The design and implementation of the Durham university seismic processing system

Poulter, Michael John

7 5

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

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 Asbery, for helping me to keep it all working.

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

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

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

Contents

Chapter 1

Introduction

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.

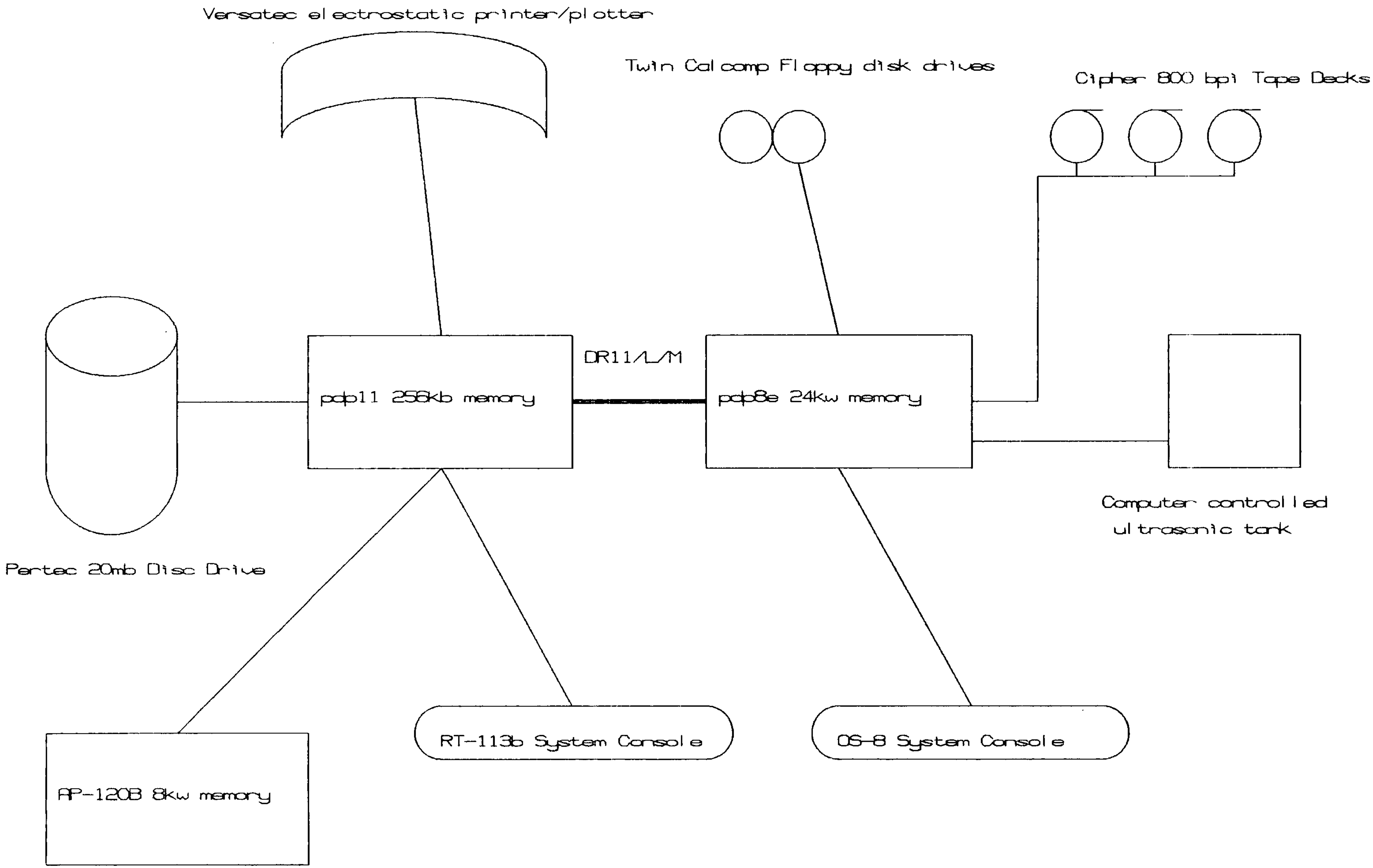


Fig 1.1: Hardware configuration of the Durham University seismic processing system

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 minimising 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.

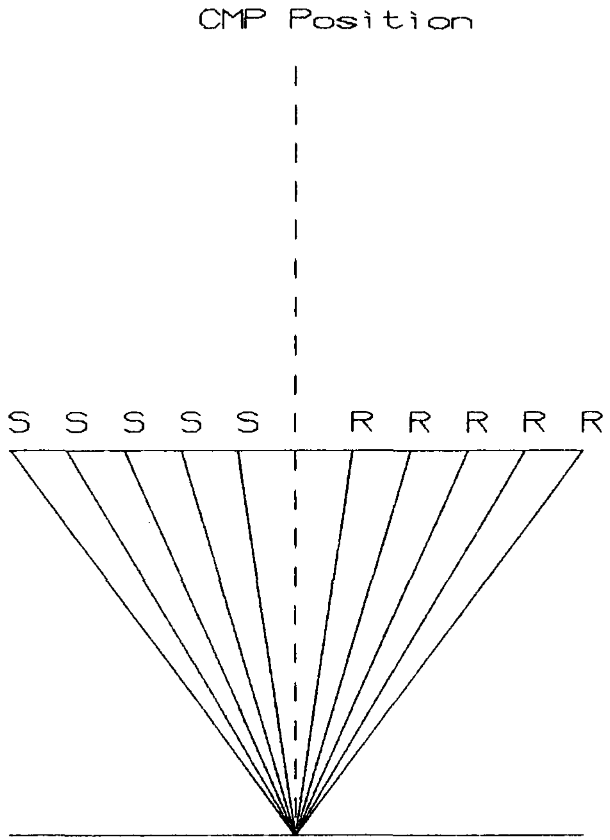


Fig 2.1 : Common Mid-Point Geometry

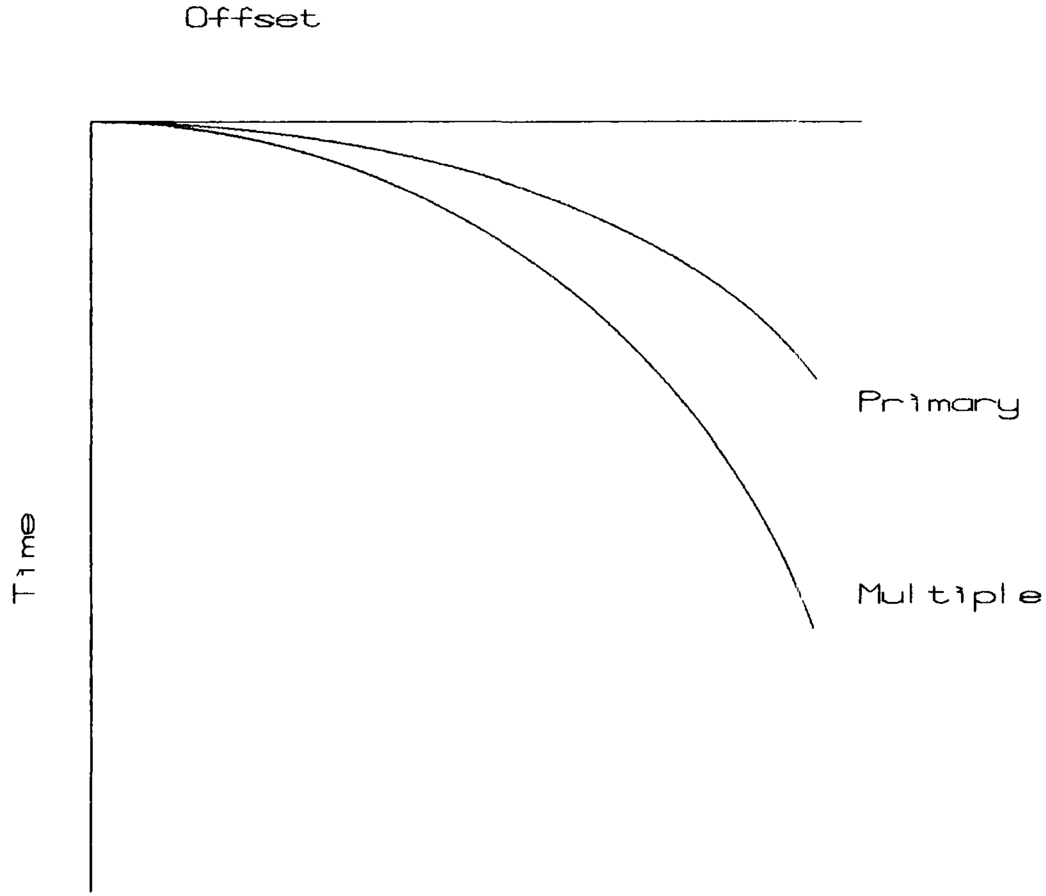


Fig 2.2 : Travel time relation in CMP geometry

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 straightfoward 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.

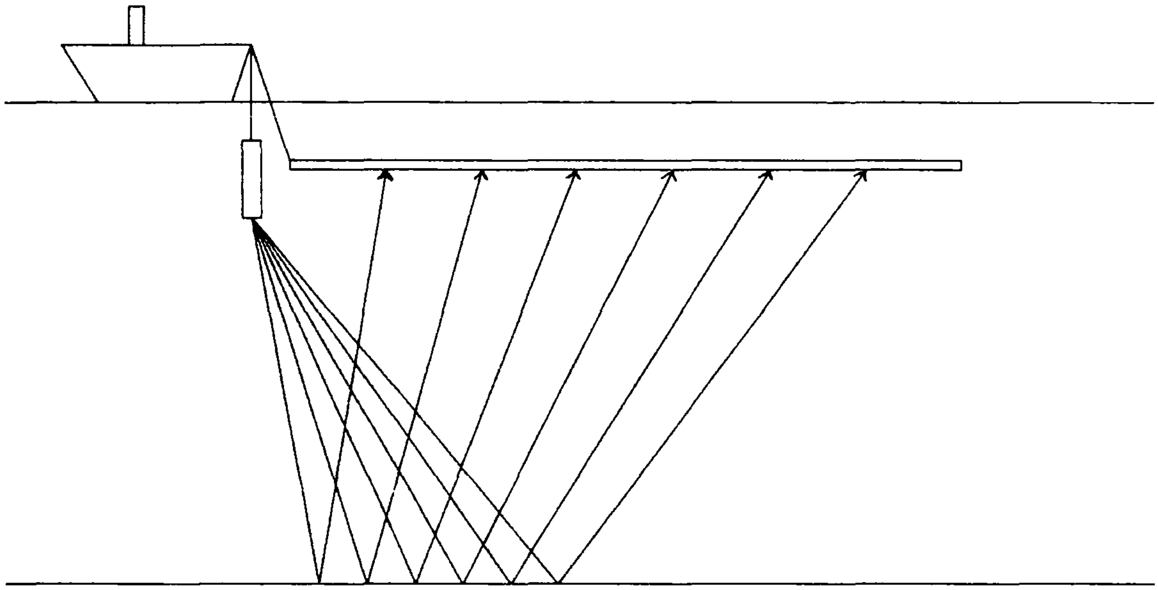


Fig 2.3 : Marine acquisition configuration

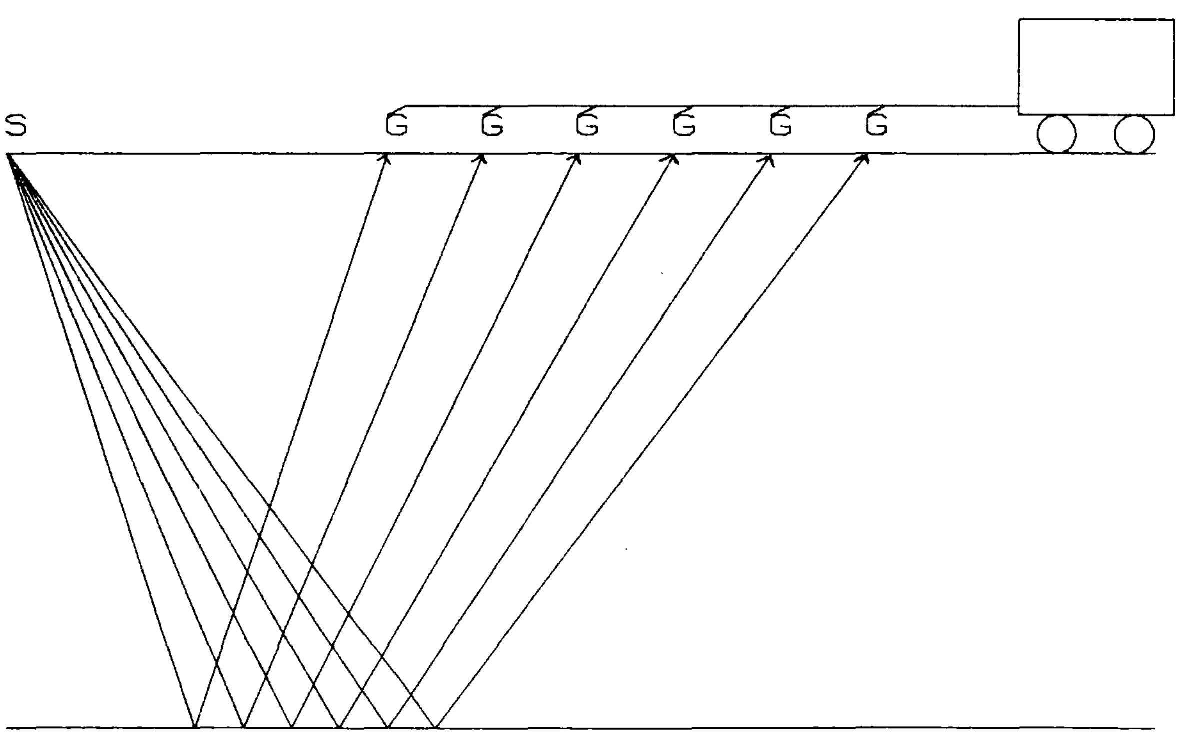


Fig 2.4 : Land acquisition configuration

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

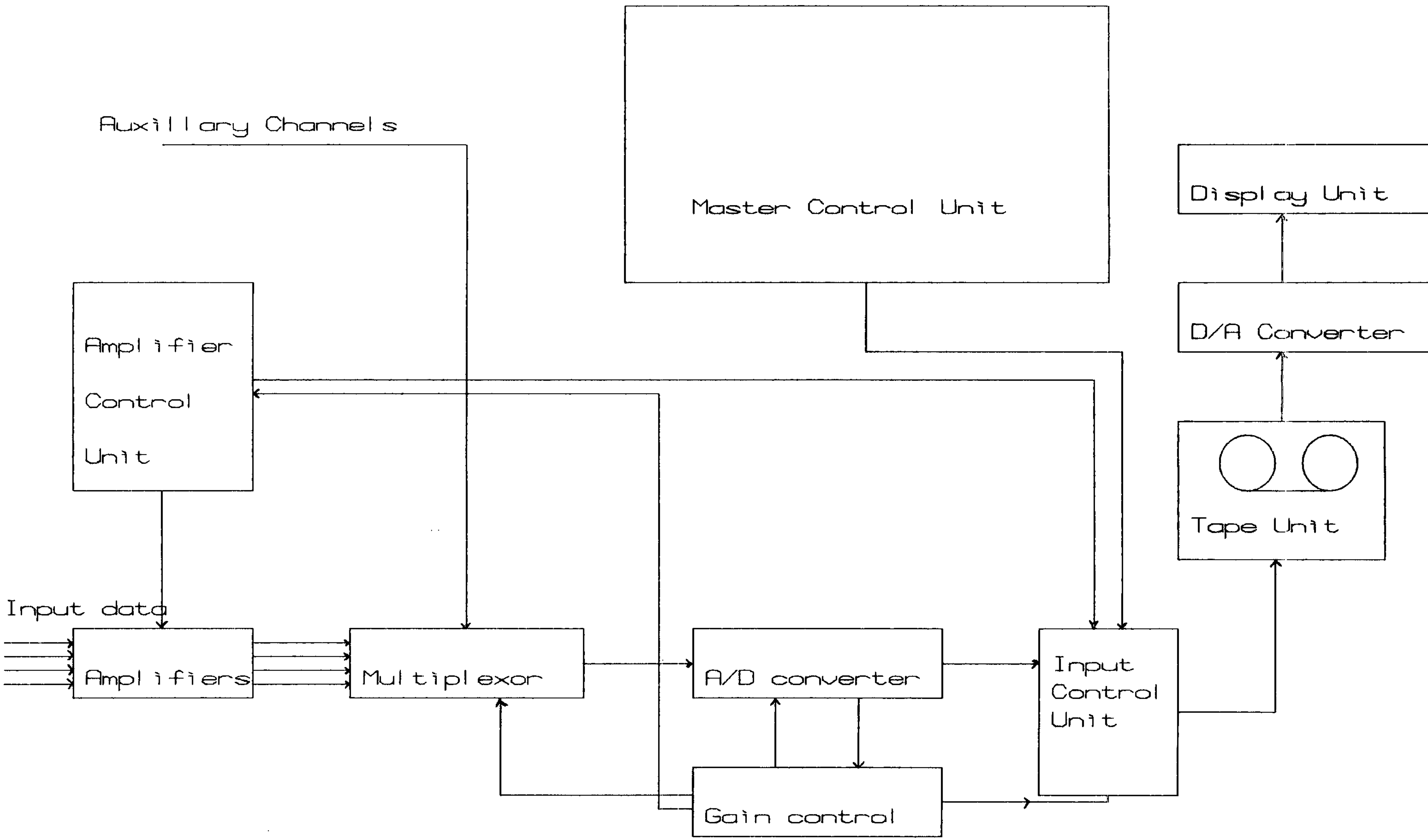


Fig 2.5 : Block diagram of a seismic data acquisition system

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

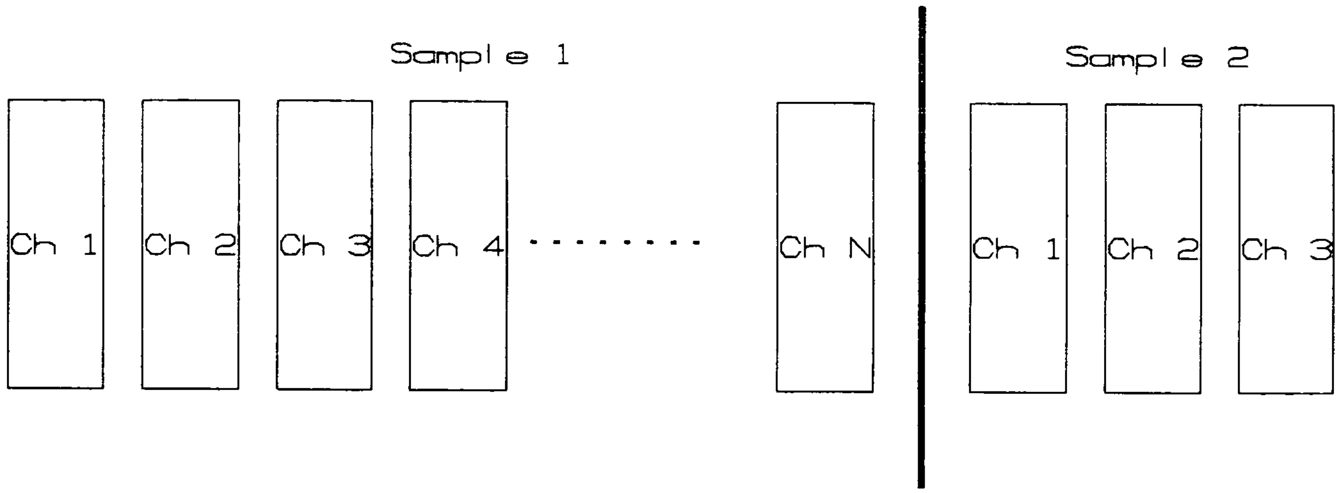
NMO correction

Stack................Improve S/N

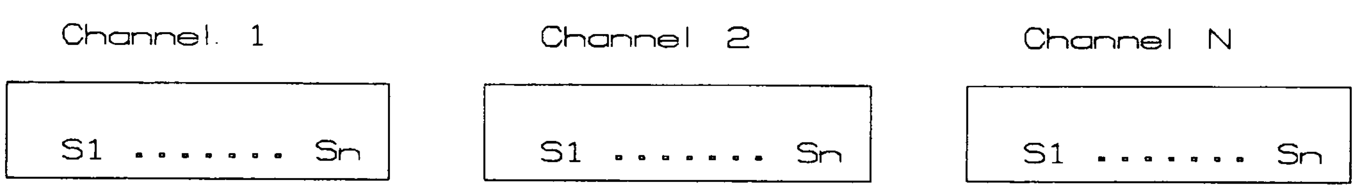
Residual statics

Deconvolution

Time varying filtering



Multiplexed format



Trace Sequential format

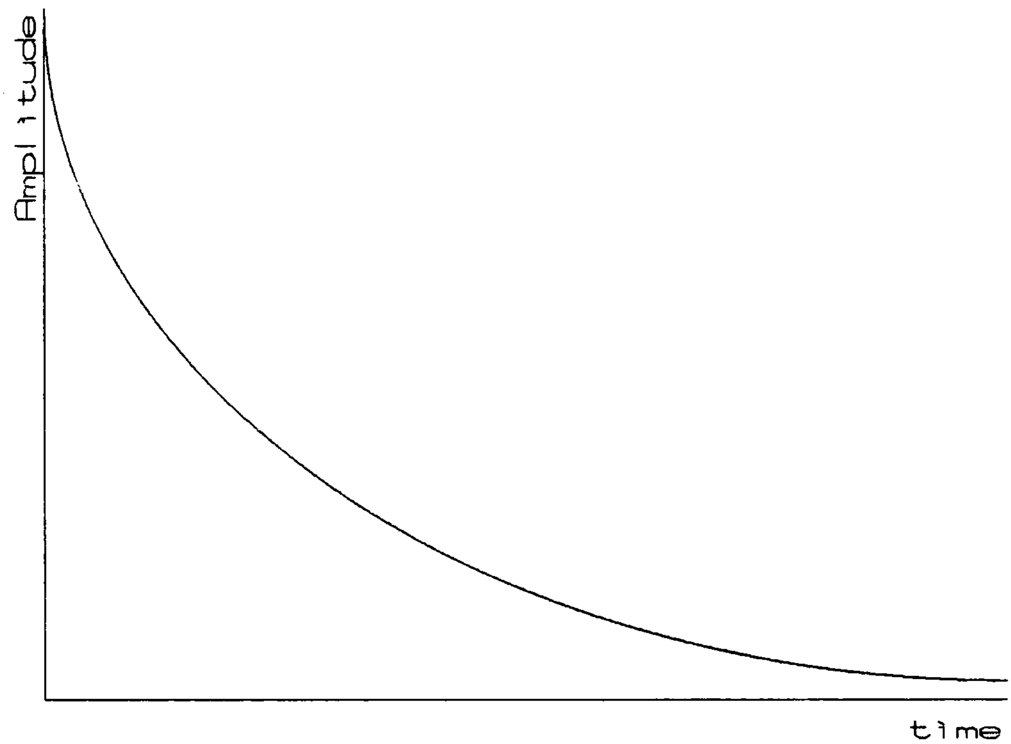
Fig 2.6 : Data organisations

Migration............Image data to correct location

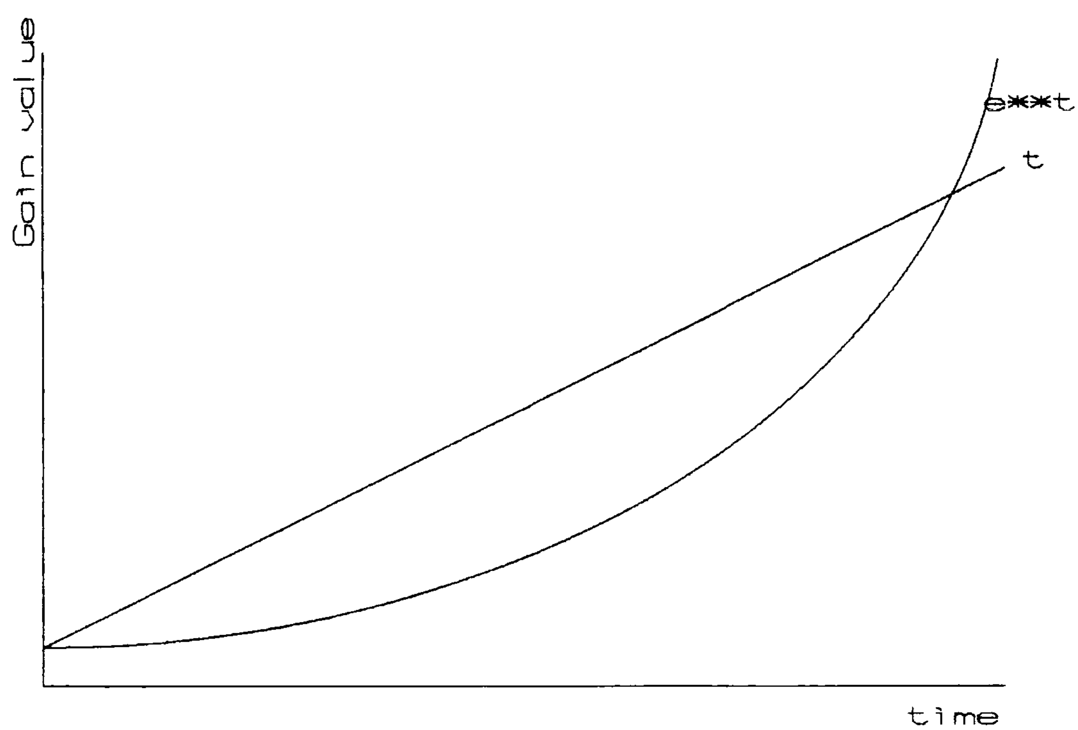
Processing

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 has been found to work well in practice.



Data amplitude



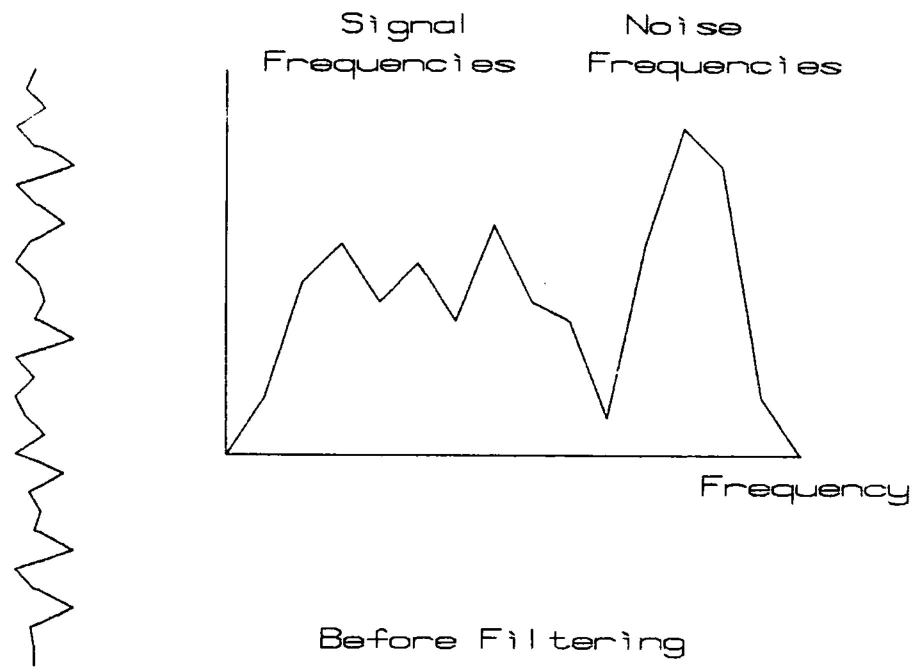
Amplitude Recovery curves

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.



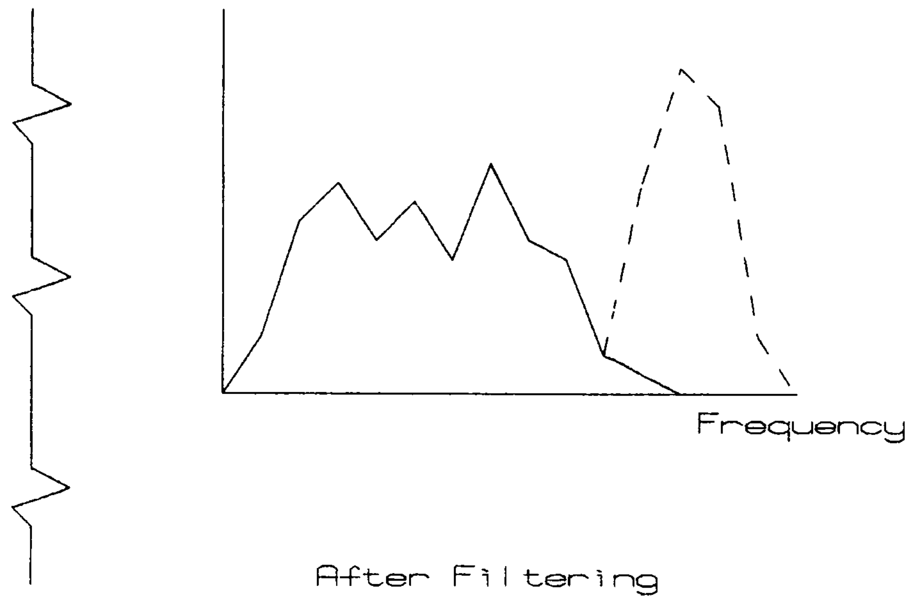


Fig 2.8 : Example of Frequency Filtering

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

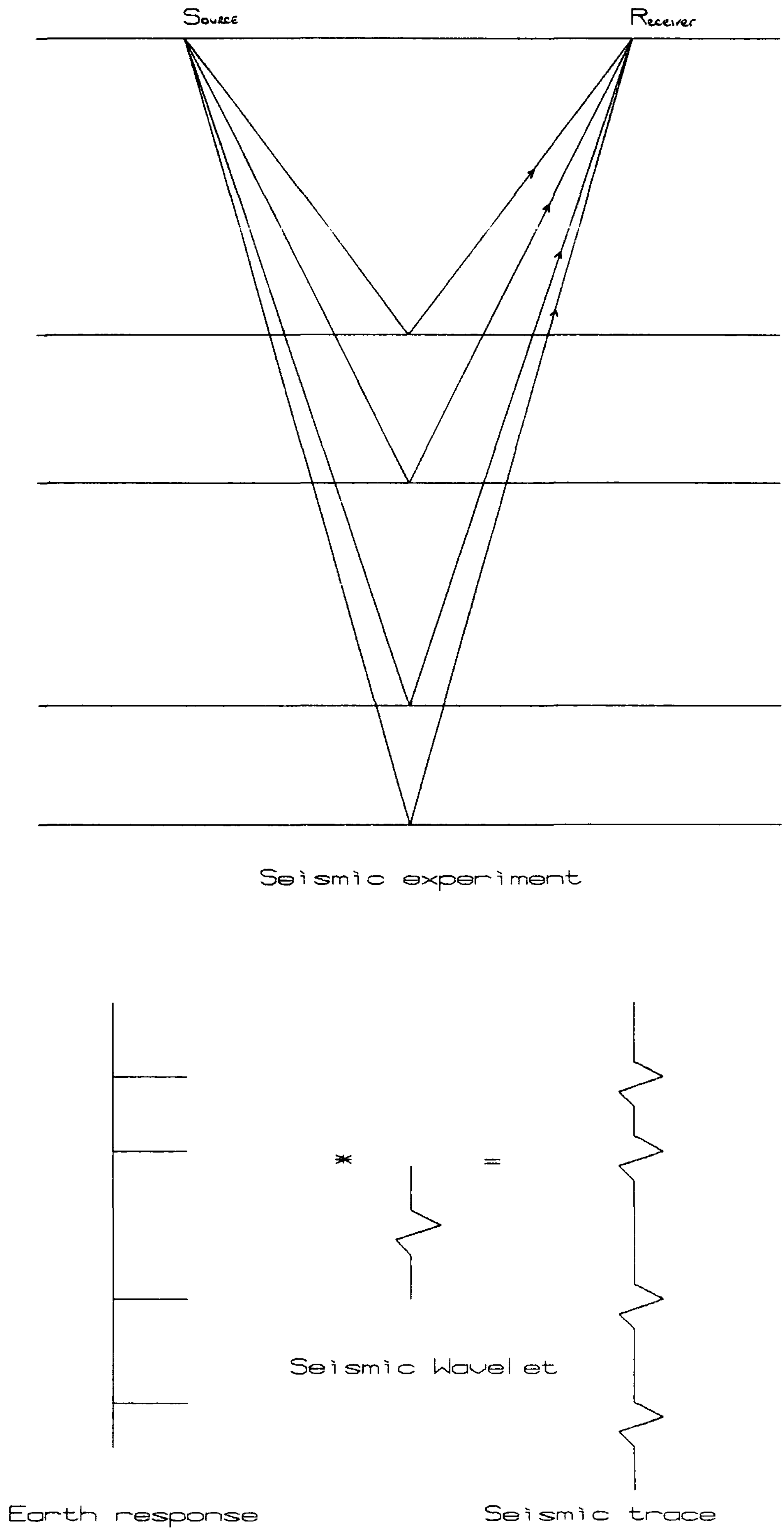


Fig 2.9 : Convolutional model of the seismic trace

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.

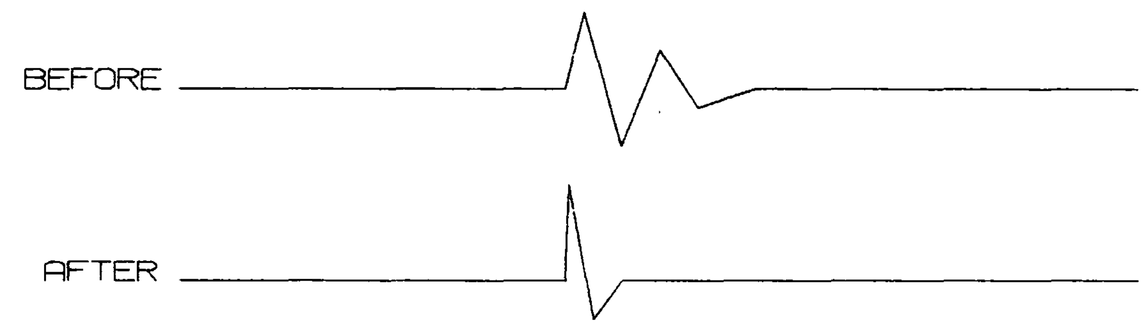
However, in the vast majority of cases the source wavelet is unknown and so an attempt to remove its effect is usually made by attempting to find an estimate of the wavelet from the statistics of the trace(Robinson and Treitel, 1967). These methods are based on the premise that as the primary reflection 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, wheras 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)

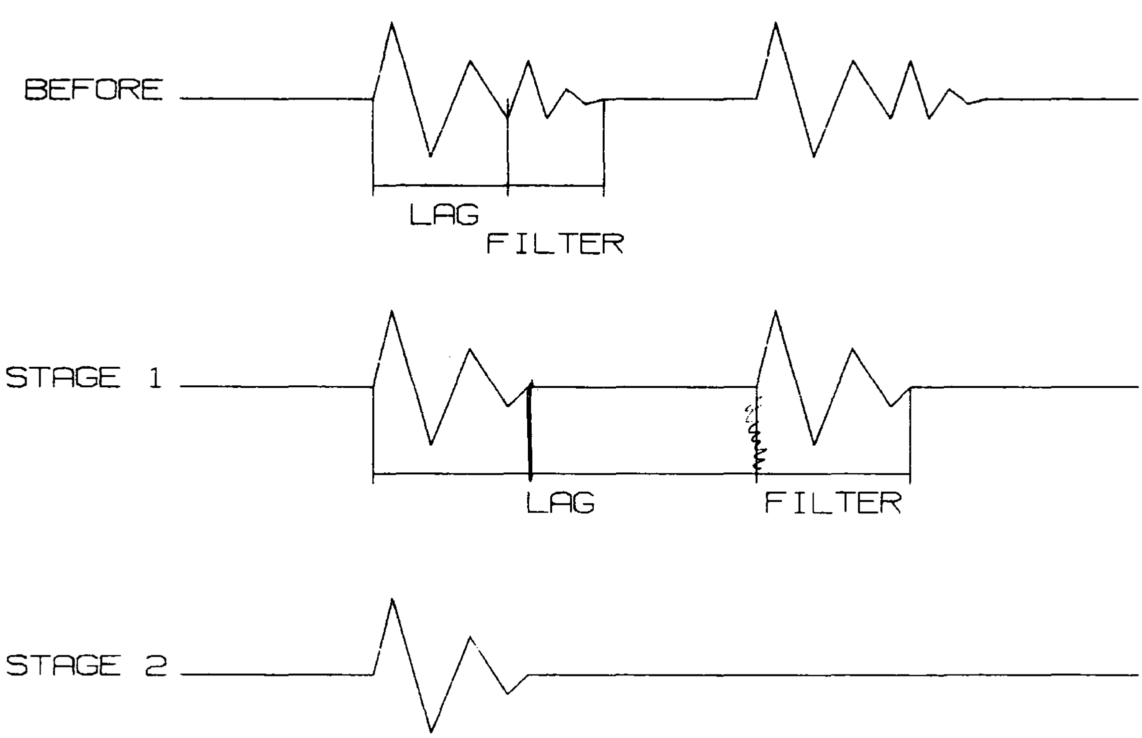
If the prediction distance is set up to be the same as the period of a long period multiple, then the prediction error filter can be used to dereverberate the trace. Also it can be used to compress an airgun wavelet, to improve resolution, by having a prediction distance just less than one wavelength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

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

Migration is an attempt to image the reflection events on the CMP section back to their correct spatial locations. A CMP section is displayed as though each event recorded on a particular trace was produced by a reflector perpendicularly below the surface,at the Common Mid-point. It can be shown quite easily that this assumption is untrue, for anything other than horizontally layered horizons. Therefore it is important to apply migration in order to display the horizons in their correct spatial locations.

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

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

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

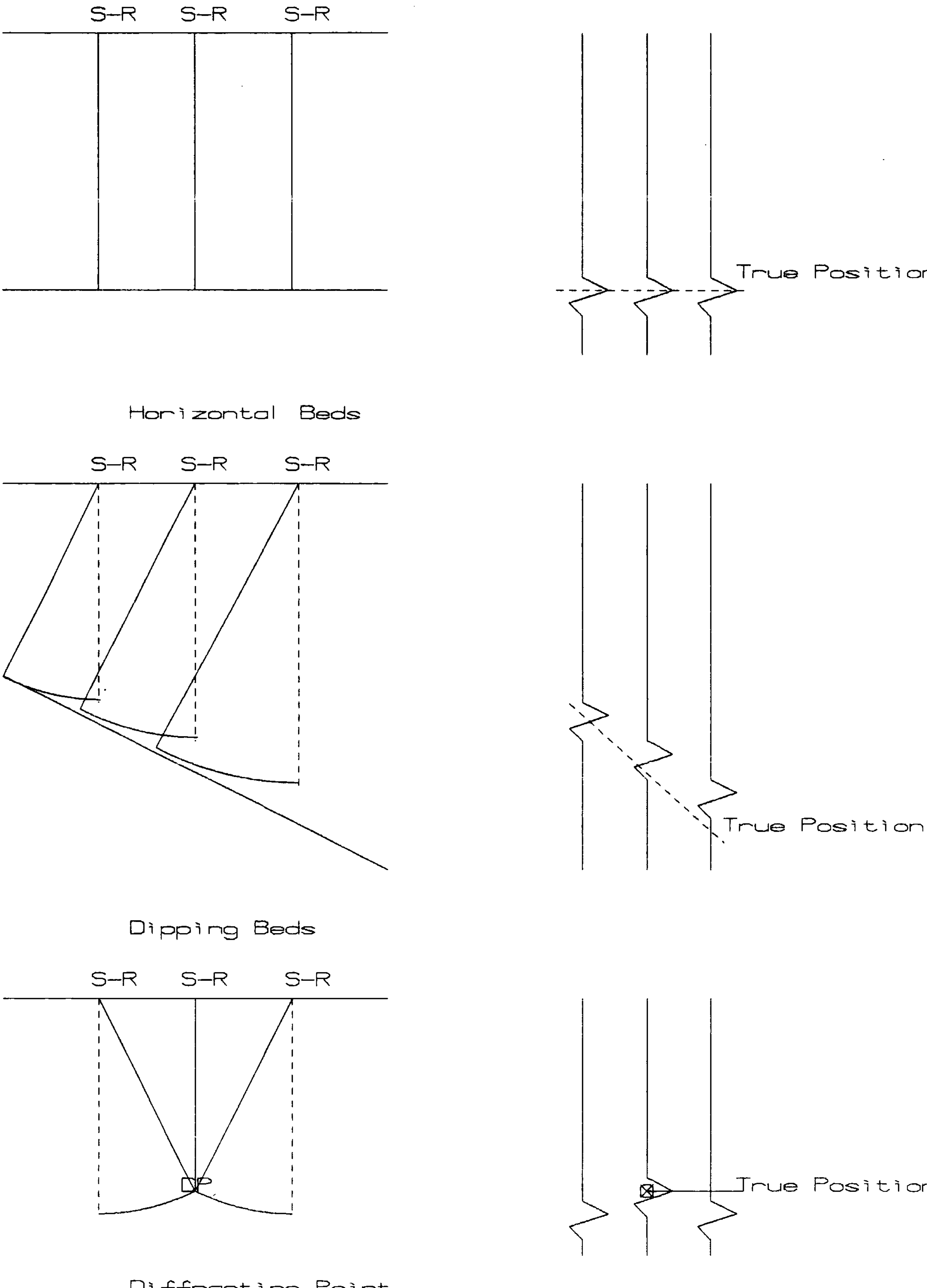


Fig 2.11 : Diagram to show mis-positioning of seismic events

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

Summary

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

Chapter 3

Hardware and Systems Software

Hardware

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

The hub of the system is the Digital Equipment Corporation pdp 11/34. This is a 16 bit word length minicomputer with 256Kilobytes (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 AP120B are via the AP executive(APEX). This software provides a means of transferring data and microcode to and from the AP and monitoring the execution of AP microcode, in order to return error conditions and check for microcode termination. A full library of microcode routines(Floating Point Systems Inc, 1977) was provided with the machine. These routines have a Fortran callable interface which links into APEX to achieve transfer of the microcode to the AP. This library provides a comprehensive suite of routines for vector operations and it is rare to find an operation which cannot be performed by using a combination of these routines. However should the user find an application he wishes to perform, which cannot be achieved using existing routines, a new microcode routine can easily be developed using the software development tools available for the AP, an assembler (APAL), linker (APLINK), simulator(APSIM) and debugger(APDBUG). A full suite of diagnostic programs were also provided with the AP.

In order to drive the electrostatic printer/plotter as a plotter, the Versaplot library(Versatec, 1978) of plotting routines was purchased. This library provides a suite of Fortran callable subroutines which emulate the standard Calcomp graphics subroutines. They are used at a high level to produce vector type plots, such as graphs and annotation. Also provided, as part of the library, are programs to perform vector to raster conversion, and an input/output package which takes the rasters produced and outputs them to the plotter.

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

The systems console on the pdp 8/e is linked into OS/8. This is a reasonably powerful disc based operating system developed for the pdp 8 series of computers. Although it has a rather rudimentary keyboard command language, it does provide a useful suite of utility programs for file and peripheral manipulation. It also provides facilities for program development in pdp 8 assembler PAL-8, and Fortran IV, with a multiple pass Fortran compiler. This compiler converts the Fortran into a pseudo-assembly language RALF, and the RALF assembler is then run to produce an object module. All the peripherals on the pdp 8/e, such as the tape decks and the video screen have OS/8 compatible device drivers and so can be manipulated by the standard utilities.

The hardware link between the pdp 11 and the pdp 8 was the only data pathway for which there was no controlling software once the system was fully configured. All the other peripherals could be manipulated, to a greater or lesser extent , using the facilities of the two operating systems and the additional software provided by the AP microcode library, APEX and the Versaplot library.

Linking the Two Minicomputers

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

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

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

<u>RT-11 to OS/8 transfer</u>

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

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

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

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

RT-11 to OS/8

```
RT-11..........COPY DK1:MPDMXA.FOR DR:*.*
OS/8...........R PIP
.............*MTA0:<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 ultasonic tank are written to tape in pdp8 floating point format, which consists 3 words, or 36 bits per floating point value. However, in order to use the facilities on the pdp 11 to handle this data, it is necessary to transfer it and simultaneously convert it to the 32 bit floating point representation used on the pdp11.

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

Tape Handling

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

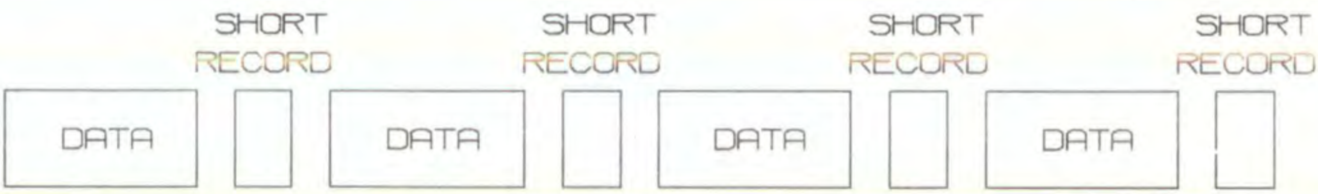
Processing System Tape Subroutines

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 treat 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

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

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

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

As virtual arrays are also supported in the less sophisticated Single Job (SJ) Monitor of RT-11, this monitor was investigated. Under this monitor the relocation of the high addresses to the I/O page is unaffected by the virtual arrays option being present, and so this problem is immediately overcome in this environment. Additionally this monitor is much less sophisticated and so has the advantage of occupying much less space in memory than the Extended Memory Monitor. However on further investigation major disadvantages were found in its implementation of the virtual arrays principle.

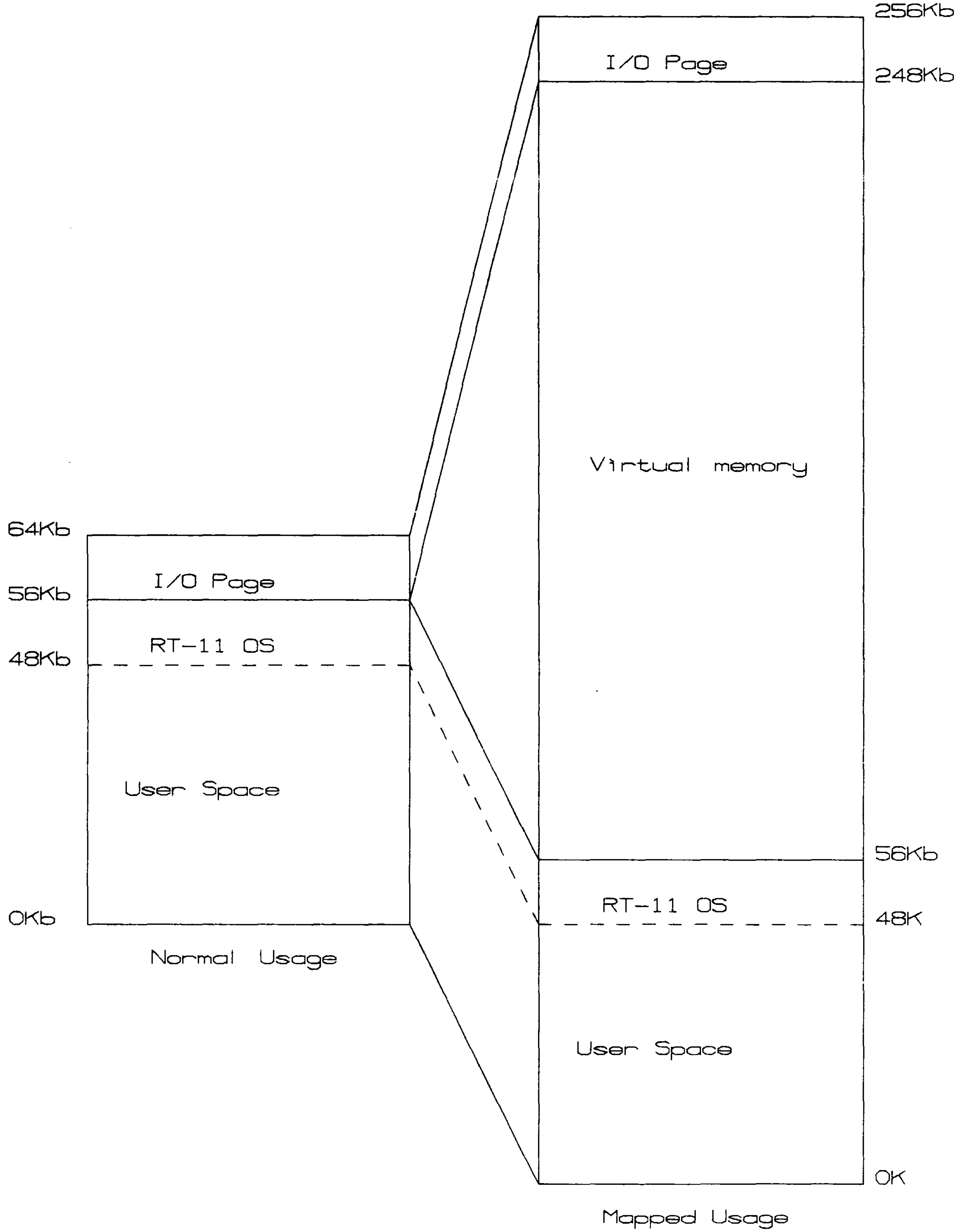


Fig 3.2 : Memory configuration of the pdp 11/34

The Fortran compiler generates a reference to a set of utility subroutines every time a virtual array element is referenced in a Fortran program. These subroutines are given the offset of the start of the virtual array from address 1600 octal(56Kbyte) in 64 Kbyte blocks, and the number of the array element referenced. By dumping the machine code from memory when such an operation was in progress, it was possible to decode the method used to access the data at this extended address.

In normal running under the SJ monitor, the KT-11 memory mapping unit is switched off and the virtual addresses 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 refering to a virtual array element.

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

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

Virtual Memory Input/Output

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

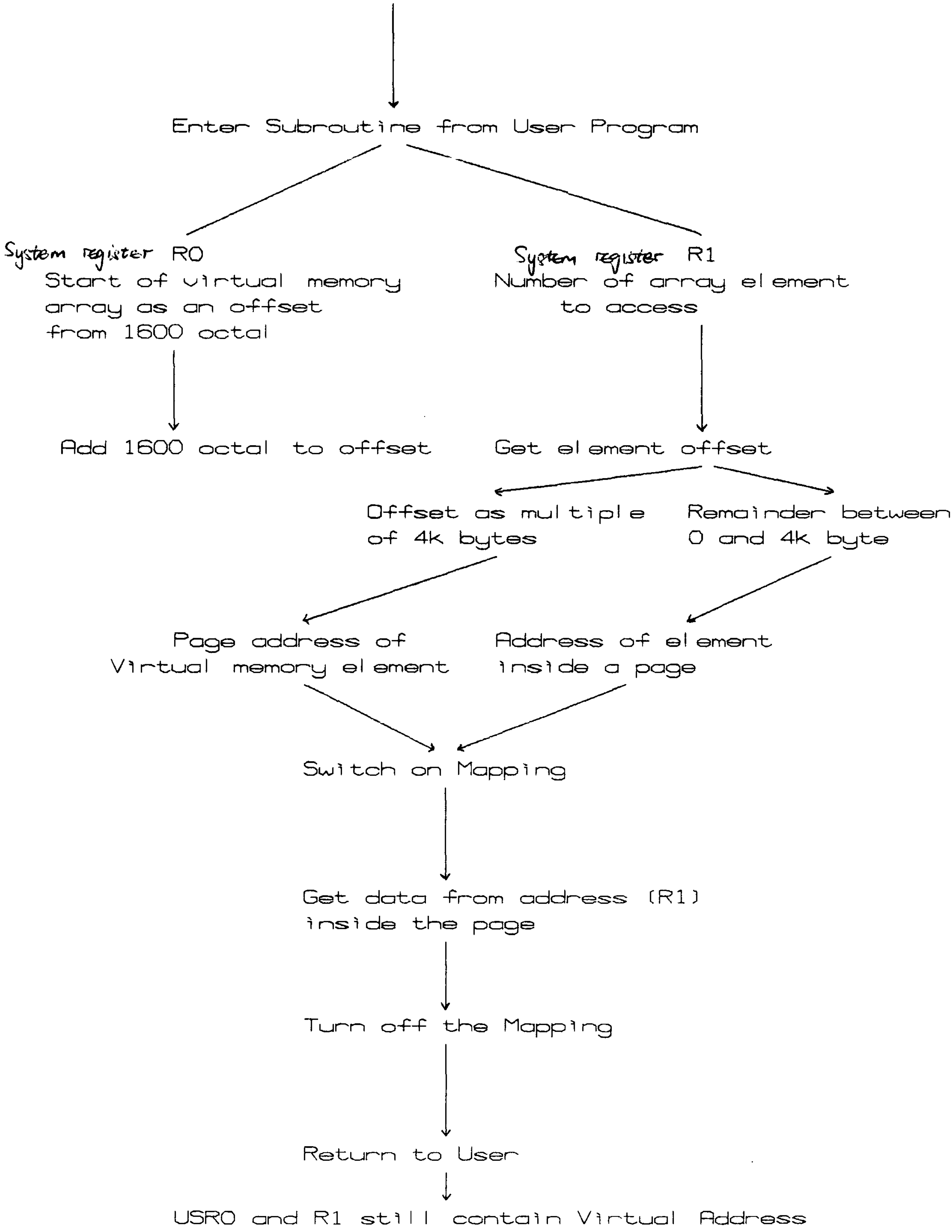


Fig 3.3 : Virtual Memory access in RT-11 Fortran

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 foward, 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 intiating 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 (Northumbrian Universities' Multiple Access Computer) IBM 370/168. Although written in Fortran it was not easily transferable to the pdp11 as, besides using several constructs unique to IBM Fortran it also used operating system calls to allocate dynamic memory for its work space during execution. On the other hand the package produces good quality contours and uses a reasonably time efficient algorithm. Therefore it was decided to convert this package to run on the pdp11. The dynamic memory allocation was rewritten to use static work arrays passed to it by the calling program, in virtual memory. Once this had been accomplished and other parts had been rewritten in standard Fortran it was linked into the Versaplot package for drawing the contour lines. This package provides a full contouring capability, which can easily be used by any display program which requires contouring.

Seismic Displays

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

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

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

In order to develop the software an interactive section plotting routine was produced, which expected its input to be on the disc and writes the raster output file back to disc. However once developed this algorithm was incorporated into a full section plotting program for the processing system.

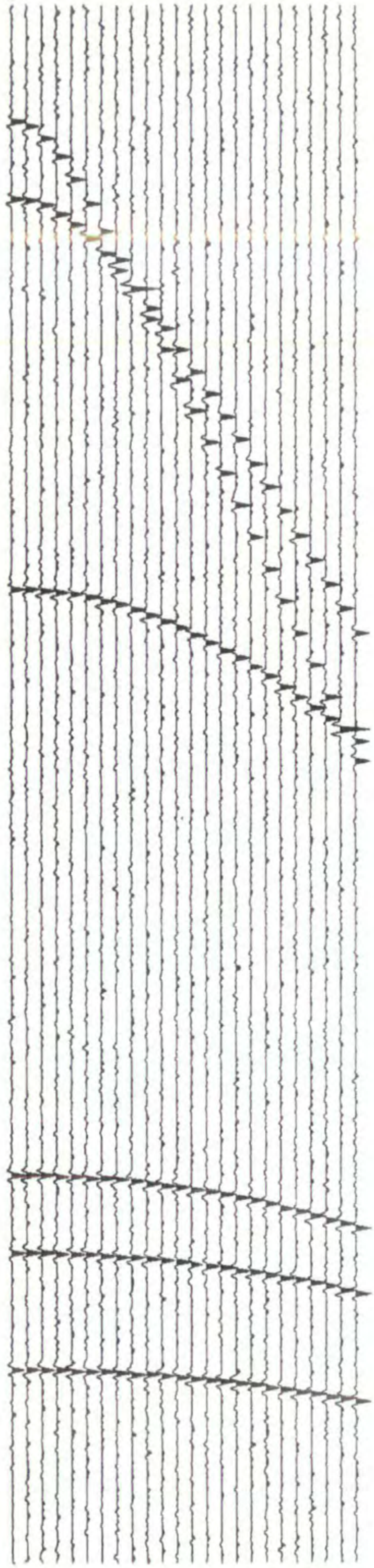


Fig 3.4: Example of output from the Seismic Display Package

It was realised that once a file of rasters have been produced, other plots could be merged with the seismic display, before the rasters are transferred to the plotter, as long as the other plot is also in raster form.

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

The next logical step was to develop a post processor which took the raster input from up to 8 files and merges them according to offsets and ranges specified by the user, and even reverses the contrast if required, before using the plotter driver software to put the final rasters onto the plotter. This software allows the seismic image produced by the special display software ,to be merged with axes, time scales and annotation which is produced by a program using Versaplot routines, and so allows each approach to be used solely in the mode in which it is most efficient.

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 acquired in order to provide an interpretation aid for seismic reflection data, as well as high quality synthetic data for research work, was much too large to fit onto the pdp11, and so had to be installed on the IBM. Examples of the ouput 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 straightfoward as it is written in standard Fortran. The only alterations necessary were to change the input/ output unit usage to be compatible with MTS usage and to alter the plotting calls so as to fit in with the *PLOTSYS system on MTS. This package provides a very powerful raytrace modelling and synthetic seismogram tool for use in conjunction with the seismic processing system.

As has already been explained, some users find the limited program debugging facilities available with the RT-11 Fortran extremely difficult after having become accustomed to more powerful facilities on large mainframes such as the NUMAC IBM 370. Hence program development could tend to take longer than usual at first. With the pdp11 being run as a single user system it means that while program development is taking place it cannot be used for processing and vice versa. Therefore it was decided to

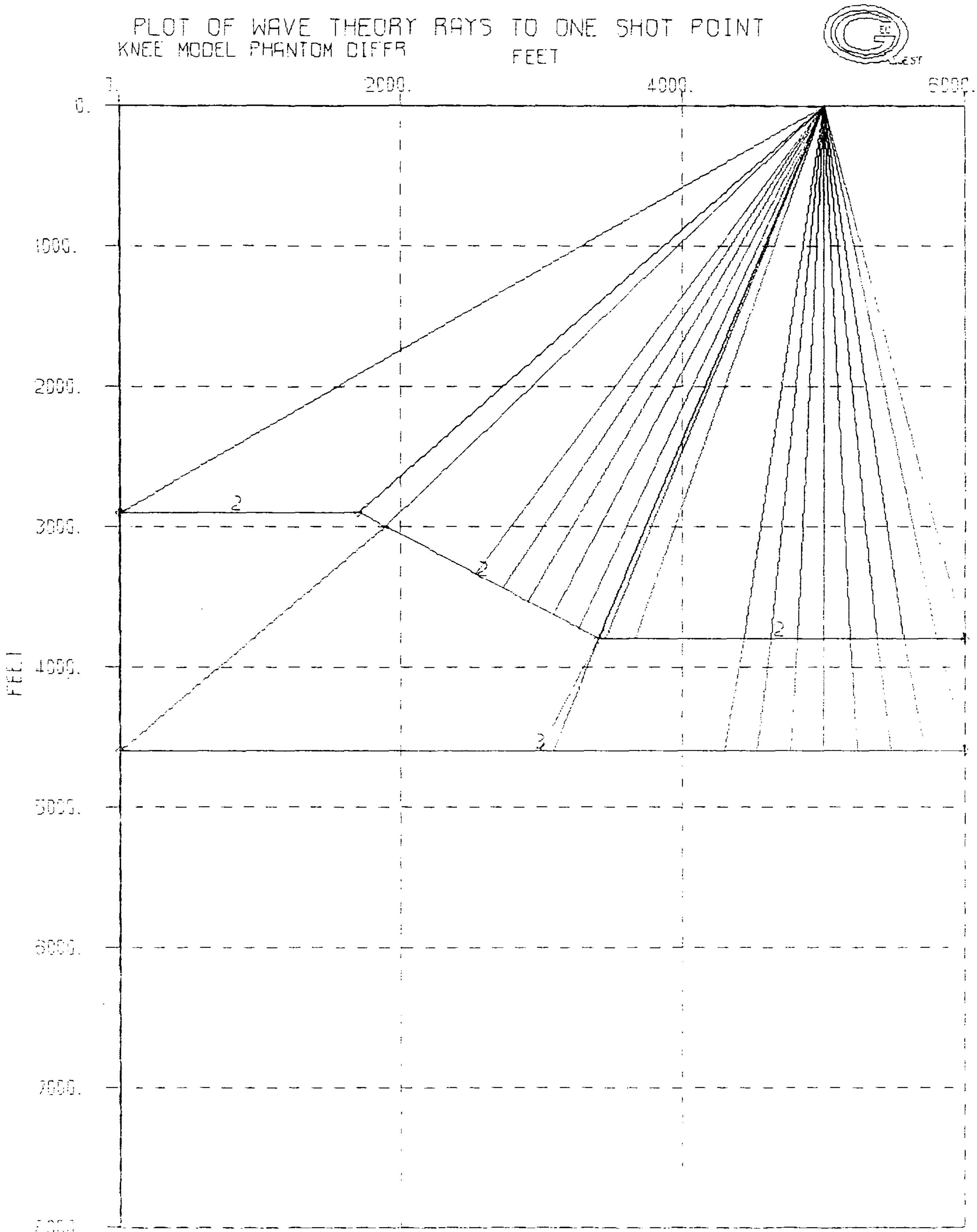


Fig 3.5a: Example of an AIMS input model

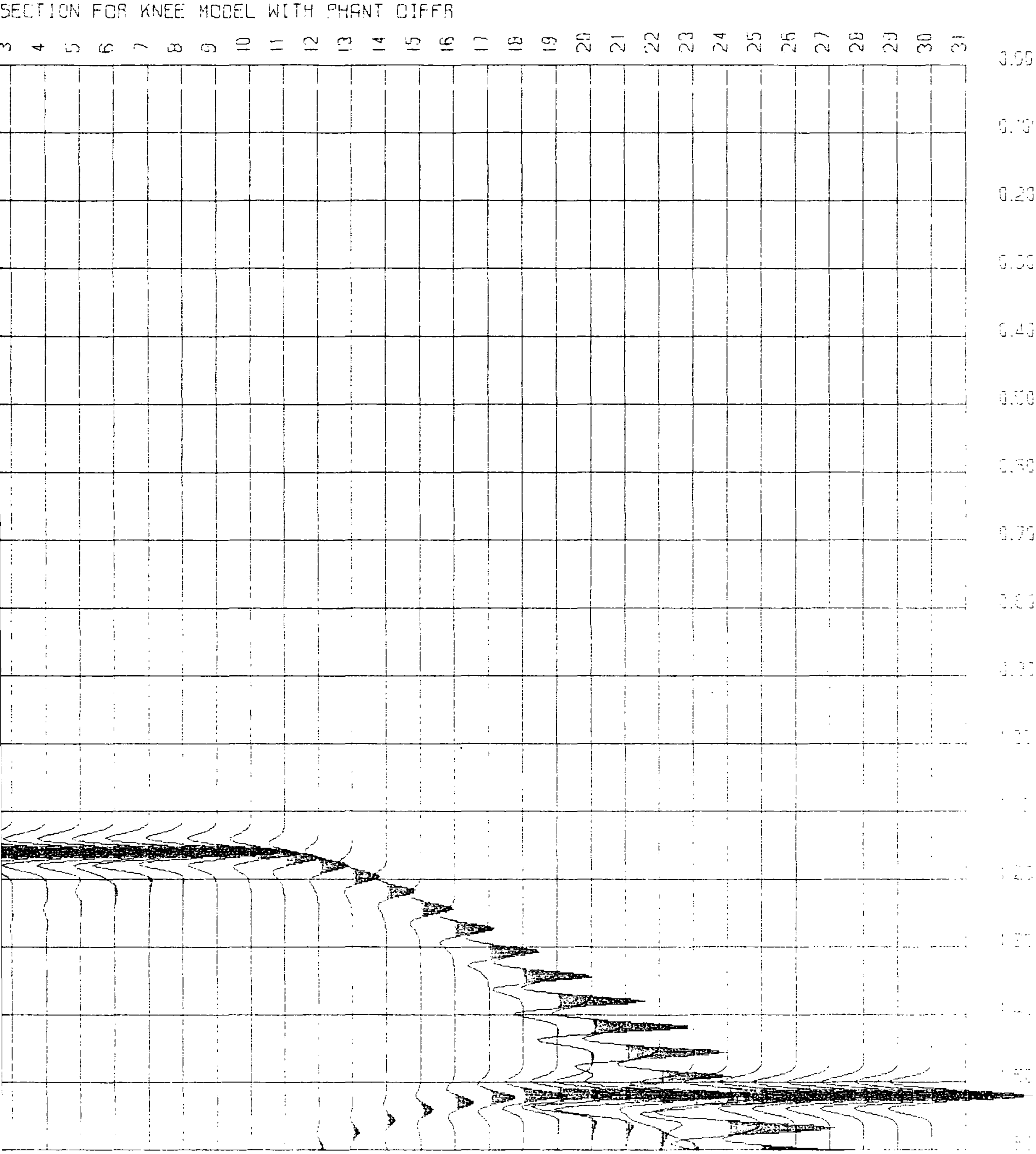


Fig 3.5b: AIMS output from the model in Fig 3.5a

provide development facilities on the NUMAC IBM so that the basic algorithm of a new program could be developed offline from the pdp11 and transferred to it, at a later date, for interfacing to the processing system.

At this time FPS, the suppliers of the AP had a Fortran simulator of their mathematics library under development. A copy was acquired and developed to run on the IBM 370, while still looking to the user as though it was running on the pdp11. By writing programs in as near standard Fortran (in this case ANSI FORTRAN 66) as possible and using this simulator, a program can be developed on the IBM which is easily transferable to the pdp11 when complete. At this point only the disc input/output needs to be changed and the tape access software added to the program, before final tests can be run. In practice this has proved to be an extremely valuable tool, having allowed MSc students to develop algorithms on the IBM, while the pdp11 is in use, transfer the programs to the pdp11 and then run them on the pdp11 on large datasets, which could not be handled in a reasonable time on the multiuser general purpose IBM system.

Summary

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

Chapter 4

Seismic Processing Software

Overview

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

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

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

Overall Design Considerations

Most commercially available seismic processing systems have a seismic monitor, which is responsible for taking menu type input for a particular job, structuring the modules required into a run stream and controlling the data flow through the system. This possibility was considered for controlling the operation of the Durham seismic procesing 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 seveal 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 stand alone 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 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

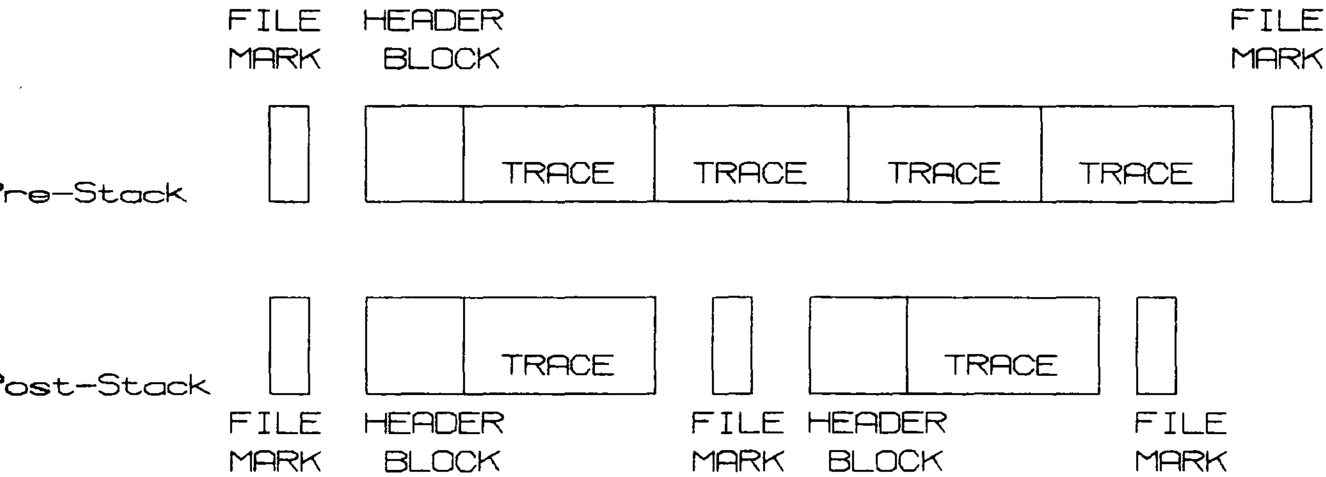
As a further aid to simplifying data transfers it was decided to make the header block, containing information on the data, consist of one 512 byte block, which would be written to block 0 in a disc file. It was felt that 512 bytes would provide adequate space for all present, and any foreseeable future,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

Byte position	Function
1–3 4–5 6–8	ASCII values from field tape header block
9	Sampling interval in msecs(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

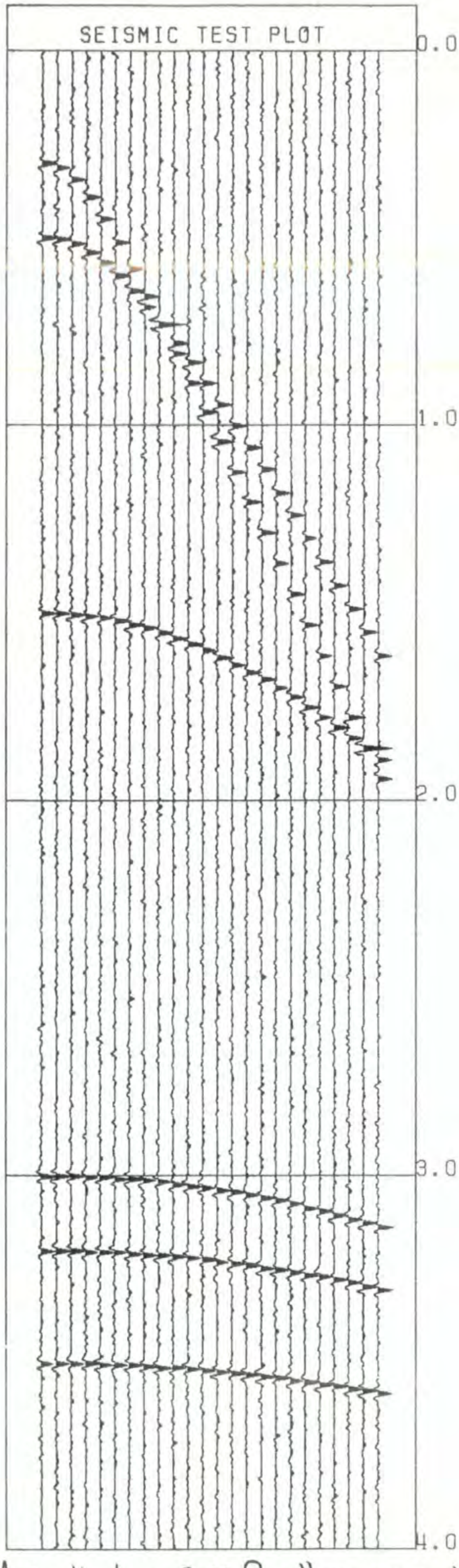


Fig 4.2: A Synthetic CMP gather generated by ANSEI

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

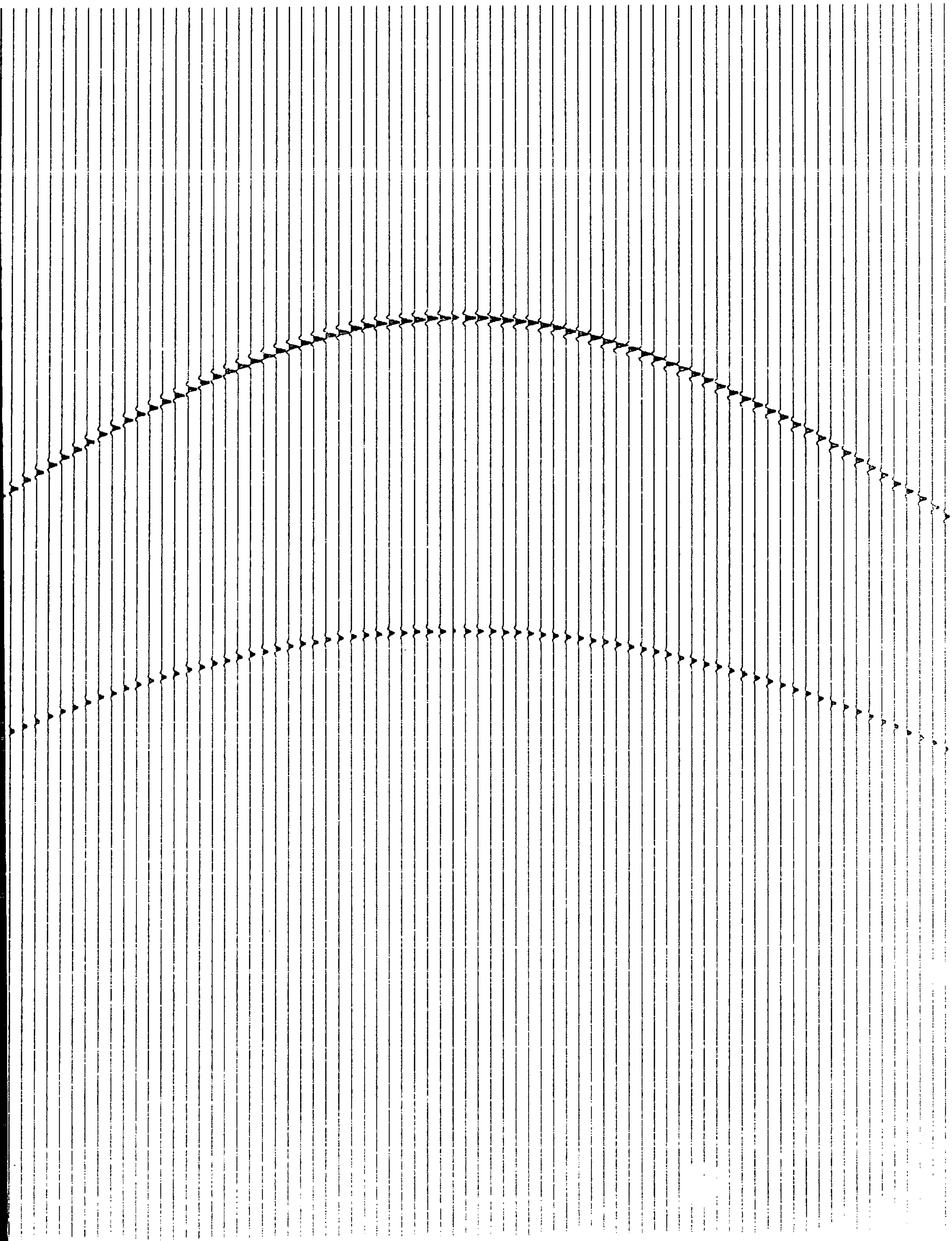


Fig 4·3: A synthetic Stacked Section Showing Diffractors output compatible with the seismic processing system.

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 impule 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 occured. 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 flexibilty 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.

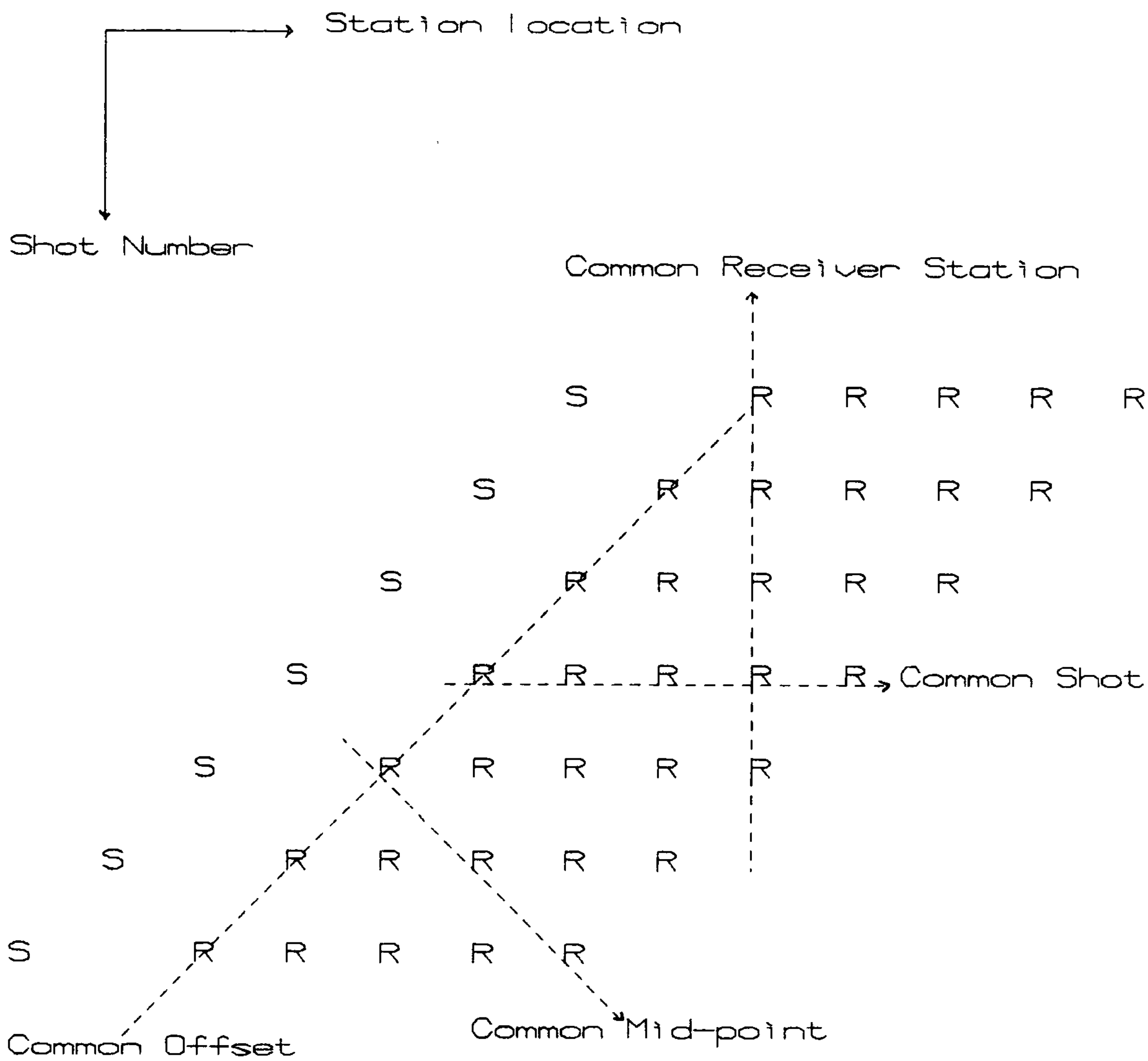


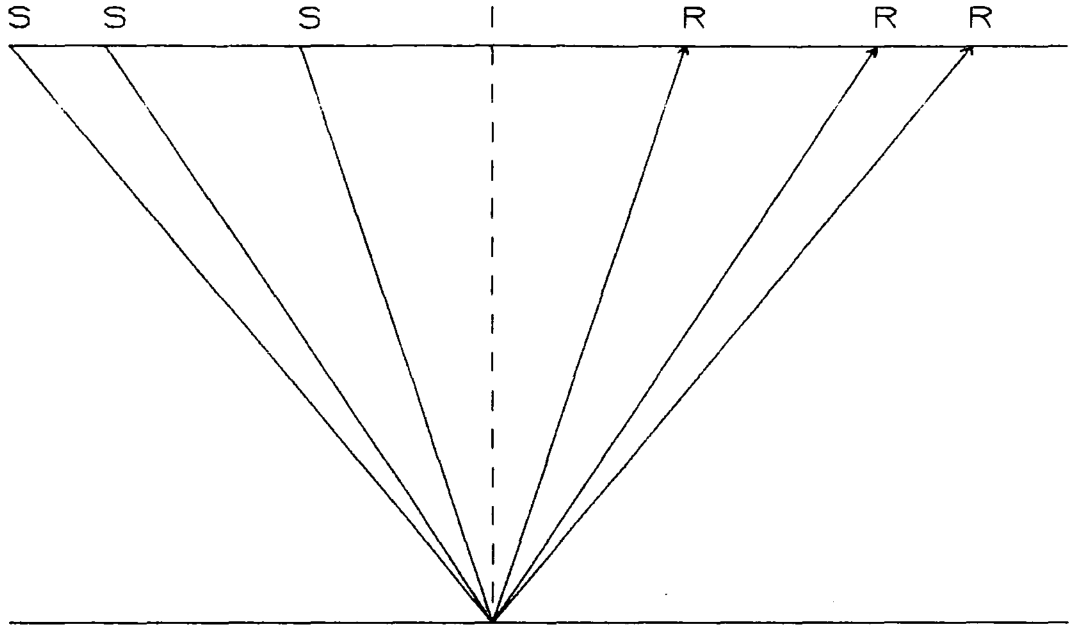
Fig 4.6 : CMP Acquisition Geometry

Sort

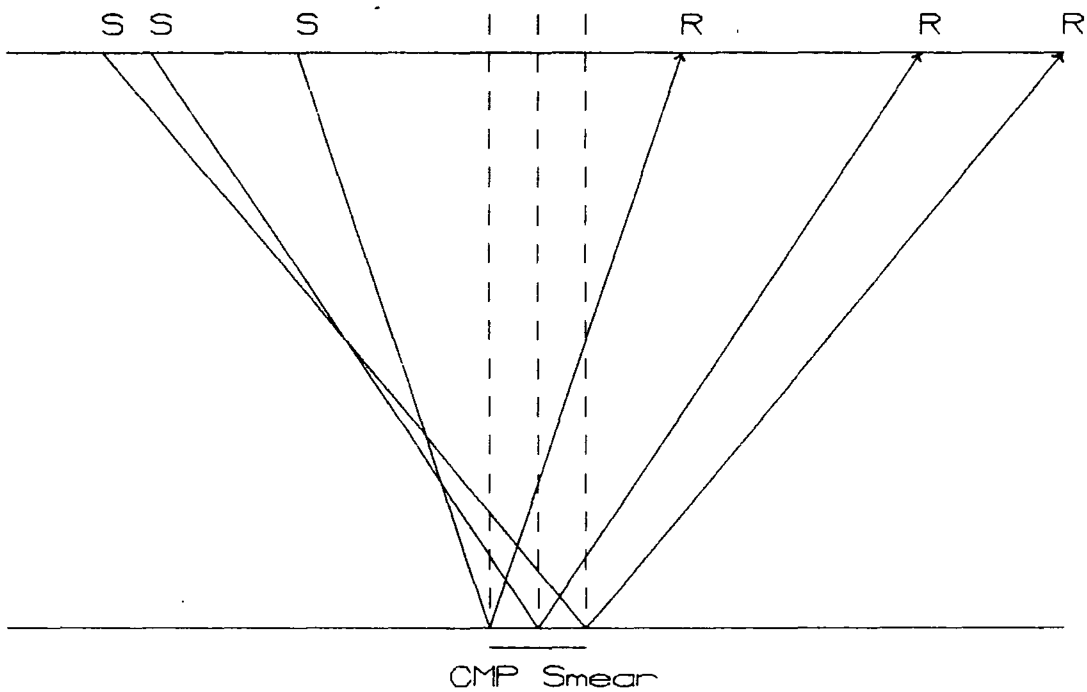
It was decided that during demultiplex all the data on the field tape should be demutliplexed and written to tape as common shot gathers, as mentioned previously, so that once it has been completed the field tapes would not have to be refered to again. Therefore, in order to select the segment of the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

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

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



Correct CMP positioning



Incorrect CMP positioning

Fig 4.7 : The Smear Effect of Incorrect CMP positioning

An example with CMP gathers as input files

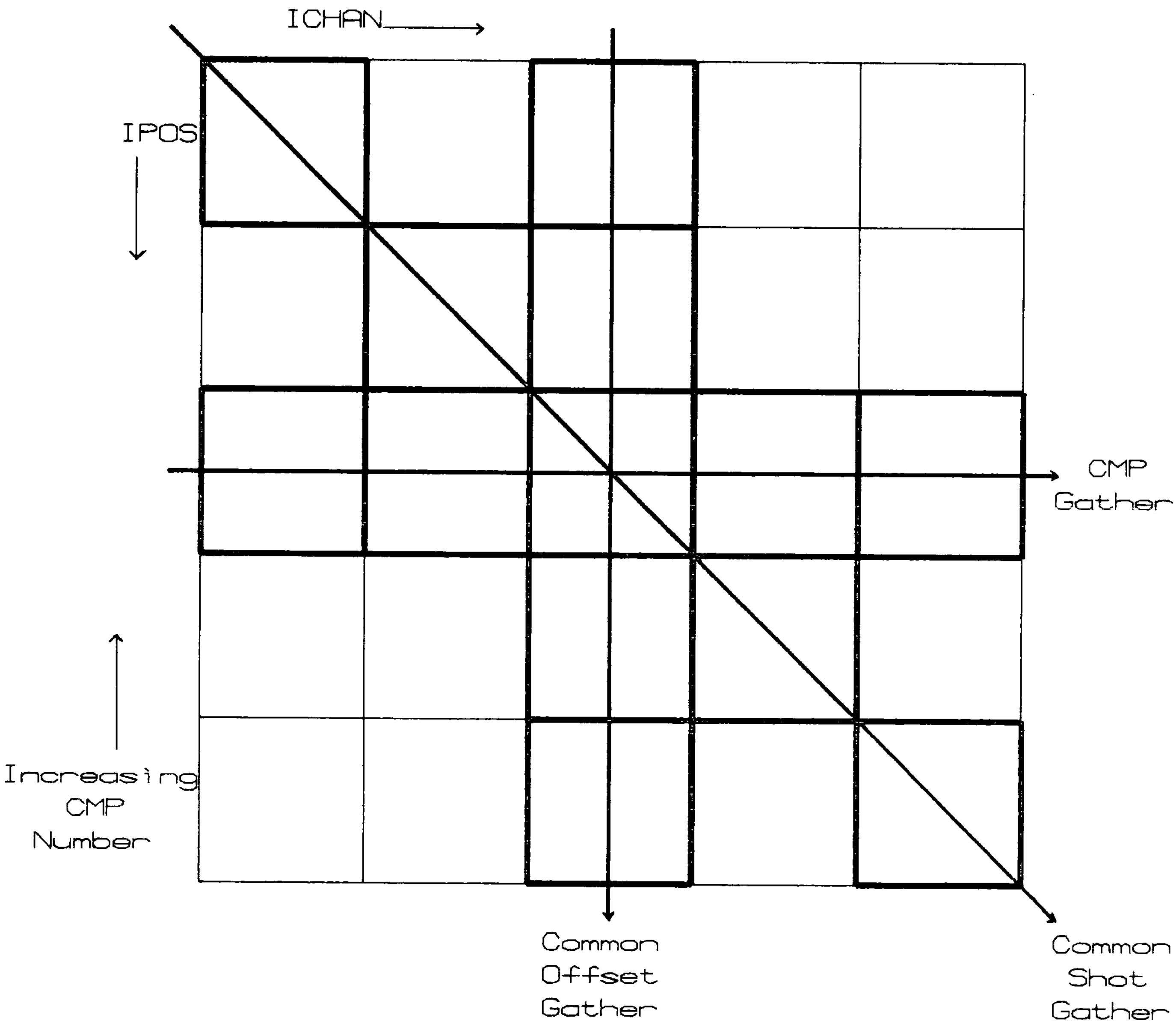


Fig 4.8 : Generating Different Trace Gathers using SORT

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

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

It is possible that tape problems may cause the program to close down. If this occurs, the last line in the Log file for the job contains an index to the order of the temporary files which it has written out before terminating. If the program is restarted this can be input, along with the restart, flag and the job will continue from the point in the sort at which it was interrupted, so that the whole job does not have to be resubmitted.

The major features of the sort algorithm were included in the demultiplex program so that reordering into CMP gathers, for test purposes, can be accomplished straight from the field tapes.

It was felt that this sort program provides sufficient flexibility to allow marine data and most land data to be reordered into any configuration desired. The only situation which would be difficult for it to cope with would be in the crooked line sorting of land data, where the optimum sort configuration for CMP gathers, is contantly 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.

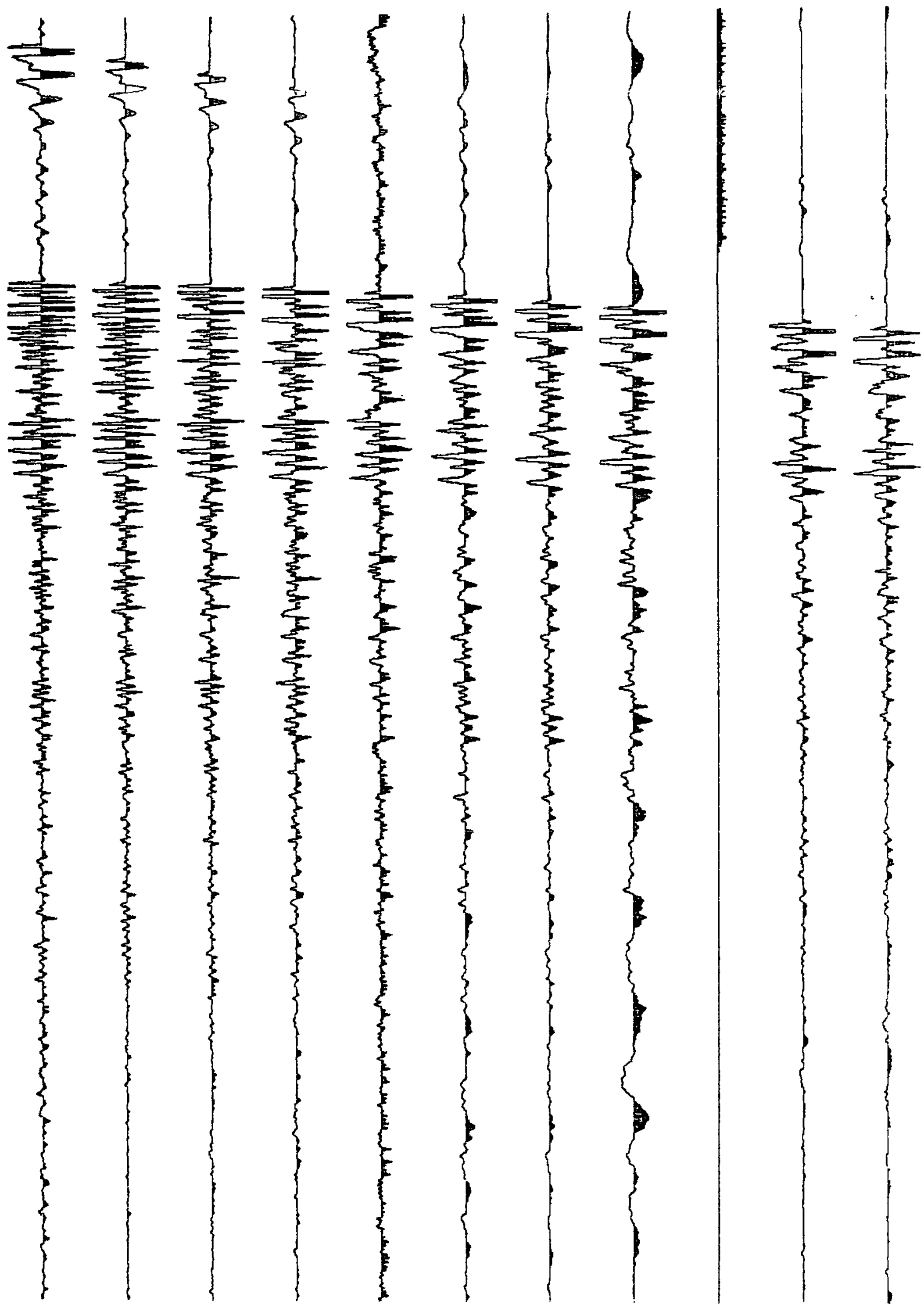


Fig 4.9: Example of data plotted with a large trace spacing

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

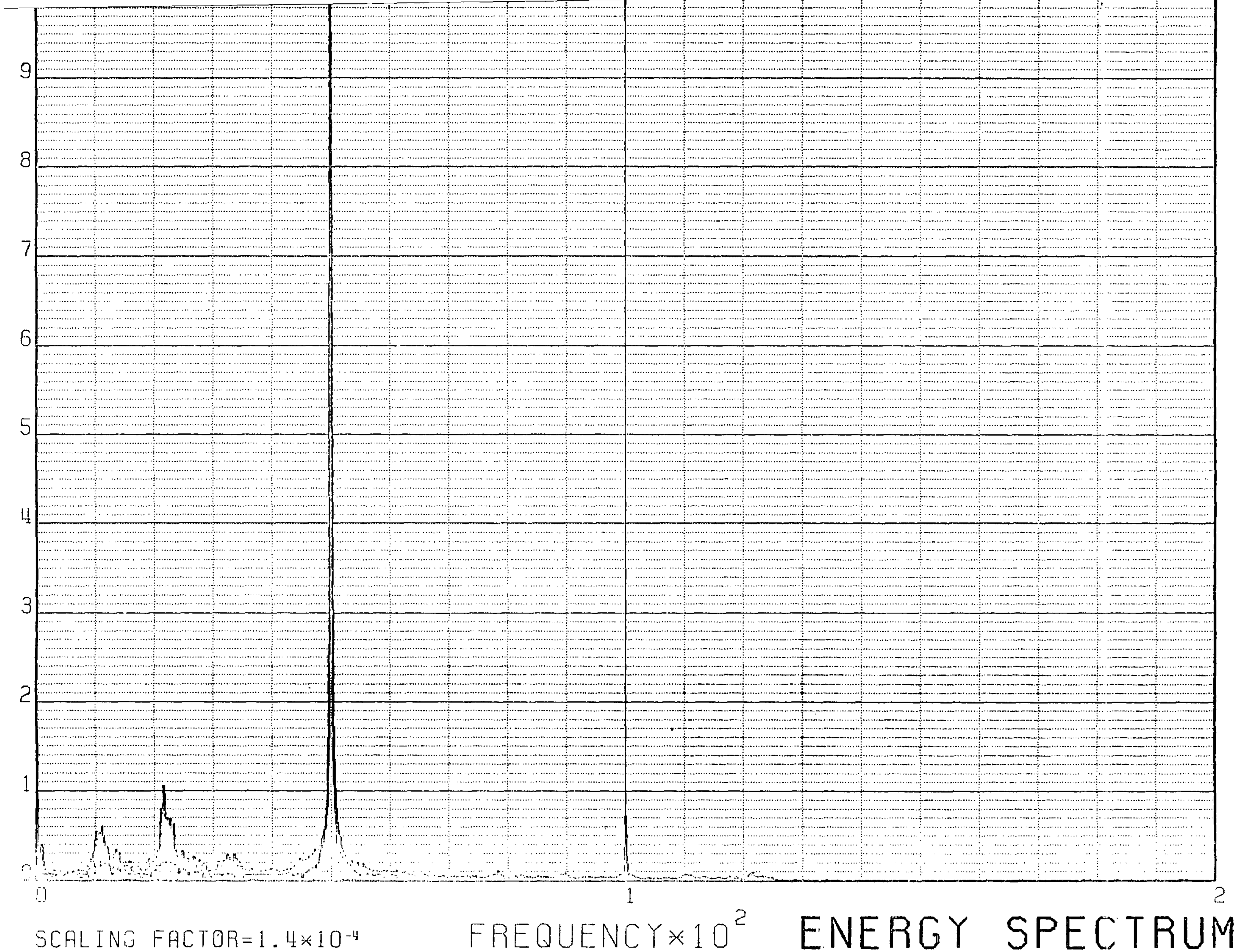


Fig 4.10: Energy Spectrum of a noisy Seismic trace produced by MPFANL

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 $\exp(-at)$ ~~e-at~~. Therefore the inverse function $\exp(at)$ ~~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.

$\exp(0.2t)$ ~~e+0.2t~~

$t\exp(0.2t)$ ~~te+0.2t~~...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.

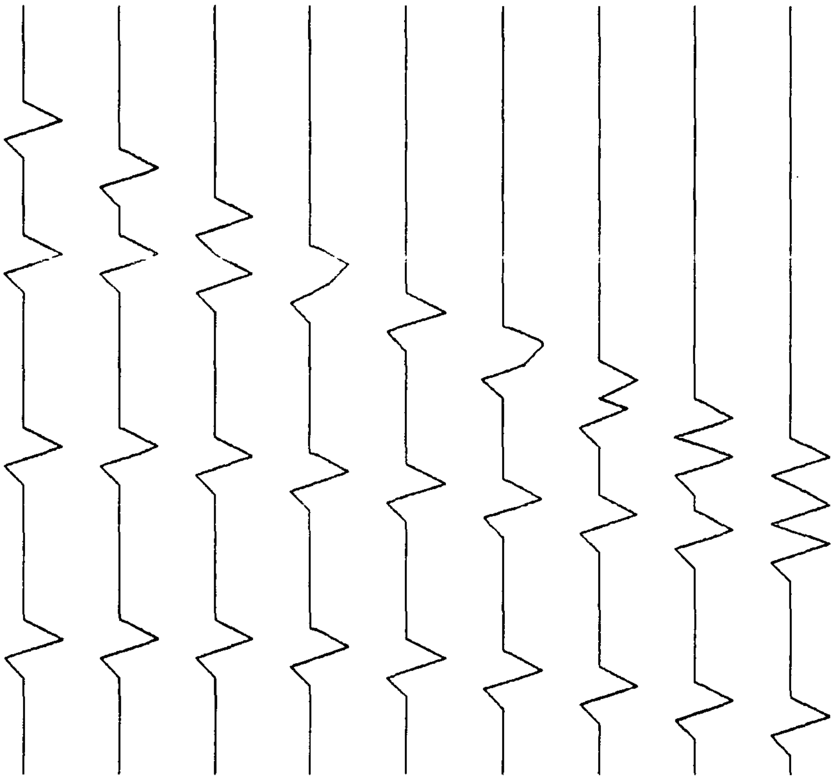
In experiments with real data it was felt that the function $t\exp(0.2t)$ ~~te+0.2t~~ gave the best results, of the ones available, in producing equalisation of amplitudes down the trace. It was also felt that these deterministic methods are preferable to AGC functions at this stage in the processing 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.

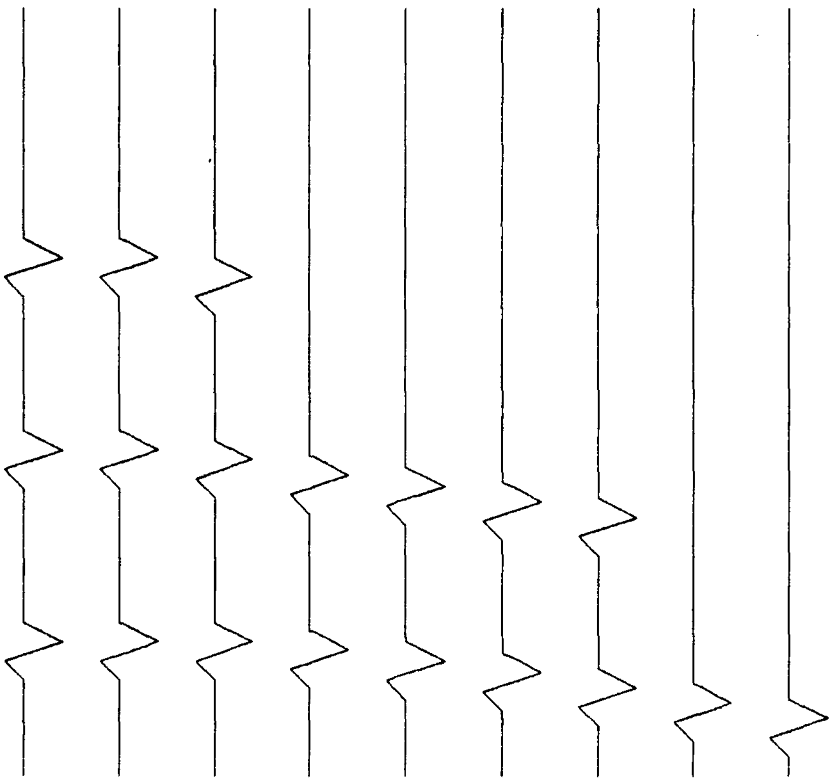
Muting

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 Schafer, 1975).

Bandreject Filter

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

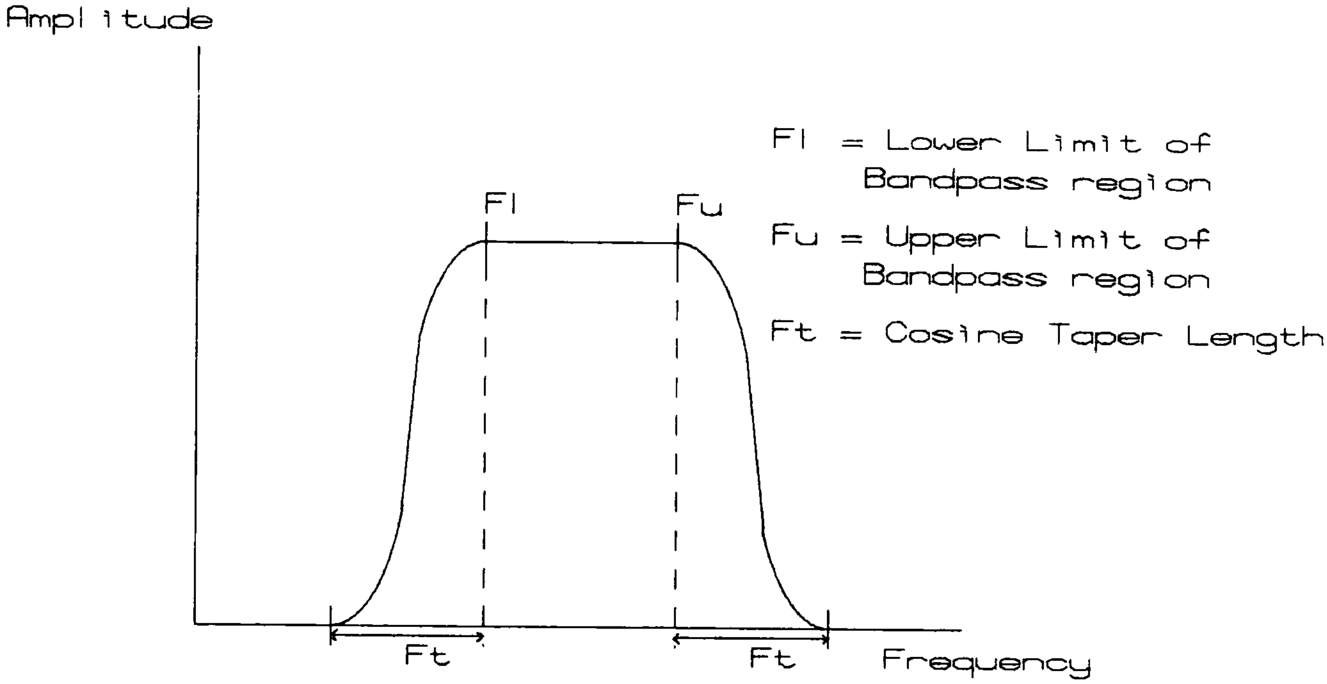


Fig 4.12 : Bandpass Filter Representation

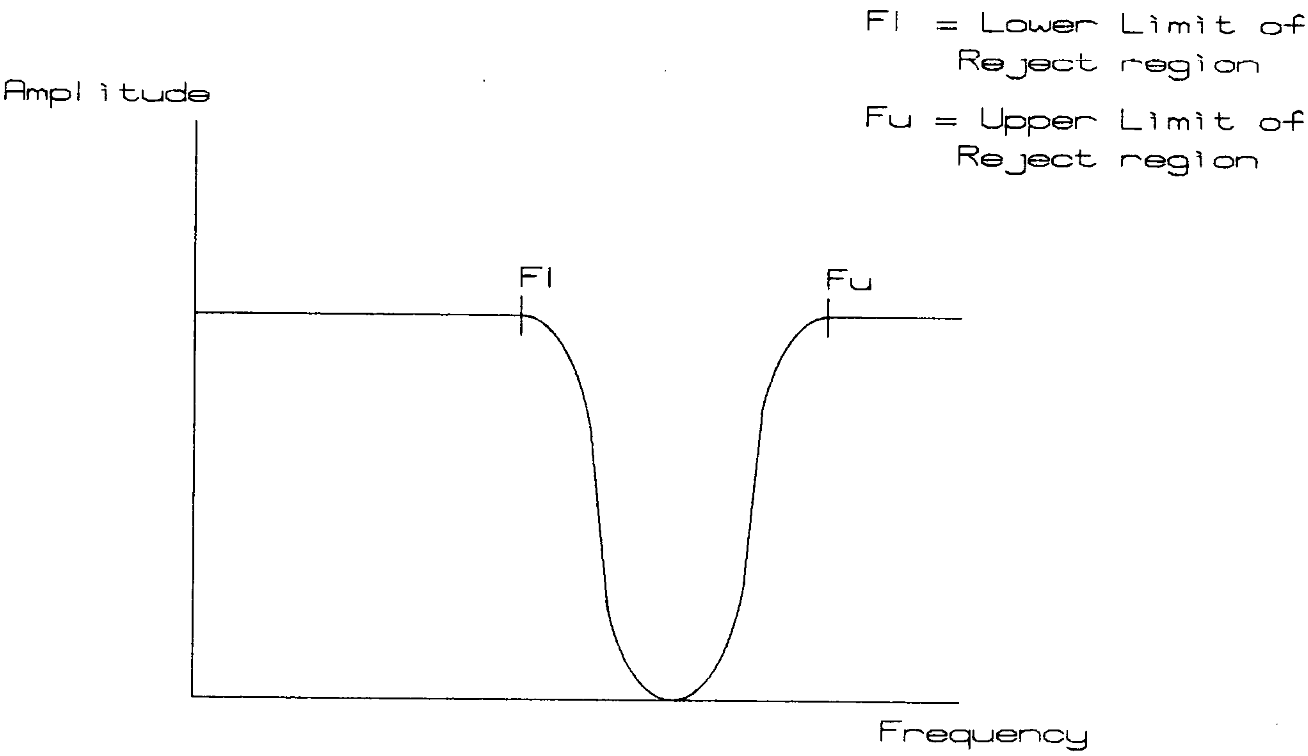


Fig 4.13 : Bandreject Filter Representation

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 Ft such that when it is convolved with an input wavelet Bt, it produces an output Ct, which is an approximation to a desired output Dt.

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

and the error energy is given by:-

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

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

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

and this reduces to

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

now

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

This is the autocorrelation function of the input wavelet

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

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

eqn 7
$$\left[\phi_{BB}\right]\left[F\right] = \left[\phi_{DB}\right]$$

This set of equations can be solved using the Levinson recursion as $\left[\phi_{BB}\right]$ is a Toeplitz 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.

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

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.

~~eqn 9~~ $$\sum_{s=0}^{M+N} \left(S_s * r_s + n_s\right)\left(S_{s+\tau} * r_{s+\tau} + n_{s+\tau}\right) = \sum_{s=0}^{M+N} S_s S_{s+\tau}$$

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.

$$I = 1 - 2c_0 + \sum_{t=0}^{M+N} C_t^2$$

eqn 10_a $$= 1 - 2B_c F_0 + \sum_{t=0}^{M+N} C_t^2$$

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

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

eqn 10_b $$[\phi_{xx}][F_k] = [\phi_{yx}]$$

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

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

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

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 to control noise amplification, and it will also 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.

eqn 1 $$\sum_{s=0}^{t} x_s P_{t-s} = \hat{x}_{t+\alpha}$$

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 Pm, the prediction filter.

eqn 2 $$\left[\phi_{xx} \right] \left[P_m \right] = \left[\phi_{x_{yx}} \right]$$

$\left[\phi_{xx} \right]$ is the autocorrelation of the input trace

$\left[\phi_{xy} \right]$ is the crosscorrelation between the input and the desired output. This represents just the alpha lag value of the autocorrelation function, for a prediction distance of alpha.

eqn 3 $$\left[\phi_{xx}^{1 \ldots N} \right] \left[P_m \right] = \left[\phi_{xx}^{\alpha \ldots \alpha+N} \right]$$

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

eqn 4 prediction filter= P0,P1,P2...Pm

eqn 5 prediction error filter= 1,0,0....-P0,-P1,-P2,...-Pm
where there are alpha-1 zeros for a lag of alpha.

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

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

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

Trace Normalisation

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

Pre-Stack processing - Summary

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

The program reads data from tape to a temporary disc file, or straight from a disc file, for tests, and reads in and operates on one trace at a time from the gather. Once the trace has ben 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:-

eqn 1 $$T_x^2 = T_0^2 + X^2/V^2$$

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.

eqn 2 Normal Moveout $= \Delta T = \left(T_0^2 - X^2/V^2\right)^{1/2} - T_0$

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 relections 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 relection 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.

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

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.

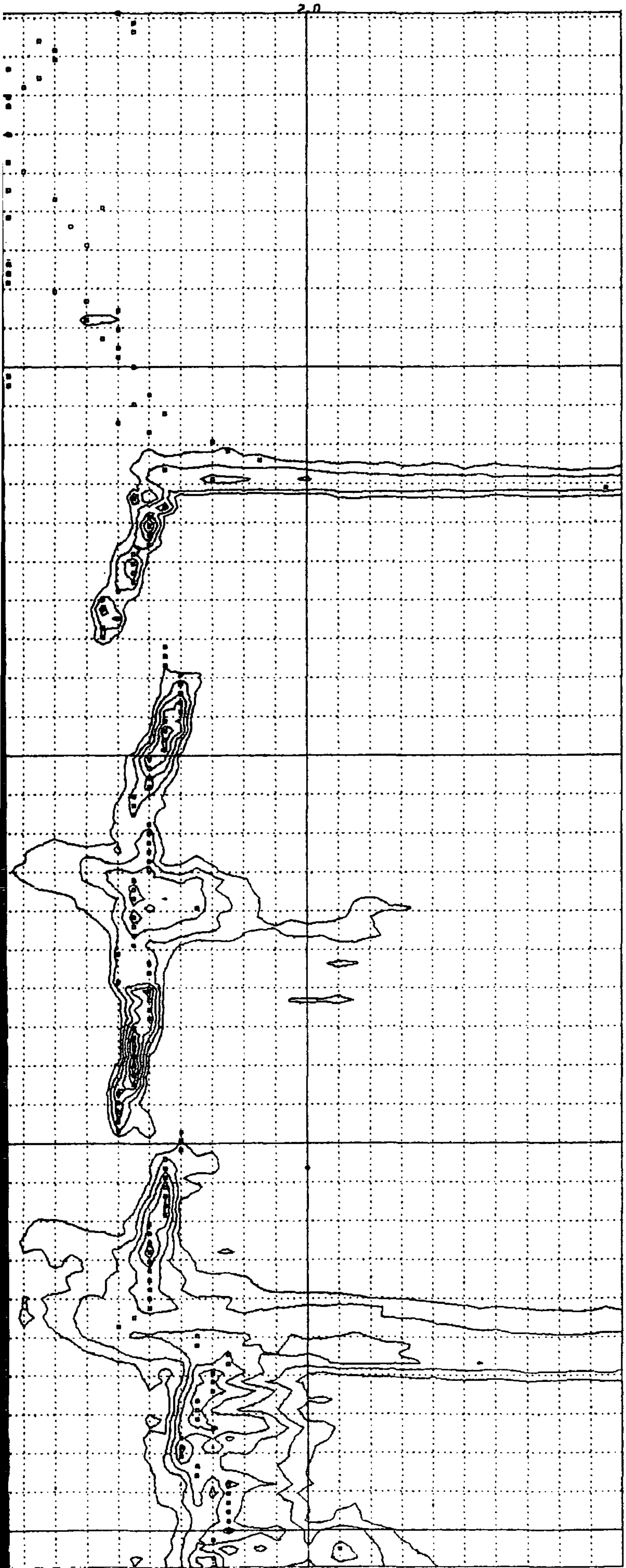
The double buffered data transfers allow the program to run at a reasonably quick rate. However the method used means that data is often read in from disc and transferred to the AP more than once during the analysis. On the other hand no constraints are placed on the number of traces in the analysis 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

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 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:-

Ti = (i-1)DT for i=1...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 Tj 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 Tj on NMO corrected trace,

sample on original trace is Ti where i is given by:-

```
i = 1 + INT(SQRT(Tj**2 + Xk**2/Vj**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 $$S_j = \sum_{j=1}^{M} C_j A_{i,j}/M$$

eqn 2 for constant energy stack $$C_j = \left(\sum_{j=1}^{M} A_{i,j}^2 \right)^{-1}$$

eqn 3 for diversity stack $$C_j = \left(\sum_{j=1}^{M} A_{i,j}^2 \right)^{-2}$$

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 can be shown that the moveout corrected signal suffers a power loss which varies with frequency, given by:-

$$1 - (\sin(\pi F dt)/\pi F dt)^{2}$$

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 origianal 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

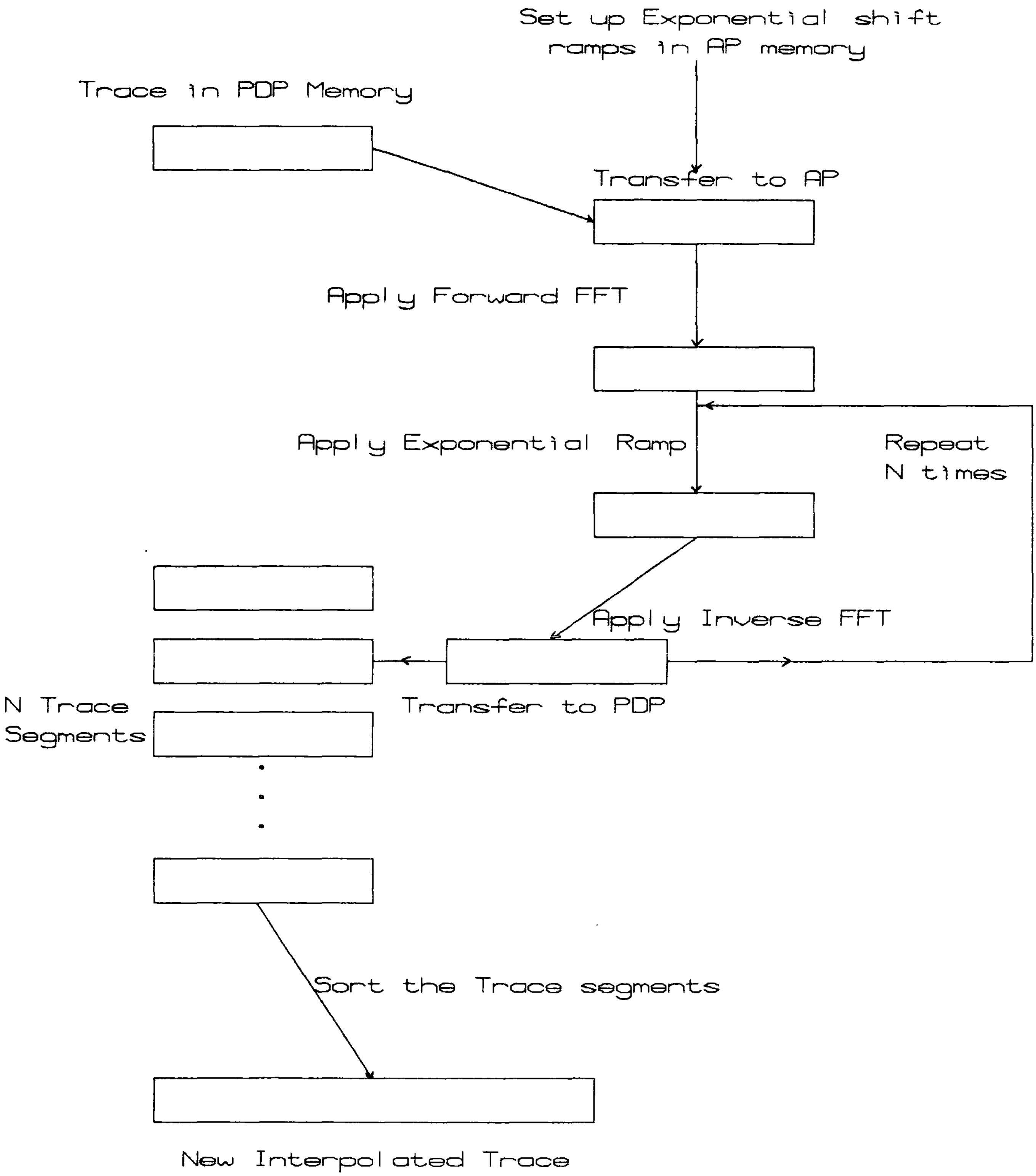


Fig 4.15 : Flow of the Interpolation Algorithm

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 ouput 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 straightfoward, 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:Ø22Ø2
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:     Ø
AND ENDING WITH SAMPLE NO: 1536
TYPE OF SOURCE= AIRGUNS
AND THE UNITS OF LENGTH USED= METRES
RECIEVER DEPTH=      1Ø.7Ø
SOURCE DEPTH=       7.3Ø
SOURCE-RECIEVER OFFSET=     297.ØØ
RECIEVER SPACING=    1ØØ.ØØ
SHOT SPACING=     5Ø.ØØ
SHOT REPITITION RATE=      2Ø.ØØ SECS
USER INFO PUT INTO HEADER BLOCK
IS GIVEN BELOW
DURHAM UNIVERSITY 198Ø 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(Ø.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:-

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

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,z,Wt) to be the Fourier counterpart of U(x,z,t), then the scalar wave equation can be written as:-

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

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

eqn 2
$$u'(K_x, z, W_t) = U'(K_x, 0, W_t)\, e^{iz\,\mathrm{sgn}(\omega)\left(\frac{\omega^2}{V^2} - K^2\right)^{1/2}} \quad \text{For } |\omega| > V|K|$$
$$U'(K_x, 0, \omega_t)\, e^{z\left(\omega^2/V^2 - K^2\right)^{1/2}} \quad \text{For } |\omega| < V|K|$$

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):-

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

eqn 3

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

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

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

It is interesting to transform the operator Mz into the Fourier domain and examine its counterpart M´z, which is given below.

eqn 4

$$M'_z = \begin{cases} e^{i z \, \mathrm{sgn}(\omega)\left(\omega^2/v^2 - K^2\right)^{1/2}} & |\omega| > V|K| \\ e^{-z\left(K^2 - \omega^2/v^2\right)^{1/2}} & |\omega| < V|K| \end{cases}$$

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.

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

eqn 5

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

$$t_0 = r/v$$

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 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.

$$U_m\left(x,\tfrac{\tau}{2}\right) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} U\left((x-x'),0,t'\right) M_\tau \, \partial x' \partial t'$$

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
$$K_0 = \left(\frac{t_0}{\Delta t}\right)$$

$$\therefore \quad \Psi_{nK} = 0 \qquad \text{for } K \leq K_0$$

$$= \left[\left(\frac{K\Delta t}{t_0}\right)^2 - 1\right]^{1/2} \qquad K > K_0$$

this gives the alternative

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

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

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:-

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 eqns7 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.

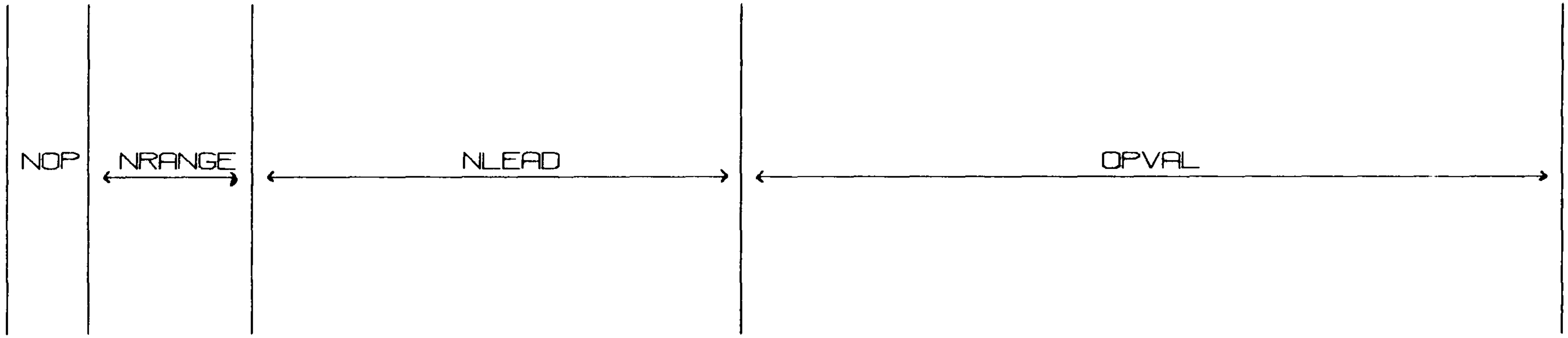
In designing the implementation of this procedure for the processing system, the results of work performed by C Godbold(Godbold, 1980), on the effect of approximating the operators used in the migration, was used to allow realistic limits to be placed on some of the migration parameters, so that the 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:-

migration aperture = X = Ztanϕ where z=depth, ϕ= 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 progam 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.

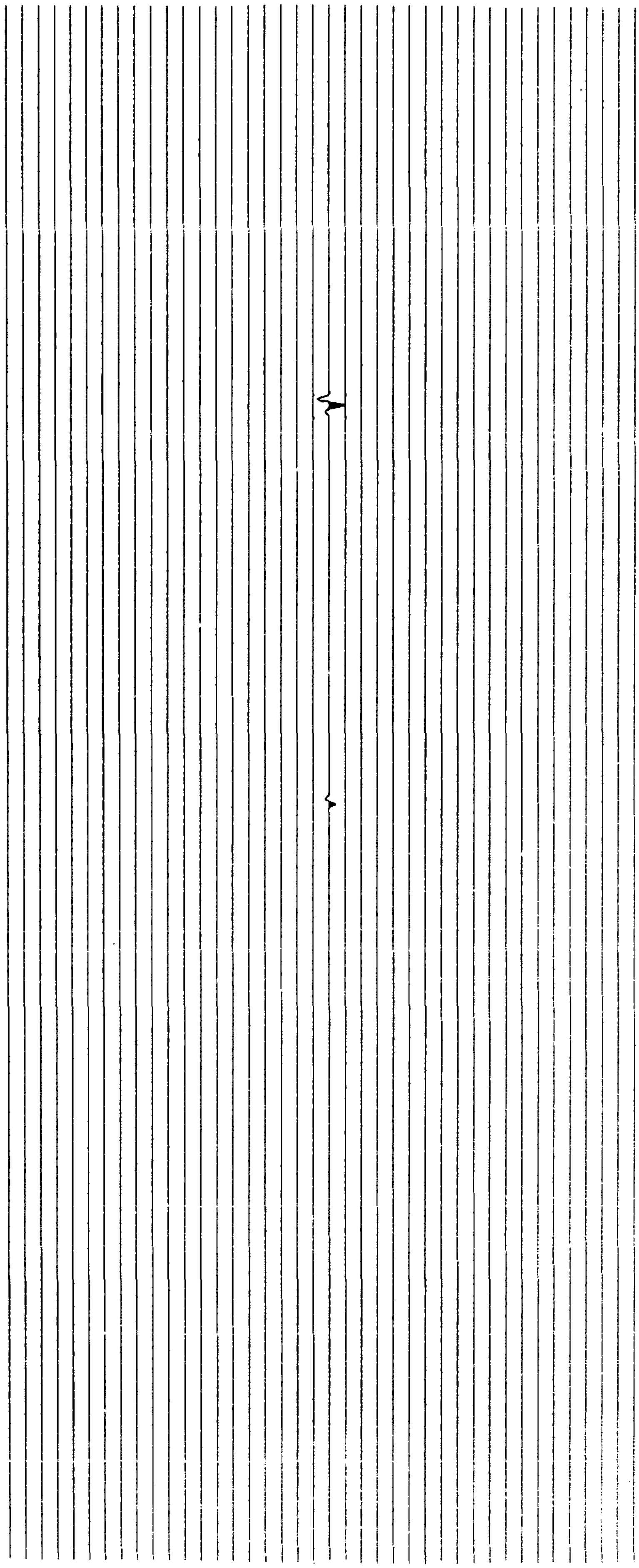


Fig 4.18: The Diffractor Model of Fig 4.3 after Kirchhoff Migration using MPKMIG

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.

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

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:-

$$t' = t - \int \frac{\partial z}{V}$$

eqn 2 $$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2} = -\frac{2}{V}\frac{\partial^2 u}{\partial z \partial t'}$$

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

eqn 3
$$\frac{V}{2}\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z \partial t'} = 0$$

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

eqn 4
$$\frac{\partial^3 u}{\partial x^2 \partial z} + \frac{\partial^3 u}{\partial z^3} = \frac{2}{V}\frac{\partial^3 u}{\partial x^2 \partial t'} + \frac{4}{V^2}\frac{\partial^3 u}{\partial z \partial t'^2}$$

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

eqn 5
$$\frac{V^2}{4}\frac{\partial^3 u}{\partial x^2 \partial z} - \frac{V}{2}\frac{\partial^3 u}{\partial x^2 \partial t'} - \frac{\partial^3 u}{\partial z \partial t'^2} = 0$$

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, is 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.

$\tau = z/v$

15 degree equation $$\frac{V^2}{2}\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial t' \partial \tau} = 0$$

45 degree equation $$\frac{V^2}{4}\frac{\partial^3 u}{\partial x^2 \partial \tau} - \frac{V^2}{2}\frac{\partial^3 u}{\partial x^2 \partial t'} - \frac{\partial^3 u}{\partial t'^2 \partial \tau} = 0$$

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:-

$$V' = V/2$$
$$t = n\Delta t$$
$$\tau = k\Delta\tau$$
$$x = j\Delta x$$

$$\overset{*}{A} = \frac{0.518 V'^2 \Delta t \Delta\tau}{16\,\Delta x^2} \approx \frac{V'^2 \Delta t \Delta \tau}{32 \Delta x^2}$$

$$\frac{\partial^2}{\partial x^2} = \frac{T}{\Delta x^2 (I + bT)} \qquad T = (E_x^{-1}, -2, E_x) \qquad b = 1/6 \;\dagger$$

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

E_z = Advance Operator $\therefore E_z f_z = f_{z+\Delta z}$

$$\left[(b-A)T + I\right] U^n_{k+1,j} = \left[(b+A)T + I\right] U^{n+1}_{k+1,j} + \left[(b+A)T + I\right] U^n_{k,j} - \left[(b-A)T + I\right] U^{n+1}_{k,j}$$

* c.f. Berkhout (1979)

† c.f. Loewenthal et al. (1976)

It can be seen that this equation is a tridiagonal matrix equation in which the solution for $U^{n}_{K+1,J}$ 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:

$$A = \frac{0.279\,V'^2 \Delta t^2}{4\,\Delta x^2} \approx \frac{V^2 \Delta t^2}{4 \Delta x^2}$$

$$B = \frac{V'^2 \Delta t \Delta \tau}{32 \Delta x^2} = \frac{V^2 \Delta t \Delta \tau}{2 \Delta x^2}$$

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

$$\frac{\partial}{\partial \tau} = \frac{2}{\Delta \tau}\left(\frac{(E_\tau - I)}{(E_\tau + I)}\right) \qquad \frac{\partial}{\partial t} = \frac{1}{2\Delta t}\left(E_t - E_t^{-1}\right)$$

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

45 degree finite difference expression

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

$$\therefore\ [I + (b - B)T_x]U^{n-1}_{K+1,J} = [I + (b - B)T_x]U^{n+1}_{K,J} - [I + (b + B)T_x]U^{n+1}_{K+1,J} - [2 + (2b + A)T_x]U^{n}_{K,J} + [2 + (2b + A)T_x]U^{n}_{K+1,J} + [I + (B + b)T_x]U^{n-1}_{K,J}$$

Once again given the boundary conditions the factors on the right hand side of the equation can be evaluated and so we are left with an tridiagonal matrix equation with the vector in x $u^{n-1}_{k+1,J}$ 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.

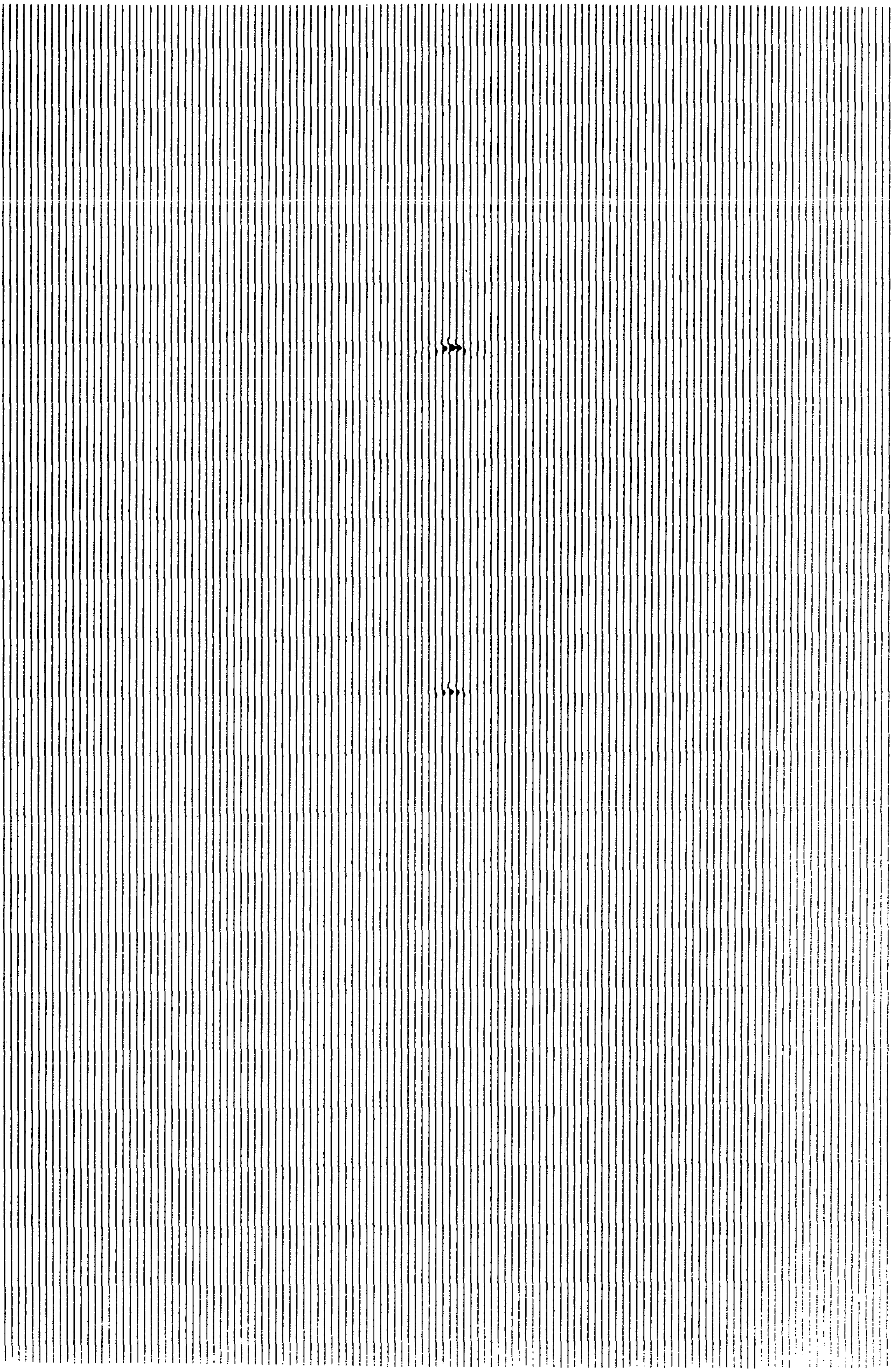


Fig 4.19: The Diffractor Model of Fig 4.3 after 15° Finite Difference Migration with MPFD15

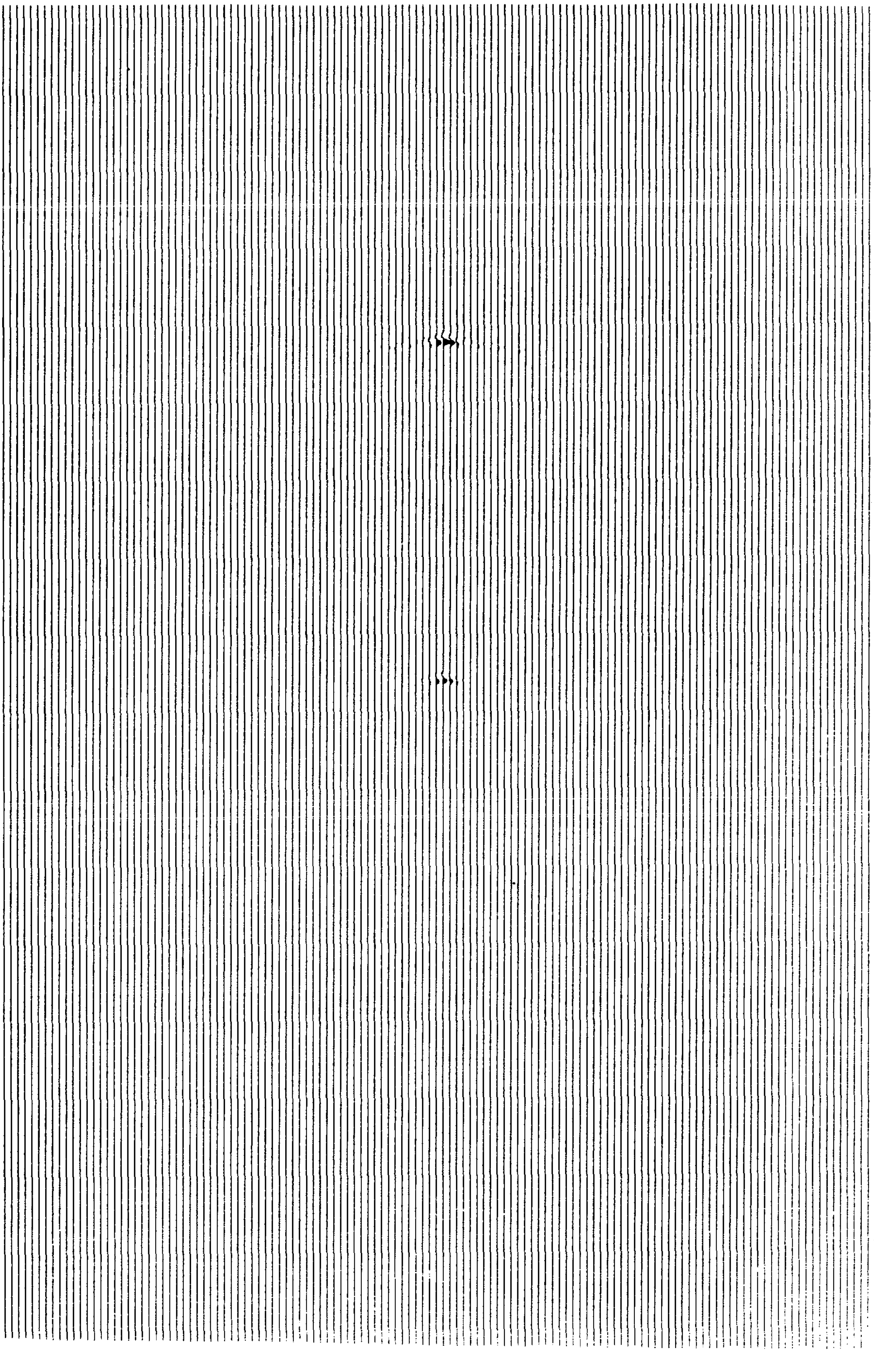


Fig 4.20: The Diffractor Model of Fig 4.3 after 45° Finite Difference Migration with MPFD45

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 then 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.

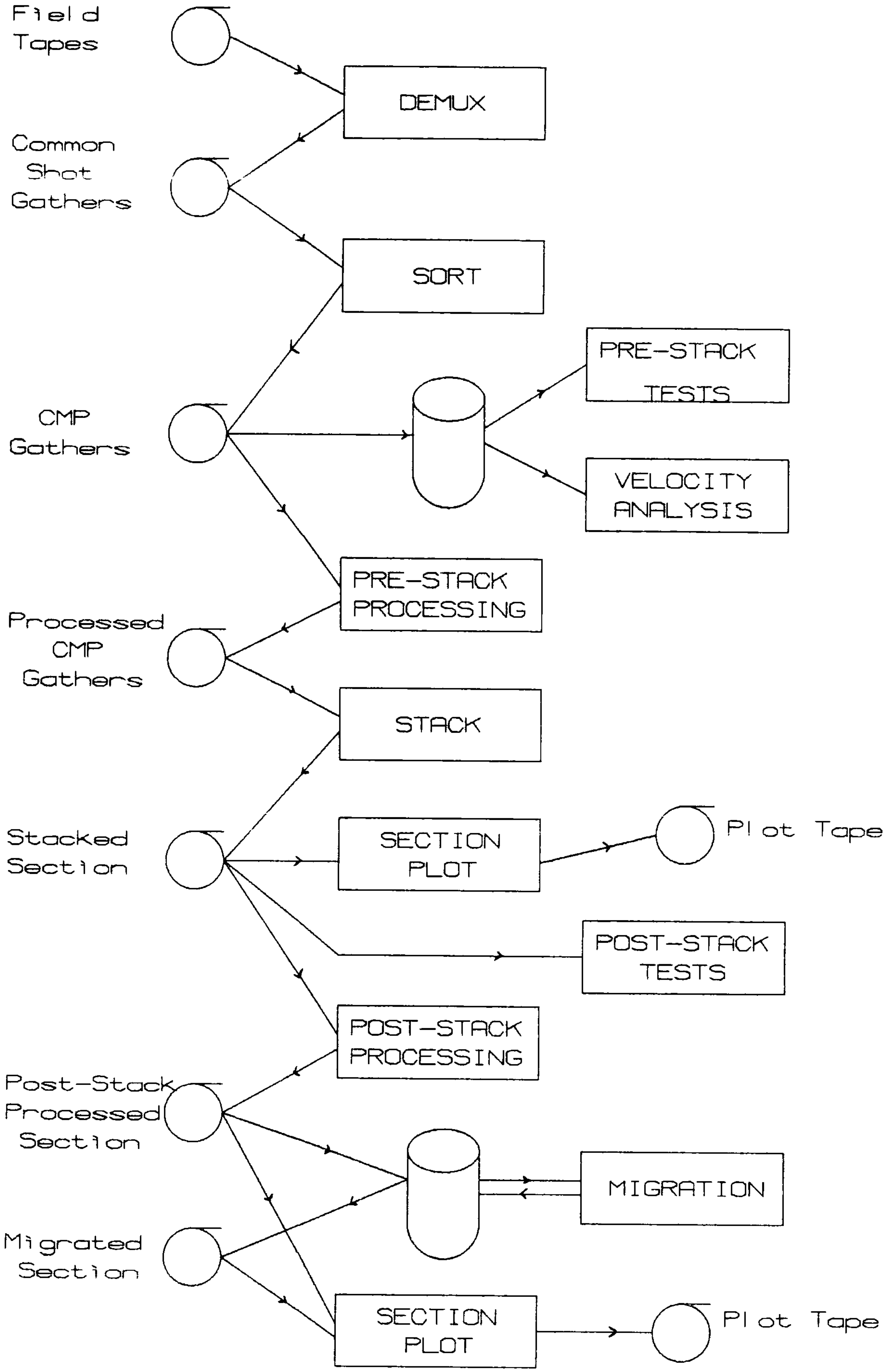


Fig 5.1 : Processing System data flow

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 antialiassing filter applied before digitising. These data were recorded in SEG-A on 9 track magnetic tape using the SDS10/10.

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

Data were chosen for tests and velocity analyses and brought down onto disc from the CMP gather tapes. These were spaced at about every 40 CMP positions over the first part of the line and then about every 100 over the area of lesser interest at the end of the line. When these gathers were displayed it was found that channel 9 was dead on some of the records, presumably due to some acquisition fault, and channel 10 had been recorded with the wrong

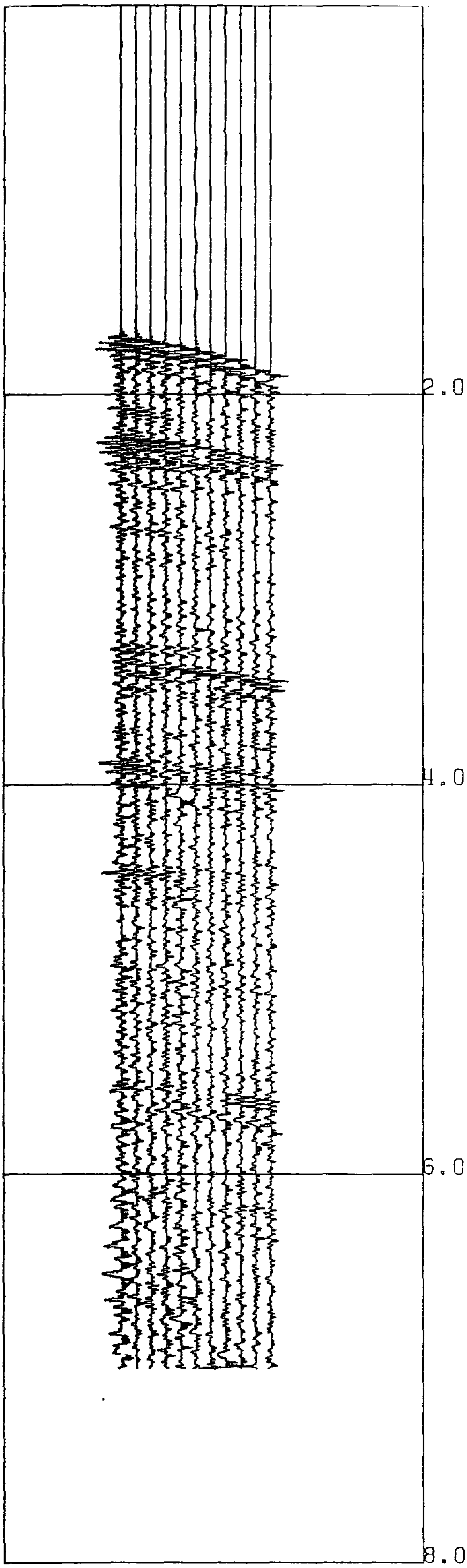


Fig 5.2a: An Example of a Jan Mayen Shot Record

polarity with respect to the rest of the data. On the whole, the data quality was quite reasonable with only minimal noise corruption.

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

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

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

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

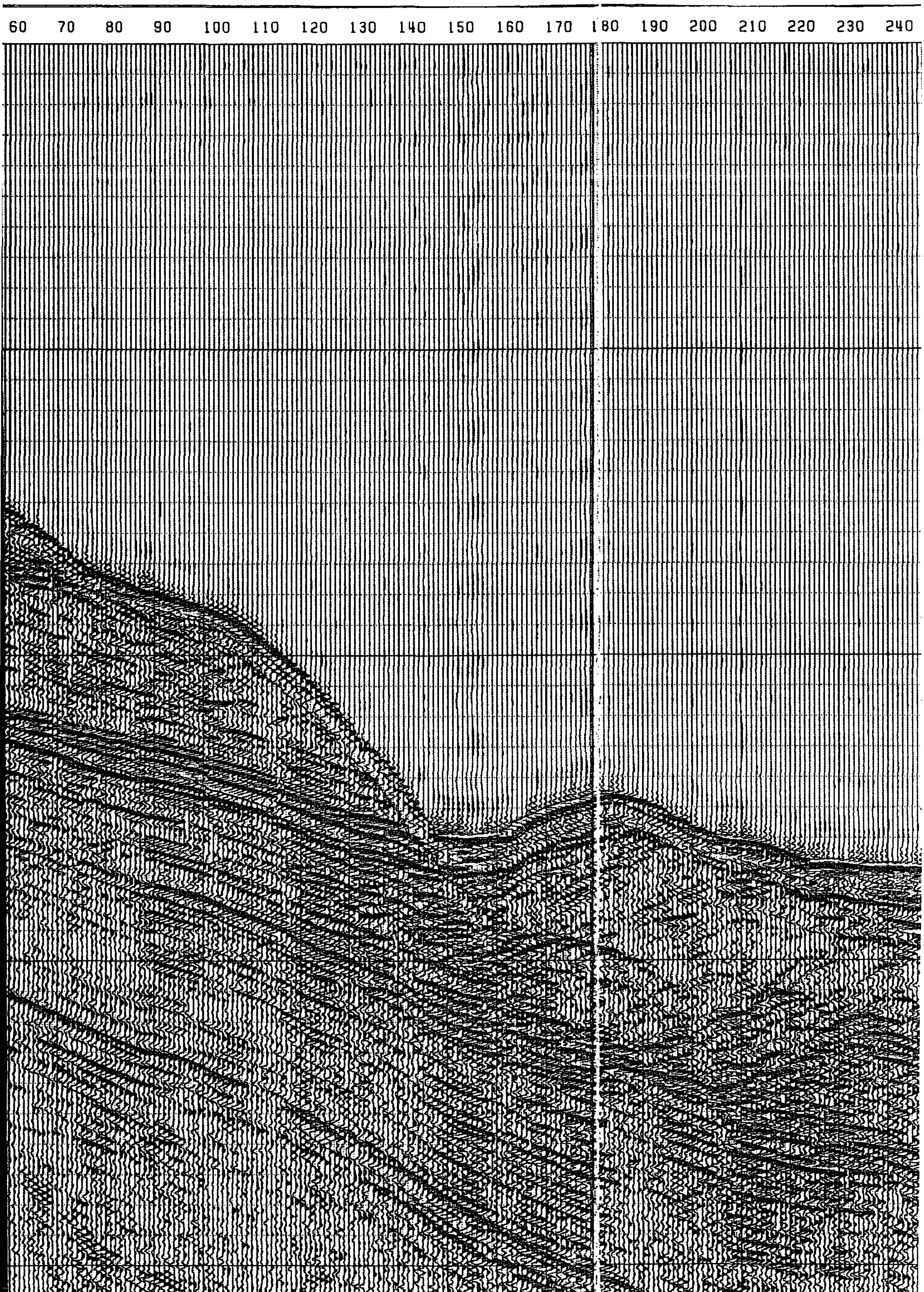


Fig 5.2.b: A Portion of the Final Section for the Jan Mayen line

Jan-Mayen Processing Details

1...Demultiplex and Sort into 11 channel CMP gathers

2...Amplitude correction- te**0.2t

3...Polarity reversal channel 10

4...150ms Spike Deconvolution, 5% prewhitening

5...Bandpass filter 5-10/40-45 Hz

6...Velocity analysis, every 40th CMP

7...NMO correction with 8 fold interpolation

8...Stack, 11 fold CMP

9...Prediction error deconvolution, 100ms gap,
100ms filter 5% prewhitening

10..Bandpass filtering 5-10/40-45

Caribbean Arc 1980

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

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

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

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

Spectral analysis plots showed quite clearly that the high frequency noise was at 50Hz, presumably due to pick up from the ship's electrical supply. The low frequency noise was centered on about 11Hz, and may have been due to the ships propellors 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

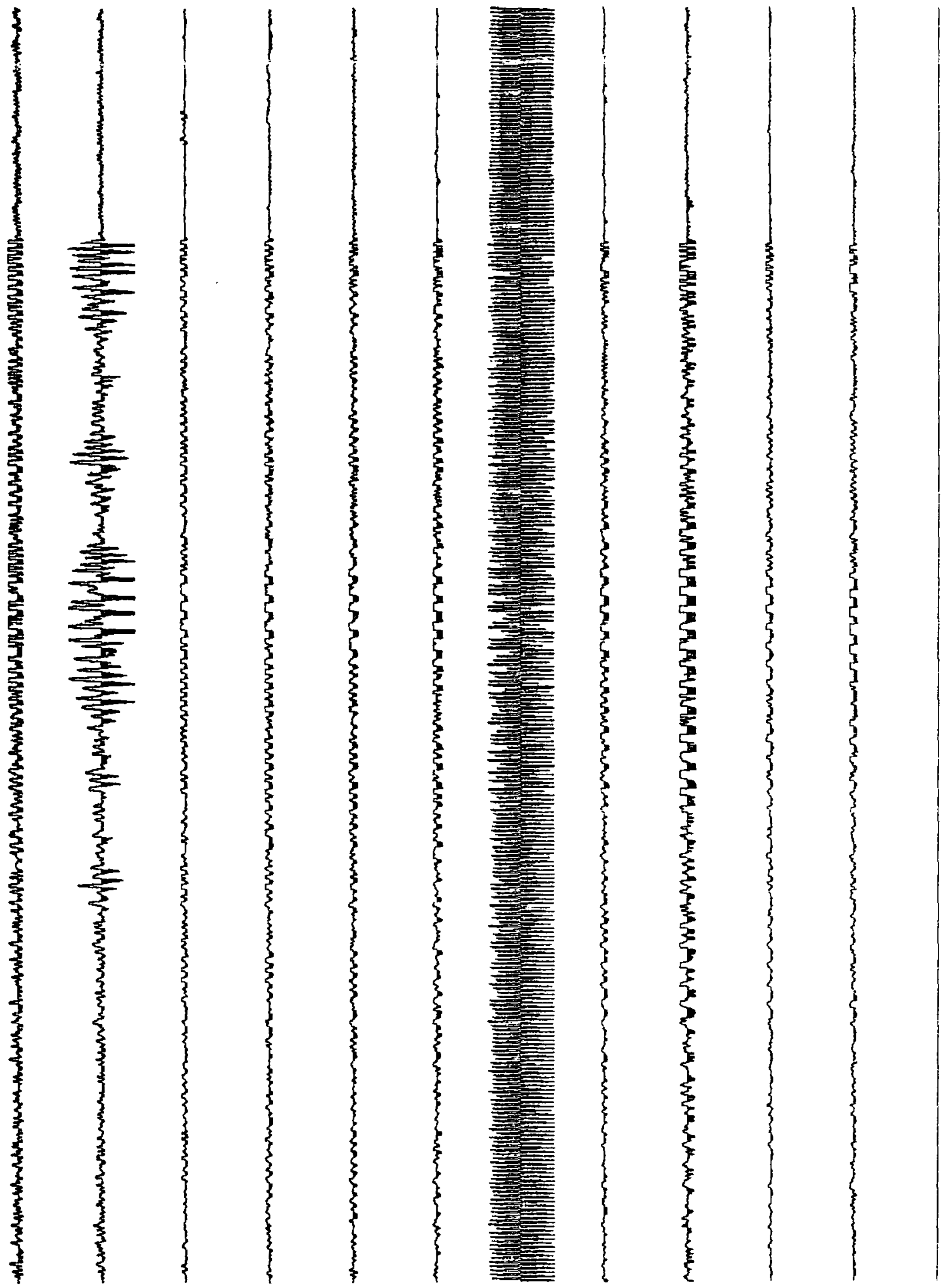


Fig 5.3: Display of a Caribbean Arc Shot Record

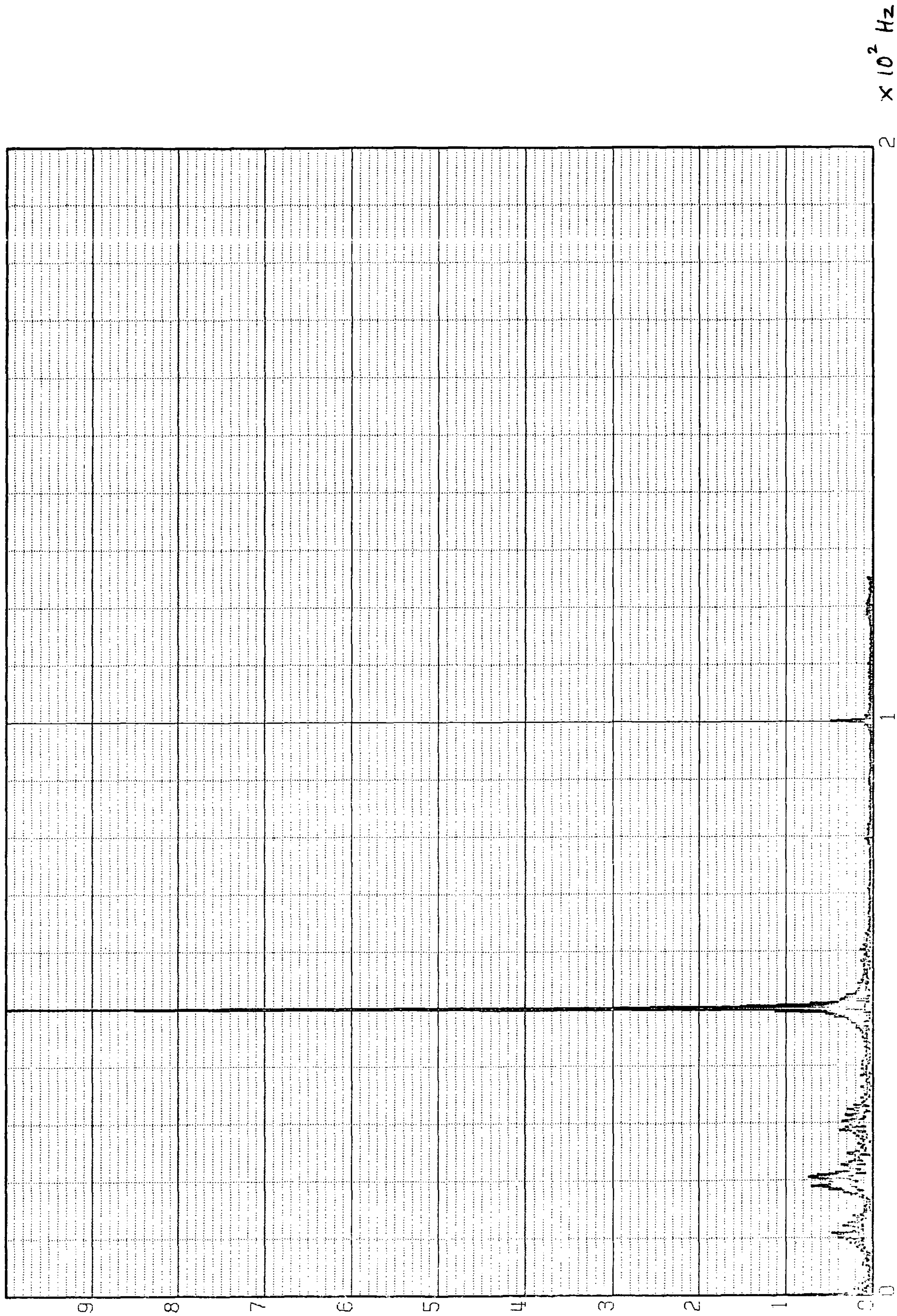


Fig 5.4: Spectral Analysis of Channel 7 from Displayed Shot Record.

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.

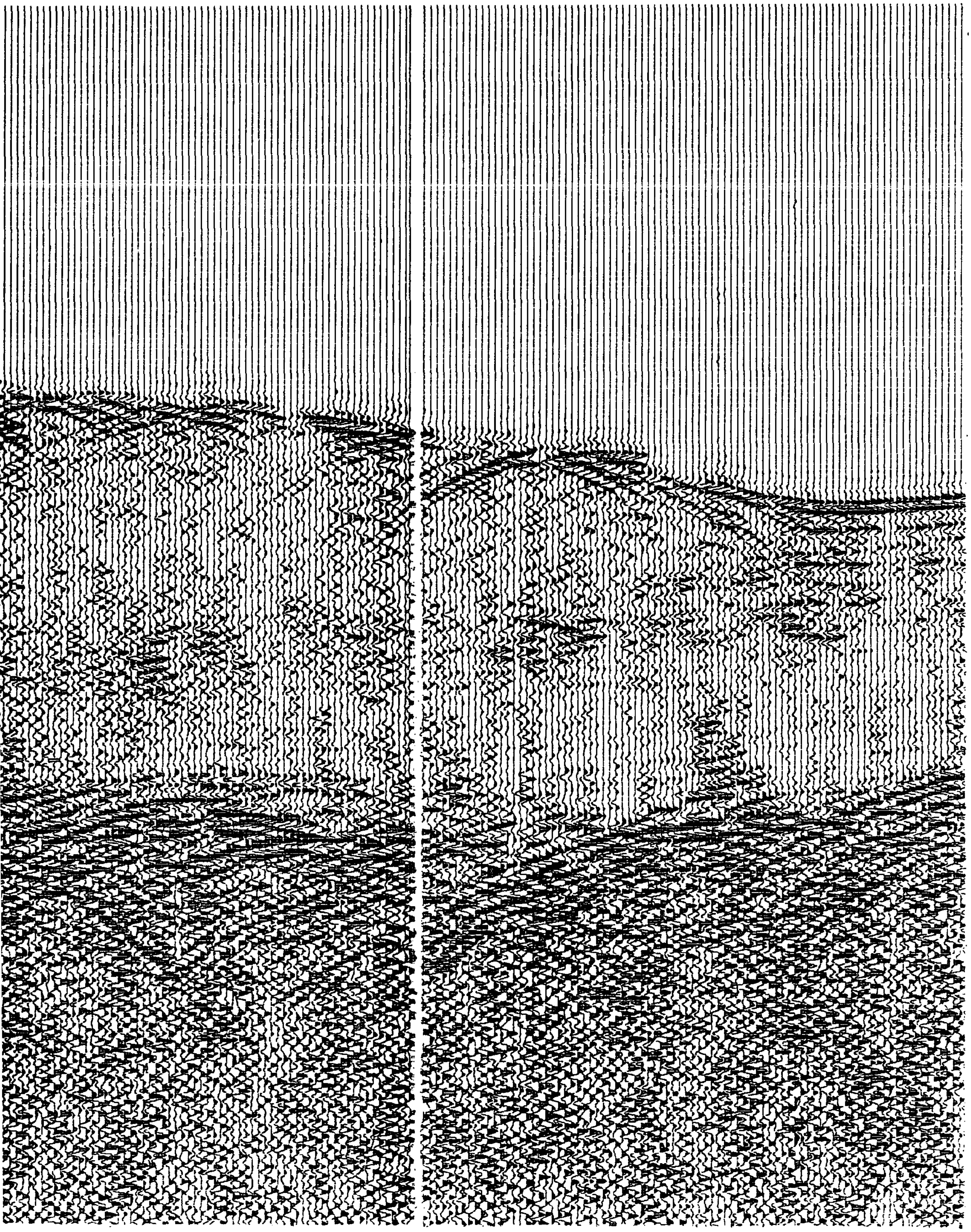


Fig 5.5: A Single Channel Display of a Portion of the Caribbean line after Pre-Stack Processing

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

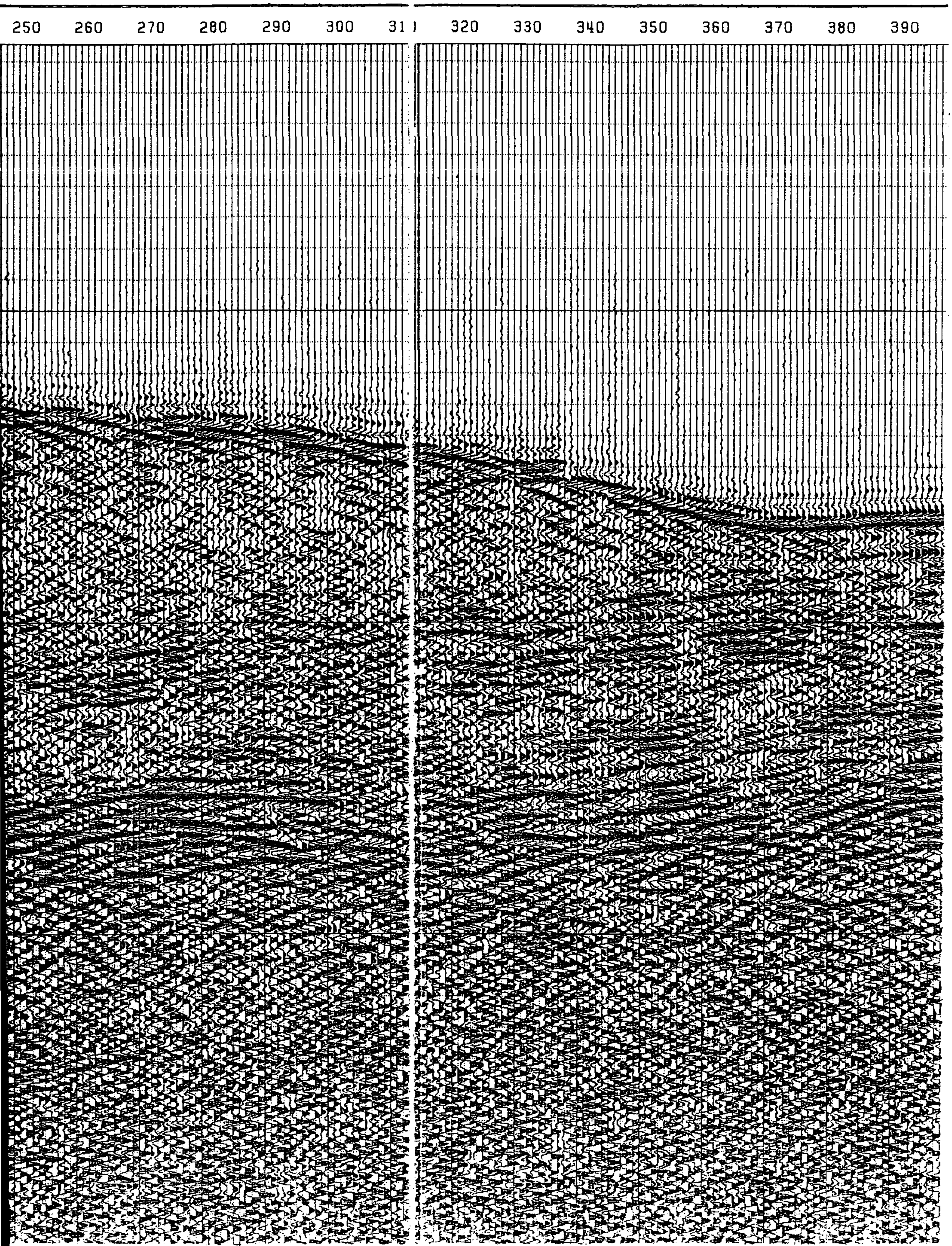


Fig 5·6: A Portion of the Caribbean line After Stack

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 acretionary 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.

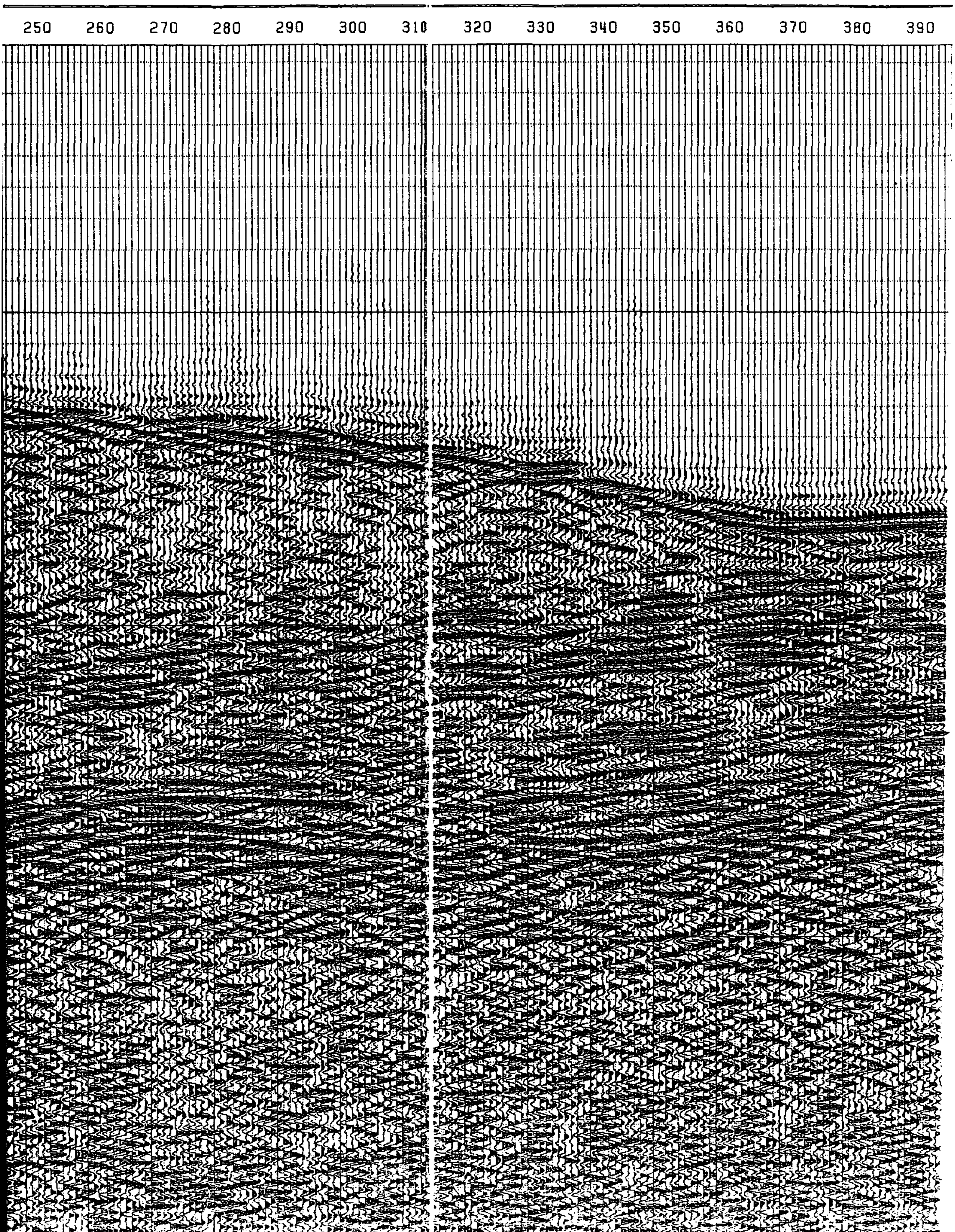


Fig 5.7: A Portion of the Caribbean line after all Processing

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 straightfoward. 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 straightfoward once the demultiplex had been performed. Most of this logic would also be shared with the SEG-A program, as the two share the same 3 byte start of scan code.

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

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

Data Exchange Format

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

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

Land Data

Although the system was designed with marine processing as the primary target, a conscious effort was made not to exclude the possibility of processing land seismic data. It would be quite feasible to process land data on the system, especially if the one glaring omission from the normal suite of land processing techniques were added to the system. This is, of course, the capability to handle static corrections.

In processing marine data, the importance of static effects is negligible, However, in order to successfully process land data the ability to apply static corrections to the data is of paramount importance, if a section of interpretable quality is to be produced.

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

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

Other Possible Software Additions

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

Vibroseis

Vibroseis is of increasing importance in the acquisition of land data. After the data has been demultiplexed the Vibroseis sweep has to be removed by performing a correlation between the recorded sweep and the data trace, which can be up to 30 seconds in length if a long source sweep has been used. At present a Vibroseis correlation capabilty 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 flexibilty 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 charcteristics 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)

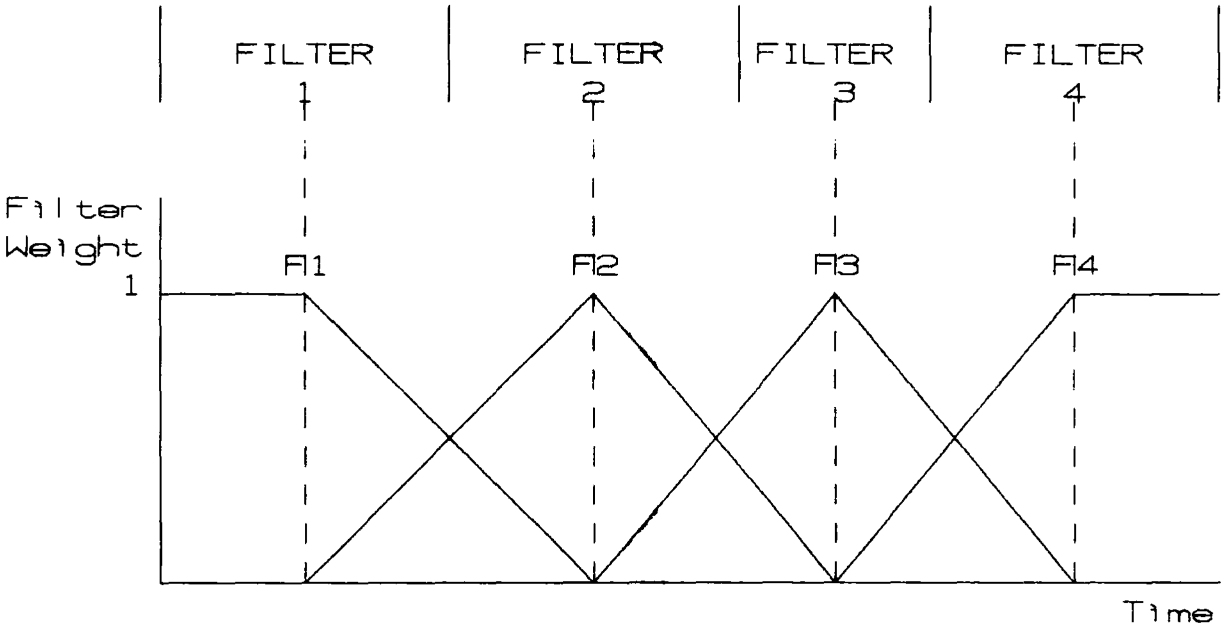


Fig 6.1 : Application of Time Variant Filters

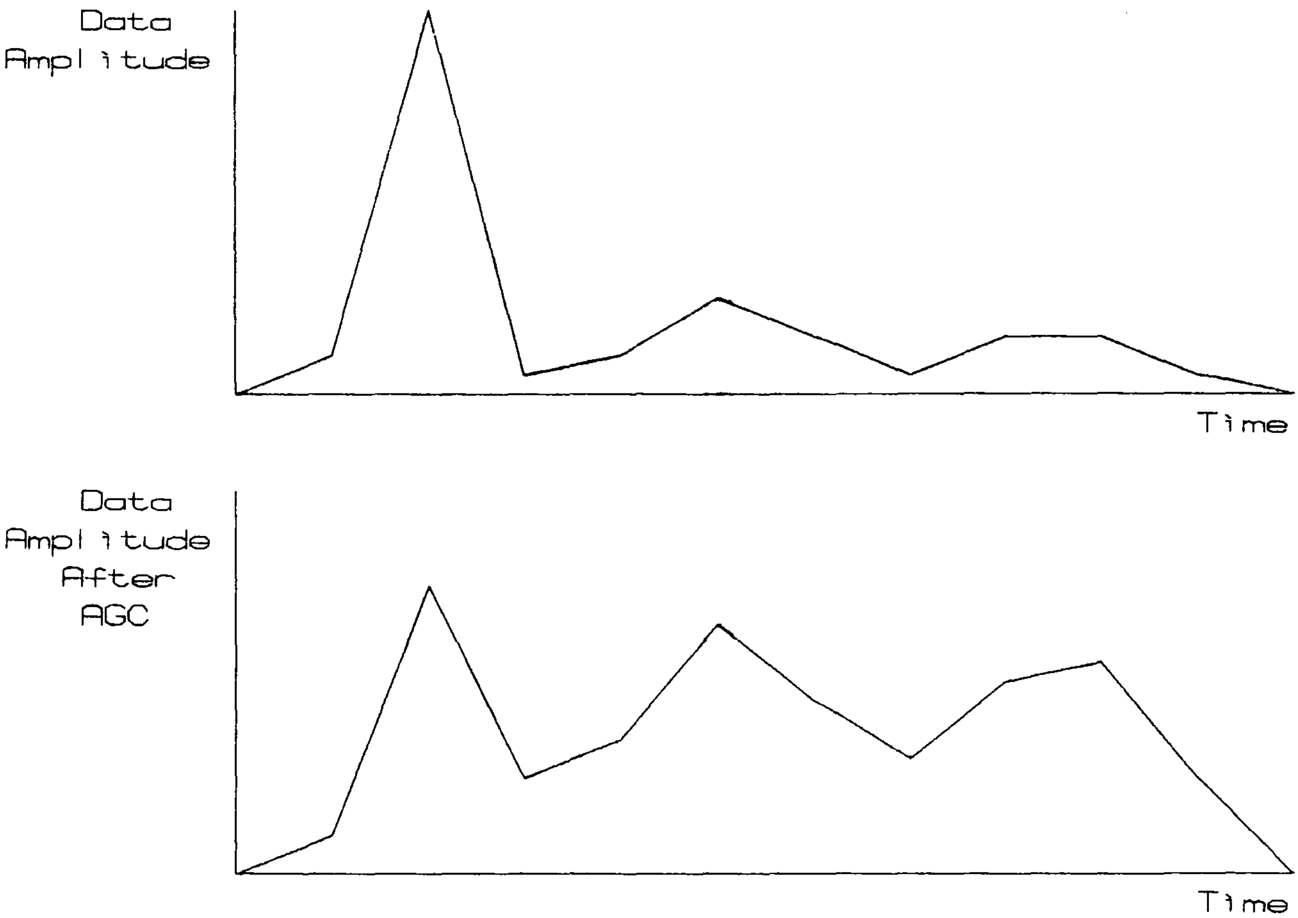


Fig 6.2 : Example of AGC for Trace scaling

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 reverbatory 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 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

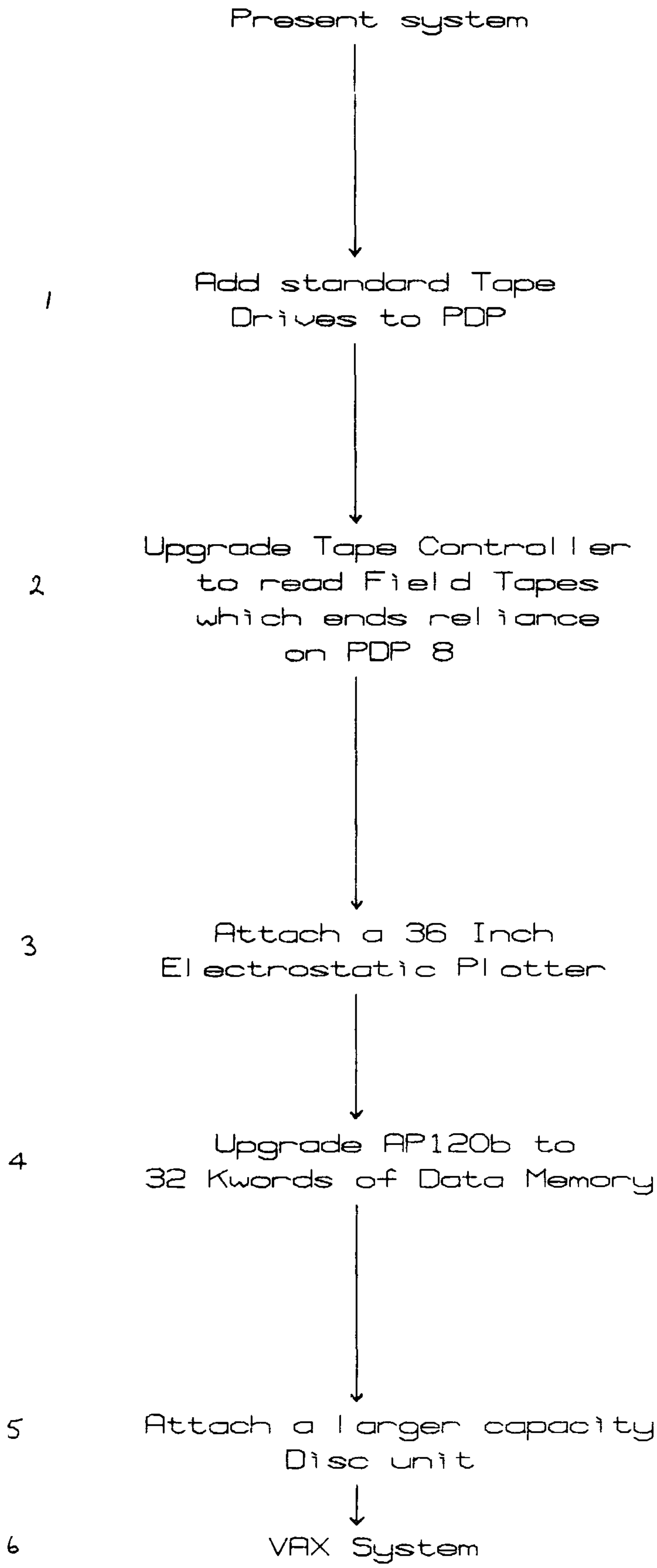


Fig 6.3 : Envisaged System upgrade path

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 ovelapped, 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 facilty 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.

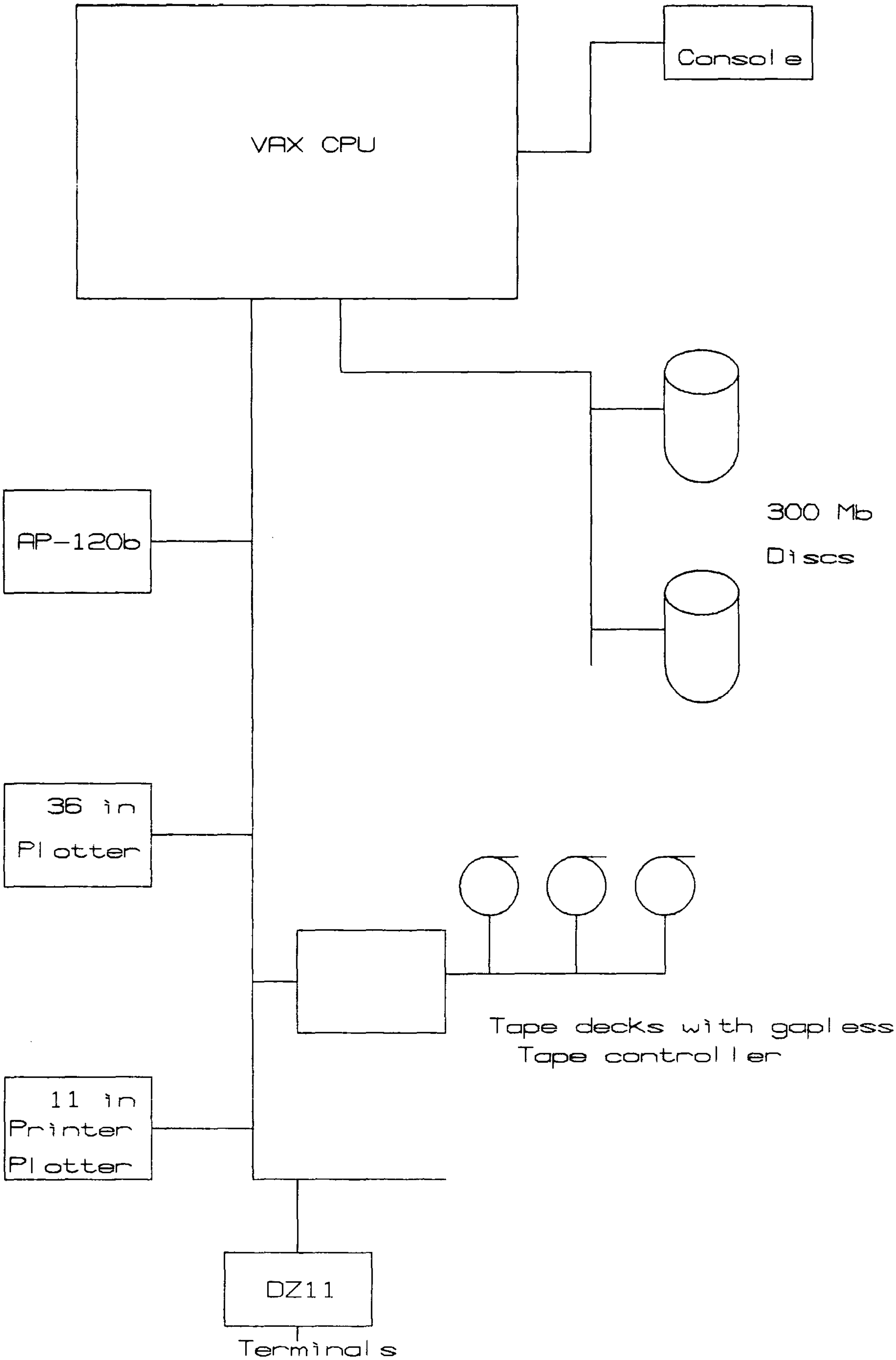


Fig 6.4 : Suggested Future Computer System

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 procesing 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

<u>READ(1,1001)(FPOS(I),I=1,NCHAN)</u>

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

<u>READ(1,1001)(VELAN(I),I=1,VELNUM)</u>

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

<u>READ(1,1000)(FNBUF(I),I=1,NROW)</u>

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

<u>READ(1,1000)(VELNAM(I),I=1,VELNUM)</u>

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

<u>READ(1,1001)NOCHAN,IGCODE,IUNITS,ISCODE</u>

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
      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
      C
      C         DATA STORAGE DECLARATIONS.
      C
      C         -------------------------
ØØØ1          VIRTUAL BUFF(8448),FNAMES(3Ø),VELNAM(6Ø),RTNAM(6Ø)
ØØØ2          REAL*8 FNAMR,FNAMES,VELNAM,DBLK(2),FMBUF,RTNAM
ØØØ3          REAL*4 DEVNAM,BUFOUT(256),FNBUF(3)
ØØØ4          INTEGER*2 FNUM,CHOFF(3Ø),FPOS(3Ø),NBLKOF(3Ø),IBLKOF(3Ø),INDEX(31),
             %USEFLG,QNUM,VNUM,BLK,BSST,EGAINS(3Ø),GCNT,
             %SPOS,FST,BSST1,BLST,OLAP,RST,ICHAN(3Ø),
             %GSAVE(3Ø),EOFFLG,MASK,SYNC,BUFF,HSBLKW(9),VELAN(6Ø),GAINS(384Ø),
             %OUTFLG,INFLG,FLEN,VELNUM
ØØØ5          LOGICAL*1 STATUS,RC,ITLEN,IGCODE,IUNITS,NOCHAN,
             %TLEN,TPDRR,TPDRW,HSBLK(256)
ØØØ6          COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
             %IERFLG,SPOS,RC
ØØØ7          COMMON /DECOM/BUFOUT,CHOFF,ICHAN,FPOS,NCHAN,NBLKOF,INDEX,GCNT,
             #IHSTRT,IHSEC
ØØØ8          COMMON /BUFCOM/FBSST1,FBLST,F256,F256D,FBSST,F1,F4Ø97,
             %BSST1,BLST,OLAP,RST,BLK
ØØØ9          COMMON/BUFS/NUERR,NALOW,ITIC
ØØ1Ø          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(2Ø)),(ISCODE,HSBLK(45)),(IBFREE,HSBLK(47)),
             %(RSPAC,HSBLK(25))
ØØ11          DATA DEVNAM/3RDK /
ØØ12          DATA CHOFF/4352,448Ø,46Ø8,4736,4864,4992,
             %512Ø,5248,5376,55Ø4,5632,576Ø,5888,6Ø16,6144,6272,
             %64ØØ,6528,6656,6784,6912,7Ø4Ø,7168,7296,7424,7552,
             %768Ø,78Ø8,7936,8Ø64/
ØØ13          DATA MASK,SYNC/"17,"177777/
ØØ14          DATA HSBLK(49)/"377/,HSBLK(5Ø)/"377/,HSBLK(255)/"377/,
             #HSBLK(256)/"377/
      C
      C CONSTANTS USED IN PROGRAM
      C
ØØ15          IPADNO=Ø
ØØ16          IEOTR=Ø
ØØ17          IEOTW=Ø
ØØ18          BSST=4Ø96+256
ØØ19          IBIAS=15
ØØ2Ø          FNUM=1
ØØ21          IAD=BSST+1
```

```
      VNUM=1
      F1=ADGET(BUFF(1))
      F4097=ADGET(BUFF(4097))
      FBSST=ADGET(BUFF(IAD))
      F256D=ADGET(BUFF(257))
      F256=APGAD(BUFF(257))
C
C  SET UP THE RT-11 INPUT-OUTPUT PROCEDURES
C
      IF(ICDFN(50).NE.0)STOP'INSUFFUCIENT CHANNEL FREE SPACE'
C  INITIALIZE THE AP
C
      CALL APINIT
C  READ IN THE NECESSARY INPUT DATA
C
      CALL ASSIGN(1,'DK1:MPDMXD.SPF',14)
      CALL ASSIGN(2,'DK1:MPDMXD.LOG',14)
      READ(1,1001)NCHAN,NFILES,ITSIZ,IHSTRT,NROW,TPDRR,TPDRW,VELNUM
 1001 FORMAT(12I5)
      READ(1,1001)USEFLG,OUTFLG,INFLG,IERFLG,NRECOV,NUERR,NALOW
      IF(INFLG.NE.0)GOTO 1
      READ(1,1002)FNBUF
 1002 FORMAT(3A4)
      CALL IRAD50(12,FNBUF,FNAMR)
      GOTO 2
    1 DO 3 IRD=1,NFILES
      READ(1,1002)FNBUF
      CALL IRAD50(12,FNBUF,FMBUF)
      RTNAM(IRD)=FMBUF
    3 CONTINUE
    2 CONTINUE
      READ(1,1001)(INDEX(I),I=1,NROW)
      READ(1,1001)(ICHAN(I),I=1,NCHAN)
      READ(1,1001)(FPOS(I),I=1,NCHAN)
      READ(1,1001)(VELAN(I),I=1,VELNUM)
      DO 5 IPL=1,NROW
      READ(1,1002)FNBUF
      CALL IRAD50(12,FNBUF,FMBUF)
    5 FNAMES(IPL)=FMBUF
      DO 6 ILP=1,VELNUM
      READ(1,1002)FNBUF
      CALL IRAD50(12,FNBUF,FMBUF)
    6 VELNAM(ILP)=FMBUF
C
C READ IN HEADER INFO
C
      READ(1,1001)NOCHAN,IGCODE,IUNITS,ISCODE
      READ(1,1003)SROFF,RSPAC,SLSPAC,STSPAC,SDEPT,RDEPT
 1003 FORMAT(6F10.0)
      READ(1,1004)(HSBLK(I),I=51,254)
 1004 FORMAT(80A1)
      HSBLKW(7)=NCHAN
      HSBLKW(8)=IHSTRT*128-128
      HSBLKW(9)=ITSIZ*128
```

```
      C
      C   SET UP CONSTANTS AND REST OF RT-11
      C   INPUT OUTPUT ROUTINES.
      C
0068        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=FNAMES(JJ)
0088        IF(IENTER(22+JJ,FMBUF,IFSIZ).LT.0)STOP' ENTER ERROR'
0090        IF(IWRITW(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=FNAMES(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 MBLKOF(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
      C OPEN FILE ON CH2Ø FOR SDS1Ø
      C
Ø11Ø        IN=IENTER(2Ø,FNAMR,-1)
Ø111        IF(IN.LT.Ø)WRITE(7,*)IN
Ø113        IF(IN.LT.Ø)STOP'FNAMR ENTER ERROR'
      C
      C CHECK ARNT AT EOT
      C
Ø115        IF(IEOTR.GE.Ø)GOTO 43
Ø117        WRITE(7,1Ø6Ø)TPDRR,IFNUM
Ø118   1Ø6Ø FORMAT(' EOT ENCOUNTERED ON DRIVE:',I2,' FILE NO:',I4)
Ø119        WRITE(7,1Ø61)
Ø12Ø   1Ø61 FORMAT(' ENTER NEW READ DRIVE NO:',$)
Ø121        READ(5,1Ø62)TPDRR
Ø122   1Ø62 FORMAT(I1)
Ø123        IEOTR=Ø
Ø124        IF(TPDRR.GT.2)GOTO 265
      C
      C CHECK IF ZERO FILES TO BE ADDED AT END OF TAPE
      C
Ø125        WRITE(7,1Ø63)
Ø127   1Ø63 FORMAT(' ENTER NO OF ZEO FILES TO BE ADDED(I2):',$)
Ø128        READ(5,1Ø64)IPADNO
Ø129   1Ø64 FORMAT(I2)
Ø13Ø        IF(IPADNO.GT.Ø)GOTO 1ØØ
      C
      C DO A READ
      C
Ø132     43 CALL TAPRED(-1,TPDRR,STATUS,TLEN,FLEN,IFNUM,IEOTR)
Ø133        IF(STATUS.LT.Ø)WRITE(2,1Ø5Ø)IFNUM
Ø135   1Ø5Ø FORMAT(' WARNING FILE NO ',I4,' RETRIES FAILED')
      C
      C DO A WIND
      C
Ø136        IF(IEOTR.LT.Ø)GOTO 46
Ø138        CALL TAPRED(Ø,TPDRR,STATUS, , ,IFNUM,IEOTR)
      C
      C START OF MAIN BUSINESS
      C
Ø139     46 CALL IWAIT(2Ø)
Ø14Ø        CALL CLOSEC(2Ø)
Ø141     45 BLK=Ø
Ø142        GCNT=Ø
Ø143        IHSEC=Ø
Ø144        EOFFLG=Ø
Ø145        RC=.FALSE.
Ø146        IF(INFLG.EQ.Ø)IOPEN=LOOKUP(2Ø,FNAMR)
Ø148        IF(INFLG.NE.Ø)FMBUF=RTNAM(FNUM)
Ø15Ø        IF(INFLG.NE.Ø)IOPEN=LOOKUP(2Ø,FMBUF)
Ø152        IF(IOPEN.LT.Ø)WRITE(7,*)IOPEN
Ø154        IF(IOPEN.LT.Ø)STOP'FNAMR LOOKUP ERROR'
```

```
C
C    READ IN FIRST DATA BLOCK FROM DISC AND EXTRACT THE
C    HEADER INFORMATION
C
        IF(IREADA(2Ø,BSST,BLK,F1).LT.Ø)STOP' READ ERROR'
        IF(IWAIT(2Ø).LT.Ø)STOP' WAIT ERROR'
        BLK=BLK+17
        IF(IREADA(2Ø,4Ø96,BLK,FBSST).LT.Ø)STOP 'READX ERROR'
        BLK=BLK+16
C
C   END OF INITIAL READS
C   BEGINNING OF HEADER BLOCK SCANS
C
        DO 5Ø LL=1,4
     5Ø HSBLKW(LL)=ISWAP(BUFF(LL)).OR."3ØØ6Ø
        HSBLKW(5)=ISWAP(BUFF(5))
        HSBLK(11)=TLEN
        ITIC=2*HSBLK(9)
        IGJL=1
        DO 6Ø JL=1,3Ø
        EGAINS(JL)=BUFF(5+JL).AND.MASK
        GSAVE(JL)=EGAINS(JL)+IBIAS
        GAINS(IGJL)=GSAVE(JL)
        IGJL=IGJL+128
     6Ø CONTINUE
        SPOS=35
     7Ø SPOS=SPOS+1
        IF(BUFF(SPOS).EQ.Ø)GO TO 7Ø
C
C    BUFFER CONSTANTS SET UP AFTER END OF HEADER BLOCK IS LOCATED
C
        FST=SPOS
        BSST1=4Ø96+FST
        BLST=8192+FST
        OLAP=257-FST
        RST=4Ø96-OLAP
C
C   CHECK SYNC WORDS FOR ERRORS AND EXTRACT THE GAINS READY
C   FOR USE AS INTEGERS.
C
        IFDIR=IDIRG(BUFF(SPOS+1))
        IF(IERFLG.NE.Ø.OR.IERR.NE.Ø)CALL BUFSCN(BUFF,Ø,IFNUM)
        IF(IERR.LT.Ø)GOTO 1ØØ
C
C   PUT FIRST BUFFER INTO AP
C
        F=APGAD(BUFF(FST))
        CALL APPUTX(Ø,4Ø96,1)
C
C FIND ADDRESS PAIRS FOR XM OPERATIONS
C
        FBSST1=APGAD(BUFF(BSST1))
        FBLST=APGAD(BUFF(BLST))
        IF(IERFLG.NE.Ø.OR.IERR.NE.Ø)GOTO 22Ø
```

```
      C---------------------------------------------------------------
      C   MAIN DEMUX LOOP FOR A SINGLE FILE
      C
      C---------------------------------------------------------------
      C
      C  COME HERE IF WANT TO DEMUX ALL THE FILE
      C WITHOUT FULL ERROR CHECKING SWITCHED ON?
Ø196    21Ø CONTINUE
Ø197        IHSEC=IHSEC+1
Ø198        CALL DMX
Ø199        IF(IERR.NE.Ø)GOTO 215
Ø2Ø1        IF(IHSEC.GE.ITSIZ)GOTO 23Ø
Ø2Ø3        IF(EOFFLG.EQ.2)GO TO 23Ø
Ø2Ø5        CALL DBLBUF(BUFF)
Ø2Ø6        GO TO 21Ø
Ø2Ø7    215 CALL IWAIT(2Ø)
Ø2Ø8        WRITE(2,1Ø3Ø)IFNUM
Ø2Ø9   1Ø3Ø FORMAT(' ERROR ON FILE:',I4,' FOUND GOING INTO RECOVERY MODE')
Ø21Ø        IF(NRECOV.EQ.Ø)GOTO 1ØØ
Ø212        DO 216 LLZ=1,NCHAN
Ø213    216 NBLKOF(LLZ)=IBLKOF(LLZ)
Ø214        GOTO 46
      C
      C  COME HERE IF WANT TO DEMUX ALL THE FILE
      C WITH FULL ERROR CHECKING SWITCHED ON
      C
Ø215    22Ø CONTINUE
Ø216        IHSEC=IHSEC+1
Ø217        IF (IHSEC.GE.IHSTRT)CALL DEMUX
Ø219        IF(IHSEC.GE.ITSIZ)GOTO 23Ø
Ø221        IF(EOFFLG.EQ.2)GOTO 23Ø
Ø223        CALL BUFSCN(BUFF,1,IFNUM)
Ø224        IF(IERR.GE.Ø)GOTO 22Ø
      C---------------------------------------------------------------
      C   END OF MAIN DEMUX LOOP
      C---------------------------------------------------------------
      C
      C BLANK PARTS OF FILES WITH FATAL ERRORS
Ø226    1ØØ CALL VCLR(Ø,1,128)
Ø227        CALL APWR
Ø228        CALL APGET(BUFOUT,Ø,128,2)
Ø229        WRITE(2,1Ø2Ø)IFNUM
Ø23Ø   1Ø2Ø FORMAT(' FILE NUMBER ',I4,' DELETED')
Ø231        CALL APWD
Ø232        DO 125 JZ=1,NCHAN
Ø233        NCH=INDEX(FPOS(JZ))
Ø234        IBLK=IBLKOF(JZ)
Ø235        DO 125 LZ=1,ITSIZO
Ø236        IF(IWRITW(256,BUFOUT,IBLK,22+NCH).LT.Ø)STOP'CLEAR ERR'
Ø238        IBLK=IBLK+1
Ø239    125 CONTINUE
Ø24Ø        IPADNO=IPADNO-1
      C
      C WRITE OUT HEADER BLOCK AND CLOSE DOWN COMPLTED GATHER FILE
```

```
C
  230 FMBUF=FNAMES(INDEX(1))
      IF(IWRITE(128,HSBLK,0,22+INDEX(NROW)).LT.0)STOP'HSBLK ERROR'
      IF(USEFLG.EQ.0.AND.FNUM.LT.NROW)GO TO 998
      CALL CLOSEC(22+INDEX(1))
      IF(OUTFLG.NE.0)GOTO 240
C
C  WRITE OUT GATHER FILE TO TAPE
C
      FLEN=LOOKUP(21,FMBUF)
      IF(FLEN.LT.0)STOP' FMBUF LOOKUP ERR'
      CALL TAPRED(1,TPDRW,STATUS,TLEN,FLEN,IFNUM,IEOTW)
      CALL CLOSEC(21)
      IF(IEOTW.GE.0)GOTO 235
      WRITE(7,1080)TPDRW,IFNUM
 1080 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
      WRITE(7,1081)
 1081 FORMAT(' ENTER NO OF NEW WRITE DRIVE:',$)
      READ(5,1062)TPDRW
      IEOTW=0
      IF(TPDRW.GT.2)GOTO 265
  235 IF(STATUS.GE.0)GOTO 240
      WRITE(2,1070)IFNUM
 1070 FORMAT(' WRITE ON FILE ',I4,' FATAL ERROR')
      GOTO 265
  240 IF(FNUM.NE.VELAN(VNUM))GOTO 250
C
C   SET UP A VELOCITY ANALYSIS FILE
C
      DBLK(1)=FMBUF
      DBLK(2)=VELNAM(VNUM)
      IF(IRENAM(21,DBLK).GT.0)STOP 'RENAME ERROR'
      VNUM=VNUM+1
      IF(IENTER(22+INDEX(1),FMBUF,IFSIZ).LT.0)STOP'ENTER ERROR'
      IF(IWRITE(256,BUFOUT,LSTBLK,22+INDEX(1)).LT.0)STOP'WRITE ERROR'
      CALL CLOSEC(22+INDEX(1))
  250 CONTINUE
      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
  998 FNUM=FNUM+1
      CALL CLOSEC(20)
  999 CONTINUE
  265 DO 260 LL=1,NROW
  260 CALL CLOSEC(22+LL)
      WRITE(2,1001)(INDEX(I),I=1,NROW)
      STOP'NORMAL TERMINATION'
      END
```

```
      SUBROUTINE DEMUX
      INTEGER*2 CHOFF(3Ø),GANADD,FPOS(3Ø),NBLKOF(3Ø),
     %EOFFLG,GAINS(384Ø),ICHAN(3Ø),GSAVE(3Ø),INDEX(31),GCNT
      REAL*4 BUFOUT(256)
      LOGICAL*1 RC
      COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
     %IERFLG,ISPOS,RC
      COMMON /DECOM/BUFOUT,CHOFF,ICHAN,FPOS,NCHAN,NBLKOF,INDEX,GCNT,
     #IHSTRT,IHSEC
C
C    DO DEMUX AND BRING IN THE SAVED GAINS
C
      NCH=Ø
      NIN=128
      IF(EOFFLG.EQ.2)NIN=NSMPIN
      NOUT=2*NIN
      IF(NOUT.EQ.Ø)RETURN
      CALL APWD
      CALL VFLT(4352,1,4352,1,384Ø)
      CALL APWR
      CALL APPUT(GAINS,Ø,384Ø,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
      CALL APWD
      DO 1Ø NJ=1,NCHAN
      NCHADD=CHOFF(ICHAN(NJ))
      GANADD=NCHADD-4352
      CALL VBINSC(NCHADD,1,NCHADD,1,GANADD,1,NIN)
      CALL APWR
      CALL IWAIT(22+NCH)
      NCH=INDEX(FPOS(NJ))
      CALL APGET(BUFOUT,NCHADD,NIN,2)
      CALL APWD
      IF(IWRITE(NOUT,BUFOUT,NBLKOF(NJ),22+NCH).LT.Ø)STOP'DEMUX ERROR'
      NBLKOF(NJ)=NBLKOF(NJ)+1
   1Ø CONTINUE
      CALL IWAIT(22+NCH)
      RETURN
      END
```

```
      SUBROUTINE DMX
      INTEGER*2 CHOFF(30),FPOS(30),NBLKOF(30),GCNT,EOFFLG,
     %GAINS(3840),ICHAN(30),GSAVE(30),INDEX(31)
      REAL*4 BUFOUT(256)
      LOGICAL*1 RC
      COMMON /SUBS/ GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
     %IERFLG,ISPOS,RC
      COMMON /DECOM/BUFOUT,CHOFF,ICHAN,FPOS,NCHAN,NBLKOF,INDEX,GCNT,
     #IHSTRT,IHSEC
C
C    DO DEMUX AND BINARY SCALING
C
      NCH=0
      NIN=128
      IF(EOFFLG.EQ.2)NIN=NSMPIN
      NOUT=2*NIN
      IF(NOUT.EQ.0)RETURN
      CALL APWD
      CALL APPUT(GSAVE,4096,30,1)
      CALL APWD
      CALL DMXA(4096,4352,128,GCNT,IFDIR,NIN)
      CALL APWR
      CALL APGSP(IERR,15)
      IF(IERR.NE.0)IERR=1
      IF(IERR.NE.0)RETURN
      CALL APGSP(GCNT,3)
      CALL APGSP(IFDIR,4)
      CALL APGET(GSAVE,4096,30,1)
      CALL IWAIT(20)
      IF(IHSEC.LT.IHSTRT)GOTO 30
      NCHADD=CHOFF(ICHAN(1))
      CALL APGET(BUFOUT(1),NCHADD,NIN,2)
      IIN=129
      IOUT=1
      IF(NCHAN.EQ.1)GOTO 20
C
C   EXTRACT EACH WANTED CHANNEL AND PUT ON DISC
C
      DO 10 NJ=2,NCHAN
      NJ1=NJ-1
      NCHADD=CHOFF(ICHAN(NJ))
      CALL IWAIT(22+NCH)
      CALL APGET(BUFOUT(IIN),NCHADD,NIN,2)
      NCH=INDEX(FPOS(NJ1))
      IF(IWRITE(NOUT,BUFOUT(IOUT),NBLKOF(NJ1),22+NCH).LT.0)STOP'DMX'
      NBLKOF(NJ1)=NBLKOF(NJ1)+1
      IT=IIN
      IIN=IOUT
      IOUT=IT
   10 CONTINUE
   20 NCH=INDEX(FPOS(NCHAN))
      CALL APWD
      IF(IWRITE(NOUT,BUFOUT(IOUT),NBLKOF(NCHAN),22+NCH).LT.0)STOP'DMX'
      NBLKOF(NCHAN)=NBLKOF(NCHAN)+1
```



```
   30 RC=.NOT.RC
      CALL IWAIT(22+NCH)
      RETURN
      END
```

```
      SUBROUTINE DBLBUF(BUFF)
      VIRTUAL BUFF(8448)
      INTEGER*2 BUFF,EOFFLG,BSST,BLST,SPOS,BSST1,
     %RST,SYNC,OLAP,BLK,GSAVE(30),GAINS(3840)
      LOGICAL*1 RC
      COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
     %IERFLG,ISPOS,RC
      COMMON /BUFCOM/FBSST1,FBLST,F256,F256D,FBSST,F1,F4097,
     %BSST1,BLST,OLAP,RST,BLK
      DATA SYNC /"177777/
C
C  THIS ROUTINE CONTROLS THE DOUBLE BUFFERING SCHEME USED
C  TO TAKE DATA FROM DISC AND PUT IT IN TO THE AP
C  WHEN IT IS NEEDED. IT IS ALSO RESPONSIBLE FOR CHECKING
C  THE SYNC WORDS AND EXTRACTING THE GAINS AS INTEGERS FOR USE
C
C
C  SET UP THE START OF THE BUFFER
C
      SPOS=BSST1
      IF(.NOT.RC)SPOS=BLST
C
C  DO A BUFFER SCAN WHEN THE INPUT PROCEDURE HAS NOTIFIED EOF
C
      IF(EOFFLG.LE.0)GOTO 50
      ISMPIN=0
      IPOS=SPOS
      DO 40 L=1,NSMPIN
      IF(BUFF(IPOS).NE.SYNC)GO TO 60
      IPOS=IPOS+32
      IF(IPOS.GT.8448)IPOS=IPOS-8192
      ISMPIN=ISMPIN+1
   40 CONTINUE
      GOTO 50
   60 NSMPIN=ISMPIN
   50 CONTINUE
C
C  PUT A BUFFER INTO THE A.P. AND START THE READ TO FILL
C  THE SECOND BUFFER FOR USE NEXT TIME
C
      IF(RC)CALL APPUTA(0,4096,1,FBSST1)
      IF(.NOT.RC)CALL APPUTA(0,OLAP,1,FBLST)
      IF(.NOT.RC)CALL APPUTA(OLAP,RST,1,F256)
      IF(EOFFLG.GT.0)EOFFLG=2
      IF(EOFFLG.GT.0)RETURN
      IF(EOFFLG.LT.0)EOFFLG=1
      IF(EOFFLG.GT.0)RETURN
      FINP=F256D
      IF(.NOT.RC)FINP=FBSST
      IIN=IREADA(20,4096,BLK,FINP)
      BLK=BLK+16
C
C  CHECK INFO RETURNED FROM INPUT ROUTINE
C  FOR ERRORS AND AN EOF SITUATION
```



```
C
      IF(IIN.EQ.4096)RETURN
      IF(IIN.GT.RST)GOTO 140
      IF(IIN.EQ.-1)IIN=0
      IF(IIN.LT.0)WRITE(7,*)IIN,BLK
      IF(IIN.LT.0)STOP'READ ERROR'
      EOFFLG=1
      NSMPIN=(OLAP+IIN)/32
      RETURN
  140 EOFFLG=-1
      IIN=IIN-RST
      NSMPIN=IIN/32
      RETURN
      END
```

```
ØØØ1          SUBROUTINE BUFSCN(BUFF,ICODE,FDONE)
ØØØ2          VIRTUAL BUFF(8448)
ØØØ3          INTEGER*2 BUFF,GAINS(384Ø),GSAVE(3Ø),SPOS,EOFFLG,DMXBUF(384Ø),
             %FRAME(33),FDONE
ØØØ4          LOGICAL*1 RC,IF(2),IBYT(2),IBYTE(2),FRAMEB(66),IBSYNC
ØØØ5          EQUIVALENCE (IWORD,IF(1)),(IWORDF,IBYT(1)),(IWORDB,IBYTE(1)),
             %(FRAME(1),FRAMEB(1))
ØØØ6          COMMON /SUBS/GAINS,GSAVE,NSMPIN,EOFFLG,IFDIR,IERR,
             %IERFLG,SPOS,RC
ØØØ7          COMMON/BUFS/NUERR,NALOW,ITIC
ØØØ8          DATA ISYNC/"177777/,IBIAS/15/
ØØØ9          DATA IBSYNC/"377/
ØØ1Ø          IF(ICODE.GT.Ø)GOTO 1
ØØ12          ICHCK=1
ØØ13          ICD=Ø
ØØ14          LPINT=Ø
ØØ15          FRAME(33)=BUFF(SPOS+1)
ØØ16          SPOS=SPOS+2
ØØ17          ITBIAS=BUFF(SPOS+24)/2*2
ØØ18          ITCONT=Ø
ØØ19          NERR=Ø
ØØ2Ø          GSAVE(3Ø)=3Ø
ØØ21          CALL FRAMFL(BUFF,FRAME,1,33,-1)
        C
        C DATA CHECK AND GAIN PREPARATION SUBROUTINE
        C
ØØ22        1 IGSCHK=GSAVE(ICHCK)
ØØ23          LDONE=Ø
ØØ24        5 LDONE=LDONE+1
ØØ25          IF(LPINT.GT.Ø)GOTO 1ØØ
ØØ27          FRAME(1)=FRAME(33)
ØØ28          CALL FRAMFL(BUFF,FRAME,2,33,ICD)
        C
        C SYNC TEST
        C
ØØ29          IF(EOFFLG.EQ.2)GOTO 9Ø
ØØ31        6 IF(FRAME(32).NE.ISYNC)GOTO 2Ø
ØØ33          IF(FRAMEB(66).EQ.IBSYNC)GOTO 3Ø
ØØ35          WRITE(2,1Ø1Ø)FDONE
ØØ36     1Ø1Ø FORMAT(' FILE NO:',I4,' ERROR DETECTED')
        C
        C TEST FOR TYPE OF DATA CORRUPTION
        C
ØØ37       2Ø IF(LDONE.EQ.128)GOTO 9Ø
ØØ39          IF(NERR.GT.NUERR)GOTO 16Ø
ØØ41          NERR=NERR+1
ØØ42          NFER=Ø
ØØ43       25 DO 4Ø I=1,32
ØØ44          IWORDB=FRAME(I)
ØØ45          IF(IWORDB.NE.ISYNC)GOTO 5Ø
ØØ47          IWORDB=FRAME(I+1)
ØØ49          IF(IBYTE(2).NE.IBSYNC)GOTO 5Ø
        C
        C DETECT PATTERN OF BYTES LOST
```

```
      C
0050          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
0078       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
0086          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
0098          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
   30 ITN=FRAME(26)/2*2
      ITP=ITCONT.OR.ITBIAS
      IF(ITN.EQ.ITP)GOTO 90
      IF(NERR.GT.NUERR)GOTO 160
      WRITE(2,1030)FDONE
 1030 FORMAT(' TIME CHECK ERROR ON FILE NO:',I4)
      NERR=NERR+1
      INTVAL=(ITN-ITP)/ITIC
      IF(INTVAL.LT.0)GOTO 105
      LPINT=INTVAL
  100 IGPOS=LDONE
      DO 110 IVI=1,29
      DMXBUF(IGPOS)=1
      GAINS(IGPOS)=GSAVE(IVI)
      IGPOS=IGPOS+128
  110 CONTINUE
      GAINS(IGPOS)=30
      LDONE=LDONE+1
      LPINT=LPINT-1
      ITCONT=ITCONT+ITIC
      ICHCK=ICHCK+1
      IF(ICHCK.GT.30)ICHCK=1
      IF(LDONE.GE.128)GOTO 150
      IF(LPINT.GT.0)GOTO 100
  105 CONTINUE
   90 ITCONT=ITCONT+ITIC
C
C GAINS CORRECTION
C
      IGSAVE=IGCHK(FRAME(1),IDIR)+IBIAS
      IF(IGSAVE.EQ.IGSCHK)GOTO 120
      WRITE(2,1040)FDONE
 1040 FORMAT(' GAIN CHECK ERROR FILE NO:',I4)
      IF(NERR.GT.NUERR)GOTO 160
      NERR=NERR+1
      GSAVE(ICHCK)=IGSAVE
  120 IF(IFDIR.EQ.IDIR)GOTO 130
      WRITE(2,1050)FDONE
 1050 FORMAT(' GAIN DIRECTION ERROR FILE NO:',I4)
      IF(NERR.GT.NUERR)GOTO 160
      NERR=NERR+1
      IFDIR=IDIR
C
C NORMAL WORK
C
  130 ICHCK=ICHCK+1
      IF(ICHCK.GT.30)ICHCK=1
      IGPOS=LDONE
      DO 140 L=1,29
      DMXBUF(IGPOS)=IGAIN(FRAME(L+1),GSAVE(L),IFDIR)
      GAINS(IGPOS)=GSAVE(L)
      IGPOS=IGPOS+128
```



```
  140 CONTINUE
      GAINS(IGPOS)=30
      IGSCHK=GSAVE(ICHCK)
      IFDIR=-IFDIR
      IF(LDONE.LT.128.AND.EOFFLG.NE.2)GOTO 5
  150 NSMPIN=LDONE
      CALL APPUT(DMXBUF,4352,3840,1)
      RETURN
  160 WRITE(2,1060) FDONE
 1060 FORMAT(' FILE NO ',I4,' DECLARED DEAD')
      IERR=-1
      RETURN
      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
0010        BUFEND=8449
0011        IBEG=4097
0012        EOF=0
0013        RETURN
      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
0060        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 EOFFLG=2
0079        RETURN
0080        END
```

```
ØØØ1          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,Ø 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
ØØØ2          INTEGER*2 MASK(8),ESTATI
ØØØ3          LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
             %IFLEN,ESTAT,ERRS(8)
ØØØ4          DATA MASK/"1,"2,"4,"1Ø,"2Ø,"4Ø,"1ØØ,"2ØØ/
ØØØ5          DATA SDSCOM/"Ø,"1,"2,"3,"4,"5,"6,"7/
ØØØ6          DATA ERRS/"377,"377,"377,"377,"377,"377,"377,"377/
ØØØ7          ITRY=Ø
ØØØ8          IF(ICOM) 1Ø,3Ø,2Ø
        C
        C SECTION CONTROLLING A READ
        C
        C
        C  CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
        C
ØØØ9       1Ø ITRY=ITRY+1
        C
        C SET UP COMMAND FOR READ
        C
ØØ1Ø          COM(1)=SDSCOM(4)
ØØ11          COM(2)=Ø
ØØ12          COM(3)=IDRV
ØØ13          COM(4)=-1
ØØ14          CALL SDS1Ø(COM,ISTAT,ITLEN,ILEN)
ØØ15          IF(ISTAT.EQ.Ø)RETURN
        C
        C ERROR DETECTED ON READ
        C
ØØ17          ISTATI=ISTAT
ØØ18          GOTO 4Ø
        C
        C IF SHORT RECORD FOUND REREAD TAPE
        C
ØØ19       5Ø ITMP=ISTATI.AND.MASK(6)
ØØ2Ø          IF(ITMP.NE.Ø)GOTO 1Ø
ØØ22          ITMP=ISTATI.AND.MASK(2)
ØØ23          IF(ITMP.EQ.Ø)RETURN
        C
        C IF CRC ERROR FOUND REWIND TAPE AND RETRY
        C
ØØ25          WRITE(2,2Ø1Ø)IFNUM
ØØ26     2Ø1Ø FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
ØØ27          IF(ITRY.GE.2)GOTO 13Ø
ØØ29          ECOM(1)=SDSCOM(6)
ØØ3Ø          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
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          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 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
0069          ISTAT=ISTAT.AND..NOT.MASK(6)
0070          IF(ISTAT.EQ.2)RETURN
0072          ISTATI=ISTAT
0073          IF(ISTAT.NE.0)GOTO 40
```

```
C
C IF ISTAT=Ø 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)=Ø
        CALL SDS1Ø(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
     4Ø WRITE(2,1Ø1Ø)ISTATI,IFNUM
   1Ø1Ø FORMAT(' STATUS=',I3,' FILE NO=',I4)
        ISTATI=ISTATI.AND..NOT.MASK(4)
        ITMP=ISTATI.AND.MASK(5)
        IF(ITMP.EQ.Ø)GOTO 8Ø
     9Ø ECOM(1)=SDSCOM(1)
        ECOM(2)=Ø
        ECOM(3)=IDRV
        ECOM(4)=Ø
        CALL SDS1Ø(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.Ø)GOTO 9Ø
        IF(ICOM) 1Ø,3Ø,2Ø
C
C TAPE OFFLINE
C
     8Ø ITMP=ISTATI.AND.MASK(1)
        IF(ITMP.EQ.Ø)GOTO 1ØØ
        TYPE 1ØØ1,IDRV
   1ØØ1 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
    11Ø ECOM(1)=SDSCOM(1)
        ECOM(2)=Ø
        ECOM(3)=IDRV
        ECOM(4)=Ø
        CALL SDS1Ø(ECOM,ESTAT, , )
        ESTATI=ESTAT
        ITMP=ESTATI.AND.MASK(1)
        IF(ITMP.NE.Ø)GOTO 11Ø
```



```
        IF(ICOM) 1Ø,3Ø,2Ø
C
C EOT
C
    1ØØ ITMP=ISTATI.AND.MASK(3)
        IF(ITMP.EQ.Ø)GOTO 12Ø
        TYPE 1ØØ2,IDRV
   1ØØ2 FORMAT(' EOT ON DRIVE ',I1)
        IEOT=-1
        RETURN
    12Ø IF(ICOM) 5Ø,35,7Ø
C
C ERROR EXIT RETURN
C
    13Ø ISTATI=-1
        RETURN
        END
```

```
        .TITLE  ISWAP

        .GLOBL  ISWAP,IGCHK,IGAIN,IDIRG
ISWAP:  MOV     @2(R5),RØ
        SWAB    RØ
        RTS     PC

IGCHK:  MOV     @2(R5),RØ
        TSTB    RØ
        BPL     1$
        MOV     #1,@4(R5)
        BR      2$
1$:     MOV     #-1.,@4(R5)
2$:     BIC     #17776Ø,RØ
        RTS     PC

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

IGAIN:  MOV     @2(R5),RØ
        ASR     RØ
        BCC     4$
        ADD     @6(R5),@4(R5)
4$:     TST     RØ
        BPL     5$
        ADD     #1,RØ
5$:     ASL     RØ
        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 8Ø
      C  MPSORT.FOR... THIS IS A
      C  GENERAL PURPOSE SORTING PROGRAM FOR
      C  SEISMIC DATA FILES
      C
ØØØ1        VIRTUAL RDNAM(2ØØ),WRTNAM(2ØØ),TPNAM(24)
ØØØ2        REAL *8 FSPECR,FSPECW,FMBUF,RDNAM,WRTNAM,TPNAM
ØØØ3        REAL*4 DEV,FNBUF(3),SEIS(2Ø48)
ØØØ4        INTEGER*2 IHBLK(256),IBLKO(24),IBLKI(24),ICHANO(24),
           %IPOS(24),USEFLG,INFLG,OUTFLG,INDEX(25),FLEN
ØØØ5        LOGICAL*1 ISTAT,TLEN,TPDRR,TPDRW,LBLK(512),IGCODE
ØØØ6        EQUIVALENCE (LBLK(1),IHBLK(1))
ØØØ7        DATA DEV/3RRK /
      C
      C SET UP I/O
      C
ØØØ8        IF(ICDFN(5Ø).NE.Ø)STOP' CHANNEL SET ERROR'
ØØ1Ø        IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
ØØ12        CALL ASSIGN(1,'DK2:MPSORT.DAT',14)
ØØ13        CALL ASSIGN(2,'DK2:MPSORT.LOG',14)
ØØ14        IRD=2Ø
ØØ15        IWRT=21
ØØ16        IEOTR=Ø
ØØ17        IEOTW=Ø
      C
      C READ INPUT DATA
      C
ØØ18        READ(1,1ØØØ)NFILES,NCHANI,NCHANO,NROW,TPDRR,TPDRW
ØØ19   1ØØØ FORMAT(12I5)
ØØ2Ø        READ(1,1ØØØ)ISECIN,ISBLKO,IFBLKO,USEFLG,INFLG,OUTFLG,IGCODE
ØØ21        READ(1,1ØØØ)(INDEX(I),I=1,NROW)
ØØ22        READ(1,1ØØØ)(ICHANO(I),I=1,NCHANO)
ØØ23        READ(1,1ØØØ)(IPOS(I),I=1,NCHANO)
      C
      C READ FILE SPECS
      C
ØØ24        IF(INFLG.NE.Ø)GOTO 1Ø
ØØ25        READ(1,1ØØ1)FNBUF
ØØ27   1ØØ1 FORMAT(3A4)
ØØ28        CALL IRAD5Ø(12,FNBUF,FSPECR)
ØØ29        GOTO 15
ØØ3Ø     1Ø DO 2Ø J=1,NFILES
ØØ31        READ(1,1ØØ1)FNBUF
ØØ32        CALL IRAD5Ø(12,FNBUF,FMBUF)
ØØ33        RDNAM(J)=FMBUF
ØØ34     2Ø CONTINUE
ØØ35     15 IF(OUTFLG.EQ.Ø)GOTO 3Ø
ØØ37        DO 35 J=1,NFILES
ØØ38        READ(1,1ØØ1)FNBUF
ØØ39        CALL IRAD5Ø(12,FNBUF,FMBUF)
ØØ4Ø        WRTNAM(J)=FMBUF
ØØ41     35 CONTINUE
      C
```

```
      C READ ARRAY SORT FILE SPECS
      C
0042     30 DO 40 J=1,NROW
            READ(1,1001)FNBUF
0043
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
0062        DO 65 L=1,NROW
0063        FMBUF=TPNAM(L)
0064        ITCH=22+L
0065        IF(IENTER(ITCH,FMBUF,IFSIZO).LT.0)STOP'ENTER ERR'
0067        IF(IWRITW(256,IHBLK,LSTBLK,ITCH).LT.0)STOP'WRITE ERR'
0069        CALL CLOSEC(ITCH)
0070     65 CONTINUE
0071     60 DO 70 L=1,NROW
0072        ITCH=22+L
0073        FMBUF=TPNAM(L)
0074        IF(LOOKUP(ITCH,FMBUF).LT.0)STOP'LOOKUP ERR'
0076     70 CONTINUE
      C
      C START OF MAIN WORK LOOP
      C
0077        DO 999 IFIL=1,NFILES
0078        IFNUM=IFIL
0079        INDEX(NROW+1)=INDEX(1)
0080        DO 100 J=1,NROW
0081    100 INDEX(J)=INDEX(J+1)
0082        IF(INFLG.NE.0)GOTO 105
      C
      C TAPE READ CONTROL
      C
```

```
ØØ84          IF(IENTER(IRD,FSPECR,NBLKR).LT.Ø)STOP'ENTER ERR'
ØØ86          ITRY=1
        C
        C EOT CONTROL
        C
ØØ87      12Ø IF(ITRY.GT.3)GOTO 125
ØØ89          IF(IEOTR.GE.Ø)GOTO 11Ø
ØØ91      125 WRITE(7,1Ø1Ø)TPDRR,IFNUM
ØØ92     1Ø1Ø FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I5)
ØØ93          WRITE(7,1Ø2Ø)
ØØ94     1Ø2Ø FORMAT(' ENTER NEW READ TAPE DRIVE:',$)
ØØ95          READ(5,1Ø3Ø)TPDRR
ØØ96     1Ø3Ø FORMAT(I1)
ØØ97          IEOTR=Ø
ØØ98          IF(TPDRR.GT.2)STOP'EOTR TERMINATE'
        C
        C TAPE READ
        C
Ø1ØØ      11Ø CALL TAPRED(-1,TPDRR,ISTAT,TLEN,FLEN,IFNUM,IEOTR)
Ø1Ø1          IF(ISTAT.LT.Ø)WRITE(2,1Ø7Ø)IFNUM
Ø1Ø3     1Ø7Ø FORMAT(' WARNING RETRY FAILED FOR FILE:',I5)
Ø1Ø4          IF(IEOTR.LT.Ø)GOTO 115
Ø1Ø6          CALL TAPRED(Ø,TPDRR,ISTAT, , ,IFNUM,IEOTR)
        C
        C CHECK IF READ A BAD FILE
        C
Ø1Ø7      115 CALL IWAIT(IRD)
Ø1Ø8          IERR=Ø
Ø1Ø9          ITRY=ITRY+1
Ø11Ø          IF(IREADW(1,IERR,Ø,IRD).LT.Ø)STOP'ERR READ ERR'
Ø112          IF(IERR.EQ."177777)GOTO 12Ø
Ø114          CALL CLOSEC(IRD)
        C
        C OPEN UP INPUT FILE FOR USE
        C
Ø115      1Ø5 FMBUF=FSPECR
Ø116          IF(INFLG.NE.Ø) FMBUF=RDNAM(IFNUM)
Ø118          IF(LOOKUP(IRD,FMBUF).LT.Ø)STOP'LOOKUP ERR'
        C
        C HEADER BLOCK MANIPULATION
        C
Ø12Ø          ITCH=22+INDEX(NROW)
Ø121          IF(IREADW(256,IHBLK,Ø,IRD).LT.Ø)STOP'READ ERR'
Ø123          LBLK(19)=IGCODE
Ø124          IHBLK(7)=NCHANO
Ø125          IHBLK(8)=IHBLK(8)+NBEG
Ø126          IHBLK(9)=IHBLK(9)+NFIN
Ø127          IF(IWRITW(256,IHBLK,Ø,ITCH).LT.Ø)STOP'WRITE ERR'
        C
        C MAIN TRANSFER LOOP
        C
Ø129          DO 2ØØ ICH=1,NCHANO
Ø13Ø          ITCH=22+INDEX(IPOS(ICH))
Ø131          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)
0166         STOP'WRITE ERROR TERMINATION'
      C
      C STORAGE ON DISC
      C
0167     220 ISV=ISV+1
0168         FSPECW=WRTNAM(ISV)
0169         IF(IENTER(IWRT,FSPECW,IFSIZO).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
0186     250 CALL CLOSEC(22+J)
0187         WRITE(2,1000)(INDEX(I),I=1,NROW)
0188         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

$e^{0.2t}$ Ramp

READ(1,1000)IAPLX

IAPLX....Application flag

0 - apply ramp

1 - remove ramp

te0.2t Ramp

READ(1,1000)IAPLTX

IAPLTX...Application flag

0 - apply ramp

1 - remove ramp

TV**2 Ramp

READ(1,1000)IAPLTV,NLYR

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

IAPLTV...Application flag

0 - apply ramp

1 - remove ramp

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

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

VLYR.....RMS velocity at above time

4....Mute

READ(1,1000)NTAP

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

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

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

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

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

5....Spiking Deconvolution

READ(1,1000)NFILT,ISPIKE,INORM

READ(1,1100)WHITE

NFILT....Number of samples in the filter

ISPIKE...Spike position

INORM....Normalisation flag

0 - no normalisation

1 - Filter unit energy

2 - constant input/output energy

WHITE....Fractional pre-whitening

6....Bandpass Filtering

READ(1,1100)FL,FU

READ(1,1100)FTPR1,FTPR2

FL.......Starting frequency for lower cutoff position Hz

FU.......Starting frequency for upper cutoff position Hz

FTPR1....Length of lower cosine taper Hz

FTPR2....Length of upper cosine taper Hz

7....Bandreject filtering

READ(1,1100)FLR,FUR

FLR......Lower frequency cutoff position Hz

FUR......Upper frequency cutoff position Hz

8....Prediction Error Deconvolution

READ(1,1000)NPFILT,NLAG,IPNORM

READ(1,1100)PRWHIT

NPFILT...Filter Length in samples

NLAG.....Prediction distance, samples

IPNORM...Normalisation flag

0 - no normalisation

1 - Filter normalised to unit energy

2 - Constant input/output energy

PRWHIT...Fractional prewhitening

9....Normalisation

READ(1,1000)NRMFLG

NRMFLG...Normalisation flag

0 - normalise to unit energy

1 - normalise to unit maximum amplitude

```
C
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
      REAL*8 FSPECR,FSPECW,FNAMR,FNAMO
      VIRTUAL FNAMR(100),FNAMO(100),EXPT(2048),TEXPT(2048),
     %TVSQ(2048),TAPER(512),BPASS(2049),BRJCT(2049)
      REAL*4 FBUF(3),T0LYR(20),VLYR(20),CONST(3),SEISM(2048)
      INTEGER*2 IORD(11),UTLFLG(11),IFILED(100),ICHAND(100),
     %ICHANP(24),MUTE(24),MUTET(24),IHBLK(256),OUTFLG
      LOGICAL*1 TPDRR,TPDRW,ISTAT,ITLEN
      DATA DEV/3RRK /
C
C SET UP VIRTUAL ADDRESSES
C
      IEOTR=0
      IEOTW=0
      ATAP=APGAD(TAPER(1))
      ATBP=APGAD(BPASS(1))
      ATVSQ=APGAD(TVSQ(1))
      ATEXPT=APGAD(TEXPT(1))
      AEXPT=APGAD(EXPT(1))
      ATBR=APGAD(BRJCT(1))
C
C SET UP I/O  CHANNELS AND READ IN CONTROL DATA
C
      IF(ICDFN(25).NE.0)STOP'CHANNEL OVERFLOW'
      CALL ASSIGN(1,'DK2:MPPRST.DAT',14)
      CALL ASSIGN(2,'DK2:MPPRST.LOG',14)
      IDCH=20
      IDCH1=21
      READ(1,1000)NFILES,NCHAN,NSAMP,NSTART,INFLG,OUTFLG
 1000 FORMAT(12I5)
      READ(1,1000)TPDRR,TPDRW
      READ(1,1100)FSAMP
 1100 FORMAT(2F10.0)
C
C READ IN FILE SPECS FOR INPUT
C
      IF(INFLG.NE.0)GOTO 10
      READ(1,1200)FBUF
 1200 FORMAT(3A4)
      CALL IRAD50(12,FBUF,FSPECR)
```

```
      GOTO 20
   10 DO 30 I=1,NFILES
      READ(1,1200)FBUF
      CALL IRAD50(12,FBUF,FSPECR)
      FNAMR(I)=FSPECR
   30 CONTINUE
C
C READ IN FILE SPECS FOR OUTPUT
C
   20 IF(OUTFLG.NE.0)GOTO 40
      READ(1,1200)FBUF
      CALL IRAD50(12,FBUF,FSPECW)
      GOTO 50
   40 DO 60 I=1,NFILES
      READ(1,1200)FBUF
      CALL IRAD50(12,FBUF,FSPECW)
      FNAMO(I)=FSPECW
   60 CONTINUE
C
C READ IN JOB SPECIFIC DATA AND SET UP
C THE FILTERS TO BE USED
C
   50 NUTIL=11
      ITX=0
      CONST(1)=FLOAT(NSTART)
      CONST(2)=0.2
      CONST(3)=1.0/(1000.0*FSAMP)
      CALL APINIT
      NSAMP2=2
   51 IF(NSAMP2.GE.NSAMP)GOTO 52
      NSAMP2=NSAMP2*2
      GOTO 51
   52 CONTINUE
C
C READ IN FLAGS FOR PROCESSES AND EXECUTION ORDER
C
      READ(1,1000)NPROC
      READ(1,1000)(UTLFLG(I),I=1,NUTIL)
      READ(1,1000)(IORD(I),I=1,NPROC)
C
C TRACE EDIT DATA
C
      IF(UTLFLG(1).EQ.0)GOTO 65
      READ(1,1000)NFILED
      READ(1,1000)(IFILED(I),ICHAND(I),I=1,NFILED)
C
C POLARITY REVERSAL DATA
C
   65 IF(UTLFLG(2).EQ.0)GOTO 70
      READ(1,1000)NCHANP
      READ(1,1000)(ICHANP(I),I=1,NCHANP)
C
C EXP(0.2T) RAMP DATA
C
```

```
    7Ø IF(UTLFLG(3).EQ.Ø)GOTO 8Ø
       READ(1,1ØØØ)IAPLX
       CALL TEXRMP(EXPT,Ø,ITX,NSAMP,CONST,AEXPT)
       ITX=1
C
C T*EXP(Ø.2T) RAMP DATA
C
    8Ø IF(UTLFLG(4).EQ.Ø)GOTO 9Ø
       READ(1,1ØØØ)IAPLTX
       CALL TEXRMP(TEXPT,1,ITX,NSAMP,CONST,ATEXPT)
       ITX=1
C
C TV**2 RAMP
C
    9Ø IF(UTLFLG(5).EQ.Ø)GOTO 1ØØ
       READ(1,1ØØØ)IAPLTV,NLYR
       READ(1,11ØØ)(TØLYR(I),VLYR(I),I=1,NLYR)
       CALL TVRMP(TVSQ,TØLYR,VLYR,NLYR,ITX,NSAMP,NSTART,CONST,
      %FSAMP,ATVSQ)
C
C MUTE DATA
C
   1ØØ IF(UTLFLG(6).EQ.Ø)GOTO 11Ø
       READ(1,1ØØØ)NTAP
       READ(1,1ØØØ)(MUTE(I),I=1,NCHAN)
       READ(1,1ØØØ)(MUTET(I),I=1,NCHAN)
       CALL COTAP(TAPER,NTAP,ATAP)
C
C DECON INPUT
C
   11Ø IF(UTLFLG(7).EQ.Ø)GOTO 12Ø
       READ(1,1ØØØ)NFILT,ISPIKE,INORM
       READ(1,11ØØ)WHITE
C
C BANDPASS FILTER
C
   12Ø IF(UTLFLG(8).EQ.Ø)GOTO 13Ø
       READ(1,11ØØ)FL,FU
       READ(1,11ØØ)FTPR1,FTPR2
       DFI=FLOAT(NSAMP2/2+1)/(FSAMP*5ØØ.Ø)
       FL1=FL-FTPR1
       FU4=FU+FTPR2
       CALL BANDPS(ATBP,BPASS,FL1,FL,FU,FU4,DFI,NSAMP2)
       NTRANF=2*NSAMP2
       NBFILT=NSAMP2+1
       NBEXP=(2*NSAMP2)+2-NSAMP
C
C BANDREJECT FILTER
C
   13Ø IF(UTLFLG(9).EQ.Ø)GOTO 14Ø
       READ(1,11ØØ)FLR,FUR
       DFI=FLOAT(NSAMP2/2+1)/(FSAMP*5ØØ.Ø)
       NTRANF=2*NSAMP2
       NRFILT=NSAMP2+1
```

```
Ø112          NBEXP=(2*NSAMP2)+2-NSAMP
Ø113          CALL BANDRJ(ATBR,BRJCT,FLR,FUR,DFI,NSAMP2)
      C
      C PREDICTION ERROR FILTER
      C
Ø114      14Ø IF(UTLFLG(1Ø).EQ.Ø)GOTO 15Ø
Ø116          READ(1,1ØØØ)NPFILT,NLAG,IPNORM
Ø117          READ(1,11ØØ)PRWHIT
      C
      C TRACE NORMALISATION
      C
Ø118      15Ø IF(UTLFLG(11).EQ.Ø)GOTO 16Ø
Ø12Ø          READ(1,1ØØØ)NRMFLG
      C
      C BLOCKING PARAMETERS
      C
Ø121      16Ø CONTINUE
Ø122          IFED=1
Ø123          NBLKW=NSAMP/128*NCHAN+1
Ø124          NBLKR=NBLKW+5
Ø125          NBLKTR=NSAMP/128
      C
      C START OF LOOP ON DIFFERENT FILES
      C
Ø126          DO 2ØØ IFNUM=1,NFILES
Ø127          IFIL=IFNUM
Ø128          IF(INFLG.NE.Ø)GOTO 21Ø
      C
      C TAPE INPUT HANDLING
      C
Ø13Ø          IF(IENTER(IDCH,FSPECR,NBLKR).LT.Ø)STOP'ENTER ERROR'
Ø132          ITRY=1
      C
      C EOT CHECK
      C
Ø133      225 IF(ITRY.GT.3)GOTO 227
Ø135          IF(IEOTR.GE.Ø)GOTO 22Ø
Ø137      227 WRITE(7,18ØØ)TPDRR,IFIL
Ø138     18ØØ FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
Ø139          WRITE(7,18Ø1)
Ø14Ø     18Ø1 FORMAT(' ENTER NEW READ DRIVE NO:',$)
Ø141          READ(5,18Ø2)TPDRR
Ø142     18Ø2 FORMAT(I1)
Ø143          IEOTR=Ø
Ø144          IF(TPDRR.GT.2)STOP' EOT TERMINATION'
      C
      C READ FROM TAPE
      C
Ø146      22Ø CALL TAPRED(-1,TPDRR,ISTAT,ITLEN,IFLEN,IFIL,IEOTR)
Ø147          IF(ISTAT.LT.Ø)WRITE(2,15ØØ)IFIL
Ø149     15ØØ FORMAT(' RETRIES ON READ FAILED ON FILE',I5)
Ø15Ø          IF(IEOTR.LT.Ø)GOTO 23Ø
      C
      C WIND OVER EOF MARK
```

```
C
      CALL TAPRED(Ø,TPDRR,ISTAT, , ,IFIL,IEOTR)
  23Ø CALL IWAIT(IDCH)
      IERR=Ø
      ITRY=ITRY+1
      IF(IREADW(1,IERR,Ø,IDCH).LT.Ø)STOP'ERR READ ERR'
      IF(IERR.EQ."177777)GOTO 225
      CALL CLOSEC(IDCH)
C
C OPEN UP READING FILES
C
  21Ø IF(INFLG.NE.Ø)FSPECR=FNAMR(IFNUM)
      IF(LOOKUP(IDCH,FSPECR).LT.Ø)STOP'LOOKUP ERR'
C
C OPEN UP OUTPUT FILES
C
      IF(OUTFLG.NE.Ø)FSPECW=FNAMO(IFNUM)
      IF(IENTER(IDCH1,FSPECW,NBLKW).LT.Ø)STOP'ENTER ERR2'
C
C HEADER BLOCK MANAGEMENT
C
      IF(IREADW(256,IHBLK,Ø,IDCH).LT.Ø)STOP'READW ERR'
C
C PUT IN CORRECT ORDER OF PROCESSING
C
      IBFREE=IHBLK(24)
      IHBLK(129+IBFREE)=1
      IBFREE=IBFREE+1
      IHBLK(129+IBFREE)=NPROC
      IBFREE=IBFREE+1
      DO 215 J=1,NPROC
      IHBLK(129+IBFREE)=IORD(J)
      IBFREE=IBFREE+1
  215 CONTINUE
      IHBLK(24)=IBFREE
C
C WRITE OUT UPDATED HEADER
C
      IF(IWRITW(256,IHBLK,Ø,IDCH1).LT.Ø)STOP'WRITW ERR'
      JBLK=1
      DO 3ØØ ICHNUM=1,NCHAN
      IF(IREADW(2*NSAMP,SEISM,JBLK,IDCH).LT.Ø)STOP'READW ERR2'
      CALL APPUT(SEISM,Ø,NSAMP,2)
      CALL APWD
      DO 4ØØ IPCNUM=1,NPROC
      GOTO(41Ø,42Ø,43Ø,44Ø,45Ø,46Ø,47Ø,48Ø,
     %49Ø,5ØØ,51Ø)IORD(IPCNUM)
C
C PROCESS EDIT COMMANDS
C
  41Ø IF(IFED.GT.NFILED)GOTO 4ØØ
      IF(IFILED(IFED).NE.IFIL)GOTO 4ØØ
      IF(ICHAND(IFED).NE.ICHNUM)GOTO 4ØØ
      IFED=IFED+1
```

```
      CALL VCLR(0,1,NSAMP)
      CALL APWR
      CALL APGET(SEISM,0,NSAMP,2)
      CALL APWD
      GOTO 310
C
C POLARITY REVERSAL
C
  420 DO 421 IP=1,NCHANP
      IF(ICHANP(IP).EQ.ICHNUM)GOTO 425
  421 CONTINUE
      GOTO 400
  425 CALL VNEG(0,1,0,1,NSAMP)
      CALL APWR
      GOTO 400
C
C AMP RECOVERY FILTERS APPLICATION AND REMOVAL
C
C
C EXP(0.2T) FILTER
C
  430 IAPL=IAPLX
      CALL APPUTA(NSAMP,NSAMP,2,AEXPT)
      GOTO 455
C
C T*EXP(0.2T) FILTER
C
  440 IAPL=IAPLTX
      CALL APPUTA(NSAMP,NSAMP,2,ATEXPT)
      GOTO 455
C
C TV**2 FILTER
C
  450 IAPL=IAPLTV
      CALL APPUTA(NSAMP,NSAMP,2,ATVSQ)
C
C COMMON CODE
C
  455 CALL APWD
      IF(IAPL.EQ.0)CALL VMUL(0,1,NSAMP,1,0,1,NSAMP)
      IF(IAPL.NE.0)CALL VDIV(NSAMP,1,0,1,0,1,NSAMP)
      CALL APWR
      GOTO 400
C
C MUTE APPLICATION
C
  460 CALL APPUTA(NSAMP,NTAP,2,ATAP)
      CALL APWD
      LMUT=MUTE(ICHNUM)
      CALL VCLR(0,1,LMUT)
      CALL VMUL(LMUT,1,NSAMP,1,LMUT,1,NTAP)
      CALL APWR
      LMUT=MUTET(ICHNUM)
      IF(LMUT.GE.NSAMP)GOTO 400
```

```
            NMUT=NSAMP-LMUT
            CALL VCLR(NSAMP-1,-1,NMUT)
            CALL VMUL(LMUT,-1,NSAMP,1,LMUT,-1,NTAP)
            CALL APWR
            GOTO 400
C
C DECON
C
        470 CALL SPIKE(NSAMP,NSAMP2,NFILT,WHITE,INORM,ISPIKE)
            GOTO 400
C
C BANDPASS FILTER
C
        480 CALL APPUTA(4100,NBFILT,2,ATBP)
            CALL APWD
            CALL VCLR(NSAMP,1,NBEXP)
            CALL RFFT(0,NTRANF,+1)
            CALL RFFTSC(0,NTRANF,3,1)
            CALL VMUL(0,2,4100,1,0,2,NBFILT)
            CALL VMUL(1,2,4100,1,1,2,NBFILT)
            CALL RFFTSC(0,NTRANF,-3,0)
            CALL RFFT(0,NTRANF,-1)
            CALL APWR
            GOTO 400
C
C BANDREJECT FILTER
C
        490 CALL APPUTA(4100,NRFILT,2,ATBR)
            CALL APWD
            CALL VCLR(NSAMP,1,NBEXP)
            CALL RFFT(0,NTRANF,1)
            CALL RFFTSC(0,NTRANF,3,1)
            CALL VMUL(0,2,4100,1,0,2,NRFILT)
            CALL VMUL(1,2,4100,1,1,2,NRFILT)
            CALL RFFTSC(0,NTRANF,-3,0)
            CALL RFFT(0,NTRANF,-1)
            CALL APWR
            GOTO 400
C
C PREDICTION ERROR FILTER
C
        500 CALL PRDICT(NSAMP,NSAMP2,NPFILT,PRWHIT,IPNORM,NLAG)
            GOTO 400
C
C NORMALISATION
C
        510 IF(NRMFLG.EQ.0)GOTO 515
            CALL MAXMGV(0,1,2050,NSAMP)
            GOTO 516
        515 CALL SVESQ(0,1,2050,NSAMP)
            CALL VSQRT(2050,1,2050,1,1)
        516 CALL VDIV(2050,0,0,1,0,1,NSAMP)
            CALL APWR
            GOTO 400
```

```
  400 CONTINUE
C
C END OF PROCESS LOOP
C
      CALL APWAIT
      CALL APGET(SEISM,0,NSAMP,2)
      CALL APWD
  310 IF(IWRITW(2*NSAMP,SEISM,JBLK,IDCH1).LT.0)STOP'WRITW ERR2'
      JBLK=JBLK+NBLKTR
  300 CONTINUE
C
C END OF CHANNEL LOOP
C
      CALL CLOSEC(IDCH)
      CALL CLOSEC(IDCH1)
C
C OUTPUT TO TAPE
C
      IF(OUTFLG.NE.0)GOTO 200
      IFLEN=LOOKUP(IDCH1,FSPECW)
      IF(IFLEN.LT.0)STOP'LOOKUP ERR3'
      CALL TAPRED(1,TPDRW,ISTAT,ITLEN,IFLEN,IFIL,IEOTW)
      CALL CLOSEC(IDCH1)
C
C CHECK FOR ERRORS
C
      IF(IEOTW.GE.0)GOTO 250
      WRITE(7,1600)TPDRW,IFIL
 1600 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
      WRITE(7,1601)
 1601 FORMAT(' ENTER DRIVE NO FOR NEW WRITE TAPE:',$)
      READ(5,1802)TPDRW
      IEOTW=0
      IF(TPDRW.GT.2)STOP ' EOT WRITE TERMINATION'
  250 IF(ISTAT.GE.0)GOTO 200
      WRITE(2,1700) IFIL
 1700 FORMAT(' FATAL ERROR ON WRITE FILE NO',I5)
      STOP' WRITE ERROR'
  200 CONTINUE
      CALL CLOSEC(IDCH)
      CALL CLOSEC(IDCH1)
      STOP'NORMAL TERMINATION'
      END
```

```
FORTRAN IV      V02.04      THU 08-JAN-81 00:25:28                PAGE 001

      SUBROUTINE TEXRMP(FILT,IFTYP,IFLG,NSAMP,CONST,AFILT,FS)
C
C IF ITYP=0 THIS ROUTINE PRODUCES AN EXP(0.2T) ARRAY
C     NSAMP LONG IN FILT
C IF ITYP=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
      VIRTUAL FILT(2048)
      DIMENSION CONST(3)
      IF(IFLG.NE.0)GOTO 10
C
C FORM THE T RAMP
C
      CALL APWAIT
      CALL APPUT(CONST,8189,3,2)
      CALL APWD
      CALL VCLR(0,1,NSAMP)
      CALL VRAMP(8189,8191,0,1,NSAMP)
C
C FORM EXP(0.2T)
C
      CALL VSMUL(0,1,8190,NSAMP,1,NSAMP)
      CALL VEXP(NSAMP,1,NSAMP,1,NSAMP)
      IF(IFTYP.EQ.0)GOTO 20
C
C FORM T*EXP(0.2T)
C
   10 CALL VMUL(NSAMP,1,0,1,NSAMP,1,NSAMP)
   20 CALL APWR
      CALL APGETA(NSAMP,NSAMP,2,AFILT)
      RETURN
      END
```

```
FORTRAN IV      V02.04      THU 08-JAN-81 00:26:11                PAGE 001

      SUBROUTINE COTAP(TAPER,NTAP,ATAP)
C
C THIS ROUTINE PRODUCES A COSINE TAPER NTAP
C SAMPLES LONG
C
      VIRTUAL TAPER(512)
      CALL APWAIT
      CALL APPUT(1.0/FLOAT(NTAP),0,1,2)
      CALL VCLR(1,1,NTAP)
      CALL VTSADD(1,1,2306,1,1,1)
      CALL VTSMUL(0,1,2306,0,1,1)
      CALL VRAMP(1,0,0,1,NTAP)
      CALL VCOS(0,1,0,1,NTAP)
      CALL VTSADD(0,1,2049,0,1,NTAP)
      CALL VTSMUL(0,1,2327,0,1,NTAP)
      CALL APWR
      CALL APGETA(0,NTAP,2,ATAP)
      CALL APWD
      RETURN
      END
```

```
      SUBROUTINE TVRMP(FILT,TØLYR,VLYR,NLYR,IFLG,NSAMP,NSTART,
     %CONST,FSAMP,AFILT)
C
C THIS ROUTINE PRODUCES A TV**2 RAMP
C FROM VELOCITY INFO IN TØLYR,VLYR
C
      VIRTUAL FILT(2Ø48)
      DIMENSION TØLYR(2Ø),VLYR(2Ø),CONST(3)
C
C CHECK IF T RAMP ALREADY FORMED
C
      IF(IFLG.NE.Ø)GOTO 1Ø
      CALL APWAIT
      CALL APPUT(CONST,8189,3,2)
      CALL APWD
      CALL VCLR(Ø,1,NSAMP)
      CALL VRAMP(8189,8191,Ø,1,NSAMP)
C
C FORM VELOCITY RAMP IN FILT
C
   1Ø N1=1
      N2=IFIX(FSAMP*TØLYR(1))-NSTART
      V1=VLYR(1)
      DO 15 I=N1,N2
   15 FILT(I)=V1
      IF(NLYR.EQ.1)GOTO 4Ø
      DO 2Ø J=2,NLYR
      N1=N2+1
      N2=IFIX(FSAMP*TØLYR(J))-NSTART
      DELV=(VLYR(J)-VLYR(J-1))/(N2-N1+2)
      V=VLYR(J-1)
      DO 3Ø I=N1,N2
      FILT(I)=V
      V=V+DELV
   3Ø CONTINUE
   2Ø CONTINUE
   4Ø N1=N2+1
      N2=NSAMP
      VN=VLYR(NLYR)
      DO 5Ø I=N1,N2
   5Ø FILT(I)=VN
C
C PUT V RAMP IN AP AND FORM TV**2
C
      CALL APWAIT
      CALL APPUTA(NSAMP,NSAMP,2,AFILT)
      CALL APWD
      CALL VSQ(NSAMP,1,NSAMP,1,NSAMP)
      CALL VMUL(Ø,1,NSAMP,1,NSAMP,1,NSAMP)
      CALL APWR
      CALL APGETA(NSAMP,NSAMP,2,AFILT)
      CALL APWD
      RETURN
      END
```

```
      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
      VIRTUAL BPASS(2049)
C
C SET UP CONSTANTS
C
      N1=2*F1*DFI
      N2=2*F2*DFI
      N3=2*F3*DFI
      N4=2*F4*DFI
      NFILT=NSAMP+1
      NTP1=N2-N1
      NTP2=N4-N3
      NOK=N3-N2
      CALL APWAIT
C
C SET UP THE FILTER IN THE AP
C
      DO 10 I=1,2
      NTAP=NTP1
      IF(I.EQ.2)NTAP=NTP2
      RNTAP=1.0/FLOAT(NTAP)
      CALL APPUT(RNTAP,0,1,2)
      CALL VCLR(1,1,NTAP)
      CALL VTSADD(1,1,2306,1,1,1)
      CALL VTSMUL(0,1,2306,0,1,1)
      CALL VRAMP(1,0,0,1,NTAP)
      CALL VCOS(0,1,0,1,NTAP)
      CALL VTSADD(0,1,2049,0,1,NTAP)
      CALL VTSMUL(0,1,2327,0,1,NTAP)
      IF(I.EQ.1)CALL VMOV(0,1,2050,1,NTAP)
      IF(I.EQ.2)CALL VMOV(0,1,4100,1,NTAP)
   10 CONTINUE
C
C NOW HAVE TAPERS FORM REST OF FILTER
C
      CALL VCLR(0,1,NFILT)
      CALL VADD(N1,1,2050,1,N1,1,NTP1)
      CALL VADD(N4,-1,4100,1,N4,-1,NTP2)
      CALL VTSADD(N2,1,2049,N2,1,NOK)
      CALL APWR
      CALL APGETA(0,NFILT,2,ATBP)
      CALL APWD
      RETURN
      END
```

```
ØØØ1        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
ØØØ2        VIRTUAL BRJCT(2Ø49)
      C
      C SET UP CONSTANTS
      C
ØØØ3        N1=2.Ø*F1*DFI
ØØØ4        N2=2.Ø*F2*DFI
ØØØ5        NTAP=N2-N1
ØØØ6        CALL APWAIT
      C
      C SET UP FILTER IN AP
      C
ØØØ7        CALL VCLR(Ø,1,NTAP)
ØØØ8        NFILT=NSAMP+1
ØØØ9        FTAP=1.Ø/FLOAT(NTAP)
ØØ1Ø        CALL APPUT(FTAP,1,1,2)
ØØ11        CALL APWD
ØØ12        CALL VTSMUL(1,1,2317,1,1,1)
ØØ13        CALL VRAMP(Ø,1,Ø,1,NTAP)
ØØ14        CALL VCOS(Ø,1,Ø,1,NTAP)
ØØ15        CALL VTSADD(Ø,1,2Ø49,Ø,1,NTAP)
ØØ16        CALL VTSMUL(Ø,1,2327,Ø,1,NTAP)
ØØ17        CALL VMOV(Ø,1,4Ø96,1,NTAP)
      C
      C SET UP FULL FILTER NOW TAPER FINISHED
      C
ØØ18        CALL VCLR(Ø,1,NFILT)
ØØ19        CALL VTSADD(Ø,1,2Ø49,Ø,1,NFILT)
ØØ2Ø        CALL VMUL(N1,1,4Ø96,1,N1,1,NTAP)
ØØ21        CALL APWR
ØØ22        CALL APGETA(Ø,NFILT,2,ATBR)
ØØ23        CALL APWD
ØØ24        RETURN
ØØ25        END
```

```
      SUBROUTINE SPIKE(NSAMP,NSAMP2,ILENTH,WHITE,IFLAG,ISPIKE)
C
C WIENER SPIKING FILTER ROUTINE
C  NSAMP=DATA LENGTH
C  NSAMP2=NEAREST POWER OF 2 TO NSAMP
C  ILENTH=FILTER LENGTH
C  IFLAG=TRACE NORMALISATION FLAG
C        Ø NO NORMALISATION
C        1 FILTER UNIT ENERGY
C        2 EQUAL INPUT-OUTPUT ENERGY
C  ISPIKE SPIKE POSITION
C
      NTRAN=2*NSAMP2
      NCLR=NTRAN-NSAMP
      NFCLR=NTRAN-ILENTH
      NTRAN2=NTRAN+2
      NM1=NSAMP2-1
      I6=NTRAN+ILENTH
      I7=I6+ILENTH
      I8=I7+ILENTH
      ISP=I6+ISPIKE
C
C HAVING SET UP CONSTANTS GET INPUT TRACE
C ENERGY IF REQD FOR NORMALISATION
C
      IF(IFLAG.NE.2)GOTO 1Ø
      CALL SVESQ(Ø,1,8191,NSAMP)
      CALL VSQRT(8191,1,8191,1,1)
      CALL APWR
      CALL APGET(EN,8191,1,2)
      CALL APWD
C
C GET AUTOCORRELATION FUNCTION
C
   1Ø CALL VCLR(NSAMP,1,NCLR)
      CALL RFFT(Ø,NTRAN,1)
      CALL VMOV(Ø,1,NTRAN,1,NTRAN)
      CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
      CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
      CALL VCLR(NTRAN+3,2,NM1)
      CALL RFFTSC(NTRAN,NTRAN,-1,-1)
      CALL RFFT(NTRAN,NTRAN,-1)
C
C AUTO FUNCTION NOW 2N LONG FROM NTRAN
C ORIGINAL FUNCTION TRANSFORMED Ø-NTRAN
C
C NOW WHITEN
C
      CALL APWR
      CALL APPUT(WHITE,8191,1,2)
      CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)
C
C SET UP SPIKE CC FUNCTION
C
```

```
      CALL VCLR(I6,1,ILENTH)
      CALL VTSADD(ISP,1,2Ø49,ISP,1,1)
C
C SOLVE EQNS
C
      CALL WIENER(ILENTH,NTRAN,I6,I7,I8,1)
      CALL APCHK(IER)
      IF(IER.NE.Ø)STOP'LEVINSON FAILURE'
      CALL VMOV(I7,1,NTRAN,1,ILENTH)
C
C NORMALISE FILTER IF ASKED FOR
C
      IF(IFLAG.NE.1)GOTO 2Ø
      CALL SVESQ(NTRAN,1,I6,ILENTH)
      CALL VSQRT(I6,1,I6,1,1)
      CALL VDIV(I6,Ø,NTRAN,1,NTRAN,1,ILENTH)
C
C APPLY FILTER
C
   2Ø CALL VCLR(I6,1,NFCLR)
      CALL RFFT(NTRAN,NTRAN,1)
      CALL VMUL(Ø,1,NTRAN,1,Ø,1,2)
      CALL CVMUL(2,2,NTRAN2,2,2,2,NM1,1)
      CALL RFFTSC(Ø,NTRAN,Ø,-1)
      CALL RFFT(Ø,NTRAN,-1)
C
C DO SCALING IF REQD
C
      IF(IFLAG.NE.2)GOTO 3Ø
      CALL SVESQ(Ø,1,8191,NSAMP)
      CALL VSQRT(8191,1,8191,1,1)
      CALL APWR
      CALL APPUT(EN,819Ø,1,2)
      CALL APWD
      CALL VDIV(8191,1,819Ø,1,819Ø,1,1)
      CALL VSMUL(Ø,1,819Ø,Ø,1,NSAMP)
   3Ø CALL APWR
      RETURN
      END
```

```
      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=Ø NO NORMALISATION
C         =1 FILTER UNIT ENERGY
C         =2 INPUT/OUTPUT TRACE ENERGY CONSTANT
C    NLAG= LAG OFFSET OF PREDICTION
C
      NTRAN=2*NSAMP2
      NCLR=NTRAN-NSAMP
      NM1=NSAMP2-1
      NTRAN2=NTRAN+2
      NLG=NTRAN+NLAG
      NFILT=ILENTH+NLAG
      I7=NLG+ILENTH
      I8=I7+ILENTH
C
C GET INPUT TRACE ENERY
C
      IF(IFLAG.NE.2)GOTO 1Ø
      CALL SVESQ(Ø,1,8191,NSAMP)
      CALL VSQRT(8191,1,8191,1,1)
      CALL APGET(EN,8191,1,2)
      CALL APWD
C
C GET AUTOCORRELATION FUNCTION
C
   1Ø CALL VCLR(NSAMP,1,NCLR)
      CALL RFFT(Ø,NTRAN,1)
      CALL VMOV(Ø,1,NTRAN,1,NTRAN)
      CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
      CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
      CALL VCLR(NTRAN+3,2,NM1)
      CALL RFFTSC(NTRAN,NTRAN,-1,-1)
      CALL RFFT(NTRAN,NTRAN,-1)
C
C NOW WHITEN IT
C
      CALL APWR
      CALL APPUT(WHITE,8191,1,2)
      CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)
C
C NOW SOLVE EQNS
C
      CALL WIENER(ILENTH,NTRAN,NLG,I7,I8,1)
      CALL APCHK(IER)
      IF(IER.NE.Ø)STOP'LEVINSON FAILURE'
      IF(IFLAG.NE.1)GOTO 2Ø
      CALL SVESQ(I7,1,I8,ILENTH)
```

```
ØØ34          CALL VSQRT(I8,1,I8,1,1)
ØØ35          CALL VDIV(I8,Ø,I7,1,I7,1,ILENTH)
      C
      C APPLY FILTER
      C
ØØ36       2Ø CALL VCLR(NTRAN,1,NFILT)
ØØ37          CALL VTSADD(NTRAN,1,2Ø49,NTRAN,1,1)
ØØ38          CALL VSUB(I7,1,NLG,1,NLG,1,ILENTH)
ØØ39          CALL VCLR(I7,1,NTRAN-NFILT)
ØØ4Ø          CALL RFFT(NTRAN,NTRAN,1)
ØØ41          CALL VMUL(Ø,1,NTRAN,1,Ø,1,2)
ØØ42          CALL CVMUL(2,2,NTRAN2,2,2,2,NM1,1)
ØØ43          CALL RFFTSC(Ø,NTRAN,Ø,-1)
ØØ44          CALL RFFT(Ø,NTRAN,-1)
      C
      C DO SCALING
      C
ØØ45          IF(IFLAG.NE.2)GOTO 3Ø
ØØ47          CALL SVESQ(Ø,1,8191,NSAMP)
ØØ48          CALL VSQRT(8191,1,8191,1,1)
ØØ49          CALL APWR
ØØ5Ø          CALL APPUT(EN,819Ø,1,2)
ØØ51          CALL APWD
ØØ52          CALL VDIV(8191,1,819Ø,1,819Ø,1,1)
ØØ53          CALL VSMUL(Ø,1,819Ø,Ø,1,NSAMP)
ØØ54       3Ø CALL APWR
ØØ55          RETURN
ØØ55          END
```

```
ØØØ1        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,Ø 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
ØØØ2        INTEGER*2 MASK(8),ESTATI
ØØØ3        LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
           %IFLEN,ESTAT,ERRS(8)
ØØØ4        DATA MASK/"1,"2,"4,"1Ø,"2Ø,"4Ø,"1ØØ,"2ØØ/
ØØØ5        DATA SDSCOM/"Ø,"1,"2,"3,"4,"5,"6,"7/
ØØØ6        DATA ERRS/"377,"377,"377,"377,"377,"377,"377,"377/
ØØØ7        ITRY=Ø
ØØØ8        IF(ICOM) 1Ø,3Ø,2Ø
      C
      C SECTION CONTROLLING A READ
      C
      C
      C  CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
      C
ØØØ9     1Ø ITRY=ITRY+1
      C
      C SET UP COMMAND FOR READ
      C
ØØ1Ø        COM(1)=SDSCOM(4)
ØØ11        COM(2)=1
ØØ12        COM(3)=IDRV
ØØ13        COM(4)=-1
ØØ14        CALL SDS1Ø(COM,ISTAT,ITLEN,ILEN)
ØØ15        IF(ISTAT.EQ.Ø)RETURN
      C
      C ERROR DETECTED ON READ
      C
ØØ17        ISTATI=ISTAT
ØØ18        GOTO 4Ø
      C
      C IF SHORT RECORD FOUND REREAD TAPE
      C
ØØ19     5Ø ITMP=ISTATI.AND.MASK(6)
ØØ2Ø        IF(ITMP.NE.Ø)GOTO 1Ø
ØØ22        ITMP=ISTATI.AND.MASK(2)
ØØ23        IF(ITMP.EQ.Ø)RETURN
      C
      C IF CRC ERROR FOUND REWIND TAPE AND RETRY
      C
ØØ25        WRITE(2,2Ø1Ø)IFNUM
ØØ26   2Ø1Ø FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
ØØ27        IF(ITRY.GE.2)GOTO 13Ø
ØØ29        ECOM(1)=SDSCOM(6)
ØØ3Ø        ECOM(2)=1
```

```
ØØ31          ECOM(3)=IDRV
ØØ32          ECOM(4)=Ø
ØØ33          CALL SDS1Ø(ECOM,ESTAT, , )
ØØ34          GOTO 1Ø
        C
        C WRITE SECTION
        C
ØØ35       2Ø ITRY=ITRY+1
ØØ36          IF(ITRY.GT.3)GOTO 13Ø
ØØ37          COM(1)=SDSCOM(7)
ØØ38          IFLEN=(ILEN+3)/4
ØØ4Ø          COM(2)=IFLEN
ØØ41          COM(3)=IDRV
ØØ42          COM(4)=1
ØØ43          CALL SDS1Ø(COM,ISTAT, , )
ØØ44          IF(ISTAT.EQ.Ø)RETURN
        C
        C WRITE ERROR DETECTED
        C
ØØ46          ISTATI=ISTAT
ØØ47          GOTO 4Ø
ØØ48       7Ø ITMP=ISTATI.AND.MASK(6)
ØØ49          ITMPI=ISTATI.AND.MASK(2)
ØØ5Ø          IF(ITMP.EQ.Ø.AND.ITMPI.EQ.Ø)RETURN
        C
        C REPORT AND RETRY
        C
ØØ52          WRITE(2,2Ø2Ø)IFNUM
ØØ53     2Ø2Ø FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
ØØ54          ECOM(1)=SDSCOM(6)
ØØ55          ECOM(2)=2
ØØ56          ECOM(3)=IDRV
ØØ57          ECOM(4)=Ø
ØØ58          CALL SDS1Ø(ECOM,ESTAT, , )
ØØ59          NBUF=8
ØØ6Ø          IFLENE=16
ØØ61          IPAD=3276Ø
ØØ62          CALL TWRIT(ERRS,NBUF,ESTAT,IPAD,IFLENE,IDRV)
ØØ63          GOTO 2Ø
        C
        C WIND FOWARD ONE FILE
        C
ØØ64       3Ø COM(1)=SDSCOM(5)
ØØ65          COM(2)=1
ØØ66          COM(3)=IDRV
ØØ67          COM(4)=Ø
ØØ68          CALL SDS1Ø(COM,ISTAT, , )
        C
        C CLEAR IRRELEVANT BITS FROM ERROR BYTE
        C
ØØ69          ISTAT=ISTAT.AND..NOT.MASK(6)
ØØ7Ø          IF(ISTAT.EQ.2)RETURN
ØØ71          ISTATI=ISTAT
ØØ72          IF(ISTAT.NE.Ø)GOTO 4Ø
```

```
C
C IF ISTAT=Ø 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)=Ø
            CALL SDS1Ø(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
       4Ø   WRITE(2,1Ø1Ø)ISTATI,IFNUM
     1Ø1Ø   FORMAT(' STATUS=',I3,' FILE NO=',I4)
            ISTATI=ISTATI.AND..NOT.MASK(4)
            ITMP=ISTATI.AND.MASK(5)
            IF(ITMP.EQ.Ø)GOTO 8Ø
       9Ø   ECOM(1)=SDSCOM(1)
            ECOM(2)=Ø
            ECOM(3)=IDRV
            ECOM(4)=Ø
            CALL SDS1Ø(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.Ø)GOTO 9Ø
            IF(ICOM) 1Ø,3Ø,2Ø
C
C TAPE OFFLINE
C
       3Ø   ITMP=ISTATI.AND.MASK(1)
            IF(ITMP.EQ.Ø)GOTO 1ØØ
            TYPE 1ØØ1,IDRV
     1ØØ1   FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
      11Ø   ECOM(1)=SDSCOM(1)
            ECOM(2)=Ø
            ECOM(3)=IDRV
            ECOM(4)=Ø
            CALL SDS1Ø(ECOM,ESTAT, , )
            ESTATI=ESTAT
            ITMP=ESTATI.AND.MASK(1)
            IF(ITMP.NE.Ø)GOTO 11Ø
```



```
            IF(ICOM) 1Ø,3Ø,2Ø
C
C EOT
C
      1ØØ   ITMP=ISTATI.AND.MASK(3)
            IF(ITMP.EQ.Ø)GOTO 12Ø
            TYPE 1ØØ2,IDRV
     1ØØ2   FORMAT(' EOT ON DRIVE ',I1)
            IEOT=-1
            RETURN
      12Ø   IF(ICOM) 5Ø,35,7Ø
C
C ERROR EXIT RETURN
C
      13Ø   ISTATI=-1
            RETURN
            END
```

Interactive Spectral Analysis Program :- MPFANL

MPFANL is totally interactive, and it expects the input data to be in a disc file in the processing system internal format.

Plots are produced on the electrostatic plotter by running RASM after this program terminates.

Input Parameters

NSAMP(I4)...Number of samples/trace

NCHAN(I2)...Number of channels to analyse

ICHAN(I),I=1,NCHAN(I3)...Channel numbers of those to be analysed

TSEC(F10.0)...Sampling period of data in seconds.

FNAMR(3A4)....Name of data file containing the data to be analysed

```
      REAL*8 FSPECR
      REAL*4 SEIS(2060),FNAMR(3)
      INTEGER*2 ICHAN(24)
      DATA DEV/3RRK /
C
C THIS IS A SPECTRAL ANALYSIS PROGRAM
C
      WRITE(7,1000)
 1000 FORMAT(' ENTER NO OF SAMPLES(I4):',$)
      READ(5,1001)NSAMP
 1001 FORMAT(I4)
      WRITE(7,1002)
 1002 FORMAT(' ENTER NO OF CHANNELS(I2):',$)
      READ(5,1007)NCHAN
 1007 FORMAT(I2)
      WRITE(7,1008)
 1008 FORMAT(' ENTER CHANNELS REQD IN ORDER(12I3)',/,' >',$)
      READ(5,1009)(ICHAN(I),I=1,NCHAN)
 1009 FORMAT(12I3)
      WRITE(7,1003)
 1003 FORMAT(' ENTER SAMPLING PERIOD(F10.0):',$)
      READ(5,1004)TSEC
 1004 FORMAT(F10.0)
      WRITE(7,1005)
 1005 FORMAT(' ENTER FILE NAME:',$)
      READ(5,1006)FNAMR
 1006 FORMAT(3A4)
      CALL IRAD50(12,FNAMR,FSPECR)
C
C SET UP READ IN
C
      ICHR=IGETC()
      IF(IFETCH(DEV).NE.0)STOP' FETCH ERR'
      IF(LOOKUP(ICHR,FSPECR).LT.0)STOP'LOOKUP ERR'
      NWDS=2*NSAMP
      NBLKS=NSAMP/128
C
C PROCESS FIRST CHANNEL
C
      NBLKST=(ICHAN(1)-1)*NBLKS+1
      IF(IREADW(NWDS,SEIS,NBLKST,ICHR).LT.0)STOP'READ ERR'
      IFLAG=+1
      CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(1))
      IF(NCHAN.EQ.1)GOTO 10
C
C DO MOST OF CHANNELS
C
      DO 20 J=2,NCHAN
      NBLKST=(ICHAN(J)-1)*NBLKS+1
      IF(IREADW(NWDS,SEIS,NBLKST,ICHR).LT.0)STOP'READW ERR'
      IFLAG=-1
      CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(J))
   20 CONTINUE
   10 IFLAG=0
```



```
      CALL SPEC(SEIS,NSAMP,IFLAG,TSEC,ICHAN(1))
      STOP' NORMAL TERMINATION'
      END
```

```
      SUBROUTINE SPEC(AVAL,INUM,IPLOT,TSEC,ICHAN)
      DIMENSION AVAL(1)
      DATA MASK/O1Ø421Ø/
C*************************************************************
C*************************************************************
C       AVAL=SOURCE DATA TO BE PADDED TO NEAREST 2**N
C       INUM=DIMENSION AVAL
C       IPLOT=DISPLAY CONTROL
C                             IPLOT>Ø NEW OR FIRST PLOT
C                             IPLOT<Ø SUBSIDIARY PLOT
C                             IPLOT=Ø TERMINATE PLOT
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
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
C      EXPAND INUM TO 2**N
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      IF(IPLOT.EQ.Ø)GOTO 999
      N=2
      ICNTRL=Ø
  1ØØ CONTINUE
      IF(N.GE.INUM) GOTO 99
      N=N*2
      GOTO 1ØØ
C     N=LOWEST 2**N GT INUM
   99 IF(N.GT.4Ø96) STOP'AP OVERFLOW ON SPECTRUM'
C     THIS FOR AP'S SAKE
      DELF=1.Ø/(N*TSEC)
C     DELF IS RETURNED TRANSFORM FREQUENCY
C     PAD OUT AVAL WITH ZEROS
      IF(INUM.EQ.N) GOTO 1Ø11
      ISTART=INUM+1
      DO 1Ø1 I=ISTART,N
  1Ø1 AVAL(I)=Ø.Ø
 1Ø11 FLIM=Ø.5*DELF*N
C     FLIM=MAX TRANSFORM FREQUENCY
C     SEEK FREQUENCY DECADE OF PLOT
      FBASE=Ø.1
      IBASE=-1
  1Ø2 CONTINUE
      IF(FLIM.LE.FBASE) GOTO 1Ø3
      FBASE=FBASE*1Ø.
      IBASE=IBASE+1
      GOTO 1Ø2
  1Ø3 FBASE=FBASE*Ø.1
      IBASE=IBASE-1
      FMAX=FLIM/FBASE
      IFMAX=IFIX(FMAX)
      IFMAX=IFMAX+1
      FMAX=FLOAT(IFMAX)
C      FMAX=NEXT HIGHEST INTEGER DECADE=1 TO 9
```

```
C       FORM PLOTTING MASK
        YSIZE=9.Ø
        ASPECT=Ø.75
        XSIZE=YSIZE*ASPECT
        XOFST=XSIZE/2Ø.Ø
        YOFST=YSIZE/2Ø.Ø
   1Ø31 CSIZE=YSIZE/8Ø.Ø
        DELX=XSIZE/1ØØ.Ø
        IF(ICNTRL.NE.Ø) DELX=XSIZE/36.Ø
        Y=FMAX*1Ø.Ø
        IY=IFIX(Y)
        DELY=YSIZE/IY
        IF(IPLOT.GT.Ø)       CALL PLOTS(Ø,Ø,Ø)
        IF(IPLOT.LE.Ø)       CALL PLOT(Ø.,Ø.,-999)
        CALL SETMSG(Ø)
        NX=1ØØ
        IF(ICNTRL.NE.Ø) NX=36
        CALL GRID(XOFST,YOFST,NX,DELX,IY,DELY,MASK)
        CALL PLOT(XOFST,YOFST,3)
        CALL NEWPEN(3)
        X=XOFST+XSIZE
        Y=YOFST+YSIZE
        CALL PLOT(X,YOFST,2)
        CALL PLOT(X,Y,2)
        CALL PLOT(XOFST,Y,2)
        CALL PLOT(XOFST,YOFST,2)
        CALL NEWPEN(1)
        CALL PLOT(X,YOFST,-3)
C       RE-ORIGIN THE PLOT
        XINC=XSIZE/1Ø.Ø
        IF(ICNTRL.NE.Ø) XINC=XSIZE/36.Ø
        XNUM=Ø.Ø
        IF(ICNTRL.NE.Ø) XNUM=-18Ø.Ø
        Y=-CSIZE
        IF(ICNTRL.NE.Ø) Y=-4.Ø*CSIZE
        DELX=1.Ø
        IF(ICNTRL.NE.Ø) DELX=1Ø.Ø
        NX=1Ø
        IF(ICNTRL.NE.Ø) NX=37
        X=Ø.Ø
        DO 1Ø4 I=1,NX
        IF(ICNTRL.NE.Ø) GOTO 1Ø41
        CALL PLOT(X,Ø.,3)
        CALL PLOT(X,YSIZE,2)
   1Ø41 CALL NUMBER(X,Y,CSIZE,XNUM,9Ø.Ø,-1)
        XNUM=XNUM+DELX
    1Ø4 X=X-XINC
        IY=IY*Ø.1
        YINC=YSIZE/IY
        IY=IY+1
        YNUM=Ø.Ø
        Y=Ø.Ø
        X=1.5*CSIZE
        XINC=-XSIZE
```

```
      DO 105 I=1,IY
      CALL PLOT(0.0,Y,3)
      CALL PLOT(XINC,Y,2)
      CALL NUMBER(X,Y,CSIZE,YNUM,90.0,-1)
      YNUM=YNUM+1
  105 Y=Y+YINC
      SSIZE=CSIZE*1.5
      X=X+2.0*SSIZE
      Y=YSIZE*0.5-7.0*SSIZE
      CALL SYMBOL(X,Y,SSIZE,12HFREQUENCY*10,90.0,12)
      Y=Y+12.0*SSIZE
      X=X-SSIZE
      DELX=FLOAT(IBASE)
      CALL NUMBER(X,Y,CSIZE,DELX,90.0,-1)
      Y=0.65*YSIZE
      X=X+SSIZE
      SSIZE=SSIZE*1.33
      CALL NEWPEN(4)
      IF(ICNTRL.EQ.0) CALL SYMBOL(X,Y,SSIZE,15HENERGY SPECTRUM,90.0,15)
      IF(ICNTRL.NE.0) CALL SYMBOL(X,Y,SSIZE,15HPHASE  SPECTRUM,90.0,15)
      FEND=FMAX*FBASE
      DFPLOT=YSIZE*DELF/FEND
      IF(ICNTRL.NE.0) GOTO 1051
C     DFPLOT=PLOT UNITS PER FREQUENCY SAMPLE
C*****************************************************
C    PROCESS TIME SERIES
C*****************************************************
      AVAL(N+1)=-XSIZE
      AVAL(N+2)=-XSIZE*7.0/44.0
      AVAL(N+3)=-0.5*XSIZE
      NX=N+3
C     EXPAND AVAL TO CONTAIN DISPLAY CONSTANTS
C     DO AP PROCESSING
      MN=N+2
      NN=MN/2
      LN=MN+1
      NNN=NN-1
      CALL APCLR
      CALL APPUT(AVAL,0,NX,2)
      CALL APWD
      CALL VMOV(N,1,8189,1,3)
C     HOLD DISPLAY CONSTANTS IN HIGH MEMORY
      CALL RFFT(0,N,1)
      CALL RFFTSC(0,N,3,1)
C     NN=COMPLEX TRANSFORM LENGTH
      CALL POLAR(0,2,0,2,NN)
      CALL MAXV(0,2,MN,NN)
C     NORMALISE AMPLITUDE SPECTRUM
      CALL VFILL(MN,LN,1,NNN)
      CALL VDIV(MN,1,0,2,0,2,NN)
C     SCALE SPECTRUM FOR PLOTTING
      CALL VSMUL(0,2,8189,0,2,NN)
C    SCALE PHASE FOR PLOTTING
      CALL VSMUL(1,2,8190,1,2,NN)
```

```
      CALL VSADD(1,2,8191,1,2,NN)
      CALL APWR
      CALL APGET(AVAL,Ø,LN,2)
      CALL APWD
C     AVAL(1,MN)=AMPLITUDE:PHASE
C     AVAL(LN)=SCALING FACTOR FOR AMPLITUDE
C     PLOT AMPLITUDE SPECTRUM
 1Ø31 CALL PLOT(Ø.,Ø.,3)
      CALL NEWPEN(3)
      Y=Ø.Ø
      ISTART=1
      IF(ICNTRL.NE.Ø) ISTART=2
      DO 1Ø6 I=ISTART,MN,2
      CALL PLOT(AVAL(I),Y,2)
  1Ø6 Y=Y+DFPLOT
      IF(ICNTRL.NE.Ø) GOTO 2Ø8
C     MARK IN SCALING FACTOR
      CALL NEWPEN(1)
      X=4.5*CSIZE
      Y=Ø.
C     FLOAT SCALING FOR DISPLAY
      ASCALE=1.Ø
      BSCALE=Ø.
      BINDEX=Ø.
      IF(AVAL(LN).LT.1.Ø) ASCALE=1Ø.Ø
      IF(AVAL(LN).GT.1.Ø) ASCALE=Ø.1
      IF(ASCALE.EQ.1Ø.Ø) BINDEX=-1.Ø
      IF(ASCALE.EQ.Ø.1) BINDEX=1.Ø
  2Ø6 CONTINUE
      IF(AVAL(LN).GE.1.Ø.AND.AVAL(LN).LT.1Ø.Ø) GOTO 2Ø7
      AVAL(LN)=AVAL(LN)*ASCALE
      BSCALE=BSCALE+BINDEX
      GOTO 2Ø6
  2Ø7 CALL SYMBOL(X,Y,CSIZE,21HSCALING FACTOR=   *1Ø,9Ø.Ø,21)
      Y=Y+15.Ø*CSIZE
      CALL NUMBER(X,Y,CSIZE,AVAL(LN),9Ø.Ø,1)
      Y=Y+6.Ø*CSIZE
      DELX=Ø.5*CSIZE
      X=X-DELX
      CALL NUMBER(X,Y,DELX,BSCALE,9Ø.Ø,-1)
      ICNTRL=1
      CALL PLOT(Ø.,Ø.,-999)
      GOTO 1Ø31
  2Ø8 X=4.5*CSIZE
      Y=Ø.Ø
      CALL SYMBOL(X,Y,CSIZE,'CHANNEL NUMBER = ',9Ø.Ø,17)
      Y=Y+16.Ø*CSIZE
      RCHAN=FLOAT(ICHAN)
      CALL NUMBER(X,Y,CSIZE,RCHAN,9Ø.Ø,-1)
      ICNTRL=Ø
  999 IF(IPLOT.EQ.Ø) CALL PLOT(Ø.,Ø.,999)
      RETURN
      END
```

Velocity Analysis :- MPVEL

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

Output file....DK2:ANOUTV.DAT

Input Parameters

```
READ(1,1000)L,NCHAN,NSTART,M

1000 FORMAT(I5)
```

L........Number of Samples per trace

NCHAN....Number of Channels in gather

NSTART...Starting sample number, from time zero

M........Level of interpolation carried out on data

```
READ(1,1200)FBUF

1200 FORMAT(12X,3A4)
```

FBUF.....Input data file name

```
READ(1,1100)XSTART,XSTEP,FSAMP

1100 FORMAT(6F10.0)
```

XSTART...Shot/First receiver offset

XSTEP....Receiver spacing

FSAMP....Sampling frequency, samples/millisecond

READ(1,1100)T01,T02,TOSTEP,TGATE

T01......Start time for analysis, milliseconds

T02......End time for analysis, milliseconds

TOSTEP...Gate step size, milliseconds

TGATE....Semblance calculation gate size, milliseconds

READ(1,1100)VSTEP,V1,V2MIN,V2MAX,VICPT,VGRAD

VSTEP....Velocity step in semblance calculation

V1.......Start velocity in analysis

V2MIN....Minimum value of end velocity

V2MAX....Maximum value of end elocity

VICPT....Intercept on V-axis of line joining V2MIN and V2MAX

VGRAD....Gradient of line joining V2MIN and V2MAX

```
      DIMENSION NMOFS(4Ø96),TØSQ(512),NV(512),VINSQ(17Ø),SMBLCE(17Ø),
     1SØ(1Ø24),S1(1Ø24),XSQ(24),
     2JSTCH(24),JST(24),NWDS(24),NEL(24),JBLK(24),CONST(7),FBUF(3)
      LOGICAL*1 SW
      REAL*8 FSPEC
      DATA K1,K2,KNCHIN,KBLK,KBLKIN,KADD2,KADD1,KHF,KFS,NBUF
     1/1,2,8185,8186,8187,8188,8189,819Ø,8191,1Ø24/
      DATA DEV/3RDK /
C
C ASSIGN INPUT AND OUTPUT CHANNELS
      CALL ASSIGN(1,'DK2:ANINV.DAT',13)
      CALL ASSIGN(2,'DK2:ANOUTV.DAT',14)
      IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
      IDCH=IGETC()
C READ IN REQUIRED INPUT PARAMETERS
      READ(1,1ØØØ) L,NCHAN,NSTART,M
 1ØØØ FORMAT(4I5)
      READ(1,12ØØ) FBUF
      READ(1,11ØØ) XSTART,XSTEP,FSAMP
 11ØØ FORMAT(1ØF1Ø.Ø)
      READ(1,11ØØ) TØ1,TØ2,TØSTEP,TGATE
      READ(1,11ØØ) VSTEP,V1,V2MIN,V2MAX,VICPT,VGRAD
 12ØØ FORMAT(12X,3A4)
C VALIDATE DATA AND CALCULATE REQUIRED ARRAYS AND CONSTANTS.
C FORM ARRAY JSTCH CORRESPONDING TO THE BEGINNING BLOCK
C NUMBERS FOR EACH TRACE ON DISK.
C AT SAME TIME FORM ARRAY XSQ OF SQUARED SHOT RECEIVER SEPARATIONS
      JSTCH(1)=1
      NREAD=L*M/128
      X=XSTART
      XSQ(1)=X**2
      DO 1Ø JCHAN=2,NCHAN
      JSTCH(JCHAN)=JSTCH(JCHAN-1)+NREAD
      X=X+XSTEP
      XSQ(JCHAN)=X**2
 1Ø   CONTINUE
C CALCULATE FSAMPM,THE SAMPLING FREQUENCY AFTER INTERPOLATION
      FSAMPM=FLOAT(M)*FSAMP
C CALCULATE TSAMP THE SAMPLE INTERVAL
      TSAMP=1.Ø/FSAMP
C CALCULATE TSTART,THE BEGINNING TIME
      TSTART=NSTART*TSAMP
C CALCULATE NSTM THE INITIAL SAMPLE NUMBER AFTER INTERPOLATION.
      NSTM=M*NSTART
C TRUNCATE TØSTEP TO BE INTEGRAL MULTIPLE OF TSAMP
      TØSTEP=TSAMP*IFIX(TØSTEP*FSAMP)
C ROUND TGATE TO BE EVEN INTEGRAL MULTIPLE OF TSAMP
C NGATE IS THE NUMBER OF ELEMENTS IN GATE(BEFORE INTERPOLATION)
C NGTM2 IS THE NUMBER IN HALF OF GATE,AFTER INTERPOLATION.
      NGATE=IFIX(TGATE*FSAMP+1.Ø)/2*2
      TGATE=FLOAT(NGATE)*TSAMP
      NGATE=NGATE+1
      NGTM2=M*(NGATE/2)
C ENSURE THAT TØ1 IS A LEAST TSTART+TGATE/2
```

```
      TØ1=AMAX1(TØ1,TSTART+Ø.5*TGATE)
C ROUND TØ1 UP TO BE INTEGRAL MULTIPLE OF TSAMP
      TØ1=TSAMP*(IFIX(TØ1*FSAMP)+1)
C TRUNCATE TØ2 IF NECESSARY SO THAT THERE IS SUFFICIENT
C DATA TO DO VELOCITY ANALYSIS AT TØ2
      TØ2MAX=SQRT((TSTART+(L-1)*TSAMP-Ø.5*TGATE)**2-
     1XSQ(NCHAN)/V1**2)
      TØ2=AMIN1(TØ2,TØ2MAX)
C CALCULATE NTØ,THE NUMBER OF TWO WAY TIMES CONSIDERED
      NTØ=IFIX((TØ2-TØ1)/TØSTEP)+1
      TØ2=TØ1+(NTØ-1)*TØSTEP
      IF(NTØ.LE.Ø)STOP'NTØ NOT POSITIVE'
C FORM ARRAYS TØSQ (SQUARED TWO WAY TIME) AND NV (NUMBER
C OF VELOCITY POINTS AT EACH TWO WAY TIME)
      TØJ=TØ1
      DO 2Ø JTØ=1,NTØ
      TØSQ(JTØ)=TØJ**2
      V2=VICPT+VGRAD*TØJ
      V2=AMAX1(V2MIN,V2)
      V2=AMIN1(V2,V2MAX)
      NV(JTØ)=IFIX((V2-V1)/VSTEP)+1
      TØJ=TØJ+TØSTEP
2Ø    CONTINUE
C VINSQ IS THE ARRAY OF INVERSE SQUARED VELOCITIES
      NVMX=NV(NTØ)
      IF(NVMX.GT.17Ø)STOP'NVMX GT 17Ø'
      V=V1
      DO 3Ø JV=1,NVMX
      VINSQ(JV)=1.Ø/V**2
      V=V+VSTEP
3Ø    CONTINUE
C NOW CALCULATE NWMX,AN UPPER BOUND ON THE SIZE OF THE
C DATA WINDOW.IF THIS EXCEEDS NBUF PROGRAM STOPS.
      NWMX=IFIX(FSAMPM*SQRT(TØSQ(1)+XSQ(NCHAN)*VINSQ(1))+Ø.5)
     1-IFIX(FSAMPM*SQRT(TØSQ(1)+XSQ(NCHAN)*VINSQ(NVMX))+Ø.5)+2*NGTM2+1
      IF(NWMX.GT.NBUF-127)STOP'NWMX TOO LARGE'
C CONST IS ARRAY OF CONSTANTS USED IN AP
      CONST(1)=1.Ø/FLOAT(NCHAN)
      CONST(2)=-128.
      CONST(3)=1.Ø/128.
      CONST(4)=FLOAT(NGTM2-NSTM+1)
      CONST(5)=-FLOAT(NGTM2+NSTM)
      CONST(6)=Ø.5
      CONST(7)=FSAMPM
C INITIALISE AP AND PLACE CONSTANTS IN TOP 7 LOCATIONS
      CALL APINIT
      CALL APWR
      CALL APPUT(CONST,KNCHIN,7,K2)
C CALCULATE THOSE ADDRESSES WHICH ARE CONSTANT IN NMO
C AND SEMBLANCE COMPUTATIONS.
      IVINSQ=NCHAN+1
      ISSQ=NWMX
      IEN=ISSQ+1
C LOOKUP FILE TO BE READ FROM
```

```
      CALL IRAD5Ø(12,FBUF,FSPEC)
      IF(LOOKUP(IDCH,FSPEC).LT.Ø)STOP'LOOKUP ERROR'
      WRITE(2) L,NCHAN,NSTART,M
      WRITE(2) XSTART,XSTEP,FSAMP
      WRITE(2) TØ1,TØ2,TØSTEP,TGATE
      WRITE(2) VSTEP,V1,V2MIN,V2MAX
      WRITE(2) NTØ
C=====================================================================
      DO 5ØØ JTØ=1,NTØ
      NVD=NV(JTØ)
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---------------------------------------------------------------------
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 JTØ.
C
C AT A TIME JTØ,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-------------------------------------------------------------------
C SET UP INITIAL ADDRESSES
      NVNC=NVD*NCHAN
      NTOT=NVNC+4*NCHAN
      INMO=IVINSQ+NVD
      I1=INMO+NVNC
      I2=I1+NCHAN
      I3=I2+NCHAN
      I4=I3+NCHAN
      I5=I4+NCHAN
      IADD=I5+NCHAN
C TRANSFER TØSQ TO Ø
      CALL APPUT(TØSQ(JTØ),Ø,K1,K2)
C TRANSFER XSQ TO (1,NCHAN)
      CALL APPUT(XSQ,1,NCHAN,K2)
C TRANSFER VINSQ TO (IVINSQ,NVD)
      CALL APPUT(VINSQ,IVINSQ,NVD,K2)
C SET UP INITIAL ADDRESSES FOR RESULTS
```

```
      IRES=INMO
      IRNV1=IRES+NVD-1
      I1J=I1
      I2J=I2
      CALL APWD
C ITERATE THROUGH CHANNELS
      DO 40 JCHAN=1,NCHAN
C FORM XSQ(JCHAN)*VINSQ(JV)+T0SQ(JT0) IN (RES,NVD)
      CALL VSMSA(IVINSQ,K1,JCHAN,0,IRES,K1,NVD)
C FORM SQRT(RES,NVD)
      CALL VSQRT(IRES,K1,IRES,K1,NVD)
C FORM INT(FSAMPM*SQRT(T0SQ(JT0)+XSQ(JCHAN)*VINSQ(JV))+0.5) USING FSAMPM
C IN KFS AND 0.5 IN KHF
      CALL VSMSA(IRES,K1,KFS,KHF,IRES,K1,NVD)
      CALL VINT(IRES,K1,IRES,K1,NVD)
C FILL IADD WITH VALUE FROM IRNV1 AND NEGATE IT
      CALL VFILL(IRNV1,IADD,K1,K1)
      CALL VNEG(IADD,K1,IADD,K1,K1)
C FILL I1J WITH VALUE FROM IRNV1 (VELOCITY V2)
C AND I2J WITH VALUE FROM IRES (VELOCITY V1)
      CALL VFILL(IRNV1,I1J,K1,K1)
      CALL VFILL(IRES,I2J,K1,K1)
C ADD VALUE IN IADD TO (IRES,NVD),I.E. SUBTRACT MOVEOUT FOR VELOCITY V2
C TO GIVE NMOFS
      CALL VSADD(IRES,K1,IADD,IRES,K1,NVD)
C REDEFINE ADDRESSES SO THAT RESULTS FOR NEXT CHANNEL FOLLOW ON
      IRES=IRES+NVD
      IRNV1=IRNV1+NVD
      I1J=I1J+1
      I2J=I2J+1
40    CONTINUE
C ADD -NSTM-NGTM2 TO (I1,NCHAN) USING VALUE IN KADD1
      CALL VSADD(I1,K1,KADD1,I1,K1,NCHAN)
C ADD -NSTM+NGTM2+1 TO (I2,NCHAN) USING VALUE IN KADD2
      CALL VSADD(I2,K1,KADD2,I2,K1,NCHAN)
C SUBTRACT (I1,NCHAN) FROM (I2,NCHAN) TO FORM (I3,NCHAN)
      CALL VSUB(I1,K1,I2,K1,I3,K1,NCHAN)
C MULTIPLY (I1,NCHAN) BY 1.0/NSBLK FROM KBLKIN TO FORM (I4,NCHAN)
      CALL VSMUL(I1,K1,KBLKIN,I4,K1,NCHAN)
C TRUNCATE (I4,NCHAN) TO INTEGER VALUES
      CALL VINT(I4,K1,I4,K1,NCHAN)
C MULTIPLY (I4,NCHAN) BY -NSBLK FROM KBLK TO FORM (I5,NCHAN)
      CALL VSMUL(I4,K1,KBLK,I5,K1,NCHAN)
C ADD (I5,NCHAN) TO (I1,NCHAN)
      CALL VADD(I1,K1,I5,K1,I1,K1,NCHAN)
C ADD 1.0 FROM TM TO (I1,NCHAN)
      CALL VTSADD(I1,K1,2049,I1,K1,NCHAN)
C ADD (I5,NCHAN) TO (I2,NCHAN) AND MULTIPLY BY 2.0 FROM TM
      CALL VADD(I2,K1,I5,K1,I2,K1,NCHAN)
      CALL VTSMUL(I2,K1,2050,I2,K1,NCHAN)
C FIX ALL VALUES  FROM INMO TO I5 PREPARATORY T
C TRANSFER TO HOST
      CALL VFIX(INMO,K1,INMO,K1,NTOT)
C TRANSFER TO HOST ARRAYS
```

```
      CALL APWR
      CALL APGET(NMOFS,INMO,NVNC,K1)
      CALL APGET(JST,I1,NCHAN,K1)
      CALL APGET(NWDS,I2,NCHAN,K1)
      CALL APGET(NEL,I3,NCHAN,K1)
      CALL APGET(JBLK,I4,NCHAN,K1)
C SET UP INITIAL ADDRESSES FOR SEMBLANCE CALCULATIONS
      ISEMB=IEN+NVD
      ISEMB1=ISEMB-1
      ISMAX=ISEMB+NVD
      ISMAX1=ISMAX+1
      ISTK=ISMAX+2
      CALL APWD
C CLEAR SECTIONS OF AP MEMORY THAT WILL HAVE DATA ADDED.
      CALL VCLR(IEN,K1,NVD)
      NVNG=NVD*NGATE
      CALL VCLR(ISTK,K1,NVNG)
C
C FILL BUFFER SØ WITH DATA WINDOW FOR CHANNEL 1
      IF(IREAD(NWDS(1),SØ(1),JBLK(1)+1,IDCH).LT.Ø)STOP'READ ERROR'
C DEFINE INITIAL VALUE OF BUFFER SWITCH;
C SW=.TRUE.,READ FROM SØ,INTO S1
C SW=.FALSE.,READ FROM S1,INTO SØ.
      SW=.TRUE.
C
C ITERATE THROUGH CHANNELS
      JNMO=1
      DO 2ØØ JCHAN=1,NCHAN
      JCHAN1=JCHAN+1
C TRANSFER DATA FROM BUFFER TO AP
      CALL APWR
      CALL IWAIT(IDCH)
      IF(SW) CALL APPUT(SØ(JST(JCHAN)),Ø,NEL(JCHAN),K2)
      IF(.NOT.SW) CALL APPUT(S1(JST(JCHAN)),Ø,NEL(JCHAN),K2)
      IF(JCHAN1.GT.NCHAN)GOTO5Ø
C TRANSFER NEW DATA FROM DISK TO BUFFERS
      IF(SW)IERR=IREAD(NWDS(JCHAN1),S1(1),JSTCH(JCHAN1)+JBLK(JCHAN1),
     1IDCH)
      IF(.NOT.SW)IERR=IREAD(NWDS(JCHAN1),SØ(1),JSTCH(JCHAN1)+
     1JBLK(JCHAN1),IDCH)
      IF(IERR.LT.Ø)STOP'READ ERROR'
      SW=.NOT.SW
C NOW DO SEMBLANCE CALCULATIONS ON DATA WITHIN AP
5Ø    CALL APWD
      DO 1ØØ JV=1,NVD
C ADD APPROPRIATELY MOVED OUT GATE ONTO STACK USING ONLY EVERY MTH
C DATA POINT
      CALL VADD(ISTK,K1,NMOFS(JNMO),M,ISTK,K1,NGATE)
C FORM SUM OF SQUARES OF ELEMENTS ADDED ONTO STACK
      CALL SVESQ(NMOFS(JNMO),M,ISSQ,NGATE)
      IENJ=ISSQ+JV
      CALL VADD(IENJ,K1,ISSQ,K1,IENJ,K1,K1)
C REDEFINE ISTK SO THAT NEXT STACK FOLLOWS ON
      ISTK=ISTK+NGATE
```

```
      JNMO=JNMO+1
100   CONTINUE
      ISTK=ISMAX+2
200   CONTINUE
C
C NOW ACCUMULATE SEMBLANCE
      DO 300 JV=1,NVD
C FIND MEAN SUM OF SQUARES FOR EACH STACK
      CALL SVESQ(ISTK,K1,ISEMB1+JV,NGATE)
      ISTK=ISTK+NGATE
300   CONTINUE
C DIVIDE MEAN STACK ENERGIES BY SUMMED ENERGY
      CALL VDIV(IEN,K1,ISEMB,K1,ISEMB,K1,NVD)
C FIND MAXIMUM VALUE OF SEMBLANCE
      CALL MAXV(ISEMB,K1,ISMAX,NVD)
C DIVIDE BY NCHAN TO NORMALISE SEMBLANCE AND ITS MAX VALUE
      NV1=NVD+1
      CALL VSMUL(ISEMB,K1,KNCHIN,ISEMB,K1,NV1)
C WRITE OUT SEMBLANCE AND ITS MAXIMUM VALUE TO SMBLCE
      CALL APWR
      CALL APGET(SMBLCE,ISEMB,NV1,K2)
C WRITE OUT ADDRESS OF MAXIMUM VALUE TO IMAX
      CALL APGET(IMAX,ISMAX1,K1,K1)
C CALCULATE JVSMX,VALUE OF JV FOR WHICH SEMBLANCE IS MAXIMUM
      JVSMX=IMAX-ISEMB1
C WRITE OUT SEMBLANCE TO OUTPUT DEVICE 2
      WRITE(2) NVD,JVSMX,(SMBLCE(JV),JV=1,NVD)
500   CONTINUE
      CALL CLOSEC(IDCH)
      STOP
      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

FORTRAN IV V02.04 THU 08-JAN-81 00:04:12 PAGE 001

```
C=================================================================
C MAIN PROGRAM
C-----------------------------------------------------------------
          REAL*8 FSPECR,FSPECW
          VIRTUAL TØSQ(2Ø48),VINSQ(2Ø48)
          DIMENSION MUTE(24),XSQ(24),FBUF(3)
          COMMON /LAYER/TØLYR(99),VLYR(99)
          DATA DEV/3RRK /
          CALL ASSIGN(1,'DK1:ANCDP.DAT',13)
          IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
          IDCH=IGETC()
          IDCH1=IGETC()
          READ(1,1ØØØ)L,NCHAN,NSTART,NLYR,M
     1ØØØ FORMAT(12I5)
          READ(1,11ØØ)XSTART,XSTEP,FSAMP
     11ØØ FORMAT(3F1Ø.Ø)
          READ(1,12ØØ)(TØLYR(I),VLYR(I),I=1,NLYR)
     12ØØ FORMAT(2F1Ø.Ø)
          READ(1,1ØØØ)INDEN,INDMUT
          IF(INDMUT.GE.Ø)READ(1,1ØØØ)(MUTE(I),I=1,NCHAN)
          READ(1,13ØØ)FBUF
     13ØØ FORMAT(3A4)
          CALL IRAD5Ø(12,FBUF,FSPECR)
          READ(1,13ØØ)FBUF
          CALL IRAD5Ø(12,FBUF,FSPECW)
          IF(LOOKUP(IDCH,FSPECR).LT.Ø)STOP'LOOKUP ERROR'
          NBLKS=L*(NCHAN+1)/128+1
          IF(IENTER(IDCH1,FSPECW,NBLKS).LT.Ø)STOP'ENTER ERROR'
          L2INT=2
          DO 5 K=1,1ØØØ
          IF(L2INT.GE.L) GOTO 1Ø
          L2INT=2*L2INT
        5 CONTINUE
       1Ø X=XSTART
          DO 2Ø JCHAN=1,NCHAN
          XSQ(JCHAN)=X**2
          X=X+XSTEP
       2Ø CONTINUE
          TSAMP=1./FSAMP
          TØ=NSTART*TSAMP
          DO 3Ø JTØ=1,L
          TØSQ(JTØ)=TØ**2
          TØ=TØ+TSAMP
       3Ø CONTINUE
          CALL SETAP(L2INT,M)
          CALL INVSQ(FSAMP,NSTART,NLYR,L,VINSQ)
          CALL CDP(L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,
         1INDEN,INDMUT,IDCH,IDCH1,TØSQ,VINSQ,XSQ)
          STOP
          END
```

FORTRAN IV V02.04 THU 08-JAN-81 00:04:53 PAGE 001

```
C=======================================================================
          BLOCK DATA
C-----------------------------------------------------------------------
          IMPLICIT INTEGER*2(I-N)
          COMMON /KONST/ K1M,KØ,K1,K2,K3,K4,K5,K6,K7,K8,K9,K1Ø,
         1IA,IB,IC,ID,ITOP
          DATA K1M,KØ,K1,K2,K3,K4,K5,K6,K7,K8,K9,K1Ø,IA,IB,IC,ID,ITOP
         1/-1,Ø,1,2,3,4,5,6,7,8,9,1Ø,Ø,2Ø48,4Ø96,6144,8191/
          END
```

```
C===========================================================================
      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.
      VIRTUAL VINSQ(2Ø48)
      COMMON /LAYER/TØLYR(99),VLYR(99)
      N1=1
      N2=IFIX(FSAMP*TØLYR(1))-NSTART
      VINSQ1=1.Ø/VLYR(1)**2
      DO 1Ø I=N1,N2
      VINSQ(I)=VINSQ1
 1Ø   CONTINUE
      IF(NLYR.EQ.1) GOTO 4Ø
      DO 3Ø J=2,NLYR
      N1=N2+1
      N2=IFIX(FSAMP*TØLYR(J))-NSTART
      DELV=(VLYR(J)-VLYR(J-1))/(N2-N1+2)
      V=VLYR(J-1)
      DO 2Ø I=N1,N2
      VINSQ(I)=1.Ø/V**2
      V=V+DELV
 2Ø   CONTINUE
 3Ø   CONTINUE
 4Ø   N1=N2+1
      N2=L
      VINSQN=1.Ø/VLYR(NLYR)**2
      DO 5Ø I=N1,N2
      VINSQ(I)=VINSQN
 5Ø   CONTINUE
      RETURN
      END
```



```
C===========================================================================
      SUBROUTINE SETAP(L2INT,M)
C---------------------------------------------------------------------------
C
C SETS UP COMPLEX EXPONENTIAL TABLE IN D+1
C AND PUTS 1.Ø IN ID
      COMMON /KONST/ K1M,KØ,K1,K2,K3,K4,K5,K6,K7,K8,K9,K1Ø,
     1IA,IB,IC,ID,ITOP
C INITIALISE AP
      CALL APINIT
C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS.
      ID1=ID+1
      ID2=ID+2
      L2I21=L2INT/2-1
C TRANSFER 1.Ø/(L2INT*M) TO KØ AND MULTIPLY BY 2PI FROM TM.
      CONST=1.Ø/FLOAT(L2INT*M)
      CALL APWR
      CALL APPUT(CONST,KØ,K1,K2)
      CALL APWD
C FORM VECTOR RAMP AND TAKE SIN AND COS OF IT
      CALL VTSMUL(KØ,K1,2317,KØ,K1,K1)
      CALL VRAMP(KØ,KØ,IB,K1,L2I21)
      CALL VCOS(IB,K1,ID1,K2,L2I21)
      CALL VSIN(IB,K1,ID2,K2,L2I21)
C PUT 1.Ø IN ID
      CALL VCLR(ID,K1,K1)
      CALL VTSADD(ID,K1,2Ø49,ID,K1,K1)
      RETURN
      END
```

```
      SUBROUTINE CDP(L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,
     1INDEN,INDMUT,IDCH,IDCH1,TØSQ,VINSQ,XSQ)
      VIRTUAL SEISM(32767),TØSQ(2Ø48),VINSQ(2Ø48)
      DIMENSION INDEX(2Ø48),MUTE(24),STACK(2Ø48),
     1DUM(1Ø),CONST(1Ø),FSTAP(2Ø),XSQ(24)
      COMMON /KONST/ K1M,KØ,K1,K2,K3,K4,K5,K6,K7,K8,K9,K1Ø,
     1IA,IB,IC,ID,ITOP
C
C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS
      IB1=IB+1
      IC1=IC+1
      IC2=IC+2
      ID1=ID+1
      L1=L+1
      L2I21=L2INT/2-1
      CONST(2)=M*FSAMP
      CONST(3)=Ø.5
      CONST(4)=-M*NSTART
      CONST(5)=Ø.Ø
      CONST(6)=L*M
      CONST(7)=L+1
      CONST(8)=1.Ø
      CONST(9)=1.Ø/M
      CONST(1Ø)=(L+1)*M-1
      JBLK=1
      NBLKTR=L/128
      FSTD=ADGET(SEISM(1))
      FTØSQ=APGAD(TØSQ(1))
      FVINSQ=APGAD(VINSQ(1))
      ISTART=1
      DO 5Ø K=1,M
      FSTAP(K)=APGAD(SEISM(ISTART))
      ISTART=ISTART+L1
   5Ø CONTINUE
C SET SEISM(L+1) TO Ø.Ø TO COPE WITH TIME OVERFLOW AND CLEAR A IN AP
      SEISM(L1)=Ø.Ø
      CALL VCLR(IA,K1,L)
C ITERATE THROUGH CHANNELS
      DO 5ØØ JCHAN=1,NCHAN
C READ IN TRACE SEISM FROM UNIT 2
      CALL IWAIT(IDCH1)
      IF(IREADA(IDCH,2*L,JBLK,FSTD).LT.Ø)STOP'READA ERROR'
      CALL IWAIT(IDCH)
C COMPUTE INDEX ARRAY
      CONST(1)=XSQ(JCHAN)
      CALL APWR
C TRANSFER (K1,K1Ø) TO HOST DUMMY ARRAY DUM AND REPLACE BY CONST
      CALL APGET(DUM,K1,K1Ø,K2)
      CALL APPUT(CONST,K1,K1Ø,K2)
C TRANSFER TØSQ TO B AND VINSQ TO IC1
      CALL APPUTA(IB,L,K2,FTØSQ)
      CALL APPUTA(IC1,L,K2,FVINSQ)
      CALL APWD
C FORM XSQ(JCHAN)*VINSQ IN C1
```

```
      CALL VSMUL(IC1,K1,K1,IC1,K1,L)
C FORM SQRT(TØSQ+XSQ(JCHAN)*VINSQ) IN B
      CALL VADD(IB,K1,IC1,K1,IB,K1,L)
      CALL VSQRT(IB,K1,IB,K1,L)
C FORM IFIX(FSAMP*M*SQRT(TØSQ+XSQ(JCHAN)*VINSQ)+Ø.5)-NSTART*M IN B
      CALL VSMSA(IB,K1,K2,K3,IB,K1,L)
      CALL VINT(IB,K1,IB,K1,L)
      CALL VSADD(IB,K1,K4,IB,K1,L)
C CLIP B BETWEEN Ø.Ø AND L*M
      CALL VCLIP(IB,K1,K5,K6,IB,K1,L)
C MULTIPLY B BY L+1 ,ADD 1.Ø AND PLACE RESULT IN C1
      CALL VSMSA(IB,K1,K7,K8,IC1,K1,L)
C MULTIPLY B BY 1.Ø/M AND TAKE INTEGER PART
      CALL VSMUL(IB,K1,K9,IB,K1,L)
      CALL VINT(IB,K1,IB,K1,L)
C MULTIPLY B BY (L+1)*M-1
      CALL VSMUL(IB,K1,K1Ø,IB,K1,L)
C SUBTRACT B FROM C1 AND PUT RESULT IN B
      CALL VSUB(IB,K1,IC1,K1,IB,K1,L)
C FIX B AND TRANSFER BACK AS HOST ARRAY INDEX
      CALL VFIX(IB,K1,IB,K1,L)
      CALL APWR
      CALL APGET(INDEX,IB,L,K1)
C TRANSFER DUM BACK TO K1
      CALL APPUT(DUM,K1,K1Ø,K2)
      CALL APWD
C
C HAVING COMPUTED INDEX ARRAY NOW START INTERPOLATION OF TRACE
C
C CLEAR B IN AP
      CALL VCLR(IB,K1,L2INT)
C TRANSFER TRACE TO B
      CALL APWR
      CALL APPUTA(IB,L,K2,FSTAP(1))
      CALL APWD
C FIND MEAN VALUE OF TRACE AND PLACE IN AP(ITOP)
      CALL MEANV(IB,K1,ITOP,L)
C SUBTRACT MEAN FROM TRACE
      CALL VNEG(ITOP,K1,ITOP,K1,K1)
      CALL VSADD(IB,K1,ITOP,IB,K1,L)
C IF INDEN IS NEGATIVE,NO SCALING OF TRACE
C IF INDEN IS ZERO SCALE TO UNIT R.M.S VALUE
C IF INDEN IS POSITIVE,USE INVERSE ENERGY SCALING (DIVERSITY STACK)
      IF(INDEN.LT.Ø) GOTO 1ØØ
      CALL RMSQV(IB,K1,ITOP,L)
      IF(INDEN.GT.Ø) CALL VSQ(ITOP,K1,ITOP,K1,K1)
C USE 1.Ø RESIDING IN ID (FROM SETAP)
      CALL VDIV(ITOP,K1,ID,K1,ITOP,K1,K1)
      CALL VSMUL(IB,K1,ITOP,IB,K1,L)
C IF INDMUT IS NEGATIVE USE NO MUTING
1ØØ   IF(INDMUT.LT.Ø) GOTO 2ØØ
      CALL VCLR(IB,K1,MUTE(ICHAN))
C TRANSFER MODIFIED TRACE BACK TO SEISM
2ØØ   CALL APWR
```

```
      CALL APGETA(IB,L,K2,FSTAP(1))
      CALL APWD
      IF(N.EQ.1) GOTO 35Ø
C TAKE TRANSFORM OF TRACE AND PUT IN C
      CALL RFFTB(IB,IC,L2INT,K1)
      CALL RFFTSC(IC,L2INT,KØ,K1)
      CALL VCLR(IC,K1,K2)
C ITERATE FROM 2 TO M
      DO 3ØØ K=2,M
C MULTIPLY TRANSFORM BY COMPLEX EXPONENTIAL ARRAY
      CALL CVMUL(IC2,K2,ID1,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
3ØØ   CONTINUE
C PICK REQUIRED ELEMENTS OUT OF SEISM AND PLACE IN STACK
35Ø   DO 4ØØ I=1,L
      STACK(I)=SEISM(INDEX(I))
4ØØ   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.Ø)STOP'WRITE ERROR'
      JBLK=JBLK+NBLKTR
      CALL VADD(IA,K1,IB1,K1,IA,K1,L)
5ØØ   CONTINUE
C SCALE STACK BY 1.Ø/NCHAN
      FNCINV=1.Ø/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.Ø)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

<u>READ(1,1000)(IVELAN(I),I=1,NVEL)</u>

IVELAN...File positions at which velocity functions are defined

<u>READ(1,1100)XSTART,XSTEP,FSAMP</u>

<u>1100 FORMAT(3F10.0)</u>

XSTART...Shot/First receiver offset

XSTEP....Receiver spacing

FSAMP....Sampling rate in samples per millisecond

<u>READ(1,1200)(TO2LYR(I),V2LYR(I),I=1,N2LYR)</u>

<u>1200 FORMAT(2F10.0)</u>

TO2LYR...Two-way travel time in milliseconds

V2LYR....RMS velocity at the above two-way travel time

<u>READ(1,1000)INDEN,INDMUT,INFLG,OUTFLG</u>

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 abcve two-way travel time

```
C==============================================================
C MAIN PROGRAM
C
C THIS IS A STACK PROGRAM CAPABLE OF VELOCITY INTERPOLATION
C
C--------------------------------------------------------------
      REAL*8 FSPECR,FSPECW,FNAMR,FNAMO
      VIRTUAL T0SQ(2048),VINSQ(2048),FNAMR(100),FNAMO(100),
     %T0LYR(20),VLYR(20),T0INT(20),VINT(20),IVELAN(100),
     %T02LYR(20),V2LYR(20)
      DIMENSION MUTE(24),XSQ(24),FBUF(3),IHBLK(256)
      INTEGER*2 OUTFLG
      LOGICAL*1 TPDRR,TPDRW,STATUS,ITLEN,LBLK(512)
      EQUIVALENCE (IHBLK(1),LBLK(1))
      COMMON/STK/ L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,INDEN,
     %INDMUT,IDCH,IDCH1,XSQ
      DATA DEV/3RRK /
C
C SET UP I/O CHANNELS AND READ IN CONTROL DATA
C
      IF(ICDFN(25).NE.0)STOP'CHANNELS FULL'
      CALL ASSIGN(1,'DK1:MPSTAK.DAT',14)
      CALL ASSIGN(2,'DK1:MPSTAK.LOG',14)
      IF(IFETCH(DEV).NE.0)STOP'FETCH ERROR'
      IEOTR=0
      IEOTW=0
      IDCH=20
      IDCH1=21
      READ(1,1000)NFILES,NVEL,L,NCHAN,NSTART,N2LYR,M
 1000 FORMAT(12I5)
      READ(1,1000)TPDRR,TPDRW,INTSW
      READ(1,1000)(IVELAN(I),I=1,NVEL)
      READ(1,1100)XSTART,XSTEP,FSAMP
 1100 FORMAT(3F10.0)
      READ(1,1200)(T02LYR(I),V2LYR(I),I=1,N2LYR)
 1200 FORMAT(2F10.0)
      READ(1,1000)INDEN,INDMUT,INFLG,OUTFLG
      IF(INDMUT.GE.0)READ(1,1000)(MUTE(I),I=1,NCHAN)
C
C READ IN FILESPECS FOR INPUT DEPENDING ON
C IF IT IS FROM TAPE OF DISC
C
      IF(INFLG.NE.0)GOTO 40
      READ(1,1300)FBUF
 1300 FORMAT(3A4)
      CALL IRAD50(12,FBUF,FSPECR)
      GOTO 50
   40 DO 60 I=1,NFILES
      READ(1,1300)FBUF
      CALL IRAD50(12,FBUF,FSPECR)
      FNAMR(I)=FSPECR
   60 CONTINUE
C
C READ IN OUTPUT FILE SPECS AGAIN THIS
```

```
C IS DEPENDANT ON IF IT GOES TO TAPE OR STAYS ON DISC.
C
    50 IF(OUTFLG.NE.0)GOTO 70
       READ(1,1300)FBUF
       CALL IRAD50(12,FBUF,FSPECW)
       GOTO 80
    70 DO 90 I=1,NFILES
       READ(1,1300)FBUF
       CALL IRAD50(12,FBUF,FSPECW)
       FNAMO(I)=FSPECW
    90 CONTINUE
C
C SET UP CONSTANTS ARRAYS
C
    80 L2INT=2
       DO 5 K=1,1000
       IF(L2INT.GE.L) GOTO 10
       L2INT=2*L2INT
5      CONTINUE
10     X=XSTART
       DO 20 JCHAN=1,NCHAN
       XSQ(JCHAN)=X**2
       X=X+XSTEP
20     CONTINUE
       TSAMP=1./FSAMP
       T0=NSTART*TSAMP
       DO 30 JT0=1,L
       T0SQ(JT0)=T0**2
       T0=T0+TSAMP
30     CONTINUE
C
C SET UP AP CONSTANTS AND CDP SUBROUTINE
C OUTSIDE OF LOOP
C
       CALL SETAP(L2INT,M)
       CALL CDP(T0SQ,VINSQ,0)
       IVEL=1
       NBLKR=L*NCHAN/128+6
       NBLKW=L/128+1
C
C START OF MAIN PROCESSING LOOP
C
       DO 999 IFNUM=1,NFILES
       IFIL=IFNUM
       IF(INFLG.NE.0)GOTO 100
C
C COME HERE IF INPUT FROM TAPE
C
       IF(IENTER(IDCH,FSPECR,NBLKR).LT.0)STOP'ENTER ERR'
C
C EOT CHECK
C
       ITRY=1
   200 IF(ITRY.GT.3)GOTO 210
```

```
      IF(IEOTR.GE.0)GOTO 105
  210 WRITE(7,1800)TPDRR,IFIL
 1800 FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
      WRITE(7,1801)
 1801 FORMAT(' ENTER NEW READ DRIVE NUMBER:',$)
      READ(5,1802)TPDRR
 1802 FORMAT(I1)
      IEOTR=0
      IF(TPDRR.GT.2)STOP' EOT READ TERMINATE'
C
C DO A TAPE READ
C
  105 CALL TAPRED(-1,TPDRR,STATUS,ITLEN,IFLEN,IFIL,IEOTR)
C
C FATAL ERROR DETECTION
C
      IF(STATUS.LT.0)WRITE(2,1500)IFNUM
 1500 FORMAT(' WARNING FILE ',I3,' RETRIES FAILED')
      IF(IEOTR.LT.0)GOTO 106
C
C WIND OVER EOF MARK
C
      CALL TAPRED(0,TPDRR,STATUS, , ,IFIL,IEOTR)
  106 CALL IWAIT(IDCH)
      IERR=0
      ITRY=ITRY+1
      IF(IREADW(1,IERR,0,IDCH).LT.0)STOP' ERR READ ERR'
      IF(IERR.EQ."177777)GOTO 200
      CALL CLOSEC(IDCH)
C
C OPEN UP FILES FOR READING IN
C
  100 IF(INFLG.NE.0)FSPECR=FNAMR(IFNUM)
      IF(LOOKUP(IDCH,FSPECR).LT.0)STOP'LOOKUP ERR'
C
C OPEN UP OUTPUT FILES
C
      IF(OUTFLG.NE.0)FSPECW=FNAMR(IFNUM)
      IF(IENTER(IDCH1,FSPECW,NBLKW).LT.0)STOP'ENTER ERR'
C
C HEADER BLOCK MANAGEMENT
C
      IF(IREADW(256,IHBLK,0,IDCH).LT.0)STOP'HBLK ERR'
C
C UPDATE HEADER
C
      IBFREE=IHBLK(24)
      IHBLK(7)=1
      LBLK(19)=3
      IHBLK(129+IBFREE)=2
      IBFREE=IBFREE+1
      IHBLK(129+IBFREE)=NCHAN
      IBFREE=IBFREE+1
      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
0138          IVEL=IVEL+1
0139          NLYR=N2LYR
0140          DO 110 I=1,NLYR
0141          T0LYR(I)=T02LYR(I)
0142          VLYR(I)=V2LYR(I)
0143      110 CONTINUE
        C
        C SET UP NEW VINSQ TABLE
        C
0144          CALL INVSQ(FSAMP,NSTART,NLYR,L,VINSQ,T0LYR,VLYR)
0145          IF(NVEL.LT.IVEL)INTSW=0
0147          IF(NVEL.LT.IVEL)GOTO 120
        C
        C READ IN NEXT ANALYSIS AND SET UP INTERPOLATION
        C
0149          READ(1,1000)N2LYR
0150          READ(1,1200)(T02LYR(I),V2LYR(I),I=1,N2LYR)
0151          IF(INTSW.EQ.0)GOTO 120
0153          INT=IVELAN(IVEL)-IVELAN(IVEL-1)
0154          IF(INT.LE.1)GOTO 120
0155          RINT=FLOAT(INT)
0157          DO 130 I=1,NLYR
0158          T0INT(I)=(T02LYR(I)-T0LYR(I))/RINT
0159          VINT(I)=(V2LYR(I)-VLYR(I))/RINT
0160      130 CONTINUE
0161          GOTO 120
        C
        C COME HERE WHEN NOT A NEW ANALYSIS
        C
0162      108 IF(INTSW.EQ.0)GOTO 120
        C
        C ADD ON INTERPOLATING PARAMETERS
        C
0164          DO 140 I=1,NLYR
0165          T0LYR(I)=T0LYR(I)+T0INT(I)
0166          VLYR(I)=VLYR(I)+VINT(I)
0167      140 CONTINUE
        C
```

```
C SET UP VINSQ TABLE AND THEN DO STACK
C
      CALL INVSQ(FSAMP,NSTART,NLVR,L,VINSQ,T0LVR,VLVR)
  120 CALL CDP(T0SQ,VINSQ,1)
      CALL CLOSEC(IDCH)
      CALL CLOSEC(IDCH1)
      IF(OUTFLG.NE.0)GOTO 999
C
C COME HERE IF OUTPUT TO GO TO TAPE
C
      IFLEN=LOOKUP(IDCH1,FSPECW)
      IF(IFLEN.LT.0)STOP'LOOKUP ERR'
C
C DO A TAPE WRITE
C
      CALL TAPRED(1,TPDRW,STATUS,ITLEN,IFLEN,IFIL,IEOTW)
      CALL CLOSEC(IDCH1)
      IF(IEOTW.GE.0)GOTO 150
      WRITE(7,1600)TPDRW,IFIL
 1600 FORMAT(' EOT ON WRITE DRIVE:',I2,' FILE NO:',I4)
      WRITE(7,1601)
 1601 FORMAT(' ENTER NUMBER OF NEW DRIVE:',$)
      READ(5,1802)TPDRW
      IEOTW=0
      IF(TPDRW.GT.2)STOP'WRITE EOT TERMINATE'
  150 IF(STATUS.GE.0)GOTO 160
      WRITE(2,1700)IFNUM
 1700 FORMAT(' FATAL ERROR ON WRITE FILE ',I3)
      STOP' FATAL ERR'
  160 CONTINUE
  999 CONTINUE
      CALL CLOSEC(IDCH)
      CALL CLOSEC(IDCH1)
      STOP' NORMAL TERMINATION'
      END
```

FORTRAN IV V02.04 THU 08-JAN-81 00:30:45 PAGE 001

```
C==========================================================================
      SUBROUTINE INVSQ(FSAMP,NSTART,NLYR,L,VINSQ,T0LYR,VLYR)
C--------------------------------------------------------------------------
C USES INPUT DATA TO GENERATE ARRAY OF INVERSE SQUARED VELOCITIES
C VELOCITIES ARE CONSTANT UP TO FIRST REFLECTOR AND BEYOND
C LAST AND ARE LINEARLY INTERPOLATED IN BETWEEN.
      VIRTUAL VINSQ(2048),T0LYR(20),VLYR(20)
      N1=1
      N2=IFIX(FSAMP*T0LYR(1))-NSTART
      VINSQ1=1.0/VLYR(1)**2
      DO 10 I=N1,N2
      VINSQ(I)=VINSQ1
10    CONTINUE
      IF(NLYR.EQ.1) GOTO 40
      DO 30 J=2,NLYR
      N1=N2+1
      N2=IFIX(FSAMP*T0LYR(J))-NSTART
      DELV=(VLYR(J)-VLYR(J-1))/(N2-N1+2)
      V=VLYR(J-1)
      DO 20 I=N1,N2
      VINSQ(I)=1.0/V**2
      V=V+DELV
20    CONTINUE
30    CONTINUE
40    N1=N2+1
      N2=L
      VINSQN=1.0/VLYR(NLYR)**2
      DO 50 I=N1,N2
      VINSQ(I)=VINSQN
50    CONTINUE
      RETURN
      END
```

FORTRAN IV V02.04 THU 08-JAN-81 00:31:09 PAGE 001

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

```
      C=======================================================================
ØØØ1        SUBROUTINE SETAP(L2INT,M)
      C-----------------------------------------------------------------------
      C
      C SETS UP COMPLEX EXPONENTIAL TABLE IN D+1
      C AND PUTS 1.Ø IN ID
ØØØ2        COMMON /KONST/ K1M,KØ,K1,K2,K3,K4,K5,K6,K7,K8,K9,K1Ø,
           1IA,IB,IC,ID,ITOP
      C INITIALISE AP
ØØØ3        CALL APINIT
      C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS.
ØØØ4        ID1=ID+1
ØØØ5        ID2=ID+2
ØØØ6        L2I21=L2INT/2-1
      C TRANSFER 1.Ø/(L2INT*M) TO KØ AND MULTIPLY BY 2PI FROM TM.
ØØØ7        CONST=1.Ø/FLOAT(L2INT*M)
ØØØ8        CALL APWR
ØØØ9        CALL APPUT(CONST,KØ,K1,K2)
ØØ1Ø        CALL APWD
      C FORM VECTOR RAMP AND TAKE SIN AND COS OF IT
ØØ11        CALL VTSMUL(KØ,K1,2317,KØ,K1,K1)
ØØ12        CALL VRAMP(KØ,KØ,IB,KI,L2I21)
ØØ13        CALL VCOS(IB,K1,ID1,K2,L2I21)
ØØ14        CALL VSIN(IB,K1,ID2,K2,L2I21)
      C PUT 1.Ø IN ID
ØØ15        CALL VCLR(ID,K1,K1)
ØØ16        CALL VTSADD(ID,K1,2Ø49,ID,K1,K1)
ØØ17        RETURN
ØØ18        END
```

```
C ================================
      SUBROUTINE CDP(TØSQ,VINSQ,ICODE)
C
C MAIN STACKING SUBROUTINE
C ===================================
      VIRTUAL SEISM(32767),TØSQ(2Ø48),VINSQ(2Ø48)
      DIMENSION INDEX(2Ø48),MUTE(24),STACK(2Ø48),
     1DUM(1Ø),CONST(1Ø),FSTAP(2Ø),XSQ(24)
      COMMON/STK/L,L2INT,NCHAN,M,NSTART,FSAMP,MUTE,INDEN,
     XINDMUT,IDCH,IDCH1,XSQ
      COMMON /KONST/ K1M,KØ,K1,K2,K3,K4,K5,K6,K7,K8,K9,K1Ø,
     1IA,IB,IC,ID,ITOP
C
C SET UP REQUIRED STARTING ADDRESSES AND CONSTANTS
C
      IF(ICODE.GT.Ø)GOTO 2Ø
      IB1=IB+1
      IC1=IC+1
      IC2=IC+2
      ID1=ID+1
      L1=L+1
      L2I21=L2INT/2-1
      CONST(2)=M*FSAMP
      CONST(3)=Ø.5
      CONST(4)=-M*NSTART
      CONST(5)=Ø.Ø
      CONST(6)=L*M
      CONST(7)=L+1
      CONST(8)=1.Ø
      CONST(9)=1.Ø/M
      CONST(1Ø)=(L+1)*M-1
      NBLKTR=L/128
      FSTD=ADGET(SEISM(1))
      FTØSQ=APGAD(TØSQ(1))
      FVINSQ=APGAD(VINSQ(1))
      ISTART=1
      DO 5Ø K=1,M
      FSTAP(K)=APGAD(SEISM(ISTART))
      ISTART=ISTART+L1
   5Ø CONTINUE
      RETURN
C SET SEISM(L+1) TO Ø.Ø TO COPE WITH TIME OVERFLOW AND CLEAR A IN AP
   2Ø JBLK=1
      IBLK=1
      SEISM(L1)=Ø.Ø
      CALL VCLR(IA,K1,L)
C ITERATE THROUGH CHANNELS
      DO 5ØØ JCHAN=1,NCHAN
C READ IN TRACE SEISM FROM UNIT 2
      IF(IREADA(IDCH,2*L,JBLK,FSTD).LT.Ø)STOP'READA ERROR'
      JBLK=JBLK+NBLKTR
      CALL IWAIT(IDCH)
C COMPUTE INDEX ARRAY
      CONST(1)=XSQ(JCHAN)
```

```
ØØ43          CALL APWR
        C TRANSFER (K1,K1Ø) TO HOST DUMMY ARRAY DUM AND REPLACE BY CONST
ØØ44          CALL APGET(DUM,K1,K1Ø,K2)
ØØ45          CALL APPUT(CONST,K1,K1Ø,K2)
        C TRANSFER TØSQ TO B AND VINSQ TO IC1
ØØ46          CALL APPUTA(IB,L,K2,FTØSQ)
ØØ47          CALL APPUTA(IC1,L,K2,FVINSQ)
ØØ48          CALL APWD
        C FORM XSQ(JCHAN)*VINSQ IN C1
ØØ49          CALL VSMUL(IC1,K1,K1,IC1,K1,L)
        C FORM SQRT(TØSQ+XSQ(JCHAN)*VINSQ) IN B
ØØ5Ø          CALL VADD(IB,K1,IC1,K1,IB,K1,L)
ØØ51          CALL VSQRT(IB,K1,IB,K1,L)
        C FORM IFIX(FSAMP*M*SQRT(TØSQ+XSQ(JCHAN)*VINSQ)+Ø.5)-NSTART*M IN B
ØØ52          CALL VSMSA(IB,K1,K2,K3,IB,K1,L)
ØØ53          CALL VINT(IB,K1,IB,K1,L)
ØØ54          CALL VSADD(IB,K1,K4,IB,K1,L)
        C CLIP B BETWEEN Ø.Ø AND L*M
ØØ55          CALL VCLIP(IB,K1,K5,K6,IB,K1,L)
        C MULTIPLY B BY L+1 ,ADD 1.Ø AND PLACE RESULT IN C1
ØØ56          CALL VSMSA(IB,K1,K7,K8,IC1,K1,L)
        C MULTIPLY B BY 1.Ø/M AND TAKE INTEGER PART
ØØ57          CALL VSMUL(IB,K1,K9,IB,K1,L)
ØØ58          CALL VINT(IB,K1,IB,K1,L)
        C MULTIPLY B BY (L+1)*M-1
ØØ59          CALL VSMUL(IB,K1,K1Ø,IB,K1,L)
        C SUBTRACT B FROM C1 AND PUT RESULT IN B
ØØ6Ø          CALL VSUB(IB,K1,IC1,K1,IB,K1,L)
        C FIX B AND TRANSFER BACK AS HOST ARRAY INDEX
ØØ61          CALL VFIX(IB,K1,IB,K1,L)
ØØ62          CALL APWR
ØØ63          CALL APGET(INDEX,IB,L,K1)
        C TRANSFER DUM BACK TO K1
ØØ64          CALL APPUT(DUM,K1,K1Ø,K2)
ØØ65          CALL APWD
        C
        C HAVING COMPUTED INDEX ARRAY NOW START INTERPOLATION OF TRACE
        C
        C CLEAR B IN AP
ØØ66          CALL VCLR(IB,K1,L2INT)
        C TRANSFER TRACE TO B
ØØ67          CALL APWR
ØØ68          CALL APPUTA(IB,L,K2,FSTAP(1))
ØØ69          CALL APWD
        C FIND MEAN VALUE OF TRACE AND PLACE IN AP(ITOP)
ØØ7Ø          CALL MEANV(IB,K1,ITOP,L)
        C SUBTRACT MEAN FROM TRACE
ØØ71          CALL VNEG(ITOP,K1,ITOP,K1,K1)
ØØ72          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)
ØØ73          IF(INDEN.LT.Ø) GOTO 1ØØ
ØØ75          CALL RMSQV(IB,K1,ITOP,L)
```

```
            IF(INDEN.GT.Ø) CALL VSQ(ITOP,K1,ITOP,K1,K1)
      C USE 1.Ø RESIDING IN ID (FROM SETAP)
            CALL VDIV(ITOP,K1,ID,K1,ITOP,K1,K1)
            CALL VSMUL(IB,K1,ITOP,IB,K1,L)
      C IF INDMUT IS NEGATIVE USE NO MUTING
      1ØØ   IF(INDMUT.LT.Ø) GOTO 2ØØ
            CALL VCLR(IB,K1,MUTE(ICHAN))
      C TRANSFER MODIFIED TRACE BACK TO SEISM
      2ØØ   CALL APWR
            CALL APGETA(IB,L,K2,FSTAP(1))
            CALL APWD
            IF(M.EQ.1) GOTO 35Ø
      C TAKE TRANSFORM OF TRACE AND PUT IN C
            CALL RFFTB(IB,IC,L2INT,K1)
            CALL RFFTSC(IC,L2INT,KØ,K1)
            CALL VCLR(IC,K1,K2)
      C ITERATE FROM 2 TO M
            DO 3ØØ K=2,M
      C MULTIPLY TRANSFORM BY COMPLEX EXPONENTIAL ARRAY
            CALL CVMUL(IC2,K2,ID1,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
      3ØØ   CONTINUE
      C PICK REQUIRED ELEMENTS OUT OF SEISM AND PLACE IN STACK
      35Ø   DO 4ØØ I=1,L
            STACK(I)=SEISM(INDEX(I))
      4ØØ   CONTINUE
      C TRANSFER STACK INTO B1 AND ADD TO STACK IN A
            CALL APWR
            CALL APPUT(STACK,IB1,L,K2)
            CALL APWD
            CALL VADD(IA,K1,IB1,K1,IA,K1,L)
      5ØØ   CONTINUE
      C SCALE STACK BY 1.Ø/NCHAN
            FNCINV=1.Ø/NCHAN
            CALL APWR
            CALL APPUT(FNCINV,ITOP,K1,K2)
            CALL APWD
            CALL VSMUL(IA,K1,ITOP,IA,K1,L)
      C TRANSFER SCALED STACK BACK TO STACK
            CALL APWR
            CALL APGET(STACK,IA,L,K2)
            CALL APWD
            IF(IWRITW(2*L,STACK,IBLK,IDCH1).LT.Ø)STOP'WRITE ERROR 2'
            RETURN
            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
      C
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
```

```
      ECOM(3)=IDRV
      ECOM(4)=0
      CALL SDS10(ECOM,ESTAT, , )
      GOTO 10
C
C WRITE SECTION
C
   20 ITRY=ITRY+1
      IF(ITRY.GT.3)GOTO 130
      COM(1)=SDSCOM(7)
      IFLEN=(ILEN+3)/4
      COM(2)=IFLEN
      COM(3)=IDRV
      COM(4)=1
      CALL SDS10(COM,ISTAT, , )
      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 TWRIT(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
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 Filtering

READ(1,1100)FL,FU

READ(1,1100)FTPR1,FTPR2

FL.......Lower cutoff frequency Hz

FU.......Upper cutoff frequency Hz

FTPR1....Length of lower cutoff taper Hz

FTPR2....Length of upper cutoff taper Hz

6....Bandreject Filtering

READ(1,1100)FLR,FUR

FLR......Lower cutoff frequency Hz

FUR......Upper cutoff frequency Hz

7....Prediction Error Deconvolution

READ(1,1000)NPFILT,NLAG,IPNORM

READ(1,1100)PRWHIT

NPFILT...Number of samples in filter

NLAG.....Prediction distance, in samples

IPNORM...Scaling flag

0 - no scaling

1 - Filter unit energy

2 - equal input/output energy

PRWHIT...Fractional prewhitening

8....Trace Normalisation

READ(1,1000)NRMFLG

NRMFLG...Normalisation flag

0 - normalise to unit energy

1 - normalise to unit maximum amplitude

```
C
C  POST STACK UTILITY PROGRAM
C  THIS INVOLVES THE FOLLOWING
C 1:EDIT
C 2:EXP(Ø.2T) AMP RECOVERY
C 3:T*EXP(Ø.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 1Ø:NORMALISATION TO UNIT ENERGY/AMPLITUDE
C
      REAL*8 FSPECR,FSPECW,FNAMR,FNAMO
      VIRTUAL FNAMR(1ØØ),FNAMO(1ØØ),EXPT(2Ø48),TEXPT(2Ø48),
     %TVSQ(2Ø48),TAPER(512),BPASS(2Ø49),BRJCT(2Ø49),MUTE(2ØØØ)
      REAL*4 FBUF(3),TØLYR(2Ø),VLYR(2Ø),CONST(3),SEISM(2Ø48)
      INTEGER*2 IORD(8),UTLFLG(8),IFILED(1ØØ),DBUF(4352),
     %IHBLK(256),OUTFLG,MNPOS(3Ø),MSAMP(3Ø),MINC(3Ø)
      LOGICAL*1 TPDRR,TPDRW,ISTAT
      EQUIVALENCE (IHBLK(1),DBUF(1)),(SEISM(1),DBUF(257))
      DATA DEV/3RRK /
C
C SET UP VIRTUAL ADDRESSES
C
      IEOTR=Ø
      IEOTW=Ø
      ATAP=APGAD(TAPER(1))
      ATBP=APGAD(BPASS(1))
      ATBR=APGAD(BRJCT(1))
      ATVSQ=APGAD(TVSQ(1))
      ATEXPT=APGAD(TEXPT(1))
      AEXPT=APGAD(EXPT(1))
C
C SET UP I/O  CHANNELS AND READ IN CONTROL DATA
C
      IF(ICDFN(25).NE.Ø)STOP'CHANNEL OVERFLOW'
      CALL ASSIGN(1,'DK2:MPPOST.DAT',14)
      CALL ASSIGN(2,'DK2:MPPOST.LOG',14)
      IDCH=2Ø
      IDCH1=21
      READ(1,1ØØØ)NFILES,NSAMP,NSTART,INFLG,OUTFLG
 1ØØØ FORMAT(12I5)
      READ(1,1ØØØ)TPDRR,TPDRW
      READ(1,11ØØ)FSAMP
 11ØØ FORMAT(2F1Ø.Ø)
C
C READ IN FILE SPECS FOR INPUT
C
 12ØØ FORMAT(3A4)
      IF(INFLG.EQ.Ø)GOTO 2Ø
      DO 3Ø I=1,NFILES
      READ(1,12ØØ)FBUF
```

```
          CALL IRAD5Ø(12,FBUF,FSPECR)
          FNAMR(I)=FSPECR
   3Ø CONTINUE
C
C READ IN FILE SPECS FOR OUTPUT
C
   2Ø IF(OUTFLG.EQ.Ø)GOTO 5Ø
      DO 6Ø I=1,NFILES
      READ(1,12ØØ)FBUF
      CALL IRAD5Ø(12,FBUF,FSPECW)
      FNAMO(I)=FSPECW
   6Ø CONTINUE
C
C READ IN JOB SPECIFIC DATA AND SET UP
C THE FILTERS TO BE USED
C
   5Ø NUTIL=1Ø
      ITX=Ø
      CONST(1)=FLOAT(NSTART)
      CONST(2)=Ø.2
      CONST(3)=1.Ø/(1ØØØ.Ø*FSAMP)
      CALL APINIT
      NSAMP2=2
   51 IF(NSAMP2.GE.NSAMP)GOTO 52
      NSAMP2=NSAMP2*2
      GOTO 51
   52 CONTINUE
C
C READ IN FLAGS FOR PROCESSES AND EXECUTION ORDER
C
      READ(1,1ØØØ)NPROC
      READ(1,1ØØØ)(UTLFLG(I),I=1,NUTIL)
      READ(1,1ØØØ)(IORD(I),I=1,NPROC)
C
C TRACE EDIT DATA
C
      IF(UTLFLG(1).EQ.Ø)GOTO 7Ø
      READ(1,1ØØØ)NFILED
      READ(1,1ØØØ)(IFILED(I),I=1,NFILED)
C
C EXP(Ø.2T) RAMP DATA
C
   7Ø IF(UTLFLG(2).EQ.Ø)GOTO 8Ø
      READ(1,1ØØØ)IAPLX
      CALL TEXRMP(EXPT,Ø,ITX,NSAMP,CONST,AEXPT)
      ITX=1
C
C T*EXP(Ø.2T) RAMP DATA
C
   8Ø IF(UTLFLG(3).EQ.Ø)GOTO 9Ø
      READ(1,1ØØØ)IAPLTX
      CALL TEXRMP(TEXPT,1,ITX,NSAMP,CONST,ATEXPT)
      ITX=1
C
```

```
C TV**2 RAMP
C
   9Ø IF(UTLFLG(4).EQ.Ø)GOTO 1ØØ
      READ(1,1ØØØ)IAPLTV,NLYR
      READ(1,11ØØ)(TØLYR(I),VLYR(I),I=1,NLYR)
      CALL TVRMP(TVSQ,TØLYR,VLYR,NLYR,ITX,NSAMP,NSTART,CONST,
     %FSAMP,ATVSQ)
C
C MUTE DATA
C
  1ØØ IF(UTLFLG(5).EQ.Ø)GOTO 11Ø
      READ(1,1ØØØ)NTAP,NMPTS
      READ(1,1ØØØ)(MNPOS(I),MSAMP(I),I=1,NMPTS)
      CALL COTAP(TAPER,NTAP,ATAP)
      DO 1Ø1 J=2,NMPTS
  1Ø1 MINC(J)=(MSAMP(J)-MSAMP(J-1))/(MNPOS(J)-MNPOS(J-1))
      MUTE(1)=MSAMP(1)
      IPOS=2
      DO 1Ø2 J=2,NFILES
      IF(J.LT.MNPOS(IPOS))GOTO 1Ø3
      MUTE(J)=MSAMP(IPOS)
      IPOS=IPOS+1
      GOTO 1Ø2
  1Ø3 MUTE(J)=MUTE(J-1)+MINC(IPOS)
  1Ø2 CONTINUE
C
C DECON INPUT
C
  11Ø IF(UTLFLG(6).EQ.Ø)GOTO 12Ø
      READ(1,1ØØØ)NFILT,ICONT,ISPIKE,INORM
      READ(1,11ØØ)WHITE
C
C BANDPASS FILTER
C
  12Ø IF(UTLFLG(7).EQ.Ø)GOTO 13Ø
      READ(1,11ØØ)FL,FU
      READ(1,11ØØ)FTPR1,FTPR2
      DFI=FLOAT(NSAMP2/2+1)/(FSAMP*5ØØ.)
      FL1=FL-FTPR1
      FU4=FU+FTPR2
      CALL BANDPS(ATBP,BPASS,FL1,FL,FU,FU4,DFI,NSAMP2)
      NTRANF=2*NSAMP2
      NBFILT=NSAMP2+1
      NBEXP=(2*NSAMP2)+2-NSAMP
C
C BANDREJECT FILTER
C
  13Ø IF(UTLFLG(8).EQ.Ø)GOTO 14Ø
      READ(1,11ØØ)FLR,FUR
      DFI=FLOAT(NSAMP2/2+1)/(FSAMP*5ØØ.)
      NTRANF=2*NSAMP2
      NRFILT=NSAMP2+1
      NBEXP=(2*NSAMP2)+2-NSAMP
      CALL BANDRJ(ATBR,BRJCT,FLR,FUR,DFI,NSAMP2)
```

```
C
C PREDICTIVE DECONVOLUTION
C
  14Ø IF(UTLFLG(9).EQ.Ø)GOTO 15Ø
      READ(1,1ØØØ)NPFILT,NLAG,IPNORM
      READ(1,11ØØ)PRWHIT
C
C TRACE NORMALISATION
C
  15Ø IF(UTLFLG(1Ø).EQ.Ø)GOTO 16Ø
      READ(1,1ØØØ)NRMFLG
C
C BLOCKING PARAMETERS
C
  16Ø CONTINUE
      IFED=1
      NBLKTR=NSAMP/128
      NBLKW=NBLKTR+1
      NBYTR=NBLKW*512
      NWDR=NBYTR/2
C
C START OF LOOP ON DIFFERENT FILES
C
      DO 2ØØ IFNUM=1,NFILES
      IFIL=IFNUM
      IF(INFLG.NE.Ø)GOTO 21Ø
C
C TAPE INPUT HANDLING
C
C
C EOT CHECK
C
      ITRY=1
  211 IF(ITRY.GT.3)GOTO 212
      IF(IEOTR.GE.Ø)GOTO 22Ø
  212 WRITE(7,14ØØ)TPDRR,IFIL
 14ØØ FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I4)
      WRITE(7,14Ø1)
 14Ø1 FORMAT(' ENTER NEW DRIVE NO:',$)
      READ(5,14Ø2)TPDRR
 14Ø2 FORMAT(I1)
      IEOTR=Ø
      IF(TPDRR.GT.2)STOP' EOT TERMINATION'
C
C READ FROM TAPE TO MEMORY
C
  22Ø CALL TAPSUB(-1,TPDRR,ISTAT,IFLEN,IFIL,DBUF,NBYTR,IEOTR)
      IF(ISTAT.LT.Ø)WRITE(2,15ØØ)IFIL
 15ØØ FORMAT(' RETRIES ON READ FAILED ON FILE',I5)
      IF(IEOTR.LT.Ø)GOTO 23Ø
C
C WIND OVER EOF MARK
C
      CALL TAPSUB(Ø,TPDRR,ISTAT, ,IFIL, , ,IEOTR)
```

```
Ø154      23Ø ITRY=ITRY+1
Ø156          IF(IHBLK(1).EQ."177777)GOTO 211
Ø157          GOTO 28Ø
        C
        C OPEN UP READING FILES
        C
Ø158      21Ø FSPECR=FNAMR(IFNUM)
Ø159          CALL CLOSEC(IDCH)
Ø16Ø          IF(LOOKUP(IDCH,FSPECR).LT.Ø)STOP'LOOKUP ERR'
Ø162          IF(IREADW(NWDR,DBUF,Ø,IDCH).LT.Ø)STOP'READW ERR'
Ø164      28Ø CONTINUE
        C
        C HEADER BLOCK MANAGEMENT
        C
Ø165          IBFREE=IHBLK(24)
Ø166          IHBLK(129+IBFREE)=3
Ø167          IBFREE=IBFREE+1
Ø168          IHBLK(129+IBFREE)=NPROC
Ø169          IBFREE=IBFREE+1
Ø17Ø          DO 215 J=1,NPROC
Ø171          IHBLK(129+IBFREE)=IORD(J)
Ø172          IBFREE=IBFREE+1
Ø173      215 CONTINUE
Ø174          IHBLK(24)=IBFREE
        C
        C OPEN UP OUTPUT FILES
        C
Ø175          IF(OUTFLG.EQ.Ø)GOTO 3ØØ
Ø177          FSPECW=FNAMO(IFNUM)
Ø178          IF(IENTER(IDCH1,FSPECW,NBLKW).LT.Ø)STOP'ENTER ERR2'
Ø18Ø          IF(IWRITW(256,IHBLK,Ø,IDCH1).LT.Ø)STOP'WRITW ERR'
Ø182          JBLK=1
Ø183      3ØØ CALL APPUT(SEISM,Ø,NSAMP,2)
Ø184          CALL APWD
Ø185          DO 4ØØ IPCNUM=1,NPROC
Ø186          GOTO(41Ø,43Ø,44Ø,45Ø,46Ø,47Ø,48Ø,
             X49Ø,5ØØ,51Ø)IORD(IPCNUM)
        C
        C PROCESS EDIT COMMANDS
        C
Ø187      41Ø IF(IFED.GT.NFILED)GOTO 4ØØ
Ø189          IF(IFILED(IFED).NE.IFIL)GOTO 4ØØ
Ø191          IFED=IFED+1
Ø192          CALL VCLR(Ø,1,NSAMP)
Ø193          CALL APWR
Ø194          CALL APGET(SEISM,Ø,NSAMP,2)
Ø195          CALL APWD
Ø196          GOTO 31Ø
        C
        C AMP RECOVERY FILTERS APPLICATION AND REMOVAL
        C
        C
        C EXP(Ø.2T) FILTER
        C
```

```
      43Ø IAPL=IAPLX
          CALL APPUTA(NSAMP,NSAMP,2,AEXPT)
          GOTO 455
C
C T*EXP(Ø.2T) FILTER
C
      44Ø IAPL=IAPLTX
          CALL APPUTA(NSAMP,NSAMP,2,ATEXPT)
          GOTO 455
C
C TV**2 FILTER
C
      45Ø IAPL=IAPLTV
          CALL APPUTA(NSAMP,NSAMP,2,ATVSQ)
C
C COMMON CODE
C
      455 CALL APWD
          IF(IAPL.EQ.Ø)CALL VMUL(Ø,1,NSAMP,1,Ø,1,NSAMP)
          IF(IAPL.NE.Ø)CALL VDIV(NSAMP,1,Ø,1,Ø,1,NSAMP)
          CALL APWR
          GOTO 4ØØ
C
C MUTE APPLICATION
C
      46Ø CALL APPUTA(NSAMP,NTAP,2,ATAP)
          CALL APWD
          MNCLR=MUTE(IFIL)
          CALL VCLR(Ø,1,MNCLR)
          CALL VMUL(MNCLR,1,NSAMP,1,MNCLR,1,NTAP)
          CALL APWR
          GOTO 4ØØ
C
C SPIKE DECON
C
      47Ø CALL SPIKE(NSAMP,NSAMP2,NFILT,WHITE,INORM,ISPIKE)
          GOTO 4ØØ
C
C BANDPASS FILTER
C
      48Ø CALL APPUTA(41ØØ,NBFILT,2,ATBP)
          CALL APWD
          CALL VCLR(NSAMP,1,NBEXP)
          CALL RFFT(Ø,NTRANF,1)
          CALL RFFTSC(Ø,NTRANF,3,1)
          CALL VMUL(Ø,2,41ØØ,1,Ø,2,NBFILT)
          CALL VMUL(1,2,41ØØ,1,1,2,NBFILT)
          CALL RFFTSC(Ø,1,NTRANF,-3,Ø)
          CALL RFFT(Ø,NTRANF,-1)
          CALL APWR
          GOTO 4ØØ
C
C BANDREJECT FILTER
C
```

```
Ø232     49Ø CALL APPUTA(41ØØ,NRFILT,2,ATBR)
Ø233         CALL APWD
Ø234         CALL VCLR(NSAMP,1,NBEXP)
Ø235         CALL RFFT(Ø,NTRANF,1)
Ø236         CALL RFFTSC(Ø,NTRANF,3,1)
Ø237         CALL VMUL(Ø,2,41ØØ,1,Ø,2,NRFILT)
Ø238         CALL VMUL(1,2,41ØØ,1,1,2,NRFILT)
Ø239         CALL RFFTSC(Ø,NTRANF,-3,Ø)
Ø24Ø         CALL RFFT(Ø,NTRANF,-1)
Ø241         CALL APWR
Ø242         GOTO 4ØØ
       C
       C PREDICT DECON
       C
Ø243     5ØØ CALL PRDICT(NSAMP,NSAMP2,NPFILT,PRWHIT,IPNORM,NLAG)
Ø244         GOTO 4ØØ
       C
       C NORMALISATION
       C
Ø245     51Ø IF(NRMFLG.EQ.Ø)GOTO 515
Ø247         CALL MAXMGV(Ø,1,2Ø5Ø,NSAMP)
Ø248         GOTO 516
Ø249     515 CALL SVESQ(Ø,1,2Ø5Ø,NSAMP)
Ø25Ø         CALL VSQRT(2Ø5Ø,1,2Ø5Ø,1,1)
Ø251     516 CALL VDIV(2Ø5Ø,Ø,Ø,1,Ø,1,NSAMP)
Ø252         CALL APWR
Ø253     4ØØ CONTINUE
       C
       C END OF PROCESS LOOP
       C
Ø254         CALL APWAIT
Ø255         CALL APGET(SEISM,Ø,NSAMP,2)
Ø256         CALL APWD
Ø257     31Ø IF(OUTFLG.EQ.Ø)GOTO 32Ø
Ø259         IF(IWRITW(2*NSAMP,SEISM,JBLK,IDCH1).LT.Ø)STOP'WRITW ERR2'
Ø261         CALL CLOSEC(IDCH1)
Ø262         GOTO 2ØØ
       C
       C OUTPUT TO TAPE
       C
Ø263     32Ø IFLEN=NBLKW
Ø264         CALL TAPSUB(1,TPDRW,ISTAT,IFLEN,IFIL,DBUF,NBYTR,IEOTW)
       C
       C CHECK FOR ERRORS
       C
Ø265         IF(IEOTW.GE.Ø)GOTO 25Ø
Ø267         WRITE(7,16ØØ)TPDRW,IFIL
Ø268    16ØØ FORMAT(' EOT ON DRIVE:',I2,' FILE NO:',I4)
Ø269         WRITE(7,16Ø1)
Ø27Ø    16Ø1 FORMAT(' ENTER NEW WRITE DRIVE NO:',$)
Ø271         READ(5,14Ø2)TPDRW
Ø272         IEOTW=Ø
Ø273         IF(TPDRW.GT.2)STOP'EOT TERMINATE'
Ø275     25Ø IF(ISTAT.GE.Ø)GOTO 2ØØ
```



```
Ø277         WRITE(2,17ØØ) IFIL
Ø278    17ØØ FORMAT(' FATAL ERROR ON WRITE FILE NO',I5)
Ø279         STOP' WRITE ERROR'
Ø28Ø     2ØØ CONTINUE
Ø281         CALL CLOSEC(IDCH)
Ø282         CALL CLOSEC(IDCH1)
Ø283         STOP'NORMAL TERMINATION'
Ø284         END
```

```
ØØØ1        SUBROUTINE TEXRMP(FILT,IFTYP,IFLG,NSAMP,CONST,AFILT,FS)
      C
      C IF ITYP=Ø THIS ROUTINE PRODUCES AN EXP(Ø.2T) ARRAY
      C    NSAMP LONG IN FILT
      C IF ITYP=1 T*EXP(Ø.2T)PRODUCED
      C
      C    CONST(1)=NSART
      C    CONST(2)=Ø.2
      C    CONST(3)=1.Ø/(1ØØØ.Ø*FSAMP)
      C
ØØØ2        VIRTUAL FILT(2Ø48)
ØØØ3        DIMENSION CONST(3)
ØØØ4        IF(IFLG.NE.Ø)GOTO 1Ø
      C
      C FORM THE T RAMP
      C
ØØØ6        CALL APWAIT
ØØØ7        CALL APPUT(CONST,8189,3,2)
ØØØ8        CALL APWD
ØØØ9        CALL VCLR(Ø,1,NSAMP)
ØØ1Ø        CALL VRAMP(8189,8191,Ø,1,NSAMP)
      C
      C FORM EXP(Ø.2T)
      C
ØØ11        CALL VSMUL(Ø,1,819Ø,NSAMP,1,NSAMP)
ØØ12        CALL VEXP(NSAMP,1,NSAMP,1,NSAMP)
ØØ13        IF(IFTYP.EQ.Ø)GOTO 2Ø
      C
      C FORM T*EXP(Ø.2T)
      C
ØØ15     1Ø CALL VMUL(NSAMP,1,Ø,1,NSAMP,1,NSAMP)
ØØ16     2Ø CALL APWR
ØØ17        CALL APGETA(NSAMP,NSAMP,2,AFILT)
ØØ18        RETURN
ØØ19        END
```

```
0001          SUBROUTINE TVRMP(FILT,TØLYR,VLYR,NLYR,IFLG,NSAMP,NSTART,
             %CONST,FSAMP,AFILT)
        C
        C THIS ROUTINE PRODUCES A TV**2 RAMP
        C FROM VELOCITY INFO IN TØLYR,VLYR
        C
0002          VIRTUAL FILT(2048)
0003          DIMENSION TØLYR(20),VLYR(20),CONST(3)
        C
        C CHECK IF T RAMP ALREADY FORMED
        C
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
        C
0011       10 N1=1
0012          N2=IFIX(FSAMP*TØLYR(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*TØLYR(J))-NSTART
0021          DELV=(VLYR(J)-VLYR(J-1))/(N2-N1+2)
0022          V=VLYR(J-1)
0023          DO 30 I=N1,N2
0024          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
        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
```

```
ØØØ1        SUBROUTINE COTAP(TAPER,NTAP,ATAP)
      C
      C THIS ROUTINE PRODUCES A COSINE TAPER NTAP
      C SAMPLES LONG
      C
ØØØ2        VIRTUAL TAPER(512)
ØØØ3        CALL APWAIT
ØØØ4        CALL APPUT(1.Ø/FLOAT(NTAP),Ø,1,2)
ØØØ5        CALL VCLR(1,1,NTAP)
ØØØ6        CALL VTSADD(1,1,23Ø6,1,1,1)
ØØØ7        CALL VTSMUL(Ø,1,23Ø6,Ø,1,1)
ØØØ8        CALL VRAMP(1,Ø,Ø,1,NTAP)
ØØØ9        CALL VCOS(Ø,1,Ø,1,NTAP)
ØØ1Ø        CALL VTSADD(Ø,1,2Ø49,Ø,1,NTAP)
ØØ11        CALL VTSMUL(Ø,1,2327,Ø,1,NTAP)
ØØ12        CALL APWR
ØØ13        CALL APGETA(Ø,NTAP,2,ATAP)
ØØ14        CALL APWD
ØØ15        RETURN
ØØ16        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)
0017          CALL APPUT(RNTAP,0,1,2)
0018          CALL VCLR(1,1,NTAP)
0019          CALL VTSADD(1,1,2306,1,1,1)
0020          CALL VTSMUL(0,1,2306,0,1,1)
0021          CALL VRAMP(1,0,0,1,NTAP)
0022          CALL VCOS(0,1,0,1,NTAP)
0023          CALL VTSADD(0,1,2049,0,1,NTAP)
0024          CALL VTSMUL(0,1,2327,0,1,NTAP)
0025          IF(I.EQ.1)CALL VMOV(0,1,2050,1,NTAP)
0027          IF(I.EQ.2)CALL VMOV(0,1,4100,1,NTAP)
0029       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
```

```
ØØØ1          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
ØØØ2          VIRTUAL BRJCT(2Ø49)
        C
        C SET UP CONSTANTS
        C
ØØØ3          N1=2.Ø*F1*DFI
ØØØ4          N2=2.Ø*F2*DFI
ØØØ5          NTAP=N2-N1
ØØØ6          CALL APWAIT
        C
        C SET UP FILTER IN AP
        C
ØØØ7          CALL VCLR(Ø,1,NTAP)
ØØØ8          NFILT=NSAMP+1
ØØØ9          FTAP=1.Ø/FLOAT(NTAP)
ØØ1Ø          CALL APPUT(FTAP,1,1,2)
ØØ11          CALL APWD
ØØ12          CALL VTSMUL(1,1,2317,1,1,1)
ØØ13          CALL VRAMP(Ø,1,Ø,1,NTAP)
ØØ14          CALL VCOS(Ø,1,Ø,1,NTAP)
ØØ15          CALL VTSADD(Ø,1,2Ø49,Ø,1,NTAP)
ØØ16          CALL VTSMUL(Ø,1,2327,Ø,1,NTAP)
ØØ17          CALL VMOV(Ø,1,4Ø96,1,NTAP)
        C
        C SET UP FULL FILTER NOW TAPER FINISHED
        C
ØØ18          CALL VCLR(Ø,1,NFILT)
ØØ19          CALL VTSADD(Ø,1,2Ø49,Ø,1,NFILT)
ØØ2Ø          CALL VMUL(N1,1,4Ø96,1,N1,1,NTAP)
ØØ21          CALL APWR
ØØ22          CALL APGETA(Ø,NFILT,2,ATBR)
ØØ23          CALL APWD
ØØ24          RETURN
ØØ25          END
```

```
0001          SUBROUTINE SPIKE(NSAMP,NSAMP2,ILENTH,WHITE,IFLAG,ISPIKE)
        C
        C WIENER SPIKING FILTER ROUTINE
        C  NSAMP=DATA LENGTH
        C  NSAMP2=NEAREST POWER OF 2 TO NSAMP
        C  ILENTH=FILTER LENGTH
        C  IFLAG=TRACE NORMALISATION FLAG
        C        0 NO NORMALISATION
        C        1 FILTER UNIT ENERGY
        C        2 EQUAL INPUT-OUTPUT ENERGY
        C  ISPIKE SPIKE POSITION
        C
0002          NTRAN=2*NSAMP2
0003          NCLR=NTRAN-NSAMP
0004          NFCLR=NTRAN-ILENTH
0005          NTRAN2=NTRAN+2
0006          NM1=NSAMP2-1
0007          I6=NTRAN+ILENTH
0008          I7=I6+ILENTH
0009          I8=I7+ILENTH
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
```

```
ØØ29        CALL VCLR(I6,1,ILENTH)
ØØ3Ø        CALL VTSADD(ISP,1,2Ø49,ISP,1,1)
      C
      C SOLVE EQNS
      C
ØØ31        CALL WIENER(ILENTH,NTRAN,I6,I7,I8,1)
ØØ32        CALL APCHK(IER)
ØØ33        IF(IER.NE.Ø)STOP'LEVINSON FAILURE'
ØØ35        CALL VMOV(I7,1,NTRAN,1,ILENTH)
      C
      C NORMALISE FILTER IF ASKED FOR
      C
ØØ36        IF(IFLAG.NE.1)GOTO 2Ø
ØØ38        CALL SVESQ(NTRAN,1,I6,ILENTH)
ØØ39        CALL VSQRT(I6,1,I6,1,1)
ØØ4Ø        CALL VDIV(I6,Ø,NTRAN,1,NTRAN,1,ILENTH)
      C
      C APPLY FILTER
      C
ØØ41     2Ø CALL VCLR(I6,1,NFCLR)
ØØ42        CALL RFFT(NTRAN,NTRAN,1)
ØØ43        CALL VMUL(Ø,1,NTRAN,1,Ø,1,2)
ØØ44        CALL CVMUL(2,2,NTRAN2,2,2,2,NM1,1)
ØØ45        CALL RFFTSC(Ø,NTRAN,Ø,-1)
ØØ46        CALL RFFT(Ø,NTRAN,-1)
      C
      C DO SCALING IF REQD
      C
ØØ47        IF(IFLAG.NE.2)GOTO 3Ø
ØØ49        CALL SVESQ(Ø,1,8191,NSAMP)
ØØ5Ø        CALL VSQRT(8191,1,8191,1,1)
ØØ51        CALL APWR
ØØ52        CALL APPUT(EN,819Ø,1,2)
ØØ53        CALL APWD
ØØ54        CALL VDIV(8191,1,819Ø,1,819Ø,1,1)
ØØ55        CALL VSMUL(Ø,1,819Ø,Ø,1,NSAMP)
ØØ56     3Ø CALL APWR
ØØ57        RETURN
ØØ58        END
```

```
      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
      NTRAN=2*NSAMP2
      NCLR=NTRAN-NSAMP
      NM1=NSAMP2-1
      NTRAN2=NTRAN+2
      NLG=NTRAN+NLAG
      NFILT=ILENTH+NLAG
      I7=NLG+ILENTH
      I8=I7+ILENTH
C
C GET INPUT TRACE ENERY
C
      IF(IFLAG.NE.2)GOTO 10
      CALL SVESQ(0,1,8191,NSAMP)
      CALL VSQRT(8191,1,8191,1,1)
      CALL APGET(EN,8191,1,2)
      CALL APWD
C
C GET AUTOCORRELATION FUNCTION
C
   10 CALL VCLR(NSAMP,1,NCLR)
      CALL RFFT(0,NTRAN,1)
      CALL VMOV(0,1,NTRAN,1,NTRAN)
      CALL VMUL(NTRAN,1,NTRAN,1,NTRAN,1,2)
      CALL CVMAGS(NTRAN2,2,NTRAN2,2,NM1)
      CALL VCLR(NTRAN+3,2,NM1)
      CALL RFFTSC(NTRAN,NTRAN,-1,-1)
      CALL RFFT(NTRAN,NTRAN,-1)
C
C NOW WHITEN IT
C
      CALL APWR
      CALL APPUT(WHITE,8191,1,2)
      CALL VSMA(NTRAN,1,8191,NTRAN,1,NTRAN,1,1)
C
C NOW SOLVE EQNS
C
      CALL WIENER(ILENTH,NTRAN,NLG,I7,I8,1)
      CALL APCHK(IER)
      IF(IER.NE.0)STOP 'LEVINSON FAILURE'
      IF(IFLAG.NE.1)GOTO 20
      CALL SVESQ(I7,1,I8,ILENTH)
```

```
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,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
```

```
      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,Ø 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,"1Ø,"2Ø,"4Ø,"1ØØ,"2ØØ/
      DATA SDSCOM/"Ø,"1,"2,"3,"4,"5,"6,"7/
      DATA ERRS/"377,"377,"377,"377,"377,"377,"377,"377/
      ITRY=Ø
      IF(ICOM) 1Ø,3Ø,2Ø
C
C SECTION CONTROLLING A READ
C
C
C  CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
C
   1Ø ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
      NBUF=NBYT
      CALL TREAD(BUF,NBUF,ISTAT,IDRV)
      IF(ISTAT.EQ.Ø)RETURN
C
C ERROR DETECTED ON READ
C
      ISTATI=ISTAT
      GOTO 4Ø
C
C IF SHORT RECORD FOUND REREAD TAPE
C
   5Ø ITMP=ISTATI.AND.MASK(6)
      IF(ITMP.NE.Ø)GOTO 1Ø
      ITMP=ISTATI.AND.MASK(2)
      IF(ITMP.EQ.Ø)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
      WRITE(2,2Ø1Ø)IFNUM
 2Ø1Ø FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
      IF(ITRY.GE.2)GOTO 13Ø
      ECOM(1)=SDSCOM(6)
      ECOM(2)=1
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(ECOM,ESTAT, , )
      GOTO 1Ø
```

```
C
C WRITE SECTION
C
   2Ø ITRY=ITRY+1
      IF(ITRY.GT.3)GOTO 13Ø
      NBUF=NBYT
      IFLEN=(ILEN+3)/4
      IF(IFLEN.LT.2)IFLEN=2
      IPAD=(IFLEN*2Ø48)-NBUF
      CALL TWRIT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
      IF(ISTAT.EQ.Ø)RETURN
C
C WRITE ERROR DETECTED
C
      ISTATI=ISTAT
      GOTO 4Ø
   7Ø ITMP=ISTATI.AND.MASK(6)
      ITMPI=ISTATI.AND.MASK(2)
      IF(ITMP.EQ.Ø.AND.ITMPI.EQ.Ø)RETURN
C
C REPORT AND RETRY
C
      WRITE(2,2Ø2Ø)IFNUM
 2Ø2Ø FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
      ECOM(1)=SDSCOM(6)
      ECOM(2)=2
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(ECOM,ESTAT, , )
      NBUF=8
      IPAD=(IFLEN*2Ø48)-NBUF
      CALL TWRIT(ERRS,NBUF,ESTAT,IPAD,IFLEN,IDRV)
      GOTO 2Ø
C
C WIND FOWARD ONE FILE
C
   3Ø COM(1)=SDSCOM(5)
      COM(2)=1
      COM(3)=IDRV
      COM(4)=Ø
      CALL SDS1Ø(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.Ø)GOTO 4Ø
C
C IF ISTAT=Ø 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)=Ø
      CALL SDS1Ø(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
   4Ø WRITE(2,1Ø1Ø)ISTATI,IFNUM
 1Ø1Ø FORMAT(' STATUS=',I3,' FILE NO=',I4)
      ISTATI=ISTATI.AND..NOT.MASK(4)
      ITMP=ISTATI.AND.MASK(5)
      IF(ITMP.EQ.Ø)GOTO 8Ø
   9Ø ECOM(1)=SDSCOM(1)
      ECOM(2)=Ø
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(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.Ø)GOTO 9Ø
      IF(ICOM) 1Ø,3Ø,2Ø
C
C TAPE OFFLINE
C
   8Ø ITMP=ISTATI.AND.MASK(1)
      IF(ITMP.EQ.Ø)GOTO 1ØØ
      TYPE 1ØØ1,IDRV
 1ØØ1 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
  11Ø ECOM(1)=SDSCOM(1)
      ECOM(2)=Ø
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(ECOM,ESTAT, , )
      ESTATI=ESTAT
      ITMP=ESTATI.AND.MASK(1)
      IF(ITMP.NE.Ø)GOTO 11Ø
      IF(ICOM) 1Ø,3Ø,2Ø
C
C EOT
C
  1ØØ ITMP=ISTATI.AND.MASK(3)
      IF(ITMP.EQ.Ø)GOTO 12Ø
```



```
      TYPE 1ØØ2,IDRV
 1ØØ2 FORMAT(' EOT ON DRIVE ',I1)
      IEOT=-1
      RETURN
  12Ø IF(ICOM) 5Ø,35,7Ø
C
C ERROR EXIT RETURN
C
  13Ø ISTATI=-1
      RETURN
      END
```

Post-Stack Trace Mix :- MPTMIX

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

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

Input Parameters

READ(1,1000)NFILES,NSAMP,TPDRR,TPDRW,INFLG,OUTFLG

1000 FORMAT(6I5)

NFILES...Number of files to process

NSAMP....Number of samples per trace

TPDRR....Input tape drive

TPDRW....Output tape drive

INFLG....Input flag

0 - Input from tape

1 - Input from disc

OUTFLG...Output flag

0 - Output to tape

1 - Output to disc

IF(INFLG.NE.0)READ(1,1100)FSPECR

1100 FORMAT(3A4)

FSPECR...Input files, 1 to NFILES

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

FSPECW...output files, 1 to NFILES

```
C
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
      REAL*8 FSPECR(200),FSPECW(200),FBUF
      REAL*4 SEIS(2048),FNBUF(3)
      INTEGER*2 BUF(4352),IHBLK(256),IHBLKS(256),OUTFLG
      EQUIVALENCE (BUF(1),IHBLK(1)),(BUF(257),SEIS(1))
      LOGICAL*1 ISTAT,TPDRR,TPDRW
      DATA DEV/3RRK /
C
C SET UP I/O PARAMETERS
C
      IEOTR=0
      IEOTW=0
      IRD=IGETC()
      IWRT=IGETC()
      IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
      CALL ASSIGN(1,'DK2:MPTMIX.DAT',14)
      CALL ASSIGN(2,'DK2:MPTMIX.LOG',14)
C
C GET INPUT DATA
C
      READ(1,1000) NFILES,NSAMP,TPDRR,TPDRW,INFLG,OUTFLG
 1000 FORMAT(6I5)
      IF(INFLG.EQ.0)GOTO 10
      DO 20 J=1,NFILES
      READ(1,1001)FNBUF
 1001 FORMAT(3A4)
      CALL IRAD50(12,FNBUF,FBUF)
      FSPECR(J)=FBUF
   20 CONTINUE
   10 IF(OUTFLG.EQ.0)GOTO 30
      DO 40 J=1,NFILES
      READ(1,1001)FNBUF
      CALL IRAD50(12,FNBUF,FBUF)
      FSPECW(J)=FBUF
   40 CONTINUE
   30 CONTINUE
C
C SET UP CONSTANTS AND INIT AP
C
      CALL APINIT
      CALL VCLR(0,1,3*NSAMP)
      NBLKTR=NSAMP/128
      NBLKW=NBLKTR+1
      NBYTR=NBLKW*512
      NWDR=NBYTR/2
      NOPS=NFILES+1
      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:',$)
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:',$)
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(IWRITW(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(IWRITW(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'
```



```
0074        END
```

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 8Ø
       C
       C  THIS PROGRAM TAKES THE OUTPUT FROM FD MIGRATION
       C  AND PUTS IT BACK INTO TIME SEQUENTIAL DATA
       C
       C
ØØØ1         VIRTUAL BUFFER(16384)
ØØØ2         REAL*4 BUFFER,INBUF(128),OUTBUF(128),FBUF(3)
ØØØ3         REAL*8 FSPECW,FSPECR
ØØØ4         INTEGER*2 IAD(128),IWRTB(128)
ØØØ5         DATA DEV/3RRK /
       C
       C INPUT SET UP
       C
ØØØ6         IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
ØØØ8         IRD=IGETC()
ØØØ9         IWRT=IGETC()
ØØ1Ø         CALL ASSIGN(1,'DK2:MPUSLC.DAT',14)
       C
       C READ IN DATA
       C
ØØ11         READ(1,1ØØØ)NCHAN,NSAMP
ØØ12    1ØØØ FORMAT(2I5)
ØØ13         READ(1,1ØØ1)FBUF
ØØ14    1ØØ1 FORMAT(3A4)
ØØ15         CALL IRAD5Ø(12,FBUF,FSPECR)
ØØ16         READ(1,1ØØ1)FBUF
ØØ17         CALL IRAD5Ø(12,FBUF,FSPECW)
       C
       C SET UP CONSTANTS
       C
ØØ18         NCHANW=Ø
ØØ19      1Ø NCHANW=NCHANW+128
ØØ2Ø         IF(NCHAN.GT.NCHANW)GOTO 1Ø
ØØ22         NBLANK=NCHANW-NCHAN
ØØ23         NBST=NBLANK/2
ØØ24         ITIM=NCHANW/128
ØØ25         NBLKW=NSAMP/128
ØØ26         NFILW=(NCHANW*NBLKW)+1
       C
       C SET UP BUFFER ADDRESSES
       C
ØØ27         IT=1
ØØ28         DO 2Ø J=1,128
ØØ29         IAD(J)=IT
ØØ3Ø      2Ø IT=IT+128
       C
       C CLEAR STORE BUFFER
       C
ØØ31         DO 3Ø J=1,16384
ØØ32      3Ø BUFFER(J)=Ø.Ø
       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 velcity 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(IENTER(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
```

```
Ø12Ø          DO 1ØØ ITORCT=1,ITORLM
Ø121          LIMIT=LIMIT-ITOR
        C
        C ZERO X ARRAYS
        C
Ø122          CALL VCLR(AØ,1,NCHANW)
Ø123          CALL APWR
Ø124          DO 11Ø I=1,4
Ø125          IST(I)=I
Ø126          FAD=ADXST(I)
Ø127          CALL APGETA(AØ,NCHANW,2,FAD)
Ø128      11Ø CONTINUE
        C
        C READ IN VELOCITIES
        C
Ø129          IBLKV=Ø
Ø13Ø          DO 12Ø L=1,NV
Ø131          IF(IREADA(IVRT,ITRLM2,IBLKV,ADVEL).LT.Ø)STOP'READV ERROR'
Ø133          IBLKV=IBLKV+NVSIZ
Ø134          CALL IWAIT(IVRT)
Ø135          V(L)=VSAVE(ITORCT)
Ø136      12Ø CONTINUE
        C
        C GEN V SLICE
        C
Ø137          CALL APPUT(V,AØ,NV,2)
Ø138          DO 13Ø I=1,NVM
Ø139      13Ø VINT(I)=(V(I+1)-V(I))/(IV(I+1)-IV(I))
Ø14Ø          CALL APPUT(VINT,A1,NVM,2)
Ø141          CALL APWD
        C
Ø142          DO 14Ø I=1,NVM
Ø143          NVD=IV(I+1)-IV(I)+1
Ø144          CALL VRAMP(AØ+I-1,A1+I-1,A2+IV(I)-1,1,NVD)
Ø145      14Ø CONTINUE
Ø146          CALL VFILL(A2+IV(1)-1,A2,1,NBST)
Ø147          CALL VFILL(A2+IV(NV)-1,A2+IV(NV),1,NEND)
Ø148          CALL VSQ(A2,1,A2,1,NCHANW)
        C
        C GEN A
        C
Ø149          CALL APPUT(ACOF,AØ,1,2)
Ø15Ø          CALL APWD
Ø151          CALL VSMUL(A2,1,AØ,AØ,1,NCHANW)
        C
        C SET UP NECESSARY COEFFS FROM SAVE
        C
Ø152          CALL APWR
Ø153          CALL APPUTA(A1,NCHANW,2,ADAPSV(1))
Ø154          CALL APPUTA(A2,NCHANW,2,ADAPSV(2))
Ø155          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)
```

```
      DO 210 I=1,3
      IA1=IA2+1
      IA2=IA1+1
      CALL APWR
      CALL APPUTA(A0,NCHANW,2,ADRHS(IA1))
      CALL APPUTA(A1,NCHANW,2,ADRHS(IA2))
      FST=ADXST(IST(I))
      CALL APPUTA(A3,NCHANW,2,FST)
      CALL APWD
      CALL VMUL(A1,1,A3,1,A4,1,NCHANW)
      CALL VMA(IA01,1,A3,1,IA41,1,IA41,1,NCHWM1)
      CALL VMA(A0,1,IA31,1,A4,1,A4,1,NCHWM1)
      CALL VADD(A4,1,A7,1,A7,1,NCHANW)
  210 CONTINUE
C
C DO SOLUTION TO EQN
C
      CALL VCLR(A3,1,NCHANW)
      CALL VCLR(A4,1,NCHANW)
      CALL APWR
      CALL APPUTA(A0,NCHANW,2,ADASV)
      CALL APPUTA(A1,NCHANW,2,ADBSV)
      CALL APPUTA(A2,NCHANW,2,ADCSV)
      CALL APWD
      CALL SOLVE(A0,1,A1,1,A7,1,A2,1,A3,1,A4,1,NCHANW)
      CALL APWR
      CALL APGSP(IER,15)
      IF(IER.NE.0)STOP'SOLVE FAILURE'
      FST=ADXST(IST(4))
      CALL APGETA(A4,NCHANW,2,FST)
      FAD=ADXDIS(IST(4))
      CALL APWD
      IF(IWRITA(IRD,NCHWR2,IBLKR,FAD).LT.0)STOP'WRITE ERROR'
C
C TURN AROUND VECTORS
C
      IST(5)=IST(1)
      IST(6)=IST(2)
      DO 220 I=1,4
  220 IST(I)=IST(I+2)
      ISTB=ISTB-1
C
C END OF LOOP
C
  200 CONTINUE
      CALL IWAIT(IRD)
  100 CONTINUE
      CALL CLOSEC(IRD)
      STOP'NORMAL TERMINATION'
      END
```

```
      C
      C M J POULTER NOV 8Ø
      C
      C THIS IS A PROGRAM FOR
      C 45 DEGREE FINITE DIFFERENCE MIGRATION
      C
ØØØ1        VIRTUAL ASAVE(1Ø24),BSAVE(1Ø24),CSAVE(1Ø24),
           #RHS(1Ø24Ø),APSAVE(2Ø48),IBLK(2Ø48),VSAVE(2Ø48),
           #XSTOR(6144)
ØØØ2        REAL*8 FSPECR,FSVEL
ØØØ3        REAL*4 VINT(1ØØ),ADXST(6),V(1ØØ),VRMS(2Ø),FBUF(3),
           #ADRHS(1Ø),ADAPSV(2),ADXDIS(6)
ØØØ4        INTEGER*2 IV(1ØØ),IST(8),AØ,A1,A2,A3,A4,A5,A6,A7,
           #IT(2Ø)
ØØØ5        DATA AØ/Ø/,A1/1Ø24/,A2/2Ø48/,A3/3Ø72/,A4/4Ø96/,
           #A5/512Ø/,A6/6144/,A7/7168/
ØØØ6        DATA DEV/3RRK /,FSVEL/12RDK4MPVTMPDAT/
      C
      C SET UP CONSTANTS AND READ IN DATA
      C
ØØØ7        CALL APINIT
ØØØ8        CALL ASSIGN(1,'DK2:MPFMIG.DAT',14)
ØØØ9        IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
ØØ11        IVRT=IGETC()
ØØ12        IRD=IGETC()
      C
      C READ IN DATA
      C
ØØ13        READ(1,1ØØØ) NSAMP,MSAMP,NTRACE,NV,NVELS
ØØ14   1ØØØ FORMAT(1ØI5)
ØØ15        READ(1,1ØØ1)DX,DT,DTOR
ØØ16   1ØØ1 FORMAT(3F1Ø.Ø)
ØØ17        READ(1,1ØØØ)(IV(I),I=1,NV)
ØØ18        READ(1,1ØØ2)FBUF
ØØ19   1ØØ2 FORMAT(3A4)
ØØ2Ø        CALL IRAD5Ø(12,FBUF,FSPECR)
      C
      C SET UP VM ADDRESSES
      C
ØØ21        IPOS=1
ØØ22        DO 1Ø J=1,1Ø
ØØ23        ADRHS(J)=APGAD(RHS(IPOS))
ØØ24        IPOS=IPOS+1Ø24
ØØ25     1Ø CONTINUE
ØØ26        ADVEL=ADGET(VSAVE(1))
ØØ27        ADAPSV(1)=APGAD(APSAVE(1))
ØØ28        ADAPSV(2)=APGAD(APSAVE(1Ø25))
ØØ29        ADASV=APGAD(ASAVE(1))
ØØ3Ø        ADBSV=APGAD(BSAVE(1))
ØØ31        ADCSV=APGAD(CSAVE(1))
ØØ32        IB=1
ØØ33        DO 15 I=1,6
ØØ34        ADXST(I)=APGAD(XSTOR(IB))
ØØ35        ADXDIS(I)=ADGET(XSTOR(IB))
```

```
ØØ36           IB=IB+1Ø24
ØØ37        15 CONTINUE
      C
      C SET UP CONSTANTS
      C
ØØ38           NCHANW=Ø
ØØ39        2Ø NCHANW=NCHANW+128
ØØ4Ø           IF(NTRACE.GT.NCHANW)GOTO 2Ø
ØØ42           IF(NCHANW.GT.1Ø24)STOP' TOO MANY TRACES TO MIGRATE'
ØØ44           ITIM=NCHANW/128
ØØ45           NBST=(NCHANW-NTRACE)/2
ØØ46           NEND=NCHANW-NTRACE-NBST
      C
      C SET UP CONSTANTS FOR LOOPS
      C
ØØ47           ITOR=DTOR/DT
ØØ48           ITORLM=MSAMP/ITOR
ØØ49           LIMIT=NSAMP
ØØ5Ø           NVM=NV-1
ØØ51           ACOF=(Ø.279*DT*DT)/(4.Ø*DX*DX)
ØØ52           BCOF=(DTOR*DT)/(32.Ø*DX*DX)
ØØ53           IBLK(1)=1
ØØ54           DO 3Ø L=2,NSAMP
ØØ55        3Ø IBLK(L)=IBLK(L-1)+ITIM
ØØ56           NVSIZ=Ø
ØØ57        4Ø NVSIZ=NVSIZ+128
ØØ58           IF(ITORLM.GT.NVSIZ)GOTO 4Ø
ØØ6Ø           NVSIZ=NVSIZ/128
ØØ61           NCHWM1=NCHANW-1
ØØ62           NCHWR2=NCHANW*2
ØØ63           ITRLM2=ITORLM*2
ØØ64           IAØ1=AØ+1
ØØ65           IA31=A3+1
ØØ66           IA41=A4+1
      C
      C SET UP FILES
      C
ØØ67           IF(LOOKUP(IRD,FSPECR).LT.Ø)STOP'LOOKUP ERROR'
ØØ69           IFILV=NV*NVSIZ
ØØ7Ø           IF(IENTER(IVRT,FSVEL,IFILV).LT.Ø)STOP'ENTER ERROR'
      C
      C SET UP I AND I/12
      C
ØØ72           CALL VCLR(A4,1,NCHANW)
ØØ73           CALL VTSADD(A4,1,2Ø49,A4,1,NCHANW)
ØØ74           CALL VMOV(A4,1,A5,1,NCHANW)
ØØ75           CALL VTSMUL(A4,1,2331,A4,1,NCHANW)
ØØ76           CALL VTSMUL(A4,1,"4427,A4,1,NCHANW)
ØØ77           CALL APWR
ØØ78           CALL APGETA(A4,NCHANW,2,ADAPSV(1))
ØØ79           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
ØØ8Ø          IBLKV=Ø
ØØ81          DO 5Ø LV=1,NV
ØØ82          IV(LV)=IV(LV)+NBST
      C
      C READ IN VELS
      C
ØØ83          DO 6Ø LL=1,NVELS
ØØ84          READ(1,1ØØ3)T,VRMS(LL)
ØØ85     1ØØ3 FORMAT(2F1Ø.Ø)
ØØ86          IT(LL)=T/DTOR+1
ØØ87       6Ø CONTINUE
      C
      C DO LINEAR INTERP ON RMS VELS
      C
ØØ88          N1=1
ØØ89          N2=IT(1)
ØØ9Ø          DO 55 LI=N1,N2
ØØ91       55 VSAVE(LI)=VRMS(1)
ØØ92          IF(NVELS.EQ.1)GOTO 65
ØØ94          DO 7Ø LJ=2,NVELS
ØØ95          N1=N2+1
ØØ96          N2=IT(LJ)
ØØ97          VT=VRMS(LJ-1)
ØØ98          DELV=(VRMS(LJ)-VT)/(N2-N1-1)
ØØ99          DO 75 LT=N1,N2
Ø1ØØ          VSAVE(LT)=VT
Ø1Ø1          VT=VT+DELV
Ø1Ø2       75 CONTINUE
Ø1Ø3       7Ø CONTINUE
Ø1Ø4       65 CONTINUE
Ø1Ø5          N1=N2+1
Ø1Ø6          N2=ITORLM+1
Ø1Ø7          DO 8Ø LL=N1,N2
Ø1Ø8       8Ø VSAVE(LL)=VRMS(NVELS)
      C
      C CHANGE INTO INT VELS
      C
Ø1Ø9          VTP1=Ø.Ø
Ø11Ø          VTP2=VSAVE(1)*VSAVE(1)
Ø111          DO 9Ø LINT=1,ITORLM
Ø112          VTP1=VTP2
Ø113          VTP2=VSAVE(LINT+1)*VSAVE(LINT+1)*(LINT+1)
Ø114          VSAVE(LINT)=SQRT(VTP2-VTP1)
Ø115       9Ø CONTINUE
Ø116          IF(IWRITA(IVRT,ITRLM2,IBLKV,ADVEL).LT.Ø)STOP'WRITV ERROR'
Ø118          IBLKV=IBLKV+NVSIZ
Ø119          CALL IWAIT(IVRT)
Ø12Ø       5Ø CONTINUE
      C
      C START MAIN LOOP
```

```
      C
Ø121          DO 1ØØ ITORCT=1,ITORLM
Ø122          LIMIT=LIMIT-ITOR
      C
      C ZERO X ARRAYS
      C
Ø123          CALL VCLR(AØ,1,NCHANW)
Ø124          CALL APWR
Ø125          DO 11Ø I=1,6
Ø126          IST(I)=I
Ø127          FAD=ADXST(I)
Ø128          CALL APGETA(AØ,NCHANW,2,FAD)
Ø129      11Ø CONTINUE
      C
      C READ IN VELOCITIES
      C
Ø13Ø          IBLKV=Ø
Ø131          DO 12Ø L=1,NV
Ø132          IF(IREADA(IVRT,ITRLM2,IBLKV,ADVEL).LT.Ø)STOP'READV ERROR'
Ø134          IBLKV=IBLKV+NVSIZ
Ø135          CALL IWAIT(IVRT)
Ø136          V(L)=VSAVE(ITORCT)
Ø137      12Ø CONTINUE
      C
      C GEN V SLICE
      C
Ø138          CALL APPUT(V,AØ,NV,2)
Ø139          DO 13Ø I=1,NVM
Ø14Ø      13Ø VINT(I)=(V(I+1)-V(I))/(IV(I+1)-IV(I))
Ø141          CALL APPUT(VINT,A1,NVM,2)
Ø142          CALL APWD
      C
Ø143          DO 14Ø I=1,NVM
Ø144          NVD=IV(I+1)-IV(I)+1
Ø145          CALL VRAMP(AØ+I-1,A1+I-1,A2+IV(I)-1,1,NVD)
Ø146      14Ø CONTINUE
Ø147          CALL VFILL(A2+IV(1)-1,A2,1,NBST)
Ø148          CALL VFILL(A2+IV(NV)-1,A2+IV(NV),1,NEND)
Ø149          CALL VSQ(A2,1,A2,1,NCHANW)
      C
      C GEN A,B
      C
Ø15Ø          CALL APPUT(ACOF,AØ,1,2)
Ø151          CALL APPUT(BCOF,A1,1,2)
Ø152          CALL APWD
Ø153          CALL VSMUL(A2,1,AØ,AØ,1,NCHANW)
Ø154          CALL VSMUL(A2,1,A1,A1,1,NCHANW)
      C
      C SET UP NECESSARY COEFFS FROM SAVE
      C
Ø155          CALL APWR
Ø156          CALL APPUTA(A2,NCHANW,2,ADAPSV(1))
Ø157          CALL APPUTA(A3,NCHANW,2,ADAPSV(2))
Ø158          CALL APWD
```

```
      C
      C GEN COEFFS
      C
      C
      C (2+(2B+ACOF)T)
      C
Ø159        CALL VTSMUL(A2,1,2Ø5Ø,A4,1,NCHANW)
Ø16Ø        CALL VADD(AØ,1,A4,1,A4,1,NCHANW)
Ø161        CALL VNEG(A4,1,A5,1,NCHANW)
Ø162        CALL VADD(A3,1,A5,1,A5,1,NCHANW)
Ø163        CALL VTSMUL(A5,1,2Ø5Ø,A5,1,NCHANW)
      C
      C (I+(B+BCOF)T)
      C
Ø164        CALL VADD(A1,1,A2,1,A6,1,NCHANW)
Ø165        CALL VNEG(A6,1,A7,1,NCHANW)
Ø166        CALL VTSMUL(A7,1,2Ø5Ø,A7,1,NCHANW)
Ø167        CALL VADD(A3,1,A7,1,A7,1,NCHANW)
Ø168        CALL APWR
      C
      C SAVE CALCD COEFFS
      C
Ø169        CALL APGETA(A4,NCHANW,2,ADRHS(7))
Ø17Ø        CALL APGETA(A5,NCHANW,2,ADRHS(8))
Ø171        CALL APGETA(A6,NCHANW,2,ADRHS(9))
Ø172        CALL APGETA(A7,NCHANW,2,ADRHS(1Ø))
Ø173        CALL APWD
      C
      C GET -VE OF ABOVE COEFFS AND SAVE
      C
Ø174        CALL VNEG(A4,1,A4,1,NCHANW)
Ø175        CALL VNEG(A5,1,A5,1,NCHANW)
Ø176        CALL VNEG(A6,1,A6,1,NCHANW)
Ø177        CALL VNEG(A7,1,A7,1,NCHANW)
Ø178        CALL APWR
Ø179        CALL APGETA(A4,NCHANW,2,ADRHS(5))
Ø18Ø        CALL APGETA(A5,NCHANW,2,ADRHS(6))
Ø181        CALL APGETA(A6,NCHANW,2,ADRHS(3))
Ø182        CALL APGETA(A7,NCHANW,2,ADRHS(4))
Ø183        CALL APWD
      C
      C (I+(B-BCOF)T)
      C
Ø184        CALL VSUB(A1,1,A2,1,A4,1,NCHANW)
Ø185        CALL VNEG(A4,1,A5,1,NCHANW)
Ø186        CALL VTSMUL(A5,1,2Ø5Ø,A5,1,NCHANW)
Ø187        CALL VADD(A3,1,A5,1,A5,1,NCHANW)
Ø188        CALL APWR
Ø189        CALL APGETA(A4,NCHANW,2,ADRHS(1))
Ø19Ø        CALL APGETA(A5,NCHANW,2,ADRHS(2))
Ø191        CALL APGETA(A4,NCHANW,2,ADCSV)
Ø192        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 Parameters

READ(1,1000)TSTEP,XSTEP

1000 FORMAT(2F10.0)

TSTEP....Sample interval in seconds

XSTEP....Trace spacing in kms

READ(1,1001)NHLFWD,NINC,NSAMP,NSTEP,IFRNGE,NVEL

1001 FORMAT(6I5)

NHLFWD...Half-width of migration operator

NINC.....Step between operator traces

NSAMP....Number of samples per trace

NSTEP....Operator update step

IFRNGE...Update flag

<0 calculate update using allowed percentage error

>0 use NSTEP to update operator

NVEL.....Number of velocity layers

READ(1,1002)FSPECO

1002 FORMAT(3A4)

FSPECO...Operator output file

<u>READ(1,1000)TMIN,TMAX</u>

TMIN.....Start time for migration, seconds

TMAX.....End time for migration, seconds

<u>READ(1,1000)(TLYR(I),VLYR(I),I=1,NVEL)</u>

TLYR.....Two-way travel time

VLYR.....RMS velocity at this time

<u>READ(1,1000)D</u>

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
ØØ31           IST=Ø
ØØ32         5 IST=IST+1
ØØ33           IF(ITMIN.GT.IT(IST))GOTO 5
ØØ35           IA=IST
ØØ36           ITMI=Ø
ØØ37           ITMA=ITMIN
ØØ38           NOPTOT=Ø
ØØ39           IPNOP=1
ØØ4Ø           IVS=Ø
ØØ41        15 ITMI=ITMA
ØØ42           ITMA=IT(IA)
ØØ43           IF(ITMA.GT.ITMAX)ITMA=ITMAX
ØØ45           CALL OPCALC(ITMI,ITMA,NRANGE,IPNOP,NSTEP,IFRNGE,NOP)
ØØ46           NOPTOT=NOPTOT+NOP
ØØ47           IVS=IVS+1
ØØ48           IOP(IVS)=NOP
ØØ49           DZ(IVS)=VEL(IA)*TSTEP
ØØ5Ø           IA=IA+1
ØØ51           IF(ITMA.LT.ITMAX)GOTO 15
ØØ53           NRANGE(IPNOP-1)=ITMAX
ØØ54           NOUT=(NOPTOT+1)
      C
      C SET UP I/O CONSTANTS
      C
ØØ55           IBNOP=Ø
ØØ56           NBRNG=(NOUT+255)/256
ØØ57           NINLD=(NHLFWD+255)/256
ØØ58           IOPINC=(NBUF+127)/128
ØØ59           NBLD=NOPTOT*NINLD
ØØ6Ø           NBOPS=NOPTOT*IOPINC
ØØ61           NFILS=NBOPS+NBLD+NBRNG+1
ØØ62           INRNGS=1
ØØ63           INLDST=1+NBRNG
ØØ64           INOPST=INLDST+NBLD
      C
      C OPEN OUTPUT FILE
      C
ØØ65           IF(IENTER(IRD,FSPEC,NFILS).LT.Ø)STOP'ENTER ERROR'
      C
      C WRITE OUT NOPTOT
      C
ØØ67           IF(IWRITW(1,NOPTOT,Ø,IRD).LT.Ø)STOP'NOPWRITW ERROR'
      C
      C WRITE OUT NRANGE
      C
ØØ69           IF(IWRITW(NOUT,NRANGE,1,IRD).LT.Ø)STOP'WRITW ERR'
      C
      C CALC XSQ
      C
ØØ71           CALL APINIT
ØØ72           CALL VCLR(AØ,1,NHLFWD)
ØØ73           CALL APWR
ØØ74           CALL APPUT(XSTEP,AØ+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
```

```
Ø1Ø7          CALL APWR
Ø1Ø8          CALL APPUT(DZINV,A15,1,2)
Ø1Ø9          CALL APWD
Ø11Ø          CALL VSMUL(A1,1,A15,A3,1,NHLFWD)
      C
      C K
      C
Ø111          CALL VINT(A3,1,A5,1,NHLFWD)
      C
      C DELTA
      C
Ø112          CALL VSUB(A3,1,A5,1,A4,1,NHLFWD)
      C
      C NLEAD
      C
Ø113          CALL VSADD(A5,1,A8+IADD,A5,1,NHLFWD)
Ø114          CALL VSUB(A6+IADD,Ø,A5,1,A5,1,NHLFWD)
Ø115          CALL VTSADD(A5,1,2Ø49,A5,1,NHLFWD)
Ø116          CALL VFIX(A5,1,A5,1,NHLFWD)
Ø117          CALL APWR
Ø118          CALL APGET(NLEAD,A5,NHLFWD,1)
Ø119          CALL APWD
Ø12Ø          IF(IWRITW(NHLFWD,NLEAD,IBLKNL,IRD).LT.Ø)STOP'NLWRIT ERR'
Ø122          IBLKNL=IBLKNL+NINLD
      C
      C GEN OPERATOR FROM INTERMEDIATE VALUES
      C
Ø123          DO 4Ø I=1,5
Ø124          AS=A(I)
Ø125          ACON=AC(I)
Ø126          CALL VTSADD(A4,1,ACON,AS,1,NHLFWD)
Ø127          CALL VDIV(A3,1,AS,1,AS,1,NHLFWD)
Ø128          CALL VTSADD(AS,1,2Ø5Ø,A9,1,NHLFWD)
Ø129          CALL VMUL(AS,1,A9,1,AS,1,NHLFWD)
Ø13Ø          CALL VSQRT(AS,1,AS,1,NHLFWD)
Ø131          CALL VMUL(AS,1,A2,1,AS,1,NHLFWD)
Ø132       4Ø CONTINUE
      C
      C OP1
      C
Ø133          CALL VNEG(A11,1,A2,1,NHLFWD)
      C
      C OP2
      C
Ø134          CALL VTSMUL(A11,1,2Ø5Ø,A3,1,NHLFWD)
Ø135          CALL VSUB(A12,1,A3,1,A3,1,NHLFWD)
      C
      C OP3
      C
Ø136          CALL VTSMUL(A12,1,2Ø5Ø,A9,1,NHLFWD)
Ø137          CALL VSUB(A11,1,A9,1,A9,1,NHLFWD)
Ø138          CALL VSUB(A13,1,A9,1,A9,1,NHLFWD)
      C
      C OP4
```

```
C
      CALL VTSMUL(A13,1,2Ø5Ø,A1Ø,1,NHLFWD)
      CALL VSUB(A12,1,A1Ø,1,A1Ø,1,NHLFWD)
      CALL VSUB(A14,1,A1Ø,1,A1Ø,1,NHLFWD)
C
C OP5
C
      CALL VTSMUL(A14,1,2Ø5Ø,A11,1,NHLFWD)
      CALL VSUB(A13,1,A11,1,A11,1,NHLFWD)
      CALL VSUB(A15,1,A11,1,A11,1,NHLFWD)
C
C WRITE OUT OPERATOR
C
      CALL APWR
      IAD=1
      CALL APGET(OPVAL(IAD),A2,NHLFWD,2)
      IAD=IAD+NHLFWD
      CALL APGET(OPVAL(IAD),A3,NHLFWD,2)
      IAD=IAD+NHLFWD
      CALL APGET(OPVAL(IAD),A9,NHLFWD,2)
      IAD=IAD+NHLFWD
      CALL APGET(OPVAL(IAD),A1Ø,NHLFWD,2)
      IAD=IAD+NHLFWD
      CALL APGET(OPVAL(IAD),A11,NHLFWD,2)
      CALL APWD
      IF(IWRITW(NBUF2,OPVAL,IBLKOP,IRD).LT.Ø)STOP'OPWRIT ERR'
      IBLKOP=IBLKOP+IOPINC
   3Ø CONTINUE
   2Ø CONTINUE
      CALL CLOSEC(IRD)
      STOP'NORMAL TERMINATION'
      END
```



```
      SUBROUTINE OPCALC(ITMIN,ITMAX,NRANGE,IPOS,NST,IFLG,NOP)
C
C THIS SUBROUTINE USES THE FORMULA
C         STEP SIZE=K*D/5Ø
C         K=CURRENT SAMPLE NO
C         D=% ERROR
C
      INTEGER*2 NRANGE(512)
      DATA IN/Ø/
      IN=IN+1
      IF(IFLG.LE.Ø.AND.IN.EQ.1)READ(1,1ØØØ)D
 1ØØØ FORMAT(F1Ø.Ø)
      NOP=Ø
      NRANGE(IPOS)=ITMIN
      IPOS=IPOS+1
      I=ITMIN-1
      IF(IFLG.LE.Ø)N=I*(D/5Ø.Ø)
   1Ø NOP=NOP+1
      IF(IFLG.GT.Ø)GOTO 2Ø
      N=(I+N)*(D/5Ø.Ø)
      IF(N.LE.Ø)N=1
      I=I+2*N-1
      GOTO 3Ø
   2Ø I=I+NST
   3Ø NRANGE(IPOS)=I
      IPOS=IPOS+1
      IF(IPOS.GT.512)STOP'TOO MANY OPERATORS'
      IF(I.LT.ITMAX)GOTO 1Ø
      RETURN
      END
```

Kirchhoff Migration :- MPKMIG

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

Input Parameters

READ(1,1000)NTRACE,NO,NSAMP,NHLFWD

1000 FORMAT(4I5)

NTRACE...Number of traces in data file

NO.......Sample number of first sample in each trace

NSAMP....Number of samples per trace

NHLFWD...Half-width of operator

READ(1,1000)ISTART,NSTART,NSTOP,NINC

ISTART...First sample to migrate

NSTART...First trace to migrate

NSTOP....Last trace to migrate

NINC.....Trace increment in migration

READ(1,1001)XSTEP,TSTEP

1001 FORMAT(2F10.0)

XSTEP....Trace spacing in kms

TSTEP....Sample interval in seconds

READ(1,1002)FSPECO

```
1002 FORMAT(3A4)
```

FSPECO...Operator input file

<u>READ(1,1002)FSPECR</u>

FSPECR...Data input file

<u>READ(1,1001)FSPECW</u>

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
      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        NFILO=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,NFILO).LT.0)STOP'ENTER ERROR'
```

```
      C
      C GET OPERATOR CONSTANTS
      C
ØØ37        IF(IREADW(1,NOP,Ø,IROP).LT.Ø)STOP'READ ERROR'
ØØ39        IF(IREADW(NOP+1,NRANGE,1,IROP).LT.Ø)STOP'READ ERR'
ØØ41        NBRNG=(NOP+255)/256
ØØ42        NBUF=NHLFWD*5
ØØ43        NBUF2=2*NBUF
ØØ44        NBUFAP=5*NOP
ØØ45        IOPINC=(NBUF+127)/128
ØØ46        NINLD=(NHLFWD+255)/256
ØØ47        NBLD=NOP*NINLD
ØØ48        INLDST=1+NBRNG
ØØ49        INOPST=INLDST+NBLD
ØØ5Ø        IBLKO=1
ØØ51        ISTOP=NRANGE(NOP+1)
      C
      C START MAIN LOOP
      C
ØØ52        DO 1Ø ITR=NSTART,NSTOP,NINC
ØØ53        IBLK=((ITR-1)*NBLKR)+1
      C
      C READ IN OUTPUT TRACE
      C
ØØ54        IF(IREADW(2*NSAMP,TRACE,IBLK,IRD).LT.Ø)STOP'READ ERR2'
ØØ56        CALL APPUT(TRACE,AØ,NSAMP,2)
ØØ57        CALL APWD
ØØ58        CALL VCLR(AØ+ISTART,1,ISTOP-ISTART)
ØØ59        IMIN=MAXØ(NW2-ITR,1)
ØØ6Ø        IMAX=MINØ(NA-ITR,NWIDTH)
ØØ61        IBLKR=((ITR-NW+IMIN-2)*NBLKR)+1
      C
      C LOOP ON OPERATOR APPLICATION
      C
ØØ62        DO 2Ø IT=IMIN,IMAX
ØØ63        IOP=IT-NW
ØØ64        IF(IOP.LT.1)IOP=2-IOP
ØØ66        AØST=AØ+ISTART
ØØ67        IF(IREADW(2*NSAMP,TRACE,IBLKR,IRD).LT.Ø)STOP'READ ERR3'
ØØ69        IBLKR=IBLKR+NBLKR
ØØ7Ø        CALL APPUT(TRACE,A1,NSAMP,2)
ØØ71        LSTOR=1
ØØ72        NSTOR=1
      C
      C GET OPERATOR AND LEADIN VALUES FROM FILE
      C
ØØ73        IBLKLD=INLDST
ØØ74        IBLKOP=INOPST
ØØ75        DO 4Ø J=1,NOP
ØØ76        IF(IREADW(NHLFWD,NLEADI,IBLKLD,IROP).LT.Ø)STOP'READ ERR4'
ØØ78        IF(IREADW(NBUF2,TRACE,IBLKOP,IROP).LT.Ø)STOP'READ ERR4'
ØØ8Ø        IBLKLD=IBLKLD+NINLD
ØØ81        IBLKOP=IBLKOP+IOPINC
ØØ82        IOFF=IOP
```

```
      ILD=IOP
      DO 50 JJ=1,5
      OPBUF(LSTOR)=TRACE(IOFF)
      LSTOR=LSTOR+1
      IOFF=IOFF+NHLFWD
   50 CONTINUE
      NLEAD(NSTOR)=NLEADI(ILD)
      NSTOR=NSTOR+1
   40 CONTINUE
      CALL APPUT(OPBUF,A2,NBUFAP,2)
      CALL APWD
C
C DO CONVOLUTIONAL MIGRATION
C
      DO 60 ILP=1,NOP
      ATRI=NLEAD(ILP)+A1
      LIMIT=NRANGE(ILP+1)
      AOP=A2+(5*(ILP-1))
      NRES=LIMIT-A0ST+1
C
C DO CONVOLVE
C
      CALL CONV(ATRI,1,AOP,1,ASUM,1,NRES,5)
      CALL VADD(A0ST,1,ASUM,1,A0ST,1,NRES)
      A0ST=LIMIT+1
   60 CONTINUE
   20 CONTINUE
C
C GET MIGRATED TRACE OUT OF AP
C
      CALL APWR
      CALL APGET(TRACE,A0,NSAMP,2)
      CALL APWD
C
C WRITE MIGRATED TRACE TO DISK
C
      IF(IWRITW(2*NSAMP,TRACE,IBLKO,IWRT).LT.0)STOP'WRIT WRR'
      IBLKO=IBLKO+NBLKR
   10 CONTINUE
      CALL CLOSEC(IWRT)
      STOP'NORMAL TERMINATION'
      END
```

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

```
ØØØ1          VIRTUAL SØ(23552),S1(23552)
ØØØ2          DIMENSION DUM(2Ø48)
ØØØ3          DIMENSION FBUFR(3),FBUFW(3),FSTAP(23)
ØØØ4          REAL*8 FSPECR,FSPECW
ØØØ5          DATA K1M,K1,K2,KTOP,IA,IB,IB1,IB2,IC,IC1,ID1
             1/-1,1,2,8191,Ø,2Ø48,2Ø49,2Ø5Ø,4Ø96,4Ø97,6145/
ØØØ6          DATA DEV/3RDK /
        C
ØØØ7          CALL ASSIGN(1,'DK2:ANINV.DAT',13)
ØØØ8          IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
ØØ1Ø          IDCHR=IGETC()
ØØ11          IDCHW=IGETC()
        C
        C READ IN REQUIRED INPUT PARAMETERS
ØØ12          READ(1,1ØØØ) L,NCHAN,M,NTAP,INDEN
ØØ13    1ØØØ  FORMAT(2I5,5X,3I5)
ØØ14          READ(1,11ØØ) FBUFR,FBUFW
ØØ15    11ØØ  FORMAT(2(3A4))
        C VALIDATE DATA
        C NREAD IS THE NUMBER OF BLOCKS READ IN/TRACE
ØØ16          NREAD=L/128
        C STOP IF L IS NOT AN INTEGER NUMBER OF BLOCKS
ØØ17          IF(L.NE.128*NREAD)STOP'L NOT INTEGER NO. OF BLOCKS'
        C L2 IS NUMBER OF WORDS READ/TRACE
ØØ19          L2=2*L
        C IAL IS POINTER NEEDED FOR COSINE TAPERING
ØØ2Ø          IAL=IA+L-1
        C FIND L2I,INTEGER POWER OF TWO G.E. THAN L.
ØØ21          L2I=2
ØØ22          DO 5 I=1,1ØØØ
ØØ23          IF(L2I.GE.L) GOTO 1Ø
ØØ25          L2I=2*L2I
ØØ26    5     CONTINUE
        C STOP IF L2I IS GREATER THAN 2Ø48
ØØ27    1Ø    IF(L2I.GT.2Ø48)STOP'L2I.GT.2Ø48'
ØØ29          L2I21=L2I/2-1
        C STOP IF M.GT.23
ØØ3Ø          IF(M.GT.23)STOP'M.GT.23'
        C LM IS NUMBER OF INTERPOLATED SAMPLES/TRACE
ØØ32          LM=L*M
        C STOP IF LM.GT.23552
ØØ33          IF(LM.GT.23552)STOP'LM.GT.23552'
        C LM2 IS THE TOTAL NUMBER OF WORDS WRITTEN OUT/INTERPOLATED TRACE
ØØ35          LM2=2*LM
        C NWRITE IS THE NUMBER OF BLOCKS WRITTEN/TRACE
ØØ36          NWRITE=LM/128
        C FSTØ IS ADDRESS NEEDED FOR DISK READ INTO SØ(1)
ØØ37          FSTØ=ADGET(SØ(1))
        C FST1 IS ADDRESS NEEDED FOR DISK WRITE FROM S1
ØØ38          FST1=ADGET(S1(1))
        C FSTAP IS ARRAY OF ADDRESSES NEEDED FOR AP TRANSFERS
ØØ39          KL=1
ØØ4Ø          DO 2Ø K=1,M
ØØ41          FSTAP(K)=APGAD(SØ(KL))
```

```
              KL=KL+L
       2Ø     CONTINUE
       C OPEN READ FILE
              CALL IRAD5Ø(12,FBUFR,FSPECR)
              IF(LOOKUP(IDCHR,FSPECR).LT.Ø)STOP'LOOKUP ERROR'
       C OPEN WRITE FILE
              CALL IRAD5Ø(12,FBUFW,FSPECW)
              IF(IENTER(IDCHW,FSPECW,NWRITE*NCHAN+1).LT.Ø)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
              CALL APINIT
       C PUT 1.Ø/FLOAT(M*L2I) IN IC AND 1.Ø/FLOAT(NTAP) IN KTOP
              CALL APWR
              CALL APPUT(1.Ø/FLOAT(M*L2I),IC,K1,K2)
              CALL APPUT(1.Ø/FLOAT(NTAP),KTOP,K1,K2)
              CALL APWD
       C MULTIPLY BY 2*PI FROM TM
              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.
              CALL VRAMP(IC,IC,IA,K1,L2I21)
              CALL CVEXP(IA,K1,IC1,K2,L2I21)
       C FORM COSINE TAPER NTAP LONG STARTING AT ID1
              CALL VCLR(ID1,K1,K1)
              CALL VTSADD(ID1,K1,23Ø6,ID1,K1,K1)
              CALL VTSMUL(KTOP,K1,23Ø6,KTOP,K1,K1)
              CALL VRAMP(ID1,KTOP,ID1,K1,NTAP)
              CALL VCOS(ID1,K1,ID1,K1,NTAP)
              CALL VTSADD(ID1,K1,2Ø49,ID1,K1,NTAP)
              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
              JBLKW=1
              JBLKR=1
       C
```

```
C NOW ITERATE THROUGH CHANNELS INTERPOLATING THE TRACES.
      DO 800 JCHAN=1,NCHAN
C READ IN TRACE FROM UNIT 2
      IF(IREADA(IDCHR,L2,JBLKR,FST0).LT.0)STOP'READA ERROR'
      JBLKR=JBLKR+NREAD
C CLEAR A AND TRANSFER TRACE INTO IT
      CALL VCLR(IA,K1,L2I)
      CALL APWR
      CALL IWAIT(IDCHR)
      CALL APPUTA(IA,L,K2,FSTAP(1))
      CALL APWD
C FIND MEAN VALUE OF TRACE AND PLACE IN AP(KTOP)
      CALL MEANV(IA,K1,KTOP,L)
C SUBTRACT MEAN FROM TRACE
      CALL VNEG(KTOP,K1,KTOP,K1,K1)
      CALL VSADD(IA,K1,KTOP,IA,K1,L)
C MULTIPLY ENDS OF DATA BY COSINE TAPER
      CALL VMUL(IA,K1,ID1,K1,IA,K1,NTAP)
      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)
      IF(INDEN.LT.0) GOTO 350
      CALL RMSQV(IA,K1,KTOP,L)
      IF(INDEN.GT.0) CALL VSQ(KTOP,K1,KTOP,K1,K1)
C PLACE 1.0 FROM TM IN IC
      CALL VCLR(IC,K1,K1)
      CALL VTSADD(IC,K1,2049,IC,K1,K1)
C USE THIS VALUE TO CREATE RECIPROCAL IN KTOP
      CALL VDIV(KTOP,K1,IC,K1,KTOP,K1,K1)
C MULTIPLY TRACE BY RECIPROCAL
      CALL VSMUL(IA,K1,KTOP,IA,K1,L)
C TRANSFER MODIFIED TRACE BACK TO S0
  350 CALL APWR
      CALL APGETA(IA,L,K2,FSTAP(1))
      CALL APWD
      IF(M.EQ.1) GOTO 500
C TAKE TRANSFORM OF TRACE AND PUT IN B
      CALL RFFTB(IA,IB,L2I,K1)
      CALL RFFTSC(IB,L2I,K2,K1)
C ITERATE FROM 2 TO M
      DO 400 K=2,M
C MULTIPLY TRANSFORM BY COMPLEX EXPONENTIAL VECTOR
      CALL CVMUL(IB2,K2,IC1,K2,IB2,K2,L2I21)
C MOVE B TO A AND TAKE IN PLACE IFFT
      CALL VMOV(IB,K1,IA,K1,L2I)
      CALL RFFT(IA,L2I,K1M)
C TRANSFER SHIFTED TRACE BACK TO S0
      CALL APWR
      CALL APGETA(IA,L,K2,FSTAP(K))
      CALL APWD
  400 CONTINUE
C INTERPOLATED TRACE IS NOW IN S0 IN SCRAMBLED ORDER.
C UNSCRAMBLE INTO S1 SO THAT RECORD IS IN CORRECT TIME SEQUENCE.
```



```
  500 J1=1
      DO 700 JL=1,L
      J0=JL
      DO 600 JM=1,M
      S1(J1)=S0(J0)
      J0=J0+L
      J1=J1+1
  600 CONTINUE
  700 CONTINUE
C NOW WRITE OUT INTERPOLATED TRACE ONTO DISK
      IF(IWRITA(IDCHW,LM2,JBLKW,FST1).LT.0)STOP'WRITA ERROR'
      JBLKW=JBLKW+NWRITE
      CALL IWAIT(IDCHW)
  800 CONTINUE
      CALL CLOSEC(IDCHR)
      CALL CLOSEC(IDCHW)
      STOP
      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
ØØØ1         REAL*8 FSPECR,RUNITS(2),SOURCE(4)
ØØØ2         REAL*4 FNAMR(3)
ØØØ3         INTEGER*2 HBLK(256)
ØØØ4         LOGICAL*1 HSBLK(512),NOCHAN,IGCODE,IUNITS,ISAMP
ØØØ5         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(2Ø)),(ISCODE,HSBLK(45)),(IBFREE,HSBLK(47)),
            %(NCHAN,HSBLK(13)),(NBEG,HSBLK(15)),(NFIN,HSBLK(17)),
            %(ISAMP,HSBLK(9)),(HBLK(1),HSBLK(1))
ØØØ6         DATA DEV/3RRK /,RUNITS(1)/'METRES  '/,RUNITS(2)/' FEET   '/
ØØØ7         DATA SOURCE(1)/'AIRGUNS '/,SOURCE(2)/'EXPLOSIV'/,
            %SOURCE(3)/'VIBROSEI'/,SOURCE(4)/'WEIGHTS '/
      C
      C GET NAME OF INPUT FILE
      C
ØØØ8         WRITE(7,5)
ØØØ9       5 FORMAT(' ENTER NAME OF FILE TO BE EXAMINED:',$)
ØØ1Ø         READ(5,1Ø)FNAMR
ØØ11      1Ø FORMAT(3A4)
ØØ12         CALL IRAD5Ø(12,FNAMR,FSPECR)
      C
      C SET UP I/O PARAMETERS
      C
ØØ13         ICHR=IGETC()
ØØ14         IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERR'
ØØ16         ILEN=LOOKUP(ICHR,FSPECR)
ØØ17         IF(ILEN.LT.Ø)STOP'LOOKUP ERR'
ØØ19         IF(IREADW(256,HSBLK,Ø,ICHR).LT.Ø)STOP'READ ERROR'
      C
      C DECODE HEADER AND PRINT OUT
      C
ØØ21         WRITE(7,2Ø)FNAMR,ILEN
ØØ22      2Ø FORMAT(' FILE ',3A4,' TO BE ANALYSED, LENGTH=',I5,' BLOCKS')
ØØ23         WRITE(7,3Ø)(HSBLK(I),I=1,8)
ØØ24      3Ø FORMAT(' FILE NO:',3A1,' TAPE NO:',5A1)
ØØ25         WRITE(7,4Ø) HSBLK(9),HSBLK(11),HSBLK(12)
ØØ26      4Ø FORMAT(' SAMPLING INTERVAL SET AT ',I1,' MSEC',/,
            %' AND RECORDING LASTED ',I2,' SECS',/,
            %' ON ',I2,' ACTIVE CHANNELS')
ØØ27         GOTO (5Ø,6Ø,6Ø,8Ø)IGCODE
ØØ28      5Ø WRITE(7,11Ø)
ØØ29         GOTO 1ØØ
ØØ3Ø      6Ø WRITE(7,12Ø)
ØØ31         GOTO 1ØØ
ØØ32      7Ø WRITE(7,13Ø)
ØØ33         GOTO 1ØØ
ØØ34      8Ø WRITE(7,14Ø)
ØØ35     1ØØ CONTINUE
```

```
      110 FORMAT(' THE FILE CONTAINS A SHOT POINT GATHER')
      12Ø FORMAT(' THE FILE CONTAINS A COMMON MIDPOINT GATHER')
      13Ø FORMAT(' THE FILE CONTAINS A STACKED TRACE')
      14Ø FORMAT(' THE FILE CONTAINS AN UNSTACKED TRACE')
          WRITE(7,9Ø)NCHAN,NBEG,NFIN
       9Ø FORMAT(' CONSISTING ',I2,' CHANNELS,STARTING AT SAMPLE NO:'
         %,I5,/,' AND ENDING WITH SAMPLE NO:'I5)
          WRITE(7,15Ø)SOURCE(ISCODE),RUNITS(IUNITS)
      15Ø FORMAT(' TYPE OF SOURCE= ',A8,/,
         %' AND THE UNITS OF LENGTH USED= ',A8)
          WRITE(7,16Ø)RDEPT,SDEPT,SROFF,RSPAC,SLSPAC,STSPAC
      16Ø FORMAT(' RECIEVER DEPTH=',F1Ø.2,/,
         %' SOURCE DEPTH=',F1Ø.2,/,
         %' SOURCE-RECIEVER OFFSET=',F1Ø.2,/,
         %' RECIEVER SPACING=',F1Ø.2,/,
         %' SHOT SPACING=',F1Ø.2,/,
         %' SHOT REPITITION RATE=',F1Ø.2,' SECS')
          WRITE(7,17Ø)(HSBLK(I),I=51,254)
      17Ø FORMAT(' USER INFO PUT INTO HEADER BLOCK',/,
         %' IS GIVEN BELOW',/,3(1X,8ØA1,/,))
          IF(IBFREE.EQ.Ø)GOTO 999
C
C PROCESS DECODE AREA
C
          WRITE(7,18Ø)
      18Ø FORMAT(/,/,' THE FOLLOWING PROCESSES HAVE BEEN
         % PERFORMED ON THE DATA',/)
          IPOS=Ø
      19Ø CONTINUE
          GOTO(2ØØ,21Ø,22Ø,23Ø,24Ø)HBLK(129+IPOS)
C
C PRE STACK DECODE
C
      2ØØ IPOS=IPOS+1
          ICNT=HBLK(129+IPOS)
          IPOS=IPOS+1
          WRITE(7,3ØØ)ICNT
      3ØØ FORMAT(' PRE-STACK PROCESSING CONSISTING OF',/,
         %I3,' OPERATIONS IN THE FOLLOWING ORDER')
          DO 31Ø J=1,ICNT
          ICD=HBLK(129+IPOS)
          IPOS=IPOS+1
          GOTO(311,312,313,314,315,316,317,318,319,32Ø,321)ICD
      311 WRITE(7,4ØØ)
      4ØØ FORMAT(' TRACE EDITING')
          GOTO 31Ø
      312 WRITE(7,4Ø1)
      4Ø1 FORMAT(' POLARITY REVERSAL')
          GOTO 31Ø
      313 WRITE(7,4Ø2)
      4Ø2 FORMAT(' EXP(Ø.2T) AMP RECOVERY')
          GOTO 31Ø
      314 WRITE(7,4Ø3)
      4Ø3 FORMAT(' T*EXP(Ø.2T) AMP RECOVERY')
```

```
      GOTO 310
  315 WRITE(7,404)
  404 FORMAT(' T*V**2 AMP RECOVERY')
      GOTO 310
  316 WRITE(7,405)
  405 FORMAT(' MUTING')
      GOTO 310
  317 WRITE(7,406)
  406 FORMAT(' SPIKE DECONVOLUTION')
      GOTO 310
  318 WRITE(7,407)
  407 FORMAT(' BANDPASS FILTERING')
      GOTO 310
  319 WRITE(7,408)
  408 FORMAT(' BANDREJECT FILTERING')
      GOTO 310
  320 WRITE(7,409)
  409 FORMAT(' PREDICTION ERROR DECONVOLUTION')
      GOTO 310
  321 WRITE(7,410)
  410 FORMAT(' AMPLITUDE/ENERGY NORMALISATION')
  310 CONTINUE
      IF(IPOS.LT.IBFREE)GOTO 190
      GOTO 999
C
C STACKING DECODE
C
  210 WRITE(7,420)
  420 FORMAT(/,' NMO CORRECTION AND CDP STACK')
      IPOS=IPOS+1
      NCHST=HBLK(129+IPOS)
      IPOS=IPOS+1
      NVST=HBLK(129+IPOS)
      IPOS=IPOS+1
      NLVRST=HBLK(129+IPOS)
      IPOS=IPOS+1
      MST=HBLK(129+IPOS)
      IPOS=IPOS+1
      INTSWS=HBLK(129+IPOS)
      IPOS=IPOS+1
      WRITE(7,421)NCHST,MST
  421 FORMAT(' WITH',I3,' CHANNELS AT A LEVEL OF',/,
     #I4,' TIMES INTERPOLATION')
      WRITE(7,422)NVST,NLVRST,INTSWS
  422 FORMAT(I5,' VELOCITY ANALYSES WERE USED EACH WITH,',/,
     #I3,' LAYERS AND THE INTERPOLATION SWITCH=',I1)
      IF(IPOS.LT.IBFREE)GOTO 190
      GOTO 999
C
C POST STACK DECODE
C
  220 IPOS=IPOS+1
      ICNT=HBLK(129+IPOS)
      IPOS=IPOS+1
```

```
        WRITE(7,430)ICNT
    430 FORMAT(' POST STACK PROCESSING CONSISTING OF',/,
       %I3,' OPERATIONS IN THE FOLLOWING ORDER')
        DO 330 J=1,ICNT
        ICD=HBLK(129+IPOS)
        IPOS=IPOS+1
        GOTO(331,332,333,334,335,336,337,338,339,340)ICD
    331 WRITE(7,400)
        GOTO 330
    332 WRITE(7,402)
        GOTO 330
    333 WRITE(7,403)
        GOTO 330
    334 WRITE(7,404)
        GOTO 330
    335 WRITE(7,405)
        GOTO 330
    336 WRITE(7,406)
        GOTO 330
    337 WRITE(7,407)
        GOTO 330
    338 WRITE(7,408)
        GOTO 330
    339 WRITE(7,409)
        GOTO 330
    340 WRITE(7,410)
    330 CONTINUE
        IF(IPOS.LT.IBFREE)GOTO 190
        GOTO 999
C
C MIGRATION
C
    230 WRITE(7,450)
    450 FORMAT(' MIGRATION')
        IPOS=IPOS+1
        IF(IPOS.LT.IBFREE)GOTO 190
        GOTO 999
C
C THREE TRACE MIX
C
    240 WRITE(7,460)
    460 FORMAT(' THREE TRACE MIX')
        IPOS=IPOS+1
        IF(IPOS.LT.IBFREE)GOTO 190
C
    999 STOP' NORMAL TERMINATION'
        END
```

Synthetic CMP Gathers :- ANSEI

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

Output file.....DK2:ANOUTS.DAT

Input Parameters

READ(1,1000)L,NCHAN,NSTART,NLYR

1000 FORMAT(4I5)

L........Number of samples per channel

NCHAN....Number of channels per gather

NSTART...Number of starting sample from time zero

NLYR.....Number of events

READ(1,1100)XSTART,XSTEP,FSAMP,FRICK,FNOISE,AMP

1100 FORMAT(6F10.0)

XSTART...Offset of first receiver from source

XSTEP....Receiver spacing

FSAMP....Sampling frequency of data samples/millisecond

FRICK....Frequency of Source wavelet(Ricker)

FNOISE...Upper of Frequency of Background noise

AMP......Amplitude of Background noise

READ(1,1200)(TOLYR(I),VLYR(I),I=1,NLYR)

1200 FORMAT(2F10.0)

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

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

```
      REAL*8 FSPEC
      DIMENSION SEISM(2048),TØLYR(99),VLYR(99),WAVE(5ØØ),CONST(4)
      DATA K1N,KØ,K1,K2,K4,KAMP,KSN,KTWOPI,KSEED,KU/-1,Ø,1,2,4,8187,
     18188,8189,819Ø,8191/
      DATA NWDBLK,DEV,FSPEC/256,3RDK ,12RDK2ANOUTSDAT/
      CALL ASSIGN(1,'DK1:ANINS.DAT',13)
      IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
      IDCH=IGETC()
      READ(1,1ØØØ) L,NCHAN,NSTART,NLYR
 1ØØØ FORMAT(4I5)
      READ(1,11ØØ) XSTART,XSTEP,FSAMP,FRICK,FNOISE,SN
 11ØØ FORMAT(6F1Ø.Ø)
      READ(1,12ØØ) (TØLYR(I),VLYR(I),I=1,NLYR)
 12ØØ FORMAT(2F1Ø.Ø)
      NBLTR=IFIX(2.Ø*FLOAT(L)/FLOAT(NWDBLK)+Ø.5)
      L=NWDBLK*NBLTR/2
      NBLTOT=NBLTR*NCHAN
      IF(IENTER(IDCH,FSPEC,NBLTOT+1).LT.Ø)STOP'ENTER ERROR'
      JBLOCK=1
      CALL RICKER(FSAMP,FRICK,NBEGIN,NWAVE,WAVE)
      L2INT=2
      DO 5 K=1,11
      IF(L2INT.GE.L) GOTO 1Ø
      L2INT=2*L2INT
    5 CONTINUE
   1Ø CONST(1)=1.Ø/SN
      CONST(2)=8.Ø*ATAN(1.Ø)
      CONST(3)=Ø.251Ø638
      CONST(4)=1.Ø
      CALL APINIT(IDUM,IDUM,IDUM)
      CALL APPUT(CONST,KSN,K4,K2)
      CALL APWD
      ISEISM=1
      IWAVE=ISEISM+L2INT+NWAVE-1
      IWAVER=IWAVE+NWAVE-1
      IRAND=IWAVE+NWAVE
      NRAND=L2INT*FNOISE/FSAMP
      INOISE=IRAND+NRAND
      INOIS2=INOISE+2
      INOIS3=INOISE+3
      NRAND2=2*NRAND
      CALL APPUT(WAVE,IWAVE,NWAVE,K2)
      CALL APWD
      XJCHAN=XSTART
      NOFFS=NSTART-NBEGIN
      DO 4Ø JCHAN=1,NCHAN
      XSQ=XJCHAN**2
      XJCHAN=XJCHAN+XSTEP
      CALL VCLR(KØ,K1,IWAVE)
      DO 3Ø JLYR=1,NLYR
      NT=IFIX(FSAMP*SQRT(TØLYR(JLYR)**2+XSQ
     #/VLYR(JLYR)**2)+Ø.5)-NOFFS
      CALL VADD(NT,K1,KU,K1,NT,K1,K1)
   3Ø CONTINUE
```

```
      CALL CONV(ISEISM,K1,IWAVER,K1N,ISEISM,K1,L2INT,NWAVE)
      CALL VRAND(KSEED,IRAND,K1,NRAND)
      CALL VSMUL(IRAND,K1,KTWOPI,IRAND,K1,NRAND)
      CALL VCLR(INOISE,K1,L2INT)
      CALL VCOS(IRAND,K1,INOIS2,K2,NRAND)
      CALL VSIN(IRAND,K1,INOIS3,K2,NRAND)
      CALL RFFT(INOISE,L2INT,K1N)
      CALL RMSQV(INOISE,K1,KAMP,L2INT)
      CALL VDIV(KAMP,K1,KSN,K1,KAMP,K1,K1)
      CALL VSMUL(INOISE,K1,KAMP,INOISE,K1,L2INT)
      CALL VADD(ISEISM,K1,INOISE,K1,ISEISM,K1,L2INT)
      CALL APWR
      CALL IWAIT(IDCH)
      CALL APGET(SEISM(1),ISEISM,L,K2)
      CALL APWD
      IF(IWRITE(2*L,SEISM,JBLOCK,IDCH).LT.Ø)STOP'WRITE ERROR'
      JBLOCK=JBLOCK+NBLTR
   6Ø CONTINUE
      CALL CLOSEC(IDCH)
      STOP
      END
```



```
      SUBROUTINE RICKER(FSAMP,FRICK,NBEGIN,NWAVE,WAVE)
      IMPLICIT INTEGER*2(I-N)
      DIMENSION WAVE(5ØØ)
      N=IFIX(FSAMP/FRICK)
      NWAVE=2*N+1
      NBEGIN=N+1
      WAVE(NBEGIN)=1.
      IF(N.EQ.Ø) RETURN
      IPLUS=NBEGIN
      IMINUS=NBEGIN
      X=Ø.
      DELX=3.14159265*FRICK/FSAMP
      DO 1Ø I=1,N
      IPLUS=IPLUS+1
      IMINUS=IMINUS-1
      X=X+DELX
      WAVE(IPLUS)=Ø.5*(1.+COS(X))*(1.-2.*X**2)*EXP(-X**2)
      WAVE(IMINUS)=WAVE(IPLUS)
   1Ø CONTINUE
      RETURN
      END
```

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'
0013          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
      CALL APINIT
      KNSAMP=NSAMP
      KNOISE=2
      IF(AMP.LT.0.0)GOTO 50
   60 KNOISE=KNOISE*2
      IF(KNOISE.LT.KNSAMP)GOTO 60
      KRAND=FNOISE*TSTEP*KNOISE/2000.0
   50 KWAVE=2*N+1
      KWAVE3=KWAVE+3
      KB=N
      KA=8192-KWAVE3
      KC=KB+KNSAMP
      KD=8189
      AUX(KWAVE+1)=2.0*PI
      AUX(KWAVE+2)=0.2510638
      AUX(KWAVE3)=AMP
      CALL APPUT(AUX,KA,KWAVE3,2)
C
C START TRACE GENERATION LOOP
C
      DO 100 L=1,NTRACE
      X=(L-1)*XSTEP
      CALL VCLR(0,1,KNSAMP)
      CALL APWR
      CALL APGET(TRACE,0,KNSAMP,2)
      CALL APWD
C
C EVALUATE ARRIVAL FOR EACH REFLECTOR
C
      IF(NPLANE.LE.0)GOTO 110
        DO 200 J=1,NPLANE
        IF(X.LT.X1(J).OR.X.GT.X2(J))GOTO 200
        XA=ABS(X-X0(J))*SINDIP(J)**2
        ZA=ABS(X-X0(J))*SINDIP(J)*COSDIP(J)
        IF(ABS(SINDIP(J)).LT.0.001)ZA=X0(J)
        CALL NEWTON(XA,ZA,T0)
        CALL IMPULS(T0)
  200   CONTINUE
C
C EVALUATE FOR EACH POINT REFLECTOR
C
  110 IF(NPOINT.LE.0)GOTO 120
        DO 210 J=1,NPOINT
        CALL NEWTON(ABS(X-XP(J)),ZP(J),T0)
        CALL IMPULS(T0)
  210   CONTINUE
C
C CONVOLVE SERIES WITH WAVEFORM
C
  120 CALL VCLR(0,1,KB)
      CALL VCLR(KC,1,KB)
      CALL APPUT(TRACE,KB,KNSAMP,2)
```

```
FORTRAN IV      VØ2.Ø4     THU Ø8-JAN-81 ØØ:52:Ø6                PAGE ØØ3

ØØ9Ø          CALL APWD
ØØ91          CALL CONV(Ø,1,KA,1,Ø,1,KNSAMP,KWAVE)
ØØ92          CALL APWR
ØØ93          IF(AMP.LE.Ø.Ø)GOTO 13Ø
        C
        C GENERATE NOISE
        C
ØØ95          CALL VRAND(KD+1,2Ø49,2,KRAND)
ØØ96          CALL VSMUL(2Ø49,2,KD,2Ø49,2,KRAND)
ØØ97          CALL CVEXP(2Ø49,2,2Ø48,2,KRAND)
ØØ98          CALL VCLR(2Ø48+(2*KRAND),1,KNOISE-(2*KRAND))
ØØ99          CALL RFFT(2Ø48,KNOISE,-1)
Ø1ØØ          CALL VSMUL(2Ø48,1,KD+2,2Ø48,1,KNSAMP)
Ø1Ø1          CALL VADD(Ø+NØ,1,2Ø48+NØ,1,Ø+NØ,1,NSAMP-NØ+1)
Ø1Ø2      13Ø CALL APWR
Ø1Ø3          CALL APGET(TRACE,Ø,NSAMP,2)
Ø1Ø4          CALL APWD
Ø1Ø5          IF(IWRITW(2*NSAMP,TRACE,IBLK,IWRT).LT.Ø)STOP'WRITE ERROR'
Ø1Ø7          IBLK=IBLK+NBLKR
Ø1Ø8      1ØØ CONTINUE
Ø1Ø9          CALL CLOSEC(IWRT)
Ø11Ø          STOP' NORMAL TERMINATION'
Ø111          END
```

```
FORTRAN IV      VØ2.Ø4     THU Ø8-JAN-81 ØØ:52:51                PAGE ØØ1

ØØØ1          SUBROUTINE RICKER(TSTEP,FRICK,N)
        C
        C SUBROUTINE TO EVALUATE A RICKER WAVELET
        C DUE TO A G NUNNS 1979--- WITH MODS
        C
ØØØ2          COMMON/BB/AUX(2Ø48)
ØØØ3          N=IFIX(1ØØØ.Ø/(FRICK*TSTEP))
ØØØ4          IPLUS=N+1
ØØØ5          AUX(IPLUS)=1.Ø
ØØØ6          IF(N.EQ.Ø)RETURN
ØØØ8          IMINUS=IPLUS
ØØØ9          X=Ø.Ø
ØØ1Ø          DELX=3.14159265*FRICK*TSTEP/1ØØØ.Ø
ØØ11            DO 1ØØ I=1,N
ØØ12            IPLUS=IPLUS+1
ØØ13            IMINUS=IMINUS-1
ØØ14            X=X+DELX
ØØ15            AUX(IPLUS)=(1.Ø+COS(X))*(Ø.5-X**2)*EXP(-X**2)
ØØ16            AUX(IMINUS)=AUX(IPLUS)
ØØ17      1ØØ CONTINUE
ØØ18          RETURN
ØØ19          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
```

```
ØØØ1          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
ØØØ2          COMMON/CC/VEL(11),THICKN(1Ø),NLAYER
ØØØ3          COMMON/DD/N,D(11)
ØØØ4          IF(NLAYER.LE.Ø)GOTO 16Ø
        C
        C FIND N
        C
ØØØ6          TH=Ø.Ø
ØØØ7          DO 1ØØ I=1,NLAYER
ØØØ8          N=I
ØØØ9          IF(Z.LE.TH+THICKN(I))GOTO 11Ø
ØØ11      1ØØ TH=TH+THICKN(I)
ØØ12          N=N+1
ØØ13      11Ø IF(N.EQ.1)GOTO 16Ø
        C
        C SET VALUES OF D(I)
        C
ØØ15          I1=N-1
ØØ16          DO 12Ø I=1,I1
ØØ17      12Ø D(I)=THICKN(I)
ØØ18          D(N)=Z-TH
ØØ19          IF(X.LE.Ø.Ø)GOTO 17Ø
        C
        C FIND N-R STARTING VALUE
        C
ØØ21          CØ=3.Ø*VEL(N)
ØØ22      13Ø CØ=(CØ+VEL(N))*Ø.5
ØØ23          IF(A(CØ,1).LE.X)GOTO 13Ø
        C
        C FIND C BY N-R
        C
ØØ25          C1=CØ
ØØ26      14Ø CØ=C1
ØØ27          C1=CØ+(A(CØ,1)-X)/(CØ*A(CØ,3))
ØØ28          IF(ABS((C1-CØ)/C1).GT.Ø.1E-Ø3)GOTO 14Ø
        C
        C EVALUATE T
        C
ØØ3Ø          T=X
ØØ31          DO 15Ø I=1,N
ØØ32      15Ø T=T+D(I)*SQRT((C1/VEL(I))**2-1.Ø)
ØØ33          T=T*1ØØØ.Ø/C1
ØØ34          RETURN
        C
        C NO VELOCITY INTERFACE BETWEEN SOURCE AND RECIEVER
        C
ØØ35      16Ø T=SQRT(X**2+Z**2)*1ØØØ.Ø/VEL(1)
ØØ36          RETURN
        C
```

```
FORTRAN IV      V02.04      THU 08-JAN-81 00:53:34                    PAGE 002

      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
```

```
FORTRAN IV      V02.04      THU 08-JAN-81 00:54:04                    PAGE 001

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
0009        IF(M.LT.1)STOP 'ERROR 3'
0011        B=VEL(I)**(M-1)
0012    100 A=A+D(I)/(B*SQRT((C/VEL(I))**2-1.0)**M)
0013        RETURN
0014        END
```

Velocity Analysis Display :- MPVCON

This routine is comletely 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(12Ø,256)
      REAL*4 X(256),Y(12Ø),CZ(2Ø),Z
      INTEGER*2 PTR(2Ø),NMAX(256)
      LOGICAL*1 SWCHES(5)
      COMMON /PPEP1/ IX1,IY1,IX2,IY2,ISCAN,NSCAN,NBAND,NIPS,NIPØ,
     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.Ø,Ø.95,Ø.9,Ø.85,Ø.8,Ø.75,Ø.7,Ø.65,Ø.6,Ø.55,Ø.5,Ø.45,
     %Ø.4,Ø.35,Ø.3,Ø.25,Ø.2,Ø.15,Ø.1,Ø.Ø5/
      DATA SWCHES/.TRUE.,.TRUE.,.TRUE.,.TRUE.,.FALSE./
      DATA NC/18/
C
C ZERO CONTOUR ARRAY
C
      DO 1 I=1,256
      DO 2 J=1,12Ø
    2 Z(J,I)=Ø.Ø
    1 CONTINUE
      CALL PLOTS(Ø,Ø,Ø)
C
C READ IN CONTROL DATA
C
      CALL ASSIGN(2,'DK2:ANOUTV.DAT',14)
      READ(2) LNUM,NCHAN,NSTART,M
      READ(2) XSTART,XSTEPR,FSAMP
      READ(2) TØ1,TØ2,TØSTEP,TGATE
      READ(2) VSTEP,V1,V2MIN,V2MAX
      READ(2) NTØ
      XSTEP=1Ø.Ø/FLOAT(NTØ-1)
      IXPOS=NTØ+1
C
C READ IN THE SEMBLANCE DATA
C
      DO 1Ø JTØ=1,NTØ
      IXPOS=IXPOS-1
      READ(2) NPT,NMAX(JTØ),(Z(J,IXPOS),J=1,NPT)
      IF(NPT.GT.MYPT)MYPT=NPT
   1Ø CONTINUE
      YSTEP=4.Ø/FLOAT(MYPT-1)
C
C SET UP COORDINATES OF CONTOUR ARRAY
C
      Y(1)=Ø.3
      DO 2Ø L=1,MYPT
   2Ø Y(L)=Y(L-1)+YSTEP
      X(1)=Ø.Ø
      DO 3Ø LL=2,NTØ
   3Ø X(LL)=X(LL-1)+XSTEP
C
C USE CONSYS CONTOUR PACKAGE
C
      CALL CONTUR(Y,MYPT,X,NTØ,Z,12Ø,CZ,NC,PTR,SWCHES,
     % , , , , )
C
C PLOT TIME GRID
C
      TØLEN=TØ2-TØ1
      TUNIT=1ØØØ.Ø*(X(NTØ)-X(1))/TØLEN
```

```
      TTUNIT=TUNIT/1Ø.Ø
      TØBEG=FLOAT(IFIX(TØ1+1ØØØ.Ø)/1ØØØ)
      XOFF=(TØBEG-(TØ1/1ØØØ.Ø))*TUNIT
C
C PLOT SECOND GRID
C
      TNUM=TØBEG
      XPT=X(NTØ)-XOFF
   5Ø CALL PLOT(Y(1),XPT,+3)
      CALL PLOT(Y(MYPT),XPT,+2)
      CALL NUMBER(Ø.Ø,XPT,Ø.Ø5,TNUM,Ø.Ø,1)
      XPT=XPT-TUNIT
      TNUM=TNUM+1.Ø
      IF(XPT.GT.X(1))GOTO 5Ø
C
C PLOT TENTHS OF SECONDS GRID
C
      LMSK="14Ø3
      MSK=+1
      XPT=X(NTØ)-XOFF
   6Ø CALL PLOT(Y(1),XPT,+3)
      CALL PLOT(Y(MYPT),XPT,+2)
      XPT=XPT-TTUNIT
      IF(XPT.GT.X(1))GOTO 6Ø
      XPT=X(NTØ)-XOFF
   7Ø CALL PLOT(Y(1),XPT,+3)
      CALL PLOT(Y(MYPT),XPT,+2)
      XPT=XPT+TTUNIT
      IF(XPT.LT.X(NTØ))GOTO 7Ø
C
C VELOCITY GRID
C
      VLEN=V2MAX-V1
      VUNIT=(Y(MYPT)-Y(1))/VLEN
      VTUNIT=VUNIT/1Ø.Ø
      VBEG=FLOAT(IFIX(V1+1.Ø))
      YOFF=(VBEG-V1)*VUNIT
C
C PLOT KM/S GRID
C
      LMSK=-1
      MSK=Ø
      YPT=Y(1)+YOFF
      VNUM=VBEG
   8Ø CALL PLOT(YPT,X(NTØ),+3)
      CALL PLOT(YPT,X(1),+2)
      CALL NUMBER(YPT-Ø.Ø5,X(NTØ),Ø.Ø5,VNUM,Ø.Ø,1)
      YPT=YPT+VUNIT
      VNUM=VNUM+1.Ø
      IF(YPT.LT.Y(MYPT))GOTO 8Ø
C
C PLOT TENTHS GRID
C
      LMSK="14Ø3
      MSK=-1
      YPT=Y(1)+YOFF
   9Ø CALL PLOT(YPT,X(NTØ),+3)
      CALL PLOT(YPT,X(1),+2)
      YPT=YPT-VTUNIT
      IF(YPT.GT.Y(1))GOTO 9Ø
      YPT=Y(1)+YOFF
  1ØØ CALL PLOT(YPT,X(NTØ),+3)
      CALL PLOT(YPT,X(1),+2)
      YPT=YPT+VTUNIT
```

```
      IF(YPT.LT.Y(MYPT))GOTO 1ØØ
C
C PLOT POSITION OF MAX PTS
C
      LMSK=-1
      MSK=Ø
      IXMP=NTØ+1
      DO 4Ø MP=1,NTØ
      IXMP=IXMP-1
      XP=X(IXMP)
      YP=Y(NMAX(MP))
      CALL SYMBOL(YP,XP,Ø.Ø2,Ø,Ø.Ø,-1)
   4Ø CONTINUE
C
C DO ANOTATION
C
      CALL SYMBOL(1.Ø,1Ø.1,Ø.1,'VELOCITY ANALYSIS CONTOURS',Ø.Ø,26)
      CALL SYMBOL(4.8,1Ø.3,Ø.1,'PROCESSING PARAMETERS',Ø.Ø,21)
      CALL PLOT(4.8,1Ø.3,+3)
      CALL PLOT(6.9,1Ø.3,+2)
      CALL SYMBOL(5.Ø,1Ø.Ø,Ø.1,'NO. OF CHANNELS = ',Ø.Ø,18)
      FNUM=NCHAN
      CALL NUMBER(6.8,1Ø.Ø,Ø.1,FNUM,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,9.8,Ø.1,'SAMPLES PER CHANNEL = ',Ø.Ø,22)
      FNUM=LNUM
      CALL NUMBER(7.2,9.8,Ø.1,FNUM,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,9.6,Ø.1,'SAMPLE DELAY = ',Ø.Ø,15)
      FNUM=NSTART
      CALL NUMBER(6.5,9.6,Ø.1,FNUM,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,9.4,Ø.1,'LEVEL OF INTERPOLATION = ',Ø.Ø,25)
      FNUM=M
      CALL NUMBER(7.5,9.4,Ø.1,FNUM,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,9.2,Ø.1,'CHANNEL 1 OFFSET = ',Ø.Ø,19)
      CALL NUMBER(6.9,9.2,Ø.1,XSTART,Ø.Ø,1)
      CALL SYMBOL(5.Ø,9.Ø,Ø.1,'CHANNEL SPACING = ',Ø.Ø,18)
      CALL NUMBER(6.8,9.Ø,Ø.1,XSTEPR,Ø.Ø,1)
      CALL SYMBOL(5.Ø,8.8,Ø.1,'SAMPLING INTERVAL MS = ',Ø.Ø,23)
      FNUM=1.Ø24/FSAMP
      CALL NUMBER(7.3,8.8,Ø.1,FNUM,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,8.6,Ø.1,'START OF ANALYSIS MS = ',Ø.Ø,23)
      CALL NUMBER(7.3,8.6,Ø.1,TØ1,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,8.4,Ø.1,'END OF ANALYSIS MS = ',Ø.Ø,21)
      CALL NUMBER(7.1,8.4,Ø.1,TØ2,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,8.2,Ø.1,'TIME STEP MS = ',Ø.Ø,15)
      CALL NUMBER(6.5,8.2,Ø.1,TØSTEP,Ø.1,-1)
      CALL SYMBOL(5.Ø,8.Ø,Ø.1,'OPERATOR GATEWIDTH MS = ',Ø.Ø,23)
      CALL NUMBER(7.3,8.Ø,Ø.1,TGATE,Ø.Ø,-1)
      CALL SYMBOL(5.Ø,7.8,Ø.1,'START VELOCITY KM/S = ',Ø.Ø,22)
      CALL NUMBER(7.2,7.8,Ø.1,V1,Ø.Ø,2)
      CALL SYMBOL(5.Ø,7.6,Ø.1,'END VELOCITY KM/S = ',Ø.Ø,2Ø)
      CALL NUMBER(7.Ø,7.6,Ø.1,V2MAX,Ø.Ø,2)
      CALL SYMBOL(5.Ø,7.4,Ø.1,'VELOCITY STEP KM/S = ',Ø.Ø,21)
      CALL NUMBER(7.1,7.4,Ø.1,VSTEP,Ø.Ø,2)
      CALL PLOT(Ø.Ø,Ø.Ø,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 TPDRR,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'
0015        CALL ASSIGN(1,'DK1:MPPLTD.DAT',14)
0016        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
0028        IF(NPINT.GT.4096) STOP' ERROR IN INTERPOLATION SPECS'
0030        NBLKR=NPT/128
0031        NWDR=NPT*2
0032        NBYTR=NWDR*2+512
0033        NBLKBF=NDSTEP/2
0034        NBLKW=(NTR+24)*NBLKBF*NTIM
0035        NWDBF=NBLKBF*256
0036        NBYTBF=NWDBF*2
0037        IOFF=2*NDSTEP
0038        IBLKR=1
0039        IBLKW1=1
0040        IBLKW2=NBLKW/NTIM
0041        IA0=0
```

```
      IA1=4Ø96
      IC1=IA1+NPTS
      IC2=IC1+1
      IC3=IC2+1
      IC4=IC3+1
C
C SET UP INPUT FILES
C
      IF(INFLG.EQ.Ø)GOTO 1Ø
      READ(1,1ØØ2)FBUF
      CALL IRAD5Ø(12,FBUF,FSPECR)
      IF(LOOKUP(IRD,FSPECR).LT.Ø)STOP'LOOKUP ERROR'
C
C SET UP OUTPUT FILES
C
   1Ø IF(OUTFLG.EQ.Ø)GOTO 2Ø
      READ(1,1ØØ2)FBUF
      CALL IRAD5Ø(12,FBUF,FSPECW)
      IF(IENTER(IWRT,FSPECW,NBLKW).LT.Ø)STOP'ENTER ERROR'
C
C SET UP ROW COUNTER AND CLEAR PLOT BUFFER
C
   2Ø CONTINUE
      DO 15 I=1,8Ø
      JROW(I)=I
      DO 15 J=1,256
      IPBUF(J,I)=Ø
   15 CONTINUE
C
C  SET UP AP
C
      CALL APINIT
      CALL VCLR(Ø,1,8192)
      CALL APWR
C
C  MAIN LOOP FOR PLOTTING DIFFERENT TRACES
C
      DO 1ØØ I=1,NTR
      IFIL=I
C
C READ IN DATA FROM DISC
C
      IF(INFLG.EQ.Ø)GOTO 11Ø
      IF(IREADW(NWDR,XBUF,IBLKR,IRD).LT.Ø)STOP'READW ERR'
      IBLKR=IBLKR+NBLKR
      GOTO 12Ø
C
C TAPE READ ROUTINE
C
  11Ø ITRY=1
  3ØØ IF(ITRY.GT.3)GOTO 31Ø
      IF(IEOTR.GE.Ø)GOTO 32Ø
  31Ø WRITE(7,11ØØ)TPDRR,IFIL
 11ØØ FORMAT(' EOT ON READ DRIVE:',I2,' FILE NO:',I5)
```

```
ØØ83          WRITE(7,11Ø1)
ØØ84     11Ø1 FORMAT(' ENTER NEW READ DRIVE NUMBER:',$)
ØØ85          READ(5,11Ø2)TPDRR
ØØ86     11Ø2 FORMAT(I1)
ØØ87          IEOTR=Ø
ØØ88          IF(TPDRR.GT.2)STOP'EOT TERMINATE'
        C
        C DO READ
        C
ØØ9Ø      32Ø CALL TAPSUB(-1,TPDRR,ISTAT,IFLEN,IFIL,BUF,NBYTR,IEOTR)
ØØ91          IF(ISTAT.LT.Ø)WRITE(2,12ØØ)IFIL
ØØ93     12ØØ FORMAT(' WARNING READ RETRY FAILED ON FILE NO:',I5)
ØØ94          IF(IEOTR.LT.Ø)GOTO 33Ø
        C
        C WIND OVER EOF MARKER
        C
ØØ96          CALL TAPSUB(Ø,TPDRR,ISTAT, ,IFIL, , ,IEOTR)
ØØ97      33Ø ITRY=ITRY+1
ØØ98          IF(IHBLK(1).EQ."177777)GOTO 3ØØ
Ø1ØØ      12Ø CONTINUE
        C
        C SET UP PROCESSING CONSTANTS
        C
Ø1Ø1          XBUF(ISFIN+1)=1.Ø/FLOAT(NINT)
Ø1Ø2          XBUF(ISFIN+2)=-FLOAT(IOFF-1)
Ø1Ø3          XBUF(ISFIN+3)=FLOAT(IOFF)
Ø1Ø4          XBUF(ISFIN+4)=XSF
        C
        C  SECTION WHICH DEALS WITH INTERPLOATION
        C  AND SCALING OF DATA BEFORE PLOTTING
        C
Ø1Ø5          CALL APPUT(XBUF,IA1,NPTS+4,2)
Ø1Ø6          CALL APWD
Ø1Ø7          CALL VMOV(IA1,1,IAØ,NINT,NPTS)
Ø1Ø8          IF(NINT.EQ.1)GOTO 125
Ø11Ø          CALL VSUB(IA1,1,IA1+1,1,IA1,1,NPTS-1)
Ø111          CALL VSMUL(IA1,1,IC1,IA1,1,NPTS-1)
Ø112          IST=IAØ
Ø113          DO 13Ø J=2,NINT
Ø114          CALL VADD(IST,NINT,IA1,1,IST+1,NINT,NPTS-1)
Ø115          IST=IST+1
Ø116      13Ø CONTINUE
Ø117      125 CALL VSMUL(IAØ,1,IC4,IAØ,1,NPINT)
Ø118          CALL VTSADD(IAØ,1,"4427,IAØ,1,NPINT)
Ø119          CALL VCLIP(IAØ,1,IC2,IC3,IAØ,1,NPINT)
Ø12Ø          CALL VINT(IAØ,1,IAØ,1,NPINT)
Ø121          CALL VFIX(IAØ,1,IAØ,1,NPINT)
Ø122          CALL APWR
Ø123          CALL APGET(IDOT,Ø,NPINT,1)
Ø124          IBIT=16
Ø125          IWORD=256
Ø126          CALL APWD
Ø127          IDOT(1)=IDOT(1)+IOFF
Ø128          DO 3Ø IP=2,NPINT
```

```
      IBITL=IBIT
      IWORDL=IWORD
      IBIT=IBIT-1
      IF(IBIT.GT.0) GOTO 40
      IWORD=IWORD-1
      IF(IWORD.EQ.0)GOTO 97
      IBIT=16
C
C   SECTION WHERE POSITIVE (SHADED) LOBES ARE PLOTTED
C
   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
   50 IDOT(IP)=IDOT(IP)+IOFF
      IF(IDOT(IP).LT.1.OR.IDOT(IP-1).LT.1)GOTO 30
      IF(IDOT(IP).GT.IDOT(IP-1))GOTO 70
      IDTB=IDOT(IP)
      IDTE=IDOT(IP-1)
      MSKB=MASK(IBIT)
      MSKE=MASK(IBITL)
      IWORDB=IWORD
      IWORDE=IWORDL
      GOTO 80
   70 IDTB=IDOT(IP-1)
      IDTE=IDOT(IP)
      MSKB=MASK(IBITL)
      MSKE=MASK(IBIT)
      IWORDB=IWORDL
      IWORDE=IWORD
   80 IDTM=(IDTB+IDTE)/2
      IROW=IDTB
   90 IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
      IROW=IROW+1
      IF(IROW.LE.IDTM)GOTO 90
      IROW=IDTM
   95 IPBUF(IWORDE,JROW(IROW))=IPBUF(IWORDE,JROW(IROW)).OR.MSKE
      IROW=IROW+1
      IF(IROW.LE.IDTE)GOTO 95
   30 CONTINUE
   97 JST=JROW(1)
      JFIN=JROW(NDSTEP)
```

```
      LIN=1
C
C FILL OUTPUT BUFFER
C
      DO 150 L=JST,JFIN
      DO 150 J=129,256
      IDOT(LIN)=IPBUF(J,L)
      IPBUF(J,L)=0
      LIN=LIN+1
  150 CONTINUE
      CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLKW1)
C
C DO SECOND SLICE IF NECESSARY
C
      IF(NTIM.LT.2)GOTO 170
      LIN=1
      DO 160 L=JST,JFIN
      DO 160 J=1,128
      IDOT(LIN)=IPBUF(J,L)
      IPBUF(J,L)=0
      LIN=LIN+1
  160 CONTINUE
      CALL BUFOUT(BUF,OUTFLG,TPDRW2,IBLKW2)
C
C SORT JROW ARRAY
C
  170 CALL APPUT(JROW,0,80,1)
      CALL APGET(JROW,NDSTEP,80-NDSTEP,1)
      CALL APGET(JROW(81-NDSTEP),0,NDSTEP,1)
      CALL APWD
  100 CONTINUE
C
C  FLUSH BUFFER AT END OF A STRIP
C
      IRBEG=1
      IRFIN=NDSTEP
      DO 190 IF=1,3
      JST=JROW(IRBEG)
      JFIN=JROW(IRFIN)
C
C DO FIRST BUFFER
C
      LIN=1
      DO 200 L=JST,JFIN
      DO 200 J=129,256
      IDOT(LIN)=IPBUF(J,L)
      LIN=LIN+1
  200 CONTINUE
      CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLKW1)
C
C DO SECOND BUFFER IF NECESSARY
C
      IF(NTIM.LT.2)GOTO 195
      LIN=1
```

```
      DO 210 L=JST,JFIN
      DO 210 J=1,128
      IDOT(LIN)=IPBUF(J,L)
      LIN=LIN+1
  210 CONTINUE
      CALL BUFOUT(BUF,OUTFLG,TPDRW2,IBLKW2)
  195 IRBEG=IRBEG+NDSTEP
      IRFIN=IRFIN+NDSTEP
  190 CONTINUE
C
C PUT OUT EXTRA LINES TO HELP WITH TONER PROBLEM
C
      DO 220 ICL=1,NWDBF
  220 IDOT(ICL)=0
C
C SEND OUT EXTRA LINES
C
      DO 230 ISEN=1,20
      CALL BUFOUT(BUF,OUTFLG,TPDRW1,IBLKW1)
      IF(NTIM.GE.2)CALL BUFOUT(BUF,OUTFLG,TPDRW2,IBLKW2)
  230 CONTINUE
      CALL CLOSEC(IWRT)
      STOP 'NORMAL TERMINATION'
      END
```

FORTRAN IV V02.04 THU 08-JAN-81 00:34:40 PAGE 001

```
      SUBROUTINE BUFOUT(IBUF,OUTFLG,TPDRW,IBLK)
C
C THIS A ROUTINE FOR PUTTING OUT THE BUFFERS FROM
C THE PLOTTING PROGRAM MPSPLT
C
      INTEGER*2 IBUF(1),OUTFLG
      LOGICAL*1 TPDRW,ISTAT
      COMMON /IO/ NBLKBF,NWDBF,NBYTBF,ICHAN
      DATA ICOUNT/0/
C
C KEEP A COUNT OF NUMBER OF BUFFERS WRITTEN
C
      ICOUNT=ICOUNT+1
      IF(OUTFLG.EQ.0)GOTO 10
      IF(IWRITW(NWDBF,IBUF(257),IBLK,ICHAN).LT.0)STOP'WRITE ERROR'
      IBLK=IBLK+NBLKBF
      RETURN
C
C DO TAPE OUTPUT
C
   10 NBYT=NBYTBF+512
      IFLEN=NBLKBF+1
      IEOTW=0
      CALL TAPSUB(1,TPDRW,ISTAT,IFLEN,ICOUNT,IBUF,NBYT,IEOTW)
C
C CHECK FOR ERRORS
C
      IF(IEOTW.GE.0)GOTO 20
      WRITE(7,1000)TPDRW,ICOUNT
 1000 FORMAT(' EOT ON WRITE DRIVE:',I2,' BUFFER NUMBER:'I5)
      WRITE(7,1100)
 1100 FORMAT(' ENTER NEW WRITE DRIVE NUMBER:',$)
      READ (5,1200)TPDRW
 1200 FORMAT(I1)
      IEOTW=0
      IF(TPDRW.GT.2)STOP' EOT TERMINATE'
   20 IF(ISTAT.GE.0)GOTO 30
      WRITE(7,1300)ICOUNT
 1300 FORMAT(' FATAL WRITE ERROR ON BUFFER:',I5)
      STOP' WRITE ERROR TERMINATION'
   30 RETURN
      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
ØØØ1        DIMENSION IVEL(2ØØ),ATIME1(2Ø),AVINT1(2Ø),AVRMS1(2Ø)
ØØØ2        COMMON /PPEP1/IX1,IY1,IX2,IY2,ISCCN,NSCCN,NBAND,NPIS,NIPØ,
           >NIPM1,LVNES,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
ØØØ3        DATA LMASK1/"1Ø421Ø/,LMASK2/"177777/
ØØØ4        CALL ASSIGN(1,'DK2:MPVDAT.DAT',14)
ØØØ5        CALL PLOTS(Ø,Ø,Ø)
      C
      C DATA INPUT
      C
ØØØ6        WRITE(7,1ØØØ)
ØØØ7   1ØØØ FORMAT(' ENTER TRACE LENGTH IN SECS(F1Ø.Ø):',$)
ØØØ8        READ(5,1ØØ1) TSEC
ØØØ9   1ØØ1 FORMAT(F1Ø.Ø)
ØØ1Ø        WRITE(7,1Ø11)
ØØ11   1Ø11 FORMAT(' ENTER TIME DELAY TO START OF TRACES(F1Ø.Ø):',$)
ØØ12        READ(5,1ØØ1)TDELAY
ØØ13        WRITE(7,1ØØ2)
ØØ14   1ØØ2 FORMAT(' ENTER TRACE INTERVAL IN INCHES(F1Ø.Ø):',$)
ØØ15        READ(5,1ØØ1)TSPACE
ØØ16        WRITE(7,1ØØ3)
ØØ17   1ØØ3 FORMAT(' ENTER START TRACE NO(I5):',$)
ØØ18        READ(5,1ØØ4)ISTART
ØØ19   1ØØ4 FORMAT(I5)
ØØ2Ø        WRITE(7,1ØØ5)
ØØ21   1ØØ5 FORMAT(' ENTER LAST TRACE NUMBER(I5):',$)
ØØ22        READ(5,1ØØ4)IEND
ØØ23        WRITE(7,1ØØ6)
ØØ24   1ØØ6 FORMAT(' ENTER THE SCAN AMPLIFICATION FACTOR(I1):',$)
ØØ25        READ(5,1ØØ9)ISCAN
ØØ26   1ØØ9 FORMAT(I1)
ØØ27        WRITE(7,1Ø1Ø)
ØØ28   1Ø1Ø FORMAT(' ENTER 1 FOR GRID PLOTTING Ø IF NOT(I1):',$)
ØØ29        READ(5,1ØØ9)IBKFLG
ØØ3Ø        WRITE(7,1ØØ7)
ØØ31   1ØØ7 FORMAT(' ENTER 1 IF WANT DOCUMENTATION Ø IF NOT(I1):',$)
ØØ32        READ(5,1ØØ9)IDSCAN
ØØ33        WRITE(7,1ØØ8)
ØØ34   1ØØ8 FORMAT(' ENTER 1 IF VELOCITIES TO BE PLOTTED Ø IF NOT(I1):',$)
ØØ35        READ(5,1ØØ9)IVSCAN
      C
      C BASIC PROCESSING PARAMETERS
      C
ØØ36        TPAPER=1Ø.56
ØØ37        TMAX=8.192
ØØ38        TLENTH=1Ø.24
ØØ39        DSHIFT=1.Ø
ØØ4Ø        TSHIFT=1.6
ØØ41        DSIZE=7.25
```

```
      VMAX=6.Ø
      VBOX=1.Ø
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,Ø=OFF
C   DSHIFT=X SHIFT FORM ORIGIN
C   TSHIFT=SEPARATION OF T SLICES
C   DSIZE=DECLARATION SIZE
C   IVSCAN=Ø-NO VELOCITIES
C          1-VELOCITY DISPLAY
C   VMAX=MAX VELOCITY
C   VBOX=VELOCITY BOX SIZE
C
C
C  FORM DISPLAY CONSTANTS AND FIND NO OF TIME SLICES
C
      IPAD=1
      TSTART=TMAX
      TEQ=TSEC*ISCAN
   1Ø IF(TEQ.LE.TSTART) GOTO 11
      IPAD=IPAD+1
      TSTART=TSTART+TMAX
      GOTO 1Ø
      IPAD=IPAD*ISCAN
   11 TSTART=TMAX/FLOAT(ISCAN)
C
C IPAD=NO OF TIME PADS
C
C TSTART=NO OF TRUE SECONDS PER PAD
C
      D1TIM=-TLENTH/TSTART
      D1ØTIM=D1TIM*Ø.1
C
C STEPS FOR 1 AND 1/1Ø SEC INTERVALS
C
      ITRACE=IEND-ISTART+1
      ITR1Ø=ITRACE/1Ø
C
C NO OF TRACES AND DECADE POINTS
C
      IT1=ISTART/1Ø
      IT2=IT1*1Ø
      IST1=IT2+1Ø-ISTART
      IF(ISTART.EQ.IT2)ITR1Ø=ITR1Ø-1
C
C GET TRACE AND DECADE STARTING POSITIONS
```

```
      C AND REDUCE NO OF DECADES BY 1 IF STARTING ON A DECADE
      C
ØØ63        XPOPST=TSPACE*Ø.5
ØØ64        XPOP1Ø=XPOPST+TSPACE*FLOAT(IST1)
      C
      C STARTING POSITIONS FOR TRACES AND DECADES
      C
ØØ65        TWIDTH=FLOAT(ITRACE)*TSPACE
ØØ66        T1ØSPA=TSPACE*1Ø.Ø
      C
      C WIDTH OF DIPLAY AND DECADE TRACE INTERVAL
      C
ØØ67        ITNUM=IT2+1Ø
      C
      C STARTING DECADE NUMBER
      C
ØØ68        TUPPR=TPAPER-Ø.Ø1
      C
      C UPPER DISPLAY LIMIT
      C
ØØ69        TSLICE=Ø.3
ØØ7Ø        TSIZE=TSLICE*Ø.5
ØØ71        PSIZE=(TUPPR-TLENTH)*Ø.25
      C
      C ANNOTATION PARAMETERS
      C
ØØ72        TOTAL=DSHIFT+FLOAT(IPAD)*(TWIDTH+TSHIFT)+
           >FLOAT(IVSCAN)*(TWIDTH+TSHIFT)+DSIZE
ØØ73        PORT(1,1)=TOTAL
      C
      C TOTAL DISPLAY SIZE
      C
ØØ74        XSTART=DSHIFT
ØØ75        YSTART=Ø.Ø
ØØ76        IF(IBKFLG.EQ.Ø)GOTO 2Ø1
      C
      C
      C  BEGIN MAIN DISPLAY LOOP
      C  SWITCHING OUT VELOCITY PLOTS WHEN
      C  NOT REQUIRED
      C
ØØ78        DO 2ØØ ISCNT=1,IPAD
ØØ79        CALL PLOT(XSTART,YSTART,-3)
ØØ8Ø        X1=-TSLICE
ØØ81        X2=TWIDTH+TSLICE
ØØ82        IF(ISCNT.NE.1)GOTO 1Ø1
      C
      C DRAW BORDER FOR FIRST FRAME
      C
ØØ84        CALL NEWPEN(3)
ØØ85        CALL PLOT(X1,Ø.Ø,3)
ØØ86        CALL PLOT(X1,TUPPR,2)
ØØ87        CALL PLOT(X2,TUPPR,2)
ØØ88        CALL PLOT(X2,Ø.Ø,2)
```

```
ØØ89          IF(IPAD.EQ.1)CALL PLOT(X1,Ø.Ø,2)
       C
       C DRAW INSIDE MARGIN
       C
ØØ91          CALL NEWPEN(1)
ØØ92          CALL PLOT(Ø.Ø,Ø.Ø,3)
ØØ93          CALL PLOT(Ø.Ø,TUPPR,2)
ØØ94          CALL PLOT(X1,TLENTH,3)
ØØ95          CALL PLOT(X2,TLENTH,2)
ØØ96          CALL PLOT(TWIDTH,TUPPR,3)
ØØ97          CALL PLOT(TWIDTH,Ø.Ø,2)
       C
       C ANNOTATE MARGIN
       C
ØØ98          YA=TLENTH+PSIZE*Ø.8
ØØ99          X=-TSLICE*Ø.33
Ø1ØØ          CALL SYMBOL(X,YA,PSIZE,3HSEC,9Ø.Ø,3)
Ø1Ø1          X=X+X2
Ø1Ø2          CALL SYMBOL(X,YA,PSIZE,3HSEC,9Ø.Ø,3)
       C
       C ALL SURROUND PLOTTED NOW
       C INSERT SHOT POINT LOCATIONS
       C
Ø1Ø3          X=XPOP1Ø
Ø1Ø4          Y=TLENTH+Ø.33*(TUPPR-TLENTH)
Ø1Ø5          COUNT=FLOAT(ITNUM)
Ø1Ø6          JLIM=ITR1Ø-1
Ø1Ø7          DO 1ØØ I=1,ITR1Ø
Ø1Ø8          CALL NUMBER(X,Y,PSIZE,COUNT,Ø.Ø,-1)
Ø1Ø9          X=X+T1ØSPA
Ø11Ø      1ØØ COUNT=COUNT+1Ø.Ø
       C
       C VELOCITY ANALYSIS POSITIONING IF REQD
       C
Ø111          IF(IVSCAN.EQ.Ø)GOTO 31
Ø113          READ(1,25)IVCNT
Ø114       25 FORMAT(12I5)
Ø115          IF(IVCNT.LE.Ø)GOTO 31
       C
       C IVCNT=NO OF VELOCITY SCANS
       C
Ø117          READ(1,25)(IVEL(I),I=1,IVCNT)
       C
       C READ IN VEL POSITIONS
       C
Ø118          XBASE=Ø.5*TSPACE
Ø119          YBASE=TLENTH
Ø12Ø          DO 29 I=1,IVCNT
Ø121          ISHIFT=IVEL(I)-ISTART
Ø122          SHIFT=FLOAT(ISHIFT)*TSPACE+XBASE-Ø.5*PSIZE
Ø123       29 CALL SYMBOL(SHIFT,YBASE,PSIZE,14,Ø.Ø,-1)
Ø124          GOTO 31
       C
       C BORDER FOR OTHER DATA PADS THAN FIRST
```

```
C
  1Ø1 CALL NEWPEN(3)
      CALL PLOT(X1,Ø.Ø,3)
      CALL PLOT(X1,TUPPR,2)
      CALL NEWPEN(1)
      CALL PLOT(Ø.Ø,TUPPR,2)
      CALL PLOT(Ø.Ø,Ø.Ø,2)
      CALL PLOT(TWIDTH,Ø.Ø,3)
      CALL PLOT(TWIDTH,TUPPR,2)
      CALL NEWPEN(3)
      CALL PLOT(X2,TUPPR,3)
      CALL PLOT(X2,Ø.Ø,2)
      IF(ISCNT.EQ.IPAD)CALL PLOT(X1,Ø.Ø,2)
C
C INSERT SHOT POSITION LINES
C
   31 ITR=ITRACE-1
      CALL GRID(XPOPST,Ø.Ø,ITR,TSPACE,-1,TLENTH,LMASK1)
      ITR=ITR1Ø-1
      CALL GRID(XPOP1Ø,Ø.Ø,ITR,T1ØSPA,-1,TLENTH,LMASK2)
C
C INSERT TIMING LINES
C
      TBASE=(FLOAT(ISCNT-1)*TSTART*1Ø.Ø)+(TDELAY*1Ø.Ø)
      IT1=IFIX(TBASE)
      T1=FLOAT(IT1)
      TERR=TBASE-T1
      TBASE=TBASE*Ø.1
      IT1=IFIX(TBASE)
      T2=FLOAT(IT1)
      TERR1Ø=TBASE-T2
      IF(TERR1Ø.GE.1.Ø)TERR1Ø=TERR1Ø-1.Ø
      T1ØST=TLENTH+D1ØTIM*(1.Ø-TERR)
      T1ST=TLENTH+D1TIM*(1.Ø-TERR1Ø)
      TINT=(TSTART+TERR)*1Ø.Ø
      ITR=IFIX(TINT)
      CALL GRID(Ø.Ø,T1ØST,-1,TWIDTH,ITR,D1ØTIM,LMASK1)
      I1=IFIX(TSTART)
      ITR=I1-1
      CALL GRID(Ø.Ø,T1ST,-1,TWIDTH,ITR,D1TIM,LMASK2)
C
C INSERT ANNOTATION ON SECONDS
C
      X=-Ø.66*TSLICE
      X1=TWIDTH+Ø.33*TSLICE
      Y=T1ST
      TIMT=IFIX(Ø.1*T1+1.Ø)
      TIM=FLOAT(TIMT)
      CALL NEWPEN(1)
      DO 11Ø I=1,I1
      CALL NUMBER(X,Y,TSIZE,TIM,Ø.Ø,-1)
      CALL NUMBER(X1,Y,TSIZE,TIM,Ø.Ø,-1)
      TIM=TIM+1.Ø
  11Ø Y=Y+D1TIM
```

```
      XSTART=TWIDTH+TSHIFT
      Y=0.0
  200 CONTINUE
C
C VELOCITY ANALYSIS PLOT PROGRAM
C
  201 IF(IVSCAN.EQ.0)GOTO 641
      CALL PLOT(XSTART,YSTART,-3)
      DO 64 I=1,IVCNT
      READ(1,25)NVEL
      READ(1,26)(ATIME1(J),AVINT1(J),AVRMS1(J),J=1,NVEL)
   26 FORMAT(3F10.0)
      IVEL1=IVEL(I)
      VBSIZE=0.22*VBOX
      VBLINE=0.11*VBOX
      VBSTX=VPLT1-0.5*VBSIZE*3.0
      VBSTY=TLENTH*(0.5-AMARK*0.25)-23.0*VBLINE
C
C DRAW BOXES FOR VELOCITY PICKS
C
      CALL PLOT(VBSTX,VBSTY,3)
      CALL GRID(VBSTX,VBSTY,3,VBSIZE,22,VBLINE,LMASK2)
      SIZE=0.04*VBOX
      COFFST=0.025*VBOX
      Y=TLENTH*(0.5-AMARK*0.25)-VBSIZE+COFFST
      X=VPLT1-2.0*SIZE
      AV=FLOAT(IVEL1)
      CALL NUMBER(X,Y,SIZE,AV,0.0,-1)
      Y=Y-VBLINE
      X=VBSTX+0.1*VBSIZE
      XST=X
C
C DO TITLES
C
      CALL SYMBOL(X,Y,SIZE,4HTIME,0.0,4)
      X=X+VBSIZE
      CALL SYMBOL(X,Y,SIZE,4HVINT,0.0,4)
      X=X+VBSIZE
      CALL SYMBOL(X,Y,SIZE,4HVRMS,0.0,4)
      Y=Y-VBLINE
      X=XST
C
C PUT IN THE NUMBERS
C
      DO 63 JJ=1,NVEL
      CALL NUMBER(X,Y,SIZE,ATIME1(JJ),0.0,3)
      X=X+VBSIZE
      CALL NUMBER(X,Y,SIZE,AVINT1(JJ),0.0,3)
      X=X+VBSIZE
      CALL NUMBER(X,Y,SIZE,AVRMS1(JJ),0.0,3)
      X=XST
   63 Y=Y-VBLINE
      IF(AMARK.EQ.0.0)GOTO 632
      AMARK=0.0
```

```
            GOTO 64
      632   AMARK=1.0
       64   CONTINUE
      641   CONTINUE
C
C PLOT COSMETICS
C
            IF(IDSCAN.EQ.0)GOTO 73
            XSTART=TWIDTH+TSHIFT
            YSTART=0.0
            CALL PLOT(XSTART,YSTART,-3)
C
C REORIGIN
C
            X2=DSIZE
            Y2=TLENTH
            CALL NEWPEN(4)
            CALL PLOT(0.0,Y2,2)
            CALL PLOT(X2,Y2,2)
            CALL PLOT(X2,0.0,2)
            CALL PLOT(0.0,0.0,2)
            X1=0.05
            X2=X2-X1
            Y1=X1
            Y2=Y2-Y1
            CALL NEWPEN(1)
            CALL PLOT(X1,Y1,3)
            CALL PLOT(X1,Y2,2)
            CALL PLOT(X2,Y2,2)
            CALL PLOT(X2,Y1,2)
            CALL PLOT(X1,Y1,2)
            CALL PLOT(0.0,0.0,3)
C
C SURROUND DONE
C
            SIZE1=0.4
            SIZE2=0.3
            SIZE3=0.2
            SIZE4=0.1
            XMID=0.5*DSIZE
            Y=TLENTH
            X=XMID-8.0*SIZE1
            Y=Y-1.5*SIZE1
C
C DRAW IN THE HEADER
C
            CALL NEWPEN(3)
            CALL SYMBOL(X,Y,SIZE1,17HDURHAM UNIVERSITY,0.0,17)
            X=XMID-9.0*SIZE2
            Y=Y-1.5*SIZE2
            CALL NEWPEN(1)
            CALL SYMBOL(X,Y,SIZE2,18HSEISMIC PROCESSING,0.0,18)
            CALL NEWPEN(1)
            X=DSIZE*0.1
```

```
      Y=Y-1.5*SIZE3
C
C READ IN TITLE AND PLOT
C
      READ(1,70)(IVEL(I),I=1,40)
   70 FORMAT(40A2)
      CALL SYMBOL(X,Y,SIZE3,IVEL,0.0,80)
      Y=Y-0.2*SIZE3
      X=0.0
      CALL PLOT(X,Y,3)
      CALL PLOT(DSIZE,Y,2)
      Y=Y-1.5*SIZE3
      X=XMID-8.0*SIZE3
      CALL SYMBOL(X,Y,SIZE3,17HSYSTEM PARAMETERS,0.0,17)
      X=0.1*DSIZE
      Y=Y-1.5*SIZE3
      READ(1,25)ILINES
C
C FILL IN PROCESSING PARAMETERS
C
      DO 72 I=1,ILINES
      READ(1,70)(IVEL(J),J=1,40)
      CALL SYMBOL(X,Y,SIZE4,IVEL,0.0,80)
   72 Y=Y-1.5*SIZE4
   73 CONTINUE
      CALL PLOT(0.0,0.0,999)
      STOP
      END
```

Section Plotting Merge and Post-Process :- MPMERG

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

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

Input Parameters

READ(1,1000)LPLT,NSTRIP,NDSTEP,IBKFLG,IPLFLG

1000 FORMAT(5I5)

LPLT.....Number of rasters in total plot

NSTRIP...Number of strips to be plotted

NDSTEP...Number of dots between traces

IBKFLG...Background flag

0 - No background to plot

1 - Plot a background

IPLFLG...Trace plot flag

0 - No trace plot

1 - Plot raster trace plot

READ(1,1000)LSTP,LENP,IOFLGP,ITPDR1,ITPDR2

LSTP.....Output raster position of first raster in trace plot

LENP.....Output raster position of last raster in trace plot

IOFLGP...Input flag for trace plot

0 - read from tape

1 - read from disc

ITPDR1...Input Tape drive number 1

ITPDR2...Input tape drive number 2

<u>IF(IOFLGP.NE.0)READ(1,1002)FBUFP</u>

FBUFP....Input file for trace plot

<u>READ(1,1000)LSTB,LFINB,IOFLGB,ITPDRB</u>

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

<u>IF(IOFLGB.NE.0)READ(1,1002)FBUFB</u>

FBUFB....Input file for background plot

```
C     M.J.POULTER OCT 80
C       POST PROCESS AND MERGE PROG FOR
C       SEISMIC PLOT SYSTEM
        INTEGER*2 IPBUF(2112),LBUF(132)
        REAL*4 FBUF(3)
        REAL*8 FSPECP,FSPECB
        LOGICAL*1 ITPDR1,ITPDR2,ITPDRP,ITPDRB
        DATA DEV/3RRK /
C
C   SET UP AP TO USE FOR ZEROING ARRAYS
C
        CALL APINIT
        CALL VCLR(0,1,2112)
C
C   SET UP I/O DEFINITIONS
C
        IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
        CALL ASSIGN(1,'DK1:MPMERG.DAT',14)
        CALL ASSIGN(2,'DK1:MPMERG.LOG',14)
C
C GET INPUT PARAMETERS
C
        READ(1,1000)LPLT,NSTRIP,NDSTEP,IBKFLG,IPLFLG
  1000  FORMAT(5I5)
        IF(IPLFLG.EQ.0)GOTO 10
        READ(1,1000)LSTP,LFINP,IOFLGP,ITPDR1,ITPDR2
        IF(IOFLGP.EQ.0)GOTO 10
        READ(1,1001)FBUF
  1001  FORMAT(3A4)
        CALL IRAD50(12,FBUF,FSPECP)
        ICHP=IGETC()
        IF(LOOKUP(ICHP,FSPECP).LT.0)STOP'LOOKUP ERROR'
    10  IF(IBKFLG.EQ.0)GOTO 20
        READ(1,1000)LSTB,LFINB,IOFLGB,ITPDRB
        IF(IOFLGB.EQ.0)GOTO 20
        READ(1,1001)FBUF
        CALL IRAD50(12,FBUF,FSPECB)
        ICHB=IGETC()
        IF(LOOKUP(ICHB,FSPECB).LT.0)STOP'LOOKUP ERROR'
    20  CONTINUE
C
C SET UP MATRIX FOR MAIN LOOP
C
        CALL MTXSET
        CALL MTX(IPBUF,0,1)
        LNUM=0
        NTIM=0
        IST=1057
        NBYTP=NDSTEP*256+512
        NBYTB=4608
        LIMP=4608
        NWDP=NDSTEP/2*256
        IF(IOFLGP.EQ.0)LIMP=2560+NWDP
        LIMB=2304
```

```
0048          ITPDRP=ITPDR1
0049          IEOFP=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          EOFFLG=1
       C
       C  THIS LOOP FILLS ONE LINE OF PLOT BUFFER
       C  WITH INPUT FROM EACH PLOT MASK
       C
       C
       C DO SEISMIC SECTION FIRST
       C
0058          IF(IPLFLG.EQ.0)GOTO 60
0060          IF(LNUM.GT.LFINP)GOTO 60
0062          IF(IEOFP.NE.0)GOTO 60
0064          EOFFLG=0
0065          IF(LNUM.LT.LSTP)GOTO 60
0067          IPOS=IPSV
0068          ICD=1
       C
       C FILL LINE BUFFER
       C
0069          CALL LINFIL(LBUF,ICD,IEOFP,ITPDRP,IOFLGP,NBYTP,LNUM,ICHP,LIMP)
0070          DO 70 J=1,132
0071          IPBUF(IPOS)=IPBUF(IPOS).OR.LBUF(J)
0072       70 IPOS=IPOS+1
       C
       C DO BACKGROUND
       C
0073       60 IF(IBKFLG.EQ.0)GOTO 80
0075          IF(LNUM.GT.LFINB)GOTO 80
0077          IF(IEOFB.NE.0)GOTO 80
0079          EOFFLG=0
0080          IF(LNUM.LT.LSTB)GOTO 80
0082          IPOS=IPSV
0083          ICD=2
0084          CALL LINFIL(LBUF,ICD,IEOFB,ITPDRB,IOFLGB,NBYTB,LNUM,ICHB,LIMB)
0085          DO 90 J=1,132
0086          IPBUF(IPOS)=IPBUF(IPOS).OR.LBUF(J)
0087       90 IPOS=IPOS+1
0088       80 IPSV=IPOS
0089       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."177777)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
      DIMENSION IPBUF(128,80),JROW(80),FBUF(3),
     %IDOT(2048),XBUF(2052),MASK(16),NSAVE(20)
      REAL*8 FSPECR,FSPECW
      EQUIVALENCE (XBUF(1),IDOT(1))
      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
      WRITE(7,1005)
 1005 FORMAT(' ENTER NO OF TRACES',/,
     %' NO OF POINTS PER TRACE',/,
     %' NO OF TIMES TO EXPAND',/,
     %' AND STEP SIZE (4I5)')
      READ(5,1000) NTR,NPT,NDPT,NDSTEP
 1000 FORMAT(4I5)
      WRITE(7,1006)
 1006 FORMAT(' ENTER SCALE FACTOR:',$)
      READ(5,1001) XSF
 1001 FORMAT(F10.0)
      WRITE(7,1007)
 1007 FORMAT(' ENTER INPUT FILE NAME:',$)
      READ(5,1002)FBUF
 1002 FORMAT(3A4)
      CALL IRAD50(12,FBUF,FSPECR)
      WRITE(7,1008)
 1008 FORMAT(' ENTER OUTPUT FILE NAME:',$)
      READ(5,1002)FBUF
      CALL IRAD50(12,FBUF,FSPECW)
C
C   SET UP CONSTANTS AND FILE ACCESS
C
      ICHR=IGETC()
      ICHW=IGETC()
      IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
      IER=LOOKUP(ICHR,FSPECR)
      IF(IER.LT.0)WRITE(7,*)IER
      IF(IER.LT.0)STOP'LOOKUP ERR'
      NFIN=NPT*NDPT-1
      NBLKR=NPT/128
      NWDR=2*NPT
      NBLKW=(NTR+4)*(NDSTEP/2)
      NBLKBF=NDSTEP/2
      NWBUF=NBLKBF*256
      IOFF=2*NDSTEP
      IBLKR=1
```

```
            IBLKW=1
            NTIM=1
            NSTRIP=Ø
            IBOFF=Ø
      C
      C  SECTION WHICH SETS UP PARAMETERS TO ALLOW
      C  STRIPPING OF PLOT IF REQUIRED
      C
            IF(NFIN.GT.2Ø48)GOTO 1
            IF(IENTER(ICHW,FSPECW,NBLKW).LT.Ø)STOP'ENTER ERR'
            NSTRIP=1
            GOTO 2
          1 NS=NFIN
          3 NSAVE(NTIM)=2Ø47
            NS=NS-2Ø48
            NTIM=NTIM+1
            IF(NTIM.GT.2Ø)STOP'TOO MANY STRIPS'
            IF(NS.GT.2Ø48)GOTO 3
            NSAVE(NTIM)=NS
            NBLKW=NBLKW*NTIM
            IF(IENTER(ICHW,FSPECW,NBLKW).LT.Ø)STOP'ENTER ERR'
      C
      C  START OF LOOP IF STRIPPING NECESSARY
      C
          4 NSTRIP=NSTRIP+1
            NFIN=NSAVE(NSTRIP)
            NPT=(NFIN+1)/NDPT
            NWDR=2*NPT
            IBLKR=1+IBOFF
            NBOF=NPT/128
            IBOFF=IBOFF+NBOF
      C
      C  SET UP ROW COUNTER FOR BUFFER USAGE
      C
          2 DO 5 I=1,8Ø
          5 JROW(I)=I
      C
      C  SET UP AP AND CLEAR PLOT BUFFER
      C
            CALL APINIT
            CALL VCLR(Ø,1,512Ø)
            CALL APWR
            CALL APGET(IPBUF(1,1),Ø,512Ø,2)
      C
      C  MAIN LOOP FOR PLOTTING DIFFERENT TRACES
      C
            DO 1Ø I=1,NTR
            IF(IREADW(NWDR,XBUF,IBLKR,ICHR).LT.Ø)STOP'READW ERR'
            IBLKR=IBLKR+NBLKR
            XBUF(NPT+1)=1.Ø/FLOAT(NDPT)
            XBUF(NPT+2)=-FLOAT(IOFF-1)
            XBUF(NPT+3)=FLOAT(IOFF)
            XBUF(NPT+4)=XSF
      C
```

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

```
      5Ø IDOT(IP)=IDOT(IP)+IOFF
         IF(IDOT(IP).LT.1.OR.IDOT(IP-1).LT.1)GOTO 3Ø
         IF(IDOT(IP).GT.IDOT(IP-1))GOTO 7Ø
         IDTB=IDOT(IP)
         IDTE=IDOT(IP-1)
         MSKB=MASK(IBIT)
         MSKE=MASK(IBITL)
         IWORDB=IWORD
         IWORDE=IWORDL
         GOTO 8Ø
      7Ø IDTB=IDOT(IP-1)
         IDTE=IDOT(IP)
         MSKB=MASK(IBITL)
         MSKE=MASK(IBIT)
         IWORDB=IWORDL
         IWORDE=IWORD
      8Ø IDTM=(IDTB+IDTE)/2
         IROW=IDTB
      9Ø IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
         IROW=IROW+1
         IF(IROW.LE.IDTM)GOTO 9Ø
         IROW=IDTM
     1ØØ IPBUF(IWORDE,JROW(IROW))=IPBUF(IWORDE,JROW(IROW)).OR.MSKE
         IROW=IROW+1
         IF(IROW.LE.IDTE)GOTO 1ØØ
      3Ø CONTINUE
     12Ø JST=JROW(1)
C
C  WRITE OUT PART OF BUFFER AND INITIALISE
C  FOR A NEW TRACE
C
         IF(IWRITW(NWBUF,IPBUF(1,JST),IBLKW,ICHW).LT.Ø)STOP'WRITE ERR'
         IBLKW=IBLKW+NBLKBF
         CALL APPUT(JROW,Ø,8Ø,1)
         CALL APGET(JROW,NDSTEP,8Ø-NDSTEP,1)
         CALL APGET(JROW(81-NDSTEP),Ø,NDSTEP,1)
         CALL APWD
         CALL VCLR(Ø,1,NWBUF)
         CALL APGET(IPBUF(1,JST),Ø,NWBUF,1)
      1Ø CONTINUE
         IROW=1
C
C  FLUSH BUFFER AT END OF A STRIP
C
         DO 11Ø IF=1,3
         IF(IWRITW(NWBUF,IPBUF(1,JROW(IROW)),IBLKW,ICHW).LT.Ø)STOP'WRT ER'
         IBLKW=IBLKW+NBLKBF
         IROW=IROW+NDSTEP
     11Ø CONTINUE
C
C  CHECK IF MORE STRIPS TO BE DONE
C
         IF(NSTRIP.LT.NTIM)GOTO 4
         CALL CLOSEC(ICHW)
```



```
         STOP 'NORMAL TERMINATION'
         END
```

Gather Ploting(Large Separation) :- MPGPLI

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
         DIMENSION IPBUF(128,80),JROW(80),FBUF(3),
        %IDOT(2048),XBUF(2052),MASK(16),NSAVE(20)
         REAL*8 FSPECR,FSPECW
         EQUIVALENCE (XBUF(1),IDOT(1))
         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
         WRITE(7,1005)
 1005    FORMAT(' ENTER NO OF TRACES',/,
        %' NO OF POINTS PER TRACE',/,
        %' NO OF TIMES TO EXPAND')
         READ(5,1000) NTR,NPT,NDPT
 1000    FORMAT(4I5)
         WRITE(7,1006)
 1006    FORMAT(' ENTER SCALE FACTOR:',$)
         READ(5,1001) XSF
 1001    FORMAT(F10.0)
         WRITE(7,1007)
 1007    FORMAT(' ENTER INPUT FILE NAME:',$)
         READ(5,1002)FBUF
 1002    FORMAT(3A4)
         CALL IRAD50(12,FBUF,FSPECR)
         WRITE(7,1008)
 1008    FORMAT(' ENTER OUTPUT FILE NAME:',$)
         READ(5,1002)FBUF
         CALL IRAD50(12,FBUF,FSPECW)
C
C     SET UP CONSTANTS AND FILE ACCESS
C
         ICHR=IGETC()
         ICHW=IGETC()
         IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
         IER=LOOKUP(ICHR,FSPECR)
         IF(IER.LT.0)WRITE(7,*)IER
         IF(IER.LT.0)STOP'LOOKUP ERR'
         NDSTEP=20
         NBUFSZ=80
         NBSPAC=NDSTEP/2
         NWSPAC=NBSPAC*256
         NFIN=NPT*NDPT-1
         NBLKR=NPT/128
         NWDR=2*NPT
         NBLKW=(NTR*NBUFSZ/2)+(NTR*10)
         NBLKBF=NBUFSZ/2
```

```
ØØ41        NWBUF=NBLKBF*256
ØØ42        IOFF=2*NDSTEP
ØØ43        IBLKR=1
ØØ44        IBLKW=1
ØØ45        NTIM=1
ØØ46        NSTRIP=Ø
ØØ47        IBOFF=Ø
      C
      C  SECTION WHICH SETS UP PARAMETERS TO ALLOW
      C  STRIPPING OF PLOT IF REQUIRED
      C
ØØ48        IF(NFIN.GT.2Ø48)GOTO 1
ØØ5Ø        IF(IENTER(ICHW,FSPECW,NBLKW).LT.Ø)STOP'ENTER ERR'
ØØ52        NSTRIP=1
ØØ53        GOTO 2
ØØ54      1 NS=NFIN
ØØ55      3 NSAVE(NTIM)=2Ø47
ØØ56        NS=NS-2Ø48
ØØ57        NTIM=NTIM+1
ØØ58        IF(NTIM.GT.2Ø)STOP'TOO MANY STRIPS'
ØØ6Ø        IF(NS.GT.2Ø48)GOTO 3
ØØ62        NSAVE(NTIM)=NS
ØØ63        NBLKW=NBLKW*NTIM
ØØ64        IF(IENTER(ICHW,FSPECW,NBLKW).LT.Ø)STOP'ENTER ERR'
      C
      C  START OF LOOP IF STRIPPING NECESSARY
      C
ØØ66      4 NSTRIP=NSTRIP+1
ØØ67        NFIN=NSAVE(NSTRIP)
ØØ68        NPT=(NFIN+1)/NDPT
ØØ69        NWDR=2*NPT
ØØ7Ø        IBLKR=1+IBOFF
ØØ71        NBOF=NPT/128
ØØ72        IBOFF=IBOFF+NBOF
      C
      C  SET UP ROW COUNTER FOR BUFFER USAGE
      C
ØØ73      2 DO 5 I=1,8Ø
ØØ74      5 JROW(I)=I
      C
      C  SET UP AP AND CLEAR PLOT BUFFER
      C
ØØ75        CALL APINIT
ØØ76        CALL VCLR(Ø,1,512Ø)
ØØ77        CALL APWR
ØØ78        CALL APGET(IPBUF(1,1),Ø,512Ø,2)
      C
      C  MAIN LOOP FOR PLOTTING DIFFERENT TRACES
      C
ØØ79        DO 1Ø I=1,NTR
ØØ8Ø        IF(IREADW(NWDR,XBUF,IBLKR,ICHR).LT.Ø)STOP'READW ERR'
ØØ82        IBLKR=IBLKR+NBLKR
ØØ83        XBUF(NPT+1)=1.Ø/FLOAT(NDPT)
ØØ84        XBUF(NPT+2)=-FLOAT(IOFF-1)
```

```
      XBUF(NPT+3)=FLOAT(IOFF)
      XBUF(NPT+4)=XSF
C
C  SECTION WHICH DEALS WITH INTERPLOATION
C  AND SCALING OF DATA BEFORE PLOTTING
C
      CALL APPUT(XBUF,2Ø48,NPT+4,2)
      CALL APWD
      CALL VMOV(2Ø48,1,Ø,NDPT,NPT)
      IF(NDPT.EQ.1)GOTO 25
      CALL VSUB(2Ø48,1,2Ø49,1,6144,1,NPT-1)
      CALL VSMUL(6144,1,2Ø48+NPT,6144,1,NPT-1)
      IBEG=Ø
      IFIN=1
      DO 2Ø J=2,NDPT
      CALL VADD(IBEG,NDPT,6144,1,IFIN,NDPT,NPT-1)
      IBEG=IBEG+1
      IFIN=IFIN+1
   2Ø CONTINUE
   25 CALL VSMUL(Ø,1,2Ø51+NPT,Ø,1,NFIN)
      CALL VTSADD(Ø,1,"4427,Ø,1,NFIN)
      CALL VCLIP(Ø,1,2Ø49+NPT,2Ø5Ø+NPT,Ø,1,NFIN)
      CALL VINT(Ø,1,Ø,1,NFIN)
      CALL VFIX(Ø,1,Ø,1,NFIN)
      CALL APWR
      CALL APGET(IDOT,Ø,NFIN,1)
      IBIT=16
      IWORD=128
      CALL APWD
      IDOT(1)=IDOT(1)+IOFF
      DO 3Ø IP=2,NFIN
      IBITL=IBIT
      IWORDL=IWORD
      IBIT=IBIT-1
      IF(IBIT.GT.Ø) GOTO 4Ø
      IWORD=IWORD-1
      IF(IWORD.EQ.Ø)GOTO 12Ø
      IBIT=16
C
C  SECTION WHERE POSITIVE (SHADED) LOBES ARE PLOTTED
C
   4Ø IF(IDOT(IP).LT.Ø)GOTO 5Ø
      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
   6Ø IPBUF(IWORD,JROW(IROW))=IPBUF(IWORD,JROW(IROW)).OR.MASK(IBIT)
      IROW=IROW+1
      IF(IROW.LT.IDT)GOTO 6Ø
      GOTO 3Ø
```

```
C
C SECTION WHERE NEGATIVE (UNSHADED) LOBES ARE PLOTTED
C
   50 IDOT(IP)=IDOT(IP)+IOFF
      IF(IDOT(IP).LT.1.OR.IDOT(IP-1).LT.1)GOTO 30
      IF(IDOT(IP).GT.IDOT(IP-1))GOTO 70
      IDTB=IDOT(IP)
      IDTE=IDOT(IP-1)
      MSKB=MASK(IBIT)
      MSKE=MASK(IBITL)
      IWORDB=IWORD
      IWORDE=IWORDL
      GOTO 80
   70 IDTB=IDOT(IP-1)
      IDTE=IDOT(IP)
      MSKB=MASK(IBITL)
      MSKE=MASK(IBIT)
      IWORDB=IWORDL
      IWORDE=IWORD
   80 IDTM=(IDTB+IDTE)/2
      IROW=IDTB
   90 IPBUF(IWORDB,JROW(IROW))=IPBUF(IWORDB,JROW(IROW)).OR.MSKB
      IROW=IROW+1
      IF(IROW.LE.IDTM)GOTO 90
      IROW=IDTM
  100 IPBUF(IWORDE,JROW(IROW))=IPBUF(IWORDE,JROW(IROW)).OR.MSKE
      IROW=IROW+1
      IF(IROW.LE.IDTE)GOTO 100
   30 CONTINUE
  120 JST=JROW(1)
C
C  WRITE OUT PART OF BUFFER AND INITIALISE
C  FOR A NEW TRACE
C
      IF(IWRITW(NWBUF,IPBUF(1,JST),IBLKW,ICHW).LT.0)STOP'WRITE ERR'
      IBLKW=IBLKW+NBLKBF
      CALL APWR
      CALL VCLR(0,1,NWBUF)
      CALL APGET(IPBUF(1,JST),0,NWBUF,1)
      CALL APWD
      IF(IWRITW(NWSPAC,IPBUF(1,JST),IBLKW,ICHW).LT.0)STOP'WRITE ERR'
      IBLKW=IBLKW+NBSPAC
   10 CONTINUE
      IROW=1
C
C  CHECK IF MORE STRIPS TO BE DONE
C
      IF(NSTRIP.LT.NTIM)GOTO 4
      CALL CLOSEC(ICHW)
      STOP 'NORMAL TERMINATION'
      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 containg the rasters to be plotted.

```
      DIMENSION IBUF(132,8),IBUFC(1056),INBUF(1024),FBUF(3)
      REAL*8 FSPECR
      EQUIVALENCE(IBUF(1),IBUFC(1))
      DATA DEV/3RRK /
      WRITE(7,1001)
 1001 FORMAT(' ENTER FILE NAME TO BE PROCESSED:',$)
      READ(5,1000)FBUF
 1000 FORMAT(3A4)
      CALL IRAD50(12,FBUF,FSPECR)
      IF(IFETCH(DEV).NE.0)STOP'FETCH ERR'
      IDCH=IGETC()
      IF(LOOKUP(IDCH,FSPECR).LT.0)STOP'LOOKUP ERR'
      CALL MTXSET
      CALL MTX(IBUF,0,1)
      IBLK=1
   10 IN=IREADW(1024,INBUF,IBLK,IDCH)
      IBLK=IBLK+4
      IF(IN.EQ.1024)GOTO 20
      IF(IN.LT.-1)STOP'READ ERR'
      IF(IN.LE.0)GOTO 40
   20 IPOS=1
      IROW=1
      NROW=IN/128
      IF((IN-(NROW*128)).NE.0)NROW=NROW+1
      NWORD=NROW*132
      CALL MWAIT
      DO 5 I=1,1056
    5 IBUFC(I)=0
      DO 30 I=1,IN
      IBUF(IPOS,IROW)=INBUF(I)
      IPOS=IPOS+1
      IF(IPOS.LE.128)GOTO 30
      IPOS=1
      IROW=IROW+1
   30 CONTINUE
      CALL MTX(IBUF,NWORD,2)
      IF(IN.EQ.1024)GOTO 10
      CALL MWAIT
   40 CALL MTX(IBUF,-1,1)
      CALL MWAIT
      STOP'NORMAL TERMINATE'
      END
```

General Purpose Raster Merge and Output :- MPMERP

Input File......DK1:MPMERD.DAT

Input Parameters

READ(1,10)NPLT,LPLT,NSTRIP

10 FORMAT(3I5)

NPLT.....Number of images to be merged

LPLT.....Length of output raster image

NSTRIP...Number of strips

For each of the NPLT images the following input is needed

READ(1,10)LST,LFIN

READ(1,10)ICOM,ILOG

READ(1,10)NPLIN

READ(1,30)FBUF

30 FORMAT(3A4)

LST......Output raster line number for first input line

LFIN.....Output raster line number for last input line

ICOM.....Data complement flag

0 - Merge in data as it is

1 - complement data before merging

ILOG.....Logical merge flag

0 - use an OR to merge the data

1 - use an AND to merge the data

NPLIN....Number of words per raster, 128 for seismic, 132 for others

FBUF.....Input raster file

```
C     M.J.POULTER SEPT 79
C       POST PROCESS AND MERGE PROG FOR
C       SEISMIC PLOT SYSTEM
        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)
        COMMON /LFIL/ INBUF,IOFF,LBUF,IBLK,NBUF,NPLIN,ISPOS,EOF,ICHAN,ICOM
        DATA IOFF/1,1Ø25,2Ø49,3Ø73,4Ø97,5721,6145,7169/
        DATA DEV/3RRK /
C
C   SET UP AP TO USE FOR ZEROING ARRAYS
C
        CALL APINIT
        CALL VCLR(Ø,1,1Ø56)
C
C   SET UP I/O DEFINITIONS
C
        IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERR'
        IF(ICDFN(3Ø).NE.Ø)STOP'CHAN DEF ERR'
        CALL ASSIGN(1,'DK1:MPMERD.DAT',14)
C
C GET INPUT PARAMETERS
C
        READ(1,1Ø) NPLT,LPLT,NSTRIP
   1Ø   FORMAT(3I5)
        IF(NPLT.GT.8)STOP'TOO MANY MASKS'
        DO 2Ø I=1,NPLT
        READ(1,1Ø)LST(I),LFIN(I)
        READ(1,1Ø)ICOM(I),ILOG(I)
        READ(1,1Ø)NPLIN(I)
        READ(1,3Ø)FBUF
   3Ø   FORMAT(3A4)
        WRITE(7,*) NPLT,LPLT,LST(I),LFIN(I),ICOM(I),ILOG(I)
        WRITE(7,*)NPLIN(I)
        WRITE(7,3Ø)FBUF
        CALL IRAD5Ø(12,FBUF,FSPEC(I))
        ICHAN(I)=2Ø+I
        IBLK(I)=1
        EOF(I)=Ø
        NBUF(I)=1Ø23+IOFF(I)
        ISPOS(I)=NBUF(I)
        IF(LOOKUP(ICHAN(I),FSPEC(I)).LT.Ø)STOP'LOOKUP ERR'
   2Ø   CONTINUE
C
C SET UP MATRIX FOR MAIN LOOP
C
        CALL MTXSET
        CALL MTX(IPBUF,Ø,1)
        LNUM=Ø
        NTIM=Ø
        IST=1Ø57
C
```

```
      C ZERO THE PLOT ARRAY
      C
       40 IST=1056-IST+2
          CALL APGET(IPBUF(IST),0,1056,1)
          IPSV=IST
          CALL APWD
      C
      C  START OF MAIN LOOP
      C
          DO 50 I =1,8
          LNUM=LNUM+1
          EOFFLG=1
      C
      C  THIS LOOP FILLS ONE LINE OF PLOT BUFFER
      C  WITH INPUT FROM EACH PLOT MASK
      C
          DO 60 J=1,NPLT
          IF(LNUM.GT.LFIN(J))GOTO 60
          IF(EOF(J).NE.0)GOTO 60
          EOFFLG=0
          IF(LNUM.LT.LST(J))GOTO 60
          IPOS=IPSV
          ID=J
          CALL LINFIL(ID)
          IF(ILOG(J).NE.0)GOTO 70
          DO 80 L=1,132
          IPBUF(IPOS)=IPBUF(IPOS).OR.LBUF(L)
       80 IPOS=IPOS+1
          GOTO 60
       70 DO 90 L=1,132
          IPBUF(IPOS)=IPBUF(IPOS).AND.LBUF(L)
       90 IPOS=IPOS+1
       60 CONTINUE
          IPSV=IPOS
       50 CONTINUE
      C
      C  WHEN A PLOT BUFFER IS FULL COME HERE TO EMPTY IT
      C
          CALL MWAIT
          CALL MTX(IPBUF(IST),1056,2)
          IF(EOFFLG.NE.0)GOTO 100
          IF(LNUM.LT.LPLT)GOTO 40
      100 CALL MWAIT
          CALL MTX(IPBUF,0,1)
          CALL MWAIT
          NTIM=NTIM+1
          LNUM=0
          IF(NTIM.LT.NSTRIP)GOTO 40
          STOP'NORMAL TERMINATION'
          END
```

```
      SUBROUTINE LINFIL(J)
      INTEGER*2 INBUF(8192),IOFF(8),LBUF(132),IBLK(8),NBUF(8),
     XNPLIN(8),ISPOS(8),EOF(8),ICHAN(8),ICOM(8)
      COMMON /LFIL/ INBUF,IOFF,LBUF,IBLK,NBUF,NPLIN,ISPOS,EOF,ICHAN,ICOM
C
C ZERO LINE ARRAY
C
      CALL APGET(LBUF,Ø,132,1)
      CALL APWD
C
C  FILL LINE BUFFER FROM INPUT BUFFER
C
      NPT=NPLIN(J)
      IPOS=ISPOS(J)
      NLIM=NBUF(J)
      IF(ICOM(J).NE.Ø)GOTO 2Ø
      DO 1Ø I=1,NPT
      IPOS=IPOS+1
      IF(IPOS.LE.NLIM)GOTO 3Ø
C
C  REFILL BUFFER FROM FILE WHEN EMPTY
C
      IN=IREADW(1Ø24,INBUF(IOFF(J)),IBLK(J),ICHAN(J))
      IBLK(J)=IBLK(J)+4
      IPOS=IOFF(J)
      IF(IN.EQ.1Ø24)GOTO 3Ø
      EOF(J)=1
      NLIM=IN+IPOS
      IF(IN.LT.-1)STOP'READ ERR'
      IF(IN.LE.Ø)RETURN
   3Ø LBUF(I)=INBUF(IPOS)
   1Ø CONTINUE
      GOTO 9Ø
   2Ø DO 4Ø I=1,NPT
      IPOS=IPOS+1
      IF(IPOS.LE.NLIM)GOTO 5Ø
C
C  REFILL BUFFER FROM FILE WHEN EMPTY
C
      IN=IREADW(1Ø24,INBUF(IOFF(J)),IBLK(J),ICHAN(J))
      IBLK(J)=IBLK(J)+4
      IPOS=IOFF(J)
      IF(IN.EQ.1Ø24)GOTO 5Ø
      EOF(J)=1
      NLIM=IN+IPOS
      IF(IN.LT.-1)STOP'READ ERR'
      IF(IN.LE.Ø)RETURN
   5Ø LBUF(I)=.NOT.INBUF(IPOS)
   4Ø CONTINUE
   9Ø NBUF(J)=NLIM
      ISPOS(J)=IPOS
      RETURN
      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 Descriptions

TREAD

```
CALL TREAD(BUFFER,NBUF,ISTAT,IDRV)
```

Purpose:- To read from tape into a specified memory buffer

Arguments:

BUFFER...Integer...Buffer to read into

NBUF.....Integer...Number of bytes to read

ISTAT....Byte......Returned tape status

IDRV.....Byte......Tape drive to read from

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 foward

5 - Space reverse

6 - Write

7 - Return to OS/8

COM(2) File length, or number of records when spacing

COM(3) Tape drive number

COM(4) SDS10 command

<0 Read from tape into file on unit 20

=0 Tape wind operation

>0 write to tape from file on unit 21

ISTAT....Byte.......Returned tape status

ITLEN....Byte.......Returned file length in seconds for
field file

ILEN.....Byte.......Returned file length in blocks on a read

TAPSUB

CALL TAPSUB(ICOM,IDRV,ISTAT,ILEN,IFNUM,BUF,NBYT,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 prupose 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 foward 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$CSR=164ØØØ
DRB=164ØØ2
DO$CSR=164Ø1Ø
DOB=164Ø12
;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),RØ          ;GET NO OF BYTES REQUIRED
        MOV      RDLST,ARGLST       ;SET UP SDS1Ø COMMAND
        MOVB     @8.(R5),ARGLT2
        JSR      PC,MSG             ;SEND COMMAND
;MAIN TRANSFER LOOP
1$:     TSTB     @#DR$CSR           ;WAIT TILL 8 READY
        BPL      1$
        MOVB     @#DRB,(R1)+        ;MOVE DATA FROM BUFFER TO TARGET
        DEC      RØ
        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
2$:     MOV      RØ,@4(R5)          ;RETURN ZERO COUNTER
        TST      @#DR$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
4$:     MOVB     (R1),R2            ;SET UP STATUS SEARCH
7$:     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
5$:     MOVB     @#DRB,R2           ;TAKE BYTE FROM 8
        BIS      #1,@#DR$CSR
        BR       7$                 ;GOBACK AND TAKE MORE DATA
;EXIT FOR EOF SIGNALLED
3$:     MOV      RØ,@4(R5)          ;RETURN NO OF BYTES LEFT
        DEC      R1
        MOVB     (R1),@6(R5)        ;RETURN STATUS
;COMMON EXIT
6$:     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),RØ          ;GET NUM OF BYTES
        MOVB     WRTLST,ARGLST      ;SET UP COMMAND
        MOVB     @1Ø.(R5),ARGLT1
        MOVB     @12.(R5),ARGLT2
        JSR      PC,MSG
;SET UP CODE
1Ø$:    TSTB     @#DO$CSR
        BPL      1Ø$                ;WAIT FOR SYNCHRONISATION
```

```
        BIC     #400,@#DOSCSR      ;CLEAR EOF BIT
;MAIN TRANSFER LOOP
11$:    BIS     #1,@#DOSCSR        ;SET ENABLE BIT
        MOVB    (R1)+,@#DOB        ;TRANSFER BYTE
12$:    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$:    TSTB    @#DRSCSR           ;SEE IF REPLY READY
        BPL     16$
        MOVB    @#DRB,@6(R5)       ;GET STATUS
        MOV     #2,R0
17$:    TSTB    @#DRSCSR
        BPL     17$
        MOVB    @#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 ARGLIST ADDRESS
        MOV     #3,R4
MLOOP:  BIS     #1,@#DOSCSR        ;SET ENABLE BIT
        CLR     R2
        MOVB    (R3)+,R2           ;GET COMMAND INTO R2
        MOV     R2,@#DOB           ;MOV TO BUFFER
9$:     TSTB    @#DOSCSR
        BPL     9$                 ;WAIT TILL ACCEPTED
        DEC     R4
        BNE     MLOOP
        BIS     #400,@#DOSCSR
        MOV     #0,@#DOB           ;SET EOF BIT
        RTS     PC                 ;RETURN
;STORAGE AREA
ARGLST: .BYTE   0
ARGLT1: .BYTE   0
ARGLT2: .BYTE   0
WRTLST: .BYTE   6
        .EVEN
RDLST:  .WORD   403
        .EVEN
        .END
```

```
        .TITLE  SDS1Ø
        .GLOBL  SDS1Ø
        .MCALL  .READW,.WRITW,.EXIT,.PRINT
DR$CSR==164ØØØ
DO$CSR==164Ø1Ø
DRB==164ØØ2
DOB==164Ø12
ERRBYT=52

SDS1Ø:                          ;COMMON ENTRY POINT
        MOV     2(R5),R1        ;MOV ADDRESS OF COMMAND->R1
        MOV     #ARGLIST,R2     ;PUT ARGLIST IN R2
        MOV     #4,RØ           ;PUT COUNTER IN RØ
1$:     MOVB    (R1)+,(R2)+     ;MOV COMMANDS TO ARGLIST
        DEC     RØ
        BNE     1$
        TSTB    FLAG            ;TEST TYPE OF COMMAND
        BMI     READ            ;GOTO APPROPRIATE SECTION
        BEQ     WIND
        JMP     WRITE
ARGLIST:.BYTE   Ø
        .BYTE   Ø
        .BYTE   Ø
FLAG:   .BYTE   Ø

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

READ:   CLR     BLKN            ;CLEAR BLOCK COUNTER
        .WRITW  #AREA,#2Ø.,#BUFF,#256.,BLKN
        JSR     PC,MSG          ;SET UP DISC POINTER
                                ;AND SEND MESSAGE
RESTRT: MOV     #BUFF,R4        ;PUT ADDR BUFF IN R4
        MOV     #2Ø48.,R3       ;PUT COUNTER IN R3
DRLOOP: TSTB    @#DR$CSR        ;TEST IF BYTE READY
        BPL     DRLOOP          ;IF NOT GO BACK AND TRY AGAIN
        MOVB    @#DRB,(R4)+     ;PUT BYTE FROM INTERFACE TO BUFFER
        DEC     R3              ;DEC THE COUNTER
        BEQ     DRDONE          ;IF Ø BUFFER FULL
        TST     @#DR$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,#2Ø.,#BUFF,#1Ø24.,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:                          ;COME HERE ON EOF
        TSTB    @#DR$CSR        ;TEST IF BYTE READY
        BPL     DREOF
        MOVB    @#DRB,R1        ;MOVE OVER LAST TWO BYTES
        DEC     R4
        MOVB    -(R4),@4(R5)    ;MOVE STATUS
        MOVB    -(R4),@6(R5)    ;AND TIME LENGTH TO RETURN ARGLIST
        ADD     #2.,R4          ;RESET R4 POINTER
        MOVB    #Ø,(R4)         ;AND ZERO THE LAST BYTE IN BUFFER
        SUB     #BUFF,R4        ;GET NO OF BYTES
        ASR     R4              ;GET NO OF WORDS
        BCC     2$              ;IF C CLEAR NO EVEN WRITE OUT
        INC     R4              ;ODD ADD ONE ON
2$:     .WRITW  #AREA,#2Ø.,#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 8/E WHAT IS REQUIRED
REPLY:  TSTB    @#DR$CSR            ;AND GO THROUGH PROC
        BPL     REPLY               ;FOR RECEIVING A REPLY
        MOVB    @#DRB,@4(R5)        ;WHICH IS RETURNED TO CALLING PROG
        MOV     #2,RØ               ;TAKE LAST TWO SYNCHRO BYTES
5$:     TSTB    @#DR$CSR
        BPL     5$
        MOVB    @#DRB,R1
        DEC     RØ
        BNE     5$
        RTS     PC

;THIS SECTION IS RESPONSIBLE FOR THE FAST TAPE WRITE
;IN CONJUNCTON WITH THE 8/E

WRITE:  CLR     BLKN                ;CLEAR BLOCK COUNTER
        CLR     EOFW                ;CLEAR EOF FLAG
        JSR     PC,MSG              ;SEND MESSAGE
9$:     TSTB    @#DO$CSR            ;TEST MESSAGE RECIEVED
        BPL     9$
        BIC     #4ØØ,@#DO$CSR       ;CLEAR EOF
        BR      DODONE              ;START TRANSFER

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

DODONE: TST     EOFW                ;TEST INTERNAL EOF FLAG
        BMI     DOEFLP              ;IF SET GOTO EOF AREA
        .READW  #AREA,#21.,#BUFF,#1Ø24.,BLKN
        BCS     RERR                ;READ IN A FULL BUFFER AND CHECK FOR ERRORS
        ADD     #4,BLKN             ;BUMP UP BLOCK COUNT
        CMP     #1Ø24.,RØ           ;SEE IF GOT A FULL BUFFER
        BEQ     WRSTRT              ;YES THEN TRANSFER
        BIS     #1ØØØØØ,EOFW        ;NO THEN SET EOF FLAG
        MOV     #1Ø24.,R1           ;SET UP COMPLETION COUNTER
        SUB     RØ,R1
        ASL     R1                  ;CONVERT WORD TO BYTE COUNT
        BR      WRSTRT              ;AND WRITE OUT AMOUNT OF BUFFER REQUIRED

DOEFLP: BIS     #1,@#DO$CSR
        MOV     #Ø,@#DOB            ;MOVE ZEROS TO OUTPUT
8$:     TSTB    @#DO$CSR            ;TEST IF OK FOR MORE
        BPL     8$                  ;IF NOT WAIT
        DEC     R1                  ;DEC THE COUNTER
        BEQ     DOEOF               ;IF Ø GOTO END
        BR      DOEFLP
```

```
DOEOF:  BIS     #4ØØ,@#DO$CSR    ;SET INTERFACE EOF FLAG
        MOV     #Ø,@#DOB         ;CLEAR INTERFACE BUFFER
        BR      REPLY            ;GET REPLY
;THIS SECTION IS THE MESSAGE SENDING
;SUBROUTINE MSG AND THE BUFFER AND ERROR MESSAGE BLOCKS

MSG:    BIC     #4ØØ,@#DO$CSR    ;CLEAR OUTPUT EOF FLAG
        MOV     #ARGLIST,R1      ;MOV ARGLIST ADDR TO R1
        MOV     #3,RØ            ;MOV COUNTER ->RØ
MLOOP:  BIS     #1,@#DO$CSR      ;SET ENABLE BIT
        CLR     R2               ;CLEAR INTER BUFFER
        MOVB    (R1)+,R2         ;MOV FROM ARGLIST TO R2
        MOV     R2,@#DOB         ;MOV R2->INTERFACE
7$:     TSTB    @#DO$CSR         ;SEE IF OK FOR NEXT BYTE
        BPL     7$
        DEC     RØ               ;DEC THE COUNTER
        BNE     MLOOP            ;AND GO BACK FOR MORE IF NONØ
        BIS     #4ØØ,@#DO$CSR    ;OR IF Ø SET EOF FLAG AND
        MOV     #Ø,@#DOB         ;FLUSH THE BUFFER
        RTS     PC               ;RETURN

RERR:   TSTB    @#ERRBYT         ;TEST FOR TYPE OF ERROR
        BEQ     8$
        .PRINT  #RMSG
        .EXIT
8$:     BIS     #4ØØ,@#DO$CSR    ;SET UP ERROR FINISH
        MOV     #Ø,@#DOB
        JMP     REPLY
WMSG:   .ASCIZ  / WRITE ERROR/
RMSG:   .ASCIZ  / READ ERROR /

;STORAGE AREA

AREA:   .BLKW   1Ø
BLKN:   .WORD   Ø
BUFF:   .BLKW   1Ø24.
EOFW:   .WORD   Ø
        .END
```

```
      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,Ø 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
      INTEGER*2 MASK(8),ESTATI
      LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
     %IFLEN,ESTAT,ERRS(8)
      DATA MASK/"1,"2,"4,"1Ø,"2Ø,"4Ø,"1ØØ,"2ØØ/
      DATA SDSCOM/"Ø,"1,"2,"3,"4,"5,"6,"7/
      DATA ERRS/"377,"377,"377,"377,"377,"377,"377,"377/
      ITRY=Ø
      IF(ICOM) 1Ø,3Ø,2Ø
C
C SECTION CONTROLLING A READ
C
C
C  CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
C
   1Ø ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
      COM(1)=SDSCOM(4)
      COM(2)=1
      COM(3)=IDRV
      COM(4)=-1
      CALL SDS1Ø(COM,ISTAT,ITLEN,ILEN)
      IF(ISTAT.EQ.Ø)RETURN
C
C ERROR DETECTED ON READ
C
      ISTATI=ISTAT
      GOTO 4Ø
C
C IF SHORT RECORD FOUND REREAD TAPE
C
   5Ø ITMP=ISTATI.AND.MASK(6)
      IF(ITMP.NE.Ø)GOTO 1Ø
      ITMP=ISTATI.AND.MASK(2)
      IF(ITMP.EQ.Ø)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
      WRITE(2,2Ø1Ø)IFNUM
 2Ø1Ø FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
      IF(ITRY.GE.2)GOTO 13Ø
      ECOM(1)=SDSCOM(6)
      ECOM(2)=1
```

```
      ECOM(3)=IDRV
      ECOM(4)=0
      CALL SDS10(ECOM,ESTAT, , )
      GOTO 10
C
C WRITE SECTION
C
   20 ITRY=ITRY+1
      IF(ITRY.GT.3)GOTO 130
      COM(1)=SDSCOM(7)
      IFLEN=(ILEN+3)/4
      COM(2)=IFLEN
      COM(3)=IDRV
      COM(4)=1
      CALL SDS10(COM,ISTAT, , )
      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=32760
      IFLENE=16
      CALL TWRIT(ERRS,NBUF,ESTAT,IPAD,IFLENE,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
```

```
      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,Ø 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,"1Ø,"2Ø,"4Ø,"1ØØ,"2ØØ/
      DATA SDSCOM/"Ø,"1,"2,"3,"4,"5,"6,"7/
      DATA ERRS/"377,"377,"377,"377,"377,"377,"377,"377/
      ITRY=Ø
      IF(ICOM) 1Ø,3Ø,2Ø
C
C SECTION CONTROLLING A READ
C
C
C  CHECK THAT ONLY A FEW RETRIES ARE ATTEMPTED
C
   1Ø ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
      NBUF=NBYT
      CALL TREAD(BUF,NBUF,ISTAT,IDRV)
      IF(ISTAT.EQ.Ø)RETURN
C
C ERROR DETECTED ON READ
C
      ISTATI=ISTAT
      GOTO 4Ø
C
C IF SHORT RECORD FOUND REREAD TAPE
C
   5Ø ITMP=ISTATI.AND.MASK(6)
      IF(ITMP.NE.Ø)GOTO 1Ø
      ITMP=ISTATI.AND.MASK(2)
      IF(ITMP.EQ.Ø)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
      WRITE(2,2Ø1Ø)IFNUM
 2Ø1Ø FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
      IF(ITRY.GE.2)GOTO 13Ø
      ECOM(1)=SDSCOM(6)
      ECOM(2)=1
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(ECOM,ESTAT, , )
      GOTO 1Ø
C
C WRITE SECTION
C
   2Ø ITRY=ITRY+1
      IF(ITRY.GT.3)GOTO 13Ø
      NBUF=NBYT
      IFLEN=(ILEN+3)/4
      IF(IFLEN.LT.2)IFLEN=2
      IPAD=(IFLEN*2Ø48)-NBUF
```

```
      CALL TWRIT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
      IF(ISTAT.EQ.Ø)RETURN
C
C WRITE ERROR DETECTED
C
      ISTATI=ISTAT
      GOTO 4Ø
   7Ø ITMP=ISTATI.AND.MASK(6)
      ITMPI=ISTATI.AND.MASK(2)
      IF(ITMP.EQ.Ø.AND.ITMPI.EQ.Ø)RETURN
C
C REPORT AND RETRY
C
      WRITE(2,2Ø2Ø)IFNUM
 2Ø2Ø FORMAT(' FILE NO ',I4,' WRITE CRC ERR RETRY PROPOSED')
      ECOM(1)=SDSCOM(6)
      ECOM(2)=2
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(ECOM,ESTAT, , )
      NBUF=8
      IPAD=(IFLEN*2Ø48)-NBUF
      CALL TWRIT(ERRS,NBUF,ESTAT,IPAD,IFLEN,IDRV)
      GOTO 2Ø
C
C WIND FOWARD ONE FILE
C
   3Ø COM(1)=SDSCOM(5)
      COM(2)=1
      COM(3)=IDRV
      COM(4)=Ø
      CALL SDS1Ø(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.Ø)GOTO 4Ø
C
C IF ISTAT=Ø 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)=Ø
      CALL SDS1Ø(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
   4Ø WRITE(2,1Ø1Ø)ISTATI,IFNUM
 1Ø1Ø FORMAT(' STATUS=',I3,' FILE NO=',I4)
      ISTATI=ISTATI.AND..NOT.MASK(4)
      ITMP=ISTATI.AND.MASK(5)
      IF(ITMP.EQ.Ø)GOTO 8Ø
   9Ø ECOM(1)=SDSCOM(1)
      ECOM(2)=Ø
```

```
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(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.Ø)GOTO 9Ø
      IF(ICOM) 1Ø,3Ø,2Ø
C
C TAPE OFFLINE
C
   8Ø ITMP=ISTATI.AND.MASK(1)
      IF(ITMP.EQ.Ø)GOTO 1ØØ
      TYPE 1ØØ1,IDRV
 1ØØ1 FORMAT(' TAPE DRIVE ',I1,' OFFLINE')
C
C HAVING ANNOUNCED ERROR SKIP UNTIL CORRECTED
C
  11Ø ECOM(1)=SDSCOM(1)
      ECOM(2)=Ø
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(ECOM,ESTAT, , )
      ESTATI=ESTAT
      ITMP=ESTATI.AND.MASK(1)
      IF(ITMP.NE.Ø)GOTO 11Ø
      IF(ICOM) 1Ø,3Ø,2Ø
C
C EOT
C
  1ØØ ITMP=ISTATI.AND.MASK(3)
      IF(ITMP.EQ.Ø)GOTO 12Ø
      TYPE 1ØØ2,IDRV
 1ØØ2 FORMAT(' EOT ON DRIVE ',I1)
      IEOT=-1
      RETURN
  12Ø IF(ICOM) 5Ø,35,7Ø
C
C ERROR EXIT RETURN
C
  13Ø ISTATI=-1
      RETURN
      END
```

Floating Point Transfers

GETNO

CALL GETNO(RNUM)

Purpose:- to find the number of floating point numbers to be transfered 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$CSR=164ØØØ
DRB=164ØØ2
DO$CSR=164Ø1Ø
DOB=164Ø12

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

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

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

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

Extended Memory Input/Output

AP to Memory

FAD = APGAD(VM(I))

Purpose:- To get a full 18 bit address for a virtual memory element

FAD......Floating point....Returned 18 bit address
VM(I)....Any...............Virtual memory element

CALL APPUTX(APAD,WCNT,FORMAT)

CALL APGETX(APAD,WCNT,FORMAT)

Purpose:- To transfer data to(PUT) and from(GET) the AP using the 18 bit address stored internally by an immediately preceeding call to APGAD.

CALL APPUTA(APAD,WCNT,FORMAT,FAD)

CALL APGETA(APAD,WCNT,FORMAT,FAD)

Purpose:- To exchange data with the AP as above except the data is provided by the value FAD which has been stored previously.

APAD......Integer.....AP memory address
WCNT......Integer.....Number of elements to transfer
FORMAT....Integer.....AP data transfer format

Disc to Memory

FAD=ADGET(VM(I))

Purpose:- To get a full 18 bit address for the virtual memory element in the format for a disc transfer.

IWRITX(CHAN,WCNT,BLK)

IREADX(CHAN,WCNT,BLK)

Purpose :- To transfer between disc and virtual memory using the 18 bit address calculated in an immediately preceeding call to ADGET.

IWRITA(CHAN,WCNT,BLK,FAD)

IREADA(CHAN,WCNT,BLK,FAD)

Purpose:- to transfer data between disc and virtual memory as above but with the 18 bit address being provided by FAD.

FAD......Floating point.....18 bit address of virtual memory element

VM(I)....Any................Virtual memory element

CHAN.....Integer............I/O channel to be used in transfer

WCNT.....Integer............Number of words to transfer

BLK......Integer............Starting block in file for transfer

```
;/***** DAPEX = HOST DEPENDENT APEX FOR PDP-11 RT-11 = REL 2.Ø  . NOV 77 ******
;C
;         FOR PDP-11 RT-11 OR DOS
;
;-----------------------------------------------------------------------------
;
;        C O N F I G U R A T I O N    P A R A M E T E R S
;
;        PDP-11 DEPENDENT
;        ----------------
;
        DOS = Ø          ;SET TO 1 FOR DOS, OR Ø FOR RT-11
;
;        AP-12ØB DEPENDENT
;        -----------------
;
        FPS = 176ØØØ     ;AP BASE DEVICE ADDRESS
;
;
;
        .TITLE DAPEX
        .GLOBL SPLDGO,ABORT,RUNDMA,RUNAP,TSTDMA,TSTRUN,WTDMA,WTRUN,APIN
        .GLOBL APIENA,APIDIS,TSTINT,APWI
        .GLOBL APOUT,APWD,APWR,APRSET,APASGN,APRLSE
;
;
;        PDP-11 DEFINITIONS
;
        RØ = %Ø
        R1 = %1
        R2 = %2
        R3 = %3
        R4 = %4
        R5 = %5
        R6 = %6
        SP = %6
        R7 = %7
        PC = %7
;
;        AP-DEVICE ADDRESSES
;
        FMTH = FPS
        FMTL = FPS + 2
        WC = FPS + 1ØØ
        HMA = FPS + 1Ø2
        CTRL = FPS + 1Ø4
        APMA = FPS + 1Ø6
        SWR = FPS + 11Ø
        FN = FPS + 112
        LITES = FPS + 114
        ABRT = FPS + 116
;
;
        .MACRO CALL X
        MOV R5,-(SP)     ;SAVE R5
        .IF EQ,<DOS-1>
        .IFT
        JSR R5,X
        BR .+2
        .IFF
        MOV #ZERO,R5
        JSR PC,X
        .IFTF
```

```
        MOV (SP)+,R5     ;RETRIEVE R5
        .ENDC
        .ENDM
;
;
        .MACRO RETURN
        .IF EQ,<DOS-1>
        .IFT
        RTS R5
        .IFF
        RTS PC
        .ENDC
        .ENDM
;
;
;C
;/***** SPLDGO = S-PAD LOAD AND GO = REL 2.Ø  , NOV 77 *************************
;C
;         SUBROUTINE SPLDGO(SLIST,NSPADS,STRT,BRKLOC)
;         INTEGER SLIST(16),NSPADS,STRT,BRKLOC
;C
;C        FIRST WAIT FOR THE LAST PROGRAM STARTED BY 'SPLDGO' OR
;C        'RUNAP' TO BE COMPLETED, THEN:
;C        LOAD 'N' VALUES INTO S-PAD FROM 'SLIST'
;C        AND START THE AP AT LOCATION 'STRT' WITH A BREAKPOINT
;C        SET AT 'BRKLOC'
;C
;C        ROUTINES USED: APWR, APOUT
;C
;C-------LOCAL STORAGE
;         INTEGER I
;C
;C        1. WAIT FOR RUNNING DONE (APWR)
;C        2. FOR EACH S-PAD PARAMETER:
;C           A. PUT S-PAD PARAMETER ADDRESS INTO SPD (CALL WREG(I-1,513))
;C           B. PUT PARAMETER VALUE INTO S-PAD (CALL WREG(SLIST(I),517))
;C        3. PUT PROGRAM STARTING LOCATION INTO TMA (CALL WREG(STRT,515)
;C        4. CALL RUNAP TO START AT LOCATION 8 OF THE BOOTSTRAP, WITH
;C           THE SPECIFIED P.S. BREAKPOINT SET
;C
;         CALL APWR
SPLDGO: CALL APWR
;C
;C        LOAD PARAMETERS INTO S-PAD (IF ANY)
;         IF (NSPADS.EQ.Ø) GOTO 2Ø
;         DO 1Ø I = 1, NSPADS
;         CALL APOUT(I-1,1)
;         CALL APOUT(513,2)
;         CALL APOUT(SLIST(I),1)
;1Ø       CALL APOUT(517,2)
;
;
;
         TST (R5)+
         MOV (R5)+,R2      ;GET S-PAD VALUE POINTER
         MOV @(R5)+,RØ     ;GET NSPADS
         BEQ SBRGO
         CLR R1           ;INITIALIZE S-PAD ADDRESS
LDSP:    MOV R1,@#SWR     ;SET S-PAD ADDRESS
         MOV #513.,@#FN   ;INTO SPD
         MOV @R2,@#SWR    ;SET PARAMETER VALUE
         MOV #517.,@#FN   ;INTO S-PAD
         TST (R2)+        ;BUMP PARAMETER VALUE POINTER
         INC R1           ; AND S-PAD ADDRESS
```

```
        DEC RØ          ;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
;2Ø     CALL APOUT(STRT,1)
;       CALL APOUT(515,2)
;'      CALL RUNAP(4,Ø,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:   Ø
;
;       END
;C
;/***** ABORT = ABORT AP-EXECUTION = REL 2.Ø  , NOV 77 ***********************
;/***** APRSET = RESET THE AP = REL 2.Ø  , NOV 77 *****************************
;C
;       SUBROUTINE ABORT
;C
;C      STOPS ANY TRANSFER, AND/OR RUN IN PROGRESS, RESETS INTERFACE AND
;C      CLEANS UP ANY SOFTWARE STATE INDICATORS
;C
;C      ROUTINES USED: APOUT
;C
;C      1. DO AN INTERFACE RESET (ORESET)
;C      2. CLEAR THE CONTROL REGISTER (OCTRL(Ø))
;C      3. DO AN INTERNAL RESET (OFN(2Ø48))
;C
;       CALL APOUT(Ø,1Ø)
;       CALL APOUT(Ø,7)
;       CALL APOUT(2Ø48,2)
;       RETURN
;
;
;
APRSET:
ABORT:  CLR @#ABRT
        CLR @#CTRL
        MOV #4ØØØ.,@#FN
        CLR @#LITES
        RETURN
;
;
;
;       END
;C
;/***** RUNDMA = START A DMA TRANSFER = REL 2.Ø  , NOV 77 ********************
```

```
;C
;         SUBROUTINE RUNDMA(HOST,APMA,N,CTRL)
;         INTEGER HOST,APMA,N,CTRL
;C
;C        WAIT FOR ANY PREVIOUS DMA TRANSFER STARTED BY 'APPUT', 'APGET', OR
;C        'RUNDMA' TO COMPLETE, THEN:
;C        START A DMA TRANSFER WITH THE ADDRESS OF 'HOST' AS THE INITIAL
;C        HOST MEMORY ADDRESS, 'APMA' AS THE INITIAL AP-12ØB MAIN DATA MEMORY
;C        ADDRESS, 'N' AS THE NUMBER OF DATA ITEMS TO BE TRANSFERED,
;C        AND 'CTRL' AS THE CONTROL REGISTER SETTING (WITH INTERRUPT CONTROL
;C        BITS MASKED OUT) TO USE.
;C
;C        ROUTINES USED: APWD, APOUT, ILOC, IAND16, IRSH16
;C
;C--------NOTE: THE DETERMINATION OF 'WC' FROM 'N' BELOW DEPENDS ON THE
;C        HOST WORD LENGTH AS IF AFFECTS THE NUMBER OF HOST WORDS PER
;C        AP-12ØB MEMORY WORD.  THIS CODE IS APPROPRIATE FOR A 16-BIT COMPUTER.
;C
;C--------LOCAL STORAGE
;         INTEGER WC
;C
;C        1. WAIT FOR DMA DONE
;C        2. SET HOST ADDRESS (OHMA)
;C        3. SET AP ADDRESS (OAPMA)
;C        4. SET WORD COUNT (OWC)
;C        5. SET CONTROL REGISTER (OCTRL)
;C
;         CALL APWD
;
;
;
RUNDMA:   CALL APWD
;
;
;
;         CALL APOUT(APMA,4)
;         CALL APOUT(ILOC(HOST),5)
;         WC = N
;
;
;
          TST (R5)+
          MOV (R5)+,@#HMA          ;SET PDP-11 ADDRESS
          MOV #Ø,@#LITES           ;CLEAR EXTENDED PDP11 ADDRESS
          MOV @(R5)+,@#APMA        ;SET AP MEMORY ADDRESS
          MOV @(R5)+,R1            ;GET DATA COUNT
;
;
;
;C        ISOLATE FMT FIELD AND ADJUST WC ACCORDINGLY
;         IF(IAND16(IRSH16(CTRL,1),3).NE.1) WC=2*N
;         CALL APOUT(WC,6)
;
;
;
          MOV @(R5)+,RØ            ;GET CONTROL WORD
          BIT RØ,#4                ;TEST 'FMT' FIELD FOR A 2
          BNE 1$
          BIT RØ,#2
          BNE 2$
1$:       ASL R1                   ;DOUBLE COUNT UNLESS FORMAT #1
2$:       MOV R1,@#WC              ;SET WORD COUNT
;
;
```

```
;
;C        CLEAR OFF HOST INTERRUPT ENABLE BITS
;         CALL APOUT(IAND16(CTRL,1Ø23), 7)
;         RETURN
;
;
;
          BIC #174ØØØ,RØ          ;CLEAR INTERUPT ENABLES
          MOV RØ,@#CTRL
          RETURN
;
;
;
;         END
;C
;/***** RUNAP = START AN AP-PROGRAM = REL 2.Ø  , NOV 77 ***********************
;C
;         SUBROUTINE RUNAP(PSA,NOLOAD,SWR,FN)
;         INTEGER PSA,NOLOAD,SWR,FN
;C
;C        WAIT FOR ANY PREVIOUS PROGRAM STARTED BY 'SPLDGO' OR 'RUNAP'
;C        TO COMPLETE, THEN:
;C        1. IF 'NOLOAD' IS ZERO, PUT 'PSA' INTO PSA
;C        2. PUT 'SWR' INTO THE SWITCH REGISTER
;C        3. PUT 'FN' (WITH 'START BIT' CLEARED AND 'CONTINUE BIT' SET)
;C           INTO THE FUNCTION REGISTER
;C
;C        ROUTINES USED: APWR, APOUT, IOR16, NAND16
;C
;C        1. WAIT FOR RUNNING DONE
;C        2. IF NO-LOAD NOT SPECIFIED, PUT 'PSA' INTO PSA
;C           (CALL WREG(PSA,512))
;C        3. PUT 'SWR' INTO SWR (OSWR)
;C        3. CLEAR POSSIBLE START BIT, OR IN CONTINUE BIT,
;C           AND PUT INTO FUNCTION (OFN)
;C
;         CALL APWR
;
;
RUNAP:    CALL APWR
;
;         CLEAR PARITY ENABLE IN STATUS REGISTER
;
         CLR @#SWR
         MOV #518.,@#FN
;
;
;         IF (NOLOAD.NE.Ø) GO TO 1ØØ
;         CALL APOUT(PSA,1)
;         CALL APOUT(512,2)
;
;
;
          TST (R5)+
          MOV @(R5)+,@#SWR        ;PUT 'PSA' INTO THE SWITCHES
          TST @(R5)+              ;SEE IF LOAD PSA??
          BNE NOLOAD
          MOV #512.,@#FN  ;PUT 'PSA' INTO PSA IF 'NOLOAD' IS ZERO
;
;
;
;1ØØ      CALL APOUT(SWR,1)
;C        CLEAR POSSIBLE SET START BIT & OR IN CONTINUE BIT TO FN REG
;         CALL APOUT(IOR16(AND16(FN,271),8192), 2)
```

```
;         RETURN
;
;
;
NOLOAD:   CLR @#SWR                 ;CLEAR SWR
          MOV #1030.,@#FN           ;CLEAR PARITY ERROR ENABLE
          MOV @(R5)+,@#SWR          ;PUT 'SWR' INTO THE SWITCHES
          MOV @(R5)+,R0             ;CLEAR ALL BUT POSSIBLE BREAKPOINT
          BIC #177360,R0
          BIS #8192.,R0             ;SET CONTINUE BIT
          MOV R0,@#FN               ;AND GO
          RETURN
;
;
;         END
;C
;/***** TSTDMA = TEST DMA TRANSFER COMPLETE = REL 2.0  , NOV 77 ***************
;C
;         SUBROUTINE TSTDMA(I)
;         INTEGER I
;C
;C        SET 'I' TO ONE IF THE LAST DMA TRANSFER STARTED BY 'APPUT'
;C        'APGET' OR 'RUNDMA' IS DONE; SET 'I' TO ZERO IF THE TRANSFER IS
;C        STILL IN PROGRESS
;C
;C        ROUTINES USED: APIN, NAND16
;C
;C        READ CTRL AND MASK TO .NOT. LOW BIT
;         CALL APIN(I,7)
;         I=NAND16(1,I)
;         RETURN
TSTDMA:   MOV #1,R1
          BIC @#CTRL,R1   ;DO NOT.CTRL.AND.1
          MOV R1,@2(R5)
          RETURN
;
;
;
;         END
;C
;/***** TSTRUN = TEST RUN COMPLETE = REL 2.0  , NOV 77 ************************
;C
;         SUBROUTINE TSTRUN(I)
;         INTEGER I
;C
;C        SET 'I' TO ONE IF THE AP-120B IS STOPPED AFTER THE LAST RUN
;C        STARTED BY 'SPLDGO' OR 'RUNAP'; ELSE SET 'I' TO ZERO IF THE AP-120B
;C        IS STILL RUNNING.
;C
;C        ROUTINES USED: APIN, NEGCHK
;C
;C        READ FN AND SHIFT DOWN TO LOW BIT
;         CALL APIN(I,2)
;         I=NEGCHK(I)
;         RETURN
;
;
;
TSTRUN:      CLR R1
             TST @#FN
             BGE 1$
             INC R1          ;RETURN A 1 IF THE HIGH BIT OF 'FN' WAS ON
1$:          MOV R1,@2(R5)
             RETURN
```

```
;
;
;
;         END
;C
;/***** WTDMA = WAIT FOR DMA TRANSFER COMPLETE = REL 2.Ø  , NOV 77 ***********
;C
;         SUBROUTINE WTDMA(IERR)
;         INTEGER IERR
;C
;C        WAITS FOR DATA TRANSFER COMPLETE, 'IERR' SET TO ONE IF A DATA LATE
;C        ERROR WAS DETECTED BY THE HARDWARE, ELSE SET TO ZERO.
;C
;C        ROUTINES USED: APIN, IAND16, IRSH16
;C
;C        SPIN WHILE THE LOW BIT OF 'CTRL' IS ON
;1ØØ      CALL APIN(IERR,7)
;         IF (IAND16(IERR,1).EQ.1) GO TO 1ØØ
;C
;C        SHIFT THE 'DATA LATE' ERROR BIT DOWN TO THE LOW END
;         IERR=IRSH16(IAND16(IERR,256),8)
;         RETURN
;
;
;
WTDMA:    BIT @#CTRL,#1
          BNE WTDMA               ;WAIT FOR THE LSB TO GO OFF
          MOV @#CTRL,RØ           ;GET CONTROL
          BIC #177377,RØ
          SWAB RØ
          MOV RØ,@2(R5)           ;RETURN THE DATA LATE BIT
          RETURN
;
;
;
;         END
;C
;/***** WTRUN = WAIT FOR RUN COMPLETE = REL 2.Ø  , NOV 77 ********************
;C
;         SUBROUTINE WTRUN(IERR)
;         INTEGER IERR
;C
;C        WAIT FOR AP RUN TO FINISH (HALT) , SET IERR TO ONE IF AN SRAO ERROR,
;C        TWO IF PARITY, ELSE ZERO.
;
WTRUN:    TST @#FN
          BGE WTRUN               ;WAIT FOR THE SIGN BIT TO GO ON
          CLR R1
          MOV #1Ø3Ø.,@#FN         ;GET AP-STATUS
          BIT @#LITES,#24Ø        ;CHECK FOR PARITY OR SRAO ERROR
          BEQ 1$                  ;IF NO ERRORS
          MOV #2,R1               ;ASSUME PARITY ERROR
          BIT @#LITES,#2ØØ        ;CHECK FOR PARITY ERROR
          BNE 1$                  ;IF PARITY ERROR
          DEC R1                  ;ELSE SRAO ERROR
1$:       MOV R1,@2(R5)
          RETURN
;
;
;
;         END
;C
;
;/****** APASGN = ASSIGN THE AP = REL 2.Ø  , 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 PAPARAMETER TO 1
        RETURN
;
;/****** APRLSE = RELEASE THE AP = REL 2.Ø  , NOV 77 **************************
;
;        APRLSE IS A 'NOP' UNDER RT-11
;
APRLSE: RETURN
;
;
;/****** APIENA = INABLE INERRUPT = REL 2.Ø  , NOV 77 *********************
;
;        NO OP UNDER RT11
APIENA: RETURN
;
;
;/****** APIDIS = DISABLE INTERRUPT = REL 2.Ø  , NOV 77 ****************
;
;        NO OP UNDER RT11
APIDIS: RETURN
;
;
;/****** TSTINT = TEST FOR INTERRUPT = REL 2.Ø  , NOV 77 ****************
;
;        NO OP UNDER RT11
TSTINT: RETURN
;
;
;/****** APWI = WAIT FOR INTERRUPT = REL 2.Ø  , NOV 77 ***************
;
;        NO OP UNDER RT11
APWI:   RETURN
;
;
;/***** APIN = INPUT AN AP-12ØB INTERFACE REGISTER = REL 2.Ø  , NOV 77 ********
;C
;         SUBROUTINE APIN(DATA,NUM)
;         INTEGER DATA,NUM
;C
;C        READ THE CONTENTS OF INTERFACE REGISTER NUMBER 'NUM' INTO 'DATA
;C
;C        PARAMETERS:
;C        DATA   - RECEIVES THE CURRENT CONTENTS OF REGISTER 'NUM'
;C        NUM    - SPECIFIES WHICH AP-12ØB 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                  1Ø. 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 LANGAUGE
;C      SINCE FORTRAN CANNOT OUTPUT DIRECTLY TO AN I/O DEVICE
;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
        BLT IMASK               ;IF EXTENDED MEMORY REGISTER READ
;
APIN1:  ASL R1                  ;CONVERT TO BYTES
        MOV @TABLE(R1),@2(R5)   ;GET FROM APPROPRIATE DEVICE ADDRESