRSET:
ABORT: CLR @#ABRT
CLR @#CTRL
MOV #4000.,@#FN
CLR @#LITES
RETURN
```

```
;
;
; END
```

```
;C
; /***** RUNDMA = START A DMA TRANSFER = REL 2.0 , NOV 77 *****/
```



```

;
; 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 PPARAMETER TO 1
        RETURN
;
; /***** APRLSE = RELEASE THE AP = REL 2.0 , NOV 77 *****/
;
; APRLSE IS A 'NOP' UNDER RT-11
;
APRLSE: RETURN
;
; /***** APIENA = INABLE INERRUPT = REL 2.0 , NOV 77 *****/
;
; NO OP UNDER RT11
;
APIENA: RETURN
;
; /***** APIDIS = DISABLE INTERRUPT = REL 2.0 , NOV 77 *****/
;
; NO OP UNDER RT11
;
APIDIS: RETURN
;
; /***** TSTINT = TEST FOR INTERRUPT = REL 2.0 , NOV 77 *****/
;
; NO OP UNDER RT11
;
TSTINT: RETURN
;
; /***** APWI = WAIT FOR INTERRUPT = REL 2.0 , NOV 77 *****/
;
; NO OP UNDER RT11
;
APWI: RETURN
;
; /***** APIN = INPUT AN AP-120B INTERFACE REGISTER = REL 2.0 , NOV 77 *****/
;C
; SUBROUTINE APIN(DATA,NUM)
; INTEGER DATA,NUM
;C
; READ THE CONTENTS OF INTERFACE REGISTER NUMBER 'NUM' INTO 'DATA'
;C
; PARAMETERS:
;C DATA - RECEIVES THE CURRENT CONTENTS OF REGISTER 'NUM'
;C NUM - SPECIFIES WHICH AP-120B INTERFACE REGISTER IS TO BE READ:
;C 1. SWR SWITCH REGISTER
;C 2. FN FUNCTION REGISTER
;C 3. LITES LITES REGISTER
;C 4. APMA AP DMA MEMORY ADDRESS REGISTER
;C 5. HMA HOST DMA MEMORY ADDRESS REGISTER
;C 6. WC DMA WORD COUNT REGISTER
;C 7. CTRL DMA CONTROL REGISTER
;C 8. FMTH FORMAT HIGH REGISTER
;C 9. FMTL FORMAT LOW REGISTER
;C 10. RESET DO AN EXTERNAL RESET (NO-OP FOR APIN)
;C 11. IFSTAT INTERFACE STATUS REGISTER (APIN READS, APOUT NO-OP)
;C 12. MASK MEMORY PROTECTION AND I/O BITS

```

```

;C          13. APMAE  AP PAGE SELECT
;C          14. MAE    DMA PAGE SELECT
;C
;C  ROUTINES USED: NONE
;C
;C-----NOTE: THIS ROUTINE WILL TYPICALLY BE IN ASSEMBLY LANGUAGE
;C          SINCE FORTRAN CANNOT OUTPUT DIRECTLY TO AN I/O DEVICE
;C
;C          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         ;IF EXTENDED MEMORY REGISTER READ
      BLT IMASK
;
APIN1: ASL R1            ;CONVERT TO BYTES
      MOV @TABLE(R1),@2(R5) ;GET FROM APPROPRIATE DEVICE ADDRESS
      RETURN
;
;          READ MASK OR APMAE OR MAE
;
;          THE EXTENDED MEMORY REGISTERS(12,13 AND 14 - SEE ABOVE) CAN NOT BE
;          READ OR WRITTEN INDIVIDUALLY. A READ OF THE RESET REGISTER WILL
;          RENDER THE MAE(BITS 0-3),APMAE(BITS 4-7) AND THE MASK(BITS 8-13).
;          A WRITE OF THE LITES REGISTER WILL SET THE MAE,APMAE AND THE MASK.
;          THE WRITE FORMAT IS THE SAME AS THAT OF THE READ.
;
;          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
      RETURN
;
IAPMAE: CMP #13.,R1     ;CHECK FOR APMAE
      BNE IMAE          ;IF MAE
      ASR R2            ;RIGHT JUSTIFY APMAE
      ASR R2
      ASR R2
      ASR R2
;
IMAE:  BIC #177760,R2   ;CLEAR ALL BUT MAE OR APMAE
      MOV R2,@2(R5)    ;RETURN REGISTER VALUE
      RETURN
;
;
;          END
;C
;
;
;          TABLE: 0
;          FPS+110   ;SWR
;          FPS+112   ;FN
;          FPS+114   ;LITES
;          FPS+106   ;APMA

```



```
:  
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  
:  
        .END
```

```
.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,.DRFIN,.QELDF
..V2..
.REGDEF
.IIF NDF TIMSIT,TIMSIT=0
.IIF NDF MMGST,MMGST=0
.IIF NDF ERLSG,ERLSG=0
.QELDF
.GLOBL DPSYS, DSSYS, DXSYS, DMSYS
.GLOBL RKSYS, RFSYS, DTSYS
DTSYS = 0
DSSYS = 0
DXSYS = 0
DPSYS = 0
RFSYS = 0
DMSYS = 0
RKDS = 177400
RKER = 177402
RKCS = 177404
RKWC = 177406
RKBA = 177410
RKDA = 177412
RKCNT = 10
RKSTS = 100000
RKDSIZ = 11300
RKIDEN = 0
RKIDS = 377
RKRCNT = 4000
RKNREG = 7
RKREGA = 177400
.DRBEG RK,220,RKDSIZ,RKSTS
.IF EQ MMGST
.IFTF
RETRY: MOV #RKCNT,(PC)+
MOV RKCQE,R5
MOV @R5,R2
MOV 2(R5),R4
ASR R4
ASR R4
ASR R4
SWAB R4
BIC #^C<160000>,R4
BR 2$
1$: ADD R2,R4
ASR R2
ASR R2
ADD R3,R2
2$: MOV R2,R3
BIC #177760,R3
BIC R3,R2
BNE 1$
CMP #12.,R3
BGT 3$
ADD #4,R3
3$: ADD R3,R4
MOV R4,DISKAD
AGAIN: MOV RKCQE,R5
MOV #103,R3
MOV #RKDA,R4
MOV (PC)+,@R4
DISKAD: 0
CMP (R5)+,(R5)+

```

```

.IFT
.IFF      MOV      (R5)+,-(R4)
.IFF      JSR      PC,@SMPPTR
          MOV      (SP)+,-(R4)
.IFTF
          MOV      (R5)+,-(R4)
          BEQ      7$
          BMI      5$
          NEG      @R4
          ADD      #2,R3
5$:
          MOV      @#RKCS,-(SP)
          BIC      #177717,(SP)
          BIS      (SP)+,R3
.IFF      BIS      (SP)+,R3
.IFTF
6$:      MOV      R3,-(R4)
          RTS      PC
7$:      MOV      #111,R3
          BR       5$
          .DRAST  RK,5
          MOV      #RKER,R5
          MOV      (R5)+,R4
          TST      RETRY
          BPL      NORMAL
          TST      @R5
          BMI      NORMAL
          BIT      #20000,@R5
          BEQ      RTSPC
          .FORK   RKFBLK
RKRETR:  CLRB    RETRY+1
          BR       AGAIN
NORMAL:  CMP      @R5,#310
          BEQ      RTSPC
          TST      @R5
          BPL      DONE
          .FORK   RKFBLK
.IF NE  ERLSG
          BIT      #62340,R4
          BNE      RKERR
          MOV      PC,R5
          ADD      #RKRBUF-.,R5
          MOV      R5,R2
          MOV      #RKREGA,R3
          MOV      #RKNREG,R4
RKREG:  MOV      (R3)+,(R5)+
          DEC      R4
          BNE      RKREG
          MOV      #RKNREG,R3
          ADD      #RKRCNT,R3
          MOV      RKCQE,R5
          MOVB    RETRY,R4
          DEC      R4
          JSR      PC,@$ELPTR
          MOV      #RKER,R5
          MOV      (R5)+,R4
.ENDC
RKERR:  MOV      #1,@R5
3$:      TSTB    @R5
          BPL      3$
          DECB   RETRY
          BEQ      HERROR

```

```

        BIT      #1100000,R4
        BEQ      RKRETR
        MOV      DISKAD,@#RKDA
        MOV      #115,@R5
        BIS      #1000000,RETRY
RTSPC:  RTS      PC
HERROR: MOV      RKCQE,R5
        BIS      #1,@-(R5)
        .IF NE  ERLSG
        BR      RKEXIT
DONE:   .FORK   RKFBLK
        MOV      #RKIDS,R4
        MOV      RKCQE,R5
        JSR     PC,@$ELPTR
        .IFF
DONE:
        .ENDC
RKEXIT: CLR      RETRY
        .DRFIN  RK
        .ENDC
RKFBLK: .WORD   0,0,0,0
        .IF NE  ERLSG
RKRBUF: .BLKW   RKNREG
        .ENDC
        .DREND  RK
        .END

```

```

.TITLE MPAPEX
.ENABL GBL
.GLOBL APGAD, APPUTX, APGETX, APGETA, APPUTA

```

```

USP0=177640
AP=176000
WC=AP+100
HMA=AP+102
CTRL=AP+104
APMA=AP+106
LITES=AP+114

```

```

APGAD:  MOV    @#USP0, R0          ;GET PAR 0 BLOCK OFFSET
        ASL    R0                  ;
        ASL    R0                  ;GET HIGHBITS INTO PLACE
        ASL    R0                  ;
        ASL    R0                  ;
        MOV    R0, HIGHBT          ;SAVE INTO HIGHBITS
        BIC    #37777, HIGHBT     ;CLEAR OTHER BITS
        ASL    R0                  ;SET UP 16 BIT PART
        ASL    R0                  ;
        BIC    #177700, R1        ;CLEAR UNWANTED BITS
        BIC    #77, R0            ;
        BIS    R1, R0             ;FORM 16 BIT PART
        MOV    R0, LOWBIT         ;AND SAVE IT
        MOV    HIGHBT, R0        ;RETURN FUNCTION VALUES
        MOV    LOWBIT, R1        ;IN R0 + R1
        RTS    PC                 ;RETURN

APPUTA:  MOV    8.(R5), R0        ;GET ADDRESS OF STORE
        MOV    (R0)+, HIGHBT     ;GET HIGHBITS FROM STORE
        MOV    (R0), LOWBIT      ;GET LOWBITS

APPUTX:  MOV    R5, -(SP)        ;SAVE R5
        JSR    PC, APWD          ;CALL APWD
        MOV    (SP)+, R5        ;RESTORE R5
        MOV    #193., R0        ;SET UP CONTROL WORD
        BR     COMMON           ;FOR A PUT AND GOTO COMMON

APGETA:  MOV    8.(R5), R0        ;GET STORE ADDRESS
        MOV    (R0)+, HIGHBT     ;GET HIGHBITS
        MOV    (R0), LOWBIT      ;AND LOWBITS

APGETX:  MOV    R5, -(SP)        ;SAVE R5
        JSR    PC, APWD          ;CALL APWD
        MOV    (SP)+, R5        ;RESTORE R5
        MOV    #225., R0        ;SET UP CONTROL WORD
        BR     COMMON           ;FOR A GET

COMMON:  MOV    @2(R5), @#APMA    ;SET AP ADDRESS
        MOV    @4(R5), R1        ;GET WORD COUNT
        ADD    @6(R5), R0        ;SET UP THE FORMAT
        ADD    @6(R5), R0        ;FOR THE TRANSFER
        BIT    R0, #4            ;AND SEE
        BNE    1$                ;IF THE WORD COUNT
        BIT    R0, #2            ;NEEDS DOUBLING
        BNE    2$                ;AS FOR THE REAL*4
1$:      ASL    R1                ;TRANSFERS
2$:      MOV    R1, @#WC          ;THEN PUT IN WORD COUNT
        MOV    LOWBIT, @#HMA     ;PUT IN LOW 16 BITS
        MOV    HIGHBT, @#LITES   ;THEN HIGH 2 BITS
        BIC    #174000, R0       ;OF PDP11 ADDRESS
        MOV    R0, @#CTRL        ;DISENABLE INTERRUPTS
        RTS    PC                 ;SETUP CTRL TO START PROCESS

LOWBIT:  .WORD 0
HIGHBT:  .WORD 0
.END

```

```

.MCALL .READC, .WRITC, .EXIT, .PRINT
.GLOBL ADGET, IREADX, IWRTX, IREADA, IWRITA
USPØ=17764Ø
RKCS=1774Ø4

ADGET:  MOV    @#USPØ, RØ           ;GET EXTENSION IN RØ
        ASL    RØ                   ;GET HIGHBITS INTO PLACE
        ASL    RØ
        MOV    RØ, R2               ;SAVE IN R2
        ASL    RØ                   ;GET LOWBITS INTO PLACE
        ASL    RØ
        ASL    RØ
        BIC    #1777ØØ, R1          ;CLEAR UNWANTED BITS
        BIC    #77, RØ              ;IN R1 + RØ
        BIS    R1, RØ
        MOV    RØ, LOWBIT           ;SAVE LOWBITS
        SWAB   R2                   ;GET HIGHBITS INTO
        BIC    #177717, R2          ;CORRECT PLACE
        MOV    R2, HIGHBT           ;SAVE IN HIGHBITS
        MOV    HIGHBT, RØ           ;PUT HIGHBITS IN RØ
        MOV    LOWBIT, R1           ;LOWBITS IN R1 TO REURN FUNCTION
        RTS    PC

IREADA: MOV    8.(R5), RØ           ;GET ADDRESS OF STORE
        MOV    (RØ)+, HIGHBT
        MOV    (RØ), 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
        BCS    ERROR
        RTS    PC

IWRITA: MOV    8.(R5), RØ
        MOV    (RØ)+, HIGHBT
        MOV    (RØ), LOWBIT

IWRITX: MOV    @2(R5), R1
        MOV    @4(R5), R3
        MOV    @6(R5), R4
        MOV    LOWBIT, R2
        BIS    HIGHBT, @#RKCS
        .WRITC #AREA, R1, R2, R3, #XMMCMP, R4
        BCS    ERROR
        RTS    PC

XMMCMP: BIC    #6Ø, @#RKCS
        RTS    PC
ERROR:  NEG    RØ                   ;GET ERROR INTO STANDARD FORTRAN TYPE
        SUB    #1, RØ
        RTS    PC

.EVEN
AREA:   .BLKW 1Ø
LOWBIT: .WORD Ø
HIGHBT: .WORD Ø
.END

```

Microcode RoutinesDemultiplex 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
        $ENTRY 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 0
        DBASE     $SEQU 1
        DINC      $SEQU 2
        GCNT      $SEQU 3
        DIR       $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          "SET UP CONSTANTS
        LDSPI TWO ;DB=2. ;FADD
        LDSPI N27 ;DB=27.
        LDSPI SYNC;DB=-1.
        LDSPI MASK;DB=200
        MOV GAIN,GAIN;SETMA                      "SET UP SAVE ON DPY
        SUB DINC,DBASE;LDDPA;DB=1.              "SET UP BASE ADDRESSES
        LDSPI N;DB=30.
        MOV DBASE,DADR;INCMA
PUSH:  DEC N;INCDPA;DPY<MD                      "MD-DPY SAVE LOOP
        INCMA;BGT PUSH
        MOV GAIN,GAIN;DPY<SPFN                  "SAVE GAIN ADDRESS
        CLR IADR;SETMA                          "SET UP MEMORY ACCESS
OUTLP: MOV MASK,TEMP                            "PUT MASK IN TEMP
        INC IADR;SETMA
        LDSPI DATA;DB=MD
        SUB SYNC,DATA                          "CHECK SYNC BITS
        LDSPI GAIN;DB=MD;BNE ERR1              "GOTO ERROR IF NE
        AND GAIN,TEMP
        LDSPI DIRCK;DB=1.;BNE SET              "CHECK DIRECTION BIT
        MOV SYNC,DIRCK
SET:   LDSPI TEMP;DB=31.
        AND TEMP,GAIN                          "CLEAR UNWANTED SYNC BITS
        SUB GAIN,TEMP
        ADD BIAS,GAIN;BEQ NEW                   "ADD BIAS TO GAIN
        INC GCNT;SETDPA
        LDSPI TEMP;DB=DPY                      "GET GAIN TO CHECK
        SUB GAIN,TEMP                          "CHECK GAIN AND SUBMUX GAIN
        SUB DIR,DIRCK;BNE ERR2                 "CHECK DIRECTION BIT
        LDDPA;DB=1.;BNE ERR3                  "SET UP FOR NEXT LOOP
        BR SKIP
ERR1:  LDSPI ERR;DB=1.                        "SET UP ERROR
        RETURN
ERR2:  MOV GAIN,GAIN;DPY<SPFN
        JMP     SKIP
ERR3:  COM DIR

```

```

INC DIR
JMP SKIP
NEW: CLR GCNT;LDDPA;DB=1.
SKIP: INC IADR;SETMA "HERE EVERY 30 TIMES
      LDSPI N;DB=30.
CONT: LDSPI GAIN;DB=DPY "START OF MAIN LOOP
      LDSPI DATA;DB=MD
      MOVR DATA,DATA "SHIFT DATA WORD R
      BZC SAME
      ADD DIR,GAIN
SAME: MOVL DATA,DATA;DPX<DB;DB=SPFN
      BGE NMI
      ADD TWO,DATA;DPX<DB;DB=SPFN
NMI: 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 DINC,DADR;SETMA;MI<FA;BGT CONT
      COM DIR
      INC DIR "TURN -1 TO +1 AND VV
      INC DBASE
      DEC N1
      MOV DBASE,DADR;BEQ FIN
      JMP OUTLP
FIN: LDDPA;DB=31.
      LDSPI TEMP;DB=DPY
      CLR ERR;LDDPA;DB=1.
      MOV TEMP,TEMP;SETMA;MI<DB;DB=DPY;INCDPA
      LDSPI N;DB=29.
POP: INCMA;MI<DB;DB=DPY;INCDPA;DEC N
      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 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 I,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 "STORE RES AND GO FOR MORE
      RETURN
      SEND

```

```

        $TITLE  FACTOR
        $ENTRY  FACTOR,13
        $EXT    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  0
    AINC   $SEQU  1
    B      $SEQU  2
    BINC   $SEQU  3
    C      $SEQU  4
    CINC   $SEQU  5
    L      $SEQU  6
    LINC   $SEQU  7
    U      $SEQU 10
    UINC   $SEQU 11
    N      $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     "SAVE L
        FSUBR FM,MD;ADD AINC,A;SETMA     "B-L*C GET NEXT A
        FADD;DEC N                       "PUSH ADDER AND DEC COUNTER
        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
SEND

```

```

$TITLE SOLVE
$ENTRY SOLVE,15
$EXT DIV
$EXT SPUFLT
" MPSOLV.APS
" THIS IS A ROUTINE TO SOLVE A TRIDIAGONAL
" MARIK SET OF EQNS 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 FACTORD 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      $EQU  0
LINC   $EQU  1
U      $EQU  2
UINC   $EQU  3
RHS    $EQU  4
RHSINC $EQU  5
C      $EQU  6
CINC   $EQU  7
Y      $EQU 10
YINC   $EQU 11
X      $EQU 12
XINC   $EQU 13
N      $EQU 14
"
" END OF ASSIGNMENTS BEGINNING OF MAIN INTRO
"
SOLVE:  MOV N,17          "GET N ONTO SPAD 15
        JSR SPUFLT       " FLOAT IT
        MOV RHS,RHS;SETMA " SET UP STARTING ADDRESSES
        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
"
LOOPA:  FMUL DPX,MD;ADD CINC,C;FADD          "INC C ADDR MANIP COUNTER
        FMUL;DPY<MD;ADD UINC,U             "PUSH MULT GET RHS
        FMUL;FSUBR TM,FA;ADD LINC,L;SETMA   "GET NEXT L
        FSUBR FM,DPY;ADD XINC,X            "RHS-L*Y
        FADD ZERO,FA;ADD RHSINC,RHS;SETMA  "PUSH ADDER GET NEXT RHS
        ADD YINC,Y;SETMA;MI<FA;DPX<FA;BFGT LOOPA " GET NEXT Y CHECK FOR LOP 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;DPY<DPX                      " DEC COUNTER GET Y(N)
        DPX<MD;ADD XINC,X                  " SET UP X(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)
        DPY<FA;DPX<MD;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

```


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.

```

0001      SUBROUTINE CONTUR(X, IX, Y, IY, Z, IDX, CZ, NC, PTR, SWCHES,
      +          MINDIS, NXL, XLOC, NYL, YLOC)
C     CONTUR 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, "CONTUR - A FORTRAN IV SUBROUTINE FOR PLOT-
C     ING CONTOUR LINES," BALLISTIC RESEARCH LABORATORIES MEMO-
C     RANDUM REPORT # 2282, ABERDEEN PROVING GROUND, MARYLAND,
C     MARCH, 1973. (NTIS ACCESSION NUMBER AD-760 437).
C     THIS ROUTINE COMPRISES 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
C     THE FOLLOWING CODE IS FOR VERSION 1.2 OF CONSYS, PRODUCED
C     15 MAY, 1976. GNC
C
0002      VIRTUAL Z(IDX, IY)
0003      REAL Z, X(IX), Y(IY), CZ(NC)
0004      REAL MINDIS, XLOC(NXL), YLOC(NYL)
0005      LOGICAL*1 SORTED, SWCHES(5)
0006      LOGICAL*1 LBLLOC, DISTOK
0007      INTEGER*2 PARERR, CONCNT, CCP1, NTEMP
0008      INTEGER*2 PL, PH
0009      INTEGER*2 PTR(NC)
C
0010      COMMON /CONCOM/ LOWER, UPPER, XL, XR, XC, YL, YU, YC,
      +          ZLL, ZUL, ZLR, ZUR
0011      INTEGER*2 LOWER, UPPER, ERRUNT
0012      DATA IOMAX/16/, ERRUNT/7/
C
C     CHECK FOR OBVIOUS ERRORS IN THE PARAMETERS.
C
0013      PARERR = 0
0014      IF( IX .GT. IDX ) PARERR = PARERR + 1
0015      IF( IX .LT. 2 ) PARERR = PARERR + 1
0016      IF( IY .LT. 2 ) PARERR = PARERR + 1
0017      IF( IDX .LT. 2 ) PARERR = PARERR + 1
0018      IF( NC .LT. 1 ) PARERR = PARERR + 1
0019      IF( PARERR .NE. 0 ) GO TO 986
0020      IF(.NOT. SWCHES(5))GOTO 1994
0021      IF(MINDIS.LT.0) PARERR = PARERR+1
0022      IF(NXL .LT. 0)PARERR=PARERR+1
0023      IF(NYL .LT. 0)PARERR=PARERR+1
0024      IF(NXL+NYL .LT. .1)PARERR=PARERR+1
0025      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
0038 1994 DO 880 I = 2, IX
0039      IF( X(I) .LE. X(I-1) ) GO TO 988
0041 880   CONTINUE
0042      DO 881 I = 2, IY
0043      IF( Y(I) .LE. Y(I-1) ) GO TO 988
0045 881   CONTINUE
C
C SORT THE ARRAY OF CONTOUR VALUES.
C
0046      IF( NC .EQ. 1 ) GO TO 5
0048      DO 1 M = 1, NC
0049      PTR(M) = M
0050 1     CONTINUE
0051      M = NC-1
0052 2     CONTINUE
0053      SORTED = .TRUE.
0054      DO 4 K = 1, M
0055      PL = PTR(K)
0056      PH = PTR(1+K)
0057      IF( CZ(PH) .GT. CZ(PL) ) GO TO 3
0058      PTR(K) = PH
0059      PTR(1+K) = PL
0061      SORTED = .FALSE.
0062 3     CONTINUE
0063 4     CONTINUE
0064      IF( SORTED ) GO TO 6
0065      M = M - 1
0067      IF( M .GE. 1 ) GO TO 2
0069      GO TO 6
0071 5     CONTINUE
0072      OFS = 0
0073      PTR(1) = 1
0074      GO TO 6
0076 6     CONTINUE
0077      CZMAX = CZ(PTR(NC))
0078      CZMIN = CZ(PTR(1) )
C
C BEGIN THE CONTOURING PROCESS BY LOOKING AT EACH CELL IN THE
C SURFACE IN TURN. FOR EACH CELL, WE ASK THE QUESTION, "DOES
C THIS CELL CONTAIN ANY CONTOUR LINES AT THE USER-SPECIFIED
C VALUES IN THE CZ ARRAY?" IF THE ANSWER IS NO, WE IMMEDIATELY
C PROCEED TO THE NEXT CELL. IF THE ANSWER IS YES, WE FIND
C THE LOWER AND UPPER LIMITS IN THE SORTED CZ ARRAY OF CONTOUR
C VALUES WHICH INTERSECT THIS CELL, AND PASS THIS INFORMATION
C AND THE CELL COORDINATES TO SUBROUTINES CTQQ (VIA CONCOM)
C WHERE THE COORDINATES FOR THE CONTOURS ARE PRODUCED.
C
0077      IYMI = IY - 1

```



```

+
          YLOC(M) .LT. YU) ) GO TO 26
0137          LBLLOC = .TRUE.
0138          GO TO 27
0139 26          CONTINUE
0140 27          CONTINUE
0141          IF( .NOT. LBLLOC ) GO TO 28
0143          IF( DISTOK(XC, YC,MINDIS) )
+          CALL NUMBER(XC, YC,0.1, Z0,0.0,4)
0145 28          CONTINUE
0146 29          CONTINUE
0147 37          CONTINUE
0148 38          CONTINUE
C DRAW BORDER LINES, IF THE USER INDICATED VIA THE SWCHES
C VECTOR THAT THEY ARE WANTED.
C
0149          XL = X(1)
0150          XR = X(IX)
0151          YL = Y(1)
0152          YU = Y(IY)
0153          IF( .NOT. SWCHES(1) ) GO TO 42
0155          CALL PLOT(XR, YL,+3)
0156          CALL PLOT(XL, YL,+2)
0157 42          CONTINUE
0158          IF( .NOT. SWCHES(2) ) GO TO 44
0159          CALL PLOT(XL, YL,+3)
0161          CALL PLOT(XL, YU,+2)
0162 44          CONTINUE
0163          IF( .NOT. SWCHES(3) ) GO TO 46
0165          CALL PLOT(XL, YU,+3)
0166          CALL PLOT(XR, YU,+2)
0167 46          CONTINUE
0168          IF( .NOT. SWCHES(4) ) GO TO 48
0170          CALL PLOT(XR, YU,+3)
0171          CALL PLOT(XR, YL,+2)
0172 48          CONTINUE
0173          RETURN
C
C
C HANDLE BAD PARAMETERS IN THE CONTUR CALL HERE.
C
C
0174 935          CONTINUE
0175          WRITE(ERRUNT, 997) PARERR, IX, IY, IDX, NC
0176          WRITE(ERRUNT, 999)
0177          CONTINUE
0178          STOP
C
0179 938          CONTINUE
0180          WRITE(ERRUNT, 994)
0181          WRITE(ERRUNT, 999)
0182          CONTINUE
0183          STOP
C
C FORMAT STATEMENTS FOR CONTUR ERROR COMMENTS.

```



```

0001      LOGICAL FUNCTION DISTOK(X, Y, MINDIS)
C      CONTUR CALLS DISTOK TO SEE WHETHER A CONTOUR LABEL CAN
C      BE PLACED AT (X, Y) AND BE MORE THAN MINDIS UNITS AWAY
C      FROM ANY OTHER LABEL PREVIOUSLY PLACED ON THE CONTOUR MAP.
C      IF A LABEL CAN BE SAFELY PLACED ON THE MAP, DISTOK RETURNS
C      THE VALUE .TRUE. AFTER SAVING THE VALUES OF X AND Y IN A
C      DYNAMICALLY ALLOCATED LOCAL ARRAY. IF THE LABEL WOULD
C      FALL WITHIN MINDIS UNITS FROM A PREVIOUS LABEL, DISTOK
C      SIMPLY RETURNS THE VALUE .FALSE.
C
C      THE FOLLOWING CODE IS FOR VERSION 1.2 OF CONSYS, PRODUCED
C      15 MAY, 1976.  GNC
C
0002      REAL X, Y, MINDIS
0003      REAL COORD(100)
0004      INTEGER*2 CURLEN
0005      DATA CURLEN/0/, NCOORD/100/
C
0006      DISTOK = .FALSE.
0007      IF( CURLEN .LT. 1 ) GO TO 9
0009      IF(CURLEN.GE.NCOORD)RETURN
0011      DO 8 I = 1, CURLEN, 2
0012          XI = COORD(I)
0013          IF( ABS(X-XI) .GT. MINDIS ) GO TO 5
0015          YI = COORD(1+I)
0016          IF( SQRT((X-XI)**2+(Y-YI)**2) .LE. MINDIS )
+          RETURN
0018      CONTINUE
0019      CONTINUE
0020      CONTINUE
C
0021      DISTOK = .TRUE.
0022      COORD(1+CURLEN) = X
0023      COORD(2+CURLEN) = Y
0024      CURLEN = CURLEN + 2
0025      RETURN
0026      END

```

```

0001      SUBROUTINE CTQQ(CZ, PTR, NC)
0002      C CTQQ IS CALLED FROM CONTUR TO PRODUCE CONTOUR COORDINATE
0003      C VALUES FOR THE GRID CELL BOUNDED BY XL AND XR, YL AND YU, AND
0004      C ZLL, ZUL, ZUR, AND ZLR. COORDINATE PAIRS THUS PRODUCED ARE
0005      C DISPOSED OF VIA CALLS TO THE USER-SPECIFIED ROUTINES PLOT
0006      C AND CKVA. THE Z VALUES TO BE CONTOURED ARE STORED IN
0007      C CZ(PTR(LOWER)), ..., CZ(PTR(UPPER)).
0008      C
0009      C THE ALGORITHM FOR THIS ROUTINE WAS TAKEN FROM:
0010      C G. W. HARTWIG, "CONTUR - A FORTRAN IV SUBROUTINE FOR
0011      C PLOTTING CONTOUR LINES," BALLISTIC RESEARCH LABORATORIES
0012      C MEMORANDUM REPORT # 2282, ABERDEEN PROVING GROUND,
0013      C MARYLAND, MARCH, 1973. (NTIS ACCESSION NUMBER AD-760 437).
0014      C
0015      C THE FOLLOWING CODE IS FOR VERSION 1.2 OF CONSYS, PRODUCED
0016      C 15 MAY, 1976. GNC
0017      C
0018      REAL CZ(NC)
0019      LOGICAL KCHK(8), CENTER
0020      REAL PX(8), PY(8), PTEMP
0021      EQUIVALENCE (PX(1), PX1), (PX(2), PX2), (PX(3), PX3),
0022      +             (PX(4), PX4), (PX(5), PX5), (PX(6), PX6),
0023      +             (PX(7), PX7), (PX(8), PX8)
0024      EQUIVALENCE (PY(1), PY1), (PY(2), PY2), (PY(3), PY3),
0025      +             (PY(4), PY4), (PY(5), PY5), (PY(6), PY6),
0026      +             (PY(7), PY7), (PY(8), PY8)
0027      INTEGER*2 PTR(NC)
0028      C
0029      COMMON /CONCOM/ LOWER, UPPER, XL, XR, XC, YL, YU, YC,
0030      +             ZLL, ZUL, ZLR, ZUR
0031      INTEGER*2 LOWER, UPPER
0032      C
0033      XC = 0.5*(XL + XR)
0034      XLMXR = XL - XR
0035      XLMXC = XL - XC
0036      XRMXC = XR - XC
0037      YLMYC = YL - YC
0038      YLMYU = YL - YU
0039      YUMYC = YU - YC
0040      ZC = 0.25*(ZLL + ZUL + ZLR + ZUR)
0041      DO 120 LEVEL = LOWER, UPPER
0042      Z0 = CZ(PTR(LEVEL))
0043      TLL = ZLL - Z0
0044      TUL = ZUL - Z0
0045      TLR = ZLR - Z0
0046      TUR = ZUR - Z0
0047      TC = ZC - Z0
0048      C
0049      IC = 0
0050      CENTER = .FALSE.
0051      DO 11 M = 1, 8

```

```

0028      KCHK(M) = .FALSE.
0029      11      CONTINUE
      C
      C      SEGMENT 1:
      C
0030      IF( TLL*TLR .GT. 0. ) GO TO 19
0031      KCHK(1) = .TRUE.
0032      IF( TLL*TLR .EQ. 0. ) GO TO 12
0033      IC = IC + 1
0034      PX(IC) = TLL * XLMXR/(ZLR-ZLL) + XL
0035      PY(IC) = YL
0036      GO TO 18
0037      12      CONTINUE
0038      IF( TLL .EQ. 0. ) GO TO 15
0039      IC = IC + 1
0040      PX(IC) = XR
0041      PY(IC) = YL
0042      GO TO 17
0043      15      CONTINUE
0044      IC = IC + 1
0045      PX(IC) = XL
0046      PY(IC) = YL
0047      IF( TLR .NE. 0. ) GO TO 16
0048      IC = IC + 1
0049      PX(IC) = XR
0050      PY(IC) = YL
0051      CONTINUE
0052      GO TO 17
0053      16      CONTINUE
0054      GO TO 18
0055      17      CONTINUE
0056      GO TO 18
0057      18      CONTINUE
0058      19      CONTINUE
      C
      C      SEGMENT 2:
      C
0061      IF( TLL*TC .GT. 0. ) GO TO 29
0062      KCHK(2) = .TRUE.
0063      IF( TLL*TC .EQ. 0. ) GO TO 22
0064      IC = IC + 1
0065      FAC = TLL/(ZC - ZLL)
0066      PX(IC) = XLMXC*FAC + XL
0067      PY(IC) = YLMYC*FAC + YL
0068      GO TO 28
0069      22      CONTINUE
0070      IF( TC .NE. 0. ) GO TO 25
0071      CENTER = .TRUE.
0072      IC = IC + 1
0073      PX(IC) = XC
0074      PY(IC) = YC
0075      CONTINUE
0076      GO TO 28
0077      25      CONTINUE
0078      GO TO 28
0079      28      CONTINUE
0080      29      CONTINUE
      C

```

```

C SEGMENT 3:
C
0082      IF( TUL*TLL .GT. 0. ) GO TO 39
0083      KCHK(3) = .TRUE.
0084      IF( TUL*TLL .EQ. 0. ) GO TO 32
0085          IC = IC + 1
0086          PX(IC) = XL
0087          PY(IC) = TLL * YLMYU/(ZUL-ZLL) + YL
0088          GO TO 38
0089  32      CONTINUE
0090      IF( TUL .NE. 0. ) GO TO 35
0091          IC = IC + 1
0092          PX(IC) = XL
0093          PY(IC) = YU
0094  35      CONTINUE
0095          GO TO 38
0096  38      CONTINUE
0097  39      CONTINUE

```

```

C SEGMENT 4:
C
0101      IF( TUL*TC .GE. 0. ) GO TO 49
0102      KCHK(4) = .TRUE.
0103      IC = IC + 1
0104      FAC = TUL/(ZC - ZUL)
0105      PX(IC) = XLMXC*FAC + XL
0106      PY(IC) = YUMYC*FAC + YU
0107  49      CONTINUE

```

```

C SEGMENT 5:
C
0109      IF( TUL*TUR .GT. 0. ) GO TO 59
0110      KCHK(5) = .TRUE.
0111      IF( TUL*TUR .EQ. 0. ) GO TO 52
0112          IC = IC + 1
0113          PX(IC) = TUL * XLMXR/(ZUR-ZUL) + XL
0114          PY(IC) = YU
0115          GO TO 58
0116  52      CONTINUE
0117      IF( TUR .NE. 0. ) GO TO 55
0118          IC = IC + 1
0119          PX(IC) = XR
0120          PY(IC) = YU
0121  55      CONTINUE
0122          GO TO 58
0123  58      CONTINUE
0124  59      CONTINUE

```

```

C SEGMENT 6:
C
0126      IF( TUR*TC .GE. 0. ) GO TO 69
0127      KCHK(6) = .TRUE.
0128      IC = IC + 1
0129      FAC = TUR/(ZC - ZUR)

```

```

0133          PX(IC) = XRMXC*FAC + XR
0134          PY(IC) = YUMYC*FAC + YU
0135 69          CONTINUE
          C
          C SEGMENT 7:
          C
0136          IF( TLR*TUR .GE. 0. ) GO TO 79
0137          KCHK(7) = .TRUE.
0138          IC = IC + 1
0139          PX(IC) = XR
0140          PY(IC) = TUR * YLMYU/(ZUR-ZLR) + YU
0141 79          CONTINUE
          C
          C SEGMENT 8:
          C
0143          IF( TLR*TC .GE. 0. ) GO TO 89
0144          KCHK(8) = .TRUE.
0145          IC = IC + 1
0146          FAC = TC/(ZC - ZLR)
0147          PX(IC) = XRMXC*FAC + XC
0148          PY(IC) = YLMYC*FAC + YC
0149 89          CONTINUE
          C
          C NOW DERIVE THE LINE SEGMENTS TO BE DRAWN FROM THE CONTENTS
          C OF THE PX AND PY ARRAYS.
          C
0151          IF( IC .LE. 1 ) GO TO 117
0152          IF( IC .GE. 6 .OR.
+          ( IC .EQ. 5 .AND. CENTER ) ) GO TO 100
0153          IF( .NOT. KCHK(8) ) GO TO 95
0154          DO 94 L = 1, 7
0155          LS=L
0156          IF( .NOT. KCHK(L) ) GO TO 93
0157          PX(IC+1) = PX(L)
0158          PY(IC+1) = PY(L)
0159          DO 92 M = 1, IC
0160          PX(M) = PX(M+1)
0161          PY(M) = PY(M+1)
0162 92          CONTINUE
0163          IF( MOD(LS, 2) .EQ. 1 ) GO TO 95
0164 93          CONTINUE
0165 94          CONTINUE
0166          GO TO 97
0167 95          IF( .NOT. CENTER .OR. KCHK(1) ) GO TO 97
0168          PTEMP = PX1
0169          PX1 = PX2
0170          PX2 = PTEMP
0171          PTEMP = PY1
0172          PY1 = PY2
0173          PY2 = PTEMP
0174          GO TO 97
0175 97          CONTINUE
0176          CALL PLOT(PX1, PY1,+3)
0177          DO 98 M = 2, IC

```

```

0184          PXM = PX(M)
0185          PYM = PY(M)
0186          CALL PLOT(PXM, PYM,+2)
0187  98      CONTINUE
0188          GO TO 109
0189  100     CONTINUE
0190          IF( IC .GE. 6 ) GO TO 104
0191          PX6 = PX5
0192          PY6 = PY5
0193          PX5 = PX2
0194          PY5 = PY2
0195          CONTINUE
0196  104     IF( KCHK(2) ) GO TO 105
0197          CALL PLOT(PX5, PY5,+3)
0198          CALL PLOT(PX6, PY6,+2)
0200          CALL PLOT(PX1, PY1,+2)
0201          CALL PLOT(PX2, PY2,+3)
0202          CALL PLOT(PX3, PY3,+2)
0203          CALL PLOT(PX4, PY4,+2)
0204          GO TO 108
0205          CONTINUE
0206  105     CALL PLOT(PX1, PY1,+3)
0207          CALL PLOT(PX2, PY2,+2)
0208          CALL PLOT(PX3, PY3,+2)
0209          CALL PLOT(PX4, PY4,+3)
0210          CALL PLOT(PX5, PY5,+2)
0211          CALL PLOT(PX6, PY6,+2)
0212          GO TO 108
0213          CONTINUE
0214  108     GO TO 109
0215          CONTINUE
0216  109     CONTINUE
0217  117     CONTINUE
0218  120     CONTINUE
0219          RETURN
C
0222          END
    
```

C-T PROGRAM RASM

C-F MAIN PROGRAM FOR VECTOR TO RASTER CONVERSION - MAPPED ALGORITHM

C COPYRIGHT C1976, VERSATEC INC., SANTA CLARA, CALIFORNIA 95051

C THE CONTENTS OF THIS DOCUMENT ARE PROPRIETARY TO VERSATEC, INC.,
C AND ARE NOT TO BE DISCLOSED TO OTHERS OR USED FOR PURPOSES OTHER
C THAN INTENDED WITHOUT THE WRITTEN APPROVAL OF VERSATEC.

C CALLS: DOPEN,DREAD,DWAIT,MWAIT,PREAD,INVECT,CYCLER,IRZERO
C CALLED BY: -NONE-
C COMMON VARIABLES USED: /PPEP2/ ISCAN,NWORD,LYNEND,IQ,MAXQ,NEPL,
C MSGLVL,LOST,IX0,IY1,NDLTX,IY2,
C IBUG,I2FLG,IR2,MXSTEP,IM
C /IOCOM/ LBLK,NBLK,LREC,JUNIT,LUNIT,IPARM,
C MUNIT

C ASSUMPTIONS: WHEN THE 'IM' OR 'INBUF' ARRAYS ARE DIMENSIONED
C OTHER CHANGES ARE AS FOLLOWS: IF IM('I') THEN SET
C 'IMD' = 'I' ; IF INBUF('J') SET 'INBUFD' = 'J'.
C 'J' MUST BE AN INTEGER MULTIPLE OF AND GREATER
C THAN OR EQUAL TO 2*(NBLK*LBLK).

C-P 50030-20503 REV. A - PART NUMBER
C-S RT-11 - OPERATING SYSTEM

C AUTHOR: M.D. DOBERVICH 07/07/76

C PROGRAM RASTER

C COMMON /PPEP2/ISCAN,NWORD,LYNEND,IQ,MAXQ,NEPL,
1 NBITS,IBT(16),KBT(16),JBT(16),MSGVLV,LOST,
2 IX0,IY1,NDLTX,IY2,
3 NDW,NDB,NRUN,ISUM,NDLTY,
4 IBUG,I2FLG,IR2,MXSTEP,
N IM(3500)

C COMMON /IOCOM/ LBLK,NBLK,LREC,LVEC,IUNIT,JUNIT,KUNIT,LUNIT,MUNIT,
1 IPARM,IPCTR,IPREC,IPBUF(256)

C DIMENSION INBUF(512)

C DATA IMD/3500/,INBUFD/512/,MAPEND/600/

C DATA IREC/0/,IOLD/1/,MAPKEY/102/

C 1 FORMAT (25H FILE/ALGORITHM MISMATCH ,16)
C-D 2 FORMAT (21H IX0,IY1,NDLTX,IY2 = 4(1X16))
2 FORMAT (21H IX0,IY1,NDLTX,IY2 = 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(IBUG),0,2)

C ... CHECK ALGORITHM KEY.

```
CALL PREAD (KEY,1)
IF (KEY.EQ.MAPKEY) GO TO 7010
WRITE (MUNIT,1) KEY
STOP
```

```
C
7010 CALL PREAD (NSCAN,1)
CALL PREAD (IR1,1)
CALL PREAD (IR2,1)
CALL PREAD (ISCAN,1)
CALL PREAD (NWORD,1)
CALL PREAD (I2FLG,1)
CALL PREAD (IOUT1,1)
CALL PREAD (MXSTP,1)
CALL PREAD (MSGLVL,1)
CALL PREAD (IWORD,1)
CALL PREAD (LYNES,1)
CALL PREAD (NBLK,1)
```

```
C
LREC = LBLK * NBLK
IBLKSZ = 2*LBLK*NBLK
```

```
C
C... INPUT AND OUTPUT BUFFERS DIMENSIONED PROPERLY?
IF (MOD(IBLKSZ,INBUFD).EQ.0.AND.IBLKSZ.LE.INBUFD.AND.IWORD.LE.IMD)
1 GO TO 7011
WRITE (MUNIT,3)
STOP
```

```
C
7011 INEW = LREC + 1
```

```
C
C... OPEN MAPPED VECTOR FILE
CALL DOPEN (JUNIT,-1,NBLK)
```

```
C
C... INPUT MAP ENTRIES FOR CURRENT PLOT AT START OF THE IM ARRAY.
```

```
C
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
7030 CALL PREAD (IM(MAPSZ),2)
```

```
C
C... END OF MAP FOR THIS PLOT?
IF (IM(MAPSZ).LT.0) GO TO 7040
MAPSZ = MAPSZ + 2
```

```
C
C... DOES MAP SIZE FIT WITHIN THE MAXIMUM MAP ALLOCATION AREA?
IF (MAPSZ.LT.MAPEND) GO TO 7030
```

```
C
C... MAP BUFFERS EXCEEDED.
WRITE (MUNIT,4)
STOP
```

```
C
C... CHECK FOR END OF ALL PLOTTING.
7040 IF (MAPSZ.EQ.1) GO TO 7700
MMAXX = IM(MAPSZ+1)
IQNDX = MAPEND
```

```
C
C... IF VECTOR QUEUING IS USED THEN MAP ALLOCATION BECOMES DYNAMIC.
IF (LYNES.NE.0) IQNDX = MAPSZ
```

```
C
C... CALCULATE NSCAN , NWORD AND ALLOCATE BUFFERS IN IM ARRAY.
NSCAN = (IWORD - (NEPL*LYNES) - IQNDX)/(I2FLG*ISCAN)
NWORD = ISCAN*NSCAN
LYNEND = IWORD-IQNDX-(I2FLG*NWORD) - 1
IBUFG = LYNEND + IQNDX
IR2 = IBUFG+NWORD+1
```

```

C
C... INITIALIZE ASSEMBLY LANGUAGE ROUTINES
CALL INIT (IM(IBUFG),IM(IR2),IM(IQNDX))
C
C... CONVERT IOUT1 AND MXSTP VALUES TO IOUT AND MXSTEP BAND VALUES.
MXSTEP = MXSTP/NSCAN
IOUT = IOUT1/NSCAN
C
C... INITIALIZE JREC AND MAP SEARCH POINTERS.
NREC = ((MAPSZ+1)/2) - 2
JREC = NREC + IREC + 1
NDXFTR = MAPSZ - 4
NDXPST = 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,INBUF(IOLD),JREC)
CALL IRZERO (IM(IBUFG))
C
C... WAIT FOR LAST READ OPERATION COMPLETE
7100 CALL DWAIT (JUNIT,IERR)
C
C-----
C... SEARCH MAP FOR VALID VECTOR BLOCK TO READ.
C
IIMP = 1
GO TO 7120
C
7110 NDXFTR = NDXFTR - 2
NREC = NREC - 1
C
C... HAS ENTIRE MAP BEEN SEARCHED?
7120 IF (NDXFTR.LT.0) GO TO 7130
C
C... DOES MAXIMUM(VECTOR) 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)
7130 NDXFTR = MAPSZ - 2
NREC = ((MAPSZ+1)/2) - 1
NBANDS = NBANDS + 1
IIMP = 0
C
C... INCREMENT CURRENT BAND LIMITS.
IXSTR = IXSTR + NSCAN
IXEND = IXEND + NSCAN
C
C... MORE BANDS IN THIS PLOT?
IF (IXSTR.LE.MMAXX) GO TO 7120

```

```

I JMP = -1
GO TO 7200

C
C... START INPUT OF NEXT BUFFER
7150 CALL DREAD (JUNIT,INBUF(INEW),JREC)
C
C - - - - -
C... RESET END OF 'OLD' BUFFER,BUFFER INDEX,AND CURRENT MAPP MINIMUM.
C
7200 IBEND = LREC + IOLD - 1
      JDX = IOLD - 4
      IM(NDXPST+1) = IM(NDXPST)
C
C... SEARCH FOR VALID VECTORS IN THE CURRENT BUFFER
GO TO 7250
C
C... UPDATE 'NEW' MAP MINIMUM IF VECTOR IS LESS THAN CURRENT MINIMUM.
7220 IF (INBUF(JDX).LT.IM(NDXPST+1)) IM(NDXPST+1) = INBUF(JDX)
7250 JDX = JDX + 4
C
C... END OF 'OLD' BUFFER?
      IF (JDX.GT.IBEND) GO TO 7270
      IX0 = INBUF(JDX) - IXSTRP
C
C... DOES 'RELATIVE' VECTOR START BEFORE BAND?
      IF (IX0.LT.0) GO TO 7260
C
C... DOES 'RELATIVE' VECTOR START AFTER BAND?
      IF (IX0.GE.NSCAN) GO TO 7220
C
C... VALID VECTOR LOCATED WITHIN BAND FOR VECTOR/RASTER CONVERSION.
      IY1 = INBUF(JDX+1)
      NDLTX = INBUF(JDX+2)
      IY2 = INBUF(JDX+3)
C-D IF (MSGVLV.GE.6) WRITE(LUNIT,2) IX0,IY1,NDLTX,IY2
      IF (MSGVLV.GE.6) WRITE(LUNIT,2) IX0,IY1,NDLTX,IY2
C
      CALL INVECT (IM(IQNDX),IM(IBUFG))
C
C... TEST FOR POSSIBLE END OF BUFFER (DATA)
7260 IF (INBUF(JDX).GE.0) GO TO 7250
7270 NDXPST = NDXPRS
C
C - - - - -
C... END OF CURRENT VECTOR INPUT BLOCK.
C... IS STATUS - END OF PLOT,END OF BAND,BAND NOT COMPLETE?
C
      EOP, EOB, BNC
      IF (I JMP) 7350,7400,7500
C
7350 CALL CYCLER (NBANDS,IM(IQNDX))
C
C... LOOP UNTIL ACTIVE LINE TABLE IS EMPTY
      NBANDS = 1
      IF (IQ.GT.1) GO TO 7350
C
C... OUTPUT LAST BAND OF CURRENT PLOT.
      CALL CYCLER (1,IM(IQNDX))
C
C... ACTION CHECK: FORM FEED ONLY, NO ACTION , IOUT AND FORM FEED?
      FFO, NA, IFF
      IF (IOUT) 7370,7380,7360
C

```

```

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 (MSGVLV.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 7020
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 7100
C
C... END OF ALL PLOTTING:
7700 CALL MTX (IM(IBUFG),-1,2)
      STOP
      END

```

```

C
C THIS IS A SUBROUTINE
C PACKAGE WHICH IN CONJUNCTION WITH
C MPRASM INTERCEPTS PLOT DATA INTENDED FOR
C THE PLOTTER AND PUTS IT TO DISC OR TAPE
C
0001      SUBROUTINE MWAIT
C
C DUMMY ROUTINE
C
0002      RETURN
0003      END

```

```

0001      SUBROUTINE MTXSET
C
C SET UP ROUTINE
C
0002      REAL*8 FSPECR
0003      REAL*4 FBUF(3)
0004      LOGICAL*1 ITPDRW
0005      COMMON /MPMTX/ ICH, IOFLG, IBLK, IWD, ITPDRW
C
C GET DATA PARAMETERS
C
0006      IBLK=1
0007      IWD=1
0008      WRITE(7,1000)
0009      1000  FORMAT(' ENTER A 0 FOR TAPE OUTPUT, '/',
*          ' OR A 1 FOR DISC OUTPUT:', $)
0010      READ(5,1001) IOFLG
0011      1001  FORMAT(I1)
C
C GET FILE NAME IF REQD
C
0012      IF(IOFLG.EQ.0) GOTO 10
0014      WRITE(7,1002)
0015      1002  FORMAT(' ENTER FILE NAME FOR OUTPUT:', $)
0016      READ(5,1003) FBUF
0017      1003  FORMAT(3A4)
0018      CALL IRAD50(12, FBUF, FSPECR)
0019      ICH=IGETC()
0020      IF(IENTER(ICH, FSPECR, -1).LT.0) STOP 'ENTER ERROR'
0022      RETURN
C
C GET TAPE NO
C
0023      10  WRITE(7,1004)
0024      1004  FORMAT(' ENTER TAPE DRIVE NUMBER:', $)
0025      READ(5,1001) ITPDRW
0026      RETURN
0027      END

```

```

0001      SUBROUTINE MTX(IPBUF,NWDS,IFLAG)
C
C MAIN TRANSFER ROUTINE
C
0002      INTEGER*2 IBUF(2048),IPBUF(1),BUF(2304)
0003      LOGICAL*1 ISTAT,ITPDRW
0004      COMMON /MPMTX/ICH,IOFLG,IBLK,IWD,ITPDRW
0005      EQUIVALENCE (BUF(257),IBUF(1))
0006      DATA NBYTBF/4608/,NBLKBF/9/,ICOUNT/0/
0007      DO 5 J=1,10
0008      5 BUF(J)=0
C
C CHECK FOR OPERATION TYPE
C
0009      IF(NWDS.EQ.0)RETURN
0010      ICOUNT=ICOUNT+1
0011      IF(NWDS.LT.0)GOTO 500
C
C SWOP DATA TO OUTPUT BUFFER
C
0014      DO 10 J=1,NWDS
0015      IBUF(IWD)=IPBUF(J)
0016      IWD=IWD+1
0017      IF(IWD.LE.2048)GOTO 10
0018      IF(IOFLG.EQ.0)GOTO 20
C
C DISC OUTPUT
C
0021      IF(IWRITW(2048,IBUF,IBLK,ICH).LT.0)STOP 'WRITE ERROR'
0022      IBLK=IBLK+8
0023      IWD=1
0024      GOTO 10
C
C TAPE OUTPUT
C
0026      20 IFLEN=NBYTBF
0027      NEYT=NBYTBF
0028      IEOTW=0
0029      CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
0030      IWD=1
C
C CHECK FOR ERRORS
C
0031      IF(IEOTW.GE.0)GOTO 30
0032      WRITE(7,1000)ITPDRW,ICOUNT
0033      1000 FORMAT(' EOT ON WRITE DRIVE:',I2,' BUFFER NUMBER:',I5)
0034      WRITE(7,1001)
0035      1001 FORMAT(' ENTER NEW WRITE DRIVE NUMBER:',S)
0036      READ(5,1002)ITPDRW
0037      1002 FORMAT(I1)
0038      IEOTW=0
0039      IF(ITPDRW.GT.2)STOP ' EOT TERMINATE '
0040      30 IF(ISTAT.GE.0)GOTO 10
0041      WRITE(7,1003)ICOUNT
0042      1003

```

```
0045 1003 FORMAT(' FATAL ERROR ON WRITING BUFFER NUMBER:',I5)
0046      STOP ' FATAL WRITE ERROR'
0047      10 CONTINUE
0048      RETURN
C
C COME HERE TO FLUSH BUFFERS
C
0049 500 CONTINUE
0050      DO 40 J=NWD,2048
0051      40 IBUF(J)=0
0052      IF(IOFLG.EQ.0)GOTO 50
0054      IF(IWRITW(2048,IBUF,IBLK,ICH).LT.0)STOP ' WRITE ERROR'
0056      CALL CLOSEC(ICH)
0057      WRITE(7,1010)ICOUNT
0058      RETURN
C
C FLUSH TO TAPE
C
0059 50 IFLEN=NBLKBF
0060      NBYT=NBYTBF
0061      IEOTW=0
0062      CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
0063      IF(IEOTW.LT.0)GOTO 60
C
C AS ONLY FLUSHING BUFFERS IGNORE ERRORS
C
0065      BUF(1)='E'
0066      BUF(2)='O'
0067      BUF(3)='D'
0068      CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
0069      50 WRITE(7,1010)ICOUNT
0070 1010 FORMAT(' BUFFERS WRITTEN=',I5)
0071      RETURN
0072      END
```

UtilitiesFile 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
C M J POULTER JUL 1980
C TAPE/DISC SAVE RESTORE PROGRAM
C VERSION 1
C
0001 REAL*8 FSPECR,FSPECW,FSPEC(1000),DATE(1000),NBUF,DBUF,FSBUF,
%EOTCOD,QSPEC(20),TPNUM,TODATE
0002 REAL*4 FBUF(3),COM(4),CBUF
0003 INTEGER*2 VNUM(1000),SIZE(1000),VBUF,SBUF,
%IVNUM(10),IFNUM(10),QNO(20),BUF(1024)
0004 LOGICAL*1 EOT,EOR,HBLK(20),ANS,YES,NO,IDRV,ISTAT
0005 EQUIVALENCE (HBLK(1),NBUF),(HBLK(9),DBUF),(HBLK(17),VBUF),
%(HBLK(19),SBUF)
0006 COMMON/SERCH/FSPEC,DATE,VNUM,SIZE,HBLK,IFILE
0007 DATA DEV/3RRK /,FSBUF/12RDK0MPTPSVDAT/,YES/'Y'/,NO/'N'/
0008 DATA COM(1)/'SAVE'/,COM(2)/'REST'/,COM(3)/'STOP'/,COM(4)/'TDIR'/
0009 DATA EOTCOD/12REOTEOTEOTEOT/
C
C SET UP I/O CHANNELS
C
0010 IF(ICDFN(25).NE.0)STOP'CHAN OVERFLOW'
0011 IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
0012 ITR=IGETC()
0013 IRD=20
0014 IWRT=21
0015 IFILE=0
0016 IQNUM=0
0017 EOR=.FALSE.
0018 EOT=.FALSE.
C
C GET TODAYS DATE
C
0021 CALL GDATE(TODATE)
0022 TYPE 10
0023 10 FORMAT(' TAPE SAVE/RESTORE PROGRAM VERSION 1')
0024 TYPE 15
0025 15 FORMAT(' ENTER TAPE NAME AND DRIVE NUMBER')
0026 ACCEPT 16,TPNUM,IDRV
0027 16 FORMAT(A8,I2)
0028 TYPE 17
0029 17 FORMAT(' IS THE TAPE TO BE INITIALISED Y/N:',$)
0030 ACCEPT 18,ANS
0031 18 FORMAT(A1)
0032 IF(ANS.EQ.NO)GOTO 19
C
C TAPE INITIALISATION CODE
C
0034 NBUF=TPNUM
0035 DBUF=TODATE
0036 ILEN=1
0037 CALL TPHAND(4,IDRV,ISTAT,ILEN,IFILE,HBLK,20)
C
C WRITE EOT BLOCK AND REWIND TO BEGINNING OF BLOCK

```

```

C
0038      NBUF=EOTCOD
0039      ILEN=1
0040      CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
0041      CALL TPHAND(6,IDRV,ISTAT,2,ILEN,IFILE, , )
0042      EOT=.TRUE.
0043      GOTO 1
C
C NORMAL INTRO TO A SESSION
C
0044      19 CALL TPHAND(2,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
0045         IF(TPNUM.NE.NBUF)TYPE 11
0047      11 FORMAT(' WARNING TAPE NAME DIFFERENT TO INIT VALUE!!!')
0048         CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
C
C START OF MAIN SOFTWARE LOOP
C
0049      1 TYPE 20
0050      20 FORMAT(' ? ',S)
0051         ACCEPT 30,CBUF
0052      30 FORMAT(A4)
C
C AFTER COMMAND RECEIVED DECODE IT
C
0053      DO 40 I=1,4
0054      40 IF(CBUF.EQ.COM(I))NCOM=I
0056         GOTO (1000,2000,3000,4000)NCOM
C
C CODE FOR SAVING A FILE
C
0057      1000 TYPE 1010
0058      1010 FORMAT(' ENTER FILENAME TO BE SAVED,WITH VERSION NUMBER')
0059         ACCEPT 1020,FBUF,NVNO
0060      1020 FORMAT(3A4,I2)
0061         CALL IRADS0(12,FBUF,FSPECR)
C
C GET TO END OF TAPE TO SAVE THE NEW FILE
C
0062      IF(.NOT.EOT)CALL FSERCH(EOTCOD,IDRV,EOT)
0064         EOT=.TRUE.
C
C OPEN FILE TO BE SAVED
C
0065      ISIZE=LOOKUP(IWRT,FSPECR)
0066      IF(ISIZE.LE.0)GOTO 9001
0068      IF(ISIZE.GT.4) GOTO 1050
0070      NWDS=ISIZE*256
0071      NBYTES=NWDS*2
0072      IF(IREADW(NWDS,BUF,0,IWRT).LT.0)GOTO 9002
C
C SET UP INTERNAL DIRECTORY
C
0074      1050 IFILE=IFILE+1
0075         FSPEC(IFILE)=FSPECR
```

```
0076     DATE(IFILE)=TODATE
0077     VNUM(IFILE)=NVNO
0078     SIZE(IFILE)=ISIZE
      C
      C SET UP HEADER BLOCK
      C
0079     NBUF=FSPECR
0080     DBUF=TODATE
0081     VBUF=NVNO
0082     SBUF=ISIZE
      C
      C WRITE HEADER BLOCK
      C
0093     ILEN=1
0094     CALL TPHAND(4, IDR, ISTAT, , ILEN, IFILE, HBLK, 20)
      C
      C WRITE OUT THE FILE
      C
0085     ILEN=ISIZE
0086     IF(ILEN.GT.4)CALL TPHAND(3, IDR, ISTAT, , ILEN, IFILE, , )
0088     IF(ILEN.LE.4)CALL TPHAND(4, IDR, ISTAT, , ILEN, IFILE, BUF, NBYTES)
      C
      C SEE IF MORE SAVES RTO BE DONE
      C
0095     CALL CLOSEC(IWRT)
0096     TYPE 1030
0097     1030 FORMAT(' MORE FILES TO BE SAVED Y/N ?', S)
0098     ACCEPT 1040, ANS
0099     1040 FORMAT(A1)
0100     IF(ANS.EQ.YES)GOTO 1000
      C
      C WRITE EOT FILE IF NO MORE
      C
0101     NBUF=EOTCOD
0102     ILEN=1
0103     CALL TPHAND(4, IDR, ISTAT, , ILEN, IFILE, HBLK, 20)
      C
      C REWIND TO BEGINNING OF EOT FILE
      C
0104     CALL TPHAND(6, IDR, ISTAT, 2, , IFILE, , )
0105     GOTO 1
      C
      C CODE FOR A RESTORE
      C
0106     2000 TYPE 2010
0107     2010 FORMAT(' ENTER FILE TO BE RESTORED')
0108     ACCEPT 2020, FBUF
0109     2020 FORMAT(3A4)
0110     CALL IRAD50(12, FBUF, FSPECW)
0111     IVER=0
      C
      C SEARCH TO SEE IF FILE ALREADY PASSED
      C
0112     DO 2030 I=1, IFILE
```

```

0109      IF(FSPECW.NE.FSPEC(I))GOTO 2030
0111      IVER=IVER+1
0112      IFNUM(IVER)=I
0113      IVNUM(IVER)=VNUM(I)
0114      2030 CONTINUE
0115      IF(IVER.LE.1)GOTO 2040
C
C IF MORE THAN ONE VERSION FIND WHICH ONE REQUIRED
C
0117      TYPE 2050
0118      2050 FORMAT(' ENTER VERSION NUMBER REQUIRED:',S)
0119      ACCEPT 2060,NVNO
0120      2060 FORMAT(I2)
0121      DO 2065 I=1,IVER
0122      2065 IF(NVNO.EQ.IVNUM(I))IFNUM(1)=IFNUM(I)
C
C FIND OUT IF NEED TO QUEUE OR CONTINUE
C
0124      2040 IF(IVER.NE.0)GOTO 2070
0125      TYPE 2050
0127      ACCEPT 2060,NVNO
0128      2042 CALL FSERCH(FSPECW,IDRV,EOT)
0129      IF(EOT)GOTO 2999
0131      CALL TPHAND(2,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
0132      CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
0133      IF(NVNO.NE.VBUF)CALL TPHAND(5,IDRV,ISTAT,2,ILEN,IFILE, , )
0135      IF(NVNO.NE.VBUF)GOTO 2042
0137      ISIZE=SBUF+16
0138      IF(IENTER(IRD,FSBUF,ISIZE).LT.0)GOTO 9003
0140      CALL TPHAND(1,IDRV,ISTAT, ,ILEN,IFILE, , )
0141      CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
0142      CALL R50ASC(12,FSPECW,FBUF)
0143      TYPE 2041,FBUF
0144      2041 FORMAT(' ENTER NEW NAME FOR TAPE FILE: ',3A4)
0145      ACCEPT 2020,FBUF
0146      CALL IRAD50(12,FBUF,FSPECW)
0147      ISIZE=SBUF
0148      IF(IENTER(ITR,FSPECW,ISIZE).LT.0)GOTO 9003
0150      IBLK=0
0151      DO 2045 I=1,ISIZE
0152      IF(IREADW(256,BUF,IBLK,IRD).LT.0)GOTO 9004
0153      IF(IWRITE(256,BUF,IBLK,ITR).LT.0)GOTO 9005
0154      IBLK=IBLK+1
0155      2045 CONTINUE
0156      CALL CLOSEC(IRD)
0157      CALL CLOSEC(ITR)
0158      GOTO 1
C
C PUT NAME IN THE QUEUE
C
0159      2070 IQNUM=IQNUM+1
0160      QSPEC(IQNUM)=FSPECW
0161      QNO(IQNUM)=IFNUM(1)
0162      2080 IF(.NOT.EOR)GOTO 1

```

```
      C
      C COME HERE AT END OF RUN
      C
0156      CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0157      IQ=0
0158      IFILE=1
0159      CALL TPHAND(5,IDRV,ISTAT,2, ,IFILE, , )
0170      N2=1
      C SORT QUEUE INTO ORDER
      C
0171      IGAP=IQNUM
0172      2081 IF(IGAP.LE.1)GOTO 2100
0173      IGAP=IGAP/2
0174      IMAX=IQNUM-IGAP
0175      2085 IEX=0
0177      DO 2086 I=1,IMAX
0178      IPLUSG=I+IGAP
0179      IF(QNO(I).LE.QNO(IPLUSG))GOTO 2086
0181      ISAVE=QNO(I)
0182      FSPECW=QSPEC(I)
0183      QNO(I)=QNO(IPLUSG)
0184      QSPEC(I)=QSPEC(IPLUSG)
0185      QNO(IPLUSG)=ISAVE
0186      QSPEC(IPLUSG)=FSPECW
0187      IEX=IEX+1
0188      2086 CONTINUE
0189      IF(IEX.GT.0)GOTO 2085
0191      GOTO 2081
      C
      C MAIN RESTORE LOOP
      C
0192      2100 CONTINUE
0193      IQ=IQ+1
0194      N1=N2
0195      N2=QNO(IQ)
0196      NSEP=N2-N1
0197      IFILE=IFILE+NSEP
0198      NFILE=NSEP*4
      C
      C WIND FOWARD TO CORRECT FILE.
      C
0199      IF(NFILE.GT.0)CALL TPHAND(5,IDRV,ISTAT,NFILE,ILEN,IFILE, , )
      C
      C READ HEADER BLOCK AND CHECK IF FOUND CORRECT FILE
      C
0201      CALL TPHAND(2,IDRV,ISTAT, , ILEN,IFILE,HBLK,20)
0202      IF(NBUF.NE.QSPEC(IQ))GOTO 2800
0203      CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
0204      FSPECW=NBUF
0205      ISIZE=SBUF+16
      C
      C READ ONTO TEMPORARY FILE
      C
0207      IF(IENTER(IRD,FSBUF,ISIZE).LT.0)GOTO 9003
```

```
0209 CALL TPHAND(1,IDRV,ISTAT, ,ILEN,IFILE, , )
0210 CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
0211 CALL R50ASC(12,FSPECW,FBUF)
0212 TYPE 2041,FBUF
0213 ACCEPT 2020,FBUF
0214 CALL IRAD50(12,FBUF,FSPECW)
0215 IF(IENTER(ITR,FSPECW,SBUF).LT.0)GOTO 9003
```

```
C
C TRANSFER TO APPROPRIATE PERMANENT FILE
C
```

```
0217 IBLK=0
0218 DO 2101 I=1,SBUF
0219 IF(IREADW(256,BUF,IBLK,IRD).LT.0)GOTO 9004
0221 IF(IWRITE(256,BUF,IBLK,ITR).LT.0)GOTO 9005
0223 IBLK=IBLK+1
0224 2101 CONTINUE
0225 CALL CLOSEC(IRD)
0226 CALL CLOSEC(ITR)
0227 IQNUM=IQNUM-1
0228 N2=N2+1
0229 IFILE=IFILE+1
0230 IF(IQNUM.NE.0)GOTO 2100
0232 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0233 STOP ' NORMAL TERMINATION'
```

```
C
C WRONG FILE FOUND
C
```

```
0234 2800 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0235 CALL R50ASC(12,NBUF,FBUF)
0236 TYPE 2810,FBUF
0237 2810 FORMAT(' FILE FOUND ON TAPE= ',3A4)
0238 CALL R50ASC(12,QSPEC(IQ),FBUF)
0239 TYPE 2820,FBUF
0242 2820 FORMAT(' FILE REQUIRED = ',3A4)
0243 STOP ' WRONG FILE FOUND FOR RESTORE'
```

```
C
C FILE NOT FOUND
C
```

```
0242 2999 TYPE 2900
0243 2900 FORMAT(' FILE NOT FOUND')
0244 GOTO 1
```

```
C
C CODE FOR STOP COMMAND
C
```

```
0245 3000 EOR=.TRUE.
0246 IF(IQNUM.NE.0)GOTO 2080
0248 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0249 CALL CLOSEC(IRD)
0250 CALL CLOSEC(IWRT)
0251 CALL CLOSEC(ITR)
0252 STOP ' NORMAL TERMINATION'
```

```
C
C CODE FOR DIRECTORY
C
```

```

0250 4000 IF(.NOT.EOT)CALL FSERCH(EOTCOD,IDRV,EOT)
0251      EOT=.TRUE.
0252      TYPE 4070
0253 4070 FORMAT(' DIR ON TERMINAL Y/N: ',S)
0254      ACCEPT 4080,ANS
0255 4080 FORMAT(A1)
0256      LUNIT=6
0257      IF(ANS.EQ.VES)LUNIT=7
0258 4010 WRITE(LUNIT,4020)TPNUM,IDRV
0259 4020 FORMAT('1 TAPE NO= ',A8,' DRIVE NO',I3,' FILE DIRECTORY')
0260      WRITE(LUNIT,4030)
0261 4030 FORMAT(' FILE NAME ',5X,' DATE SAVED ',5X,' VERSION ',
0262      X5X,' SIZE')
0263      DO 4040 I=1,IFILE
0264      CALL R50ASC(12,FSPEC(I),FBUF)
0265 4040 WRITE(LUNIT,4050)FBUF,DATE(I),VNUM(I),SIZE(I)
0266 4050 FORMAT(1X,3A4,3X,A8,14X,I2,10X,I5)
0267      WRITE(LUNIT,4060)IFILE
0268 4060 FORMAT(' TOTAL NUMBER OF FILES ON TAPE=',I3)
0269      GOTO 1
C
C ERROR FINISHES
C
0274 9001 NBUF=EOTCOD
0275      ILEN=1
0276      CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
0277      CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0278      CALL CLOSEC(IRD)
0279      CALL CLOSEC(IWRT)
0280      CALL CLOSEC(ITR)
0281      STOP 'LOOKUP ERROR'
0282 9002 NBUF=EOTCOD
0283      ILEN=1
0284      CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
0285      CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0286      CALL CLOSEC(IRD)
0287      CALL CLOSEC(IWRT)
0288      CALL CLOSEC(ITR)
0289      STOP 'READ ERR FOR SAVE'
0290 9003 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0291      CALL CLOSEC(IRD)
0292      CALL CLOSEC(IWRT)
0293      CALL CLOSEC(ITR)
0294      STOP ' ENTER ERR FOR RESTORE '
0295 9004 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0296      CALL CLOSEC(IRD)
0297      CALL CLOSEC(IWRT)
0298      CALL CLOSEC(ITR)
0299      STOP ' READ ERR ON RESTORE TRANSFER'
0300 9005 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
0301      CALL CLOSEC(IRD)
0302      CALL CLOSEC(IWRT)
0303      CALL CLOSEC(ITR)
0304      STOP ' WRITE ERR ON RESTORE TRANSFER'

```

```
0001      SUBROUTINE FSERCH(FNAME, IDRV, EOT)
      C
      C SUBROUTINE WHICH SEARCHES FOWARD TO SPECIFIED
      C FILE NAME ON THE TAPE
      C
0002      REAL*8 FNAME, FSPEC(1000), DATE(1000), NBUF, DBUF, EOTCOD
0003      INTEGER*2 SIZE(1000), VNO(1000), VBUF, SBUF
0004      LOGICAL*1 HBLK(20), IDRV, EOT, ISTAT
0005      COMMON/SERCH/ FSPEC, DATE, VNO, SIZE, HBLK, IFILE
0006      EQUIVALENCE(HBLK(1), NBUF), (HBLK(9), DBUF), (HBLK(17), VBUF),
      X(HBLK(19), SBUF)
0007      DATA EOTCOD/12REOTEOTEOTEOT/
0008      10 CALL TPHAND(2, IDRV, ISTAT, , ILEN, IFILE, HBLK, 20)
0009      IF(NBUF.EQ.EOTCOD)GOTO 15
      C
      C STICK HEADER INFO IN DIRECTORY
      C
0011      IFILE=IFILE+1
0012      FSPEC(IFILE)=NBUF
0013      DATE(IFILE)=DBUF
0014      VNO(IFILE)=VBUF
0015      SIZE(IFILE)=SBUF
0016      IF(NBUF.EQ.FNAME)GOTO 20
      C
      C POSITION TAPE NEXT TO NEXT HEADER
      C
0018      CALL TPHAND(5, IDRV, ISTAT, 3, ILEN, IFILE, , )
0019      GOTO 10
      C
      C REWIND TO BEGINNING OF FOUND HEADER
      C
0020      15 EOT=.TRUE.
0021      20 CALL TPHAND(6, IDRV, ISTAT, 1, ILEN, IFILE, , )
0022      RETURN
0023      END
```

```

0001      SUBROUTINE TPHAND(ICOM,IDRV,ISTAT,ITLEN,ILEN,IFNUM,BUF,NBYT)
C
C TAPE HANDLING SUBROUTINE
C ICOM IS THE COMMAND SIGNAL
C ICOM IS THE COMMAND SIGNAL
C 1=READ FROM TAPE TO DISC
C 2=READ FROM TAPE TO MEMORY
C 3=WRITE TO TAPE FROM DISC
C 4=WRITE TO TAPE FROM MEMORY
C 5=WIND FOWARD
C 6=WIND REVERSE
C 7= REWIND TO START
C IDRV IS THE DRIVE BEING USED
C ISTAT IS THE STATUS ON RETURN
C ITLEN IS THE NO OF TAPE FILES TO MOVE PAST
C ILEN IS THE BLOCK LENGTH OF A FILE READ OR WRITTEN
C BUF IS THE MEMORY AREA USED BY TWRIT AND TREAD
C NBYT IS THE SIZE OF BUF
C
0002      INTEGER*2 MASK(8),ESTATI
0003      LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
          %IFLEN,ESTAT,BUF(1)
0004      DATA MASK/"1","2","4","10","20","40","100","200"/
0005      DATA SDSCOM/"0","1","2","3","4","5","6","7"/
0006      ITRY=0
0007      GOTO (1000,2000,3000,4000,5000,5000,5000)ICOM
C
C SECTION CONTROLLING A READ
C
C
C CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
C
0008      1000 ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
0009      COM(1)=SDSCOM(4)
0010      COM(2)=1
0011      COM(3)=IDRV
0012      COM(4)=-1
0013      CALL SDS10(COM,ISTAT,ITLEN,ILEN)
0014      IF(ISTAT.EQ.0)RETURN
C
C ERROR DETECTED ON READ
C
0016      ISTAT=ISTAT
0017      GOTO 40
C
C READ FROM TAPE TO MEMORY
C
0018      2000 NBUF=NBYT
0019      CALL TREAD(BUF,NBUF,ISTAT,IDRV)
0020      ITRY=ITRY+1
0021      IF(ISTAT.EQ.0)RETURN
    
```

```
0023       ISTATI=ISTAT
0024       GOTO 40
C
C IF SHORT RECORD FOUND REREAD TAPE
C
0025       50 ITMP=ISTATI.AND.MASK(6)
0026       IF(ITMP.NE.0)GOTO (1000,2000)ICOM
0028       ITMP=ISTATI.AND.MASK(2)
0029       IF(ITMP.EQ.0)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
0031       TYPE 2010,IFNUM
0032       2010 FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
0033       IF(ITRY.GE.2)GOTO 130
0035       ECOM(1)=SDSCOM(6)
0036       ECOM(2)=1
0037       ECOM(3)=IDRV
0038       ECOM(4)=0
0039       CALL SDS10(ECOM,ESTAT, , )
0040       GOTO (1000,2000)ICOM
C
C WRITE SECTION
C
0041       3000 ITRY=ITRY+1
0042       IF(ITRY.GT.2)GOTO 130
0044       COM(1)=SDSCOM(7)
0045       IFLEN=(ILEN+3)/4
0046       COM(2)=IFLEN
0047       COM(3)=IDRV
0048       COM(4)=1
0049       CALL SDS10(COM,ISTAT, , )
0050       IF(ISTAT.EQ.0)RETURN
C
C WRITE ERROR DETECTED
C
0051       ISTATI=ISTAT
0052       GOTO 40
C
C MEMORY TO TAPE WRITE
C
0054       4000 CONTINUE
0055       NBUF=NBYT
0056       IFLEN=(ILEN+3)/4
0057       IF(IFLEN.LT.2)IFLEN=2
0059       IPAD=(IFLEN*2048)-NBUF
0060       CALL TWRIT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
0061       IF(ISTAT.EQ.0)RETURN
0062       ISTATI=ISTAT
0063       GOTO 40
C
C ERROR RETURN POSITION
C
0065       70 ITMP=ISTATI.AND.MASK(6)
```

```
0060      ITMPI=ISTATI.AND.MASK(2)
0061      IF(ITMP.EQ.0.AND.ITMPI.EQ.0)RETURN
      C
      C REPORT AND RETRY
      C
0069      TYPE 2020,IFNUM
0070      2020 FORMAT(' FILE NO ',I4,' WRITE CRC ERR ')
0071      RETURN
      C
      C WIND FOWARD ONE FILE
      C
0072      5000 IF(ICOM.EQ.5)COM(1)=SDSCOM(5)
0074      IF(ICOM.EQ.6)COM(1)=SDSCOM(6)
0076      IF(ICOM.EQ.7)COM(1)=SDSCOM(2)
0078      COM(2)=ITLEN
0079      COM(3)=IDRV
0080      COM(4)=0
0081      CALL SDS10(COM,ISTAT, , )
      C
      C CLEAR IRRELEVANT BITS FROM ERROR BYTE
      C
0082      ISTAT=ISTAT.AND..NOT.MASK(6)
0083      ISTAT=ISTAT.AND..NOT.MASK(2)
0084      IF(ISTAT.EQ.0)RETURN
0086      ISTATI=ISTAT
0087      GOTO 40
0088      35 RETURN
      C
      C IN THIS SECTION THE MAIN TAPE ERRORS ARE
      C HANDLED SUCH AS:= TAPE BUSY,TAPE OFFLINE
      C SOT,EOT
      C
      C TAPE BUSY SECTION...AFTER CLEARING BOT FLAG
      C
0089      40 TYPE 1010,ISTATI,IFNUM
0090      1010 FORMAT(' STATUS=',I3,' FILE NO=',I4)
0091      ISTATI=ISTATI.AND..NOT.MASK(4)
0092      ITMP=ISTATI.AND.MASK(5)
0093      IF(ITMP.EQ.0)GOTO 80
0095      90 ECOM(1)=SDSCOM(1)
0096      ECOM(2)=0
0097      ECOM(3)=IDRV
0098      ECOM(4)=0
0099      CALL SDS10(ECOM,ESTAT, , )
      C
      C HAVING EXAMINED STATUS IF TAPE STILL
      C BUSY, LOOP AGAIN,IF NOT TRY COMMAND AGAIN
      C
0100      ESTATI=ESTAT
0101      ITMP=ESTATI.AND.MASK(5)
0102      IF(ITMP.NE.0)GOTO 90
0104      ITMP=ESTATI.AND.MASK(4)
0105      IF(ITMP.EQ.0.AND.ICOM.EQ.7)GOTO 90
0107      IF(ICOM.EQ.7)GOTO 35
```

```
0109      GOTO (1000,2000,3000,4000,5000,5000,5000)ICOM
      C
      C TAPE OFFLINE
      C
0110      60 ITMP=ISTATI.AND.MASK(1)
0111      IF(ITMP.EQ.0)GOTO 100
0112      TYPE 1001,IDRV
0114      1001 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
      C
      C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
      C
0115      110 ECOM(1)=SDSCOM(1)
0116      ECOM(2)=0
0117      ECOM(3)=IDRV
0118      ECOM(4)=0
0119      CALL SDS10(ECOM,ESTAT, , )
0120      ESTATI=ESTAT
0121      ITMP=ESTATI.AND.MASK(1)
0122      IF(ITMP.NE.0)GOTO 110
0124      GOTO (1000,2000,3000,4000,5000,5000,5000)ICOM
      C
      C EOT
      C
0125      100 ITMP=ISTATI.AND.MASK(3)
0126      IF(ITMP.EQ.0)GOTO 120
0128      TYPE 1002,IDRV
0129      1002 FORMAT(' EOT ON DRIVE ',I1)
0130      IDRVS=3
0131      RETURN
0132      120 GOTO(50,50,70,70,35,35,35)ICOM
      C
      C ERROR EXIT RETURN
      C
0133      130 ISTAT=-1
0134      RETURN
0135      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

READ(1,1000)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

```

0001 REAL*8 FSPEC
0002 REAL*4 DEVNAM,FMBUF(3)
0003 LOGICAL*1 SDSCOM(4),COM(8),LUN,HBLK(10),FNUM,ITYP
0004 INTEGER*2 STATUS,TLEN,FLEN
0005 DATA COM/'0','1','2','3','4','5','6','7/
0006 DATA DEVNAM/3RRK /
0007 IF(IFETCH(DEVNAM).NE.0)STOP'FETCH ERROR'
0009 IF(ICDFN(30).NE.0)STOP'CDFN ERROR'
0011 1 CONTINUE
0012 TYPE 10
0013 10 FORMAT(' ENTER TEST OPTION ',/,
           %' 0 FOR A READ',/,
           %' 1 FOR A WRITE',/,
           %' 2 FOR UTILITY COMMANDS',/,
           %' 3 TO LEAVE THE PROGRAM :',S)
0014 ACCEPT 20 ,IOPT
0015 20 FORMAT(I1)
0016 IF(IOPT.EQ.3)STOP'EXECUTION TERMINATED'
0018 IF(IOPT.EQ.2)GO TO 60
0020 TYPE 30
0021 30 FORMAT(' ENTER THE FILE SPEC,NO COLONS OR DOTS',/,
           %' BUT WITH IMPLIED BLANKS INCLUDED:',S)
0022 ACCEPT 40,FMBUF
0023 40 FORMAT(3A4)
0024 CALL IRAD50(12,FMBUF,FSPEC)
0025 IF(IOPT.GT.0)GOTO 50
C
C READ
C
0027 IENT=IENTER(20,FSPEC,400)
0029 IF(IENT.LT.0)TYPE 45,IENT
0030 45 FORMAT(' ENTER ERROR IENT=',I3)
0031 IF(IENT.LT.0)STOP
0032 TYPE 70
0033 70 FORMAT(' ENTER TAPE UNIT TO BE READ FROM :',S)
0034 ACCEPT 20,LUN
0035 TYPE 71
0036 71 FORMAT(' ENTER 0 FOR FIELD TAPE 1 FOR INTERNAL:',S)
0037 ACCEPT 20,ITYP
0038 SDSCOM(1)=COM(4)
0039 SDSCOM(2)=ITYP
0040 SDSCOM(3)=LUN
0041 SDSCOM(4)="200"
0042 CALL SDS10(SDSCOM,STATUS,TLEN,FLEN)
0043 TYPE 35,STATUS,FLEN
0044 35 FORMAT(' STATUS EQUALS...',I3,' FILE LENGTH= ',I3)
0045 IF(ITYP.EQ.1)GOTO 91
0046 IF(IREADW(5,HBLK,0,20).LT.0)STOP'READ HBLK ERROR'
0047 DO 30 I=1,10
0048 30 HBLK(I)=HBLK(I)+"60"
0049 TYPE 90,HBLK(2),HBLK(1),HBLK(4),HBLK(3),HBLK(6),HBLK(5),
           %HBLK(8),HBLK(7),HBLK(10),TLEN
0050 90 FORMAT(' TAPE HEADER INFO',/,
           %' FILE NO = ',3A1,/,

```

```

% DATA CONSTANTS = ',5A1,/,
% SAMPLING PERIOD = ',A1,/,
% DATA TIME LENGTH = ',I2)
0054 91 CALL CLOSEC(20)
0055 GO TO 1
C
C WRITE
C
0056 50 FLEN=LOOKUP(21,FSPEC)
0057 IF(FLEN.LT.0)STOP 'LOOKUP ERROR'
0059 FLEN=(FLEN+3)/4
0060 TYPE 35,STATUS,FLEN
0061 TYPE 95
0062 95 FORMAT(' ENTER THE TAPE UNIT TO BE WRITTEN TO:',S)
0063 ACCEPT 20,LUN
0064 SDSCOM(1)=COM(7)
0065 SDSCOM(2)=FLEN
0066 SDSCOM(3)=LUN
0067 SDSCOM(4)=1
0068 CALL SDS10(SDSCOM,STATUS,,FLEN)
0069 TYPE 35,STATUS,FLEN
0070 CALL CLOSEC(21)
0071 GOTO 1
C
C WIND
C
0072 60 FNUM=0
0073 LNUM=0
0074 TYPE 100
0075 100 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-8E TO OS8',/,
%' ENTER YOUR OPTION :',S)
ACCEPT 20,NOPT
0077 IF(NOPT.EQ.8)GOTO 110
0078 TYPE 120
0079 120 FORMAT(' ENTER TAPE UNIT NO:',S)
0080 ACCEPT 20,LUN
0081 IF(NOPT.EQ.5.OR.NOPT.EQ.6)TYPE 130
0082 130 FORMAT(' ENTER NO OF FILES TO BE WOUND PAST :',S)
0083 IF(NOPT.EQ.5.OR.NOPT.EQ.6)ACCEPT 20,FNUM
0084 110 SDSCOM(1)=COM(NOPT)
0085 SDSCOM(2)=FNUM
0086 SDSCOM(3)=LUN
0087 SDSCOM(4)=0
0088 CALL SDS10(SDSCOM,STATUS,,)
0089 TYPE 35,STATUS,FLEN
0090 GOTO 1
0091 END

```

```

REAL*8 FSPECR,FSPECW
REAL*4 FBUF(3),SEISM(2048)
LOGICAL*1 TPDRR,ISTAT,ITLEN
DATA DEV/3RRK /
IF(ICDFN(25).NE.0)STOP 'CHAN ERR'
IF(IFETCH(DEV).NE.0)STOP 'FETCH ERROR'
CALL ASSIGN(1,'DK2:MPTPK.DAT',14)
CALL ASSIGN(2,'DK2:MPTPK.LOG',14)
IRD=20
IWRT=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,FSPECR)
NWDS=NBLKS*256
IBLKOT=1
IBLKSZ=NFILIN*NBLKS+1
IF(IENTER(IWRT,FSPECW,IBLKSZ).LT.0)STOP 'ENTER ERR'
DO 20 I=1,NFILIN
IFIL=1
IF(IENTER(IRD,FSPECR,300).LT.0)STOP 'ENTZ ERR'
IF(TPDRR.LE.2)GOTO 30
WRITE(2,1400)IFIL
1400 FORMAT(' FILE NO ',I5,' EOT ')
CALL CLOSEC(IWRT)
STOP 'EOT'
30 CALL TAPRED(-1,TPDRR,ISTAT,ITLEN,IFLEN,IFIL)
IF(ISTAT.LT.0)WRITE(2,1500)IFIL
1500 FORMAT(' FILE NO ',I5,' RETRIES FAILE D LAST READ USED')
IF(TPDRR.GT.2)GOTO 40
CALL TAPRED(0,TPDRR,ISTAT, , ,IFIL)
40 CALL CLOSEC(IRD)
IF(LOOKUP(IRD,FSPECR).LT.0)STOP 'LOOKUP ERR'
IF(IREADW(NWDS,SEISM,IBLKST,IRD).LT.0)STOP 'READ ERR'
IF(IWRITE(NWDS,SEISM,IBLKOT,IWRT).LT.0)STOP ' WRITE ERR'
IBLKOT=IBLKOT+NBLKS
CALL CLOSEC(IRD)
20 CONTINUE
CALL CLOSEC(IWRT)
STOP 'NORMAL TERMINATION'
END

```

IBM NUMAC MTS Software

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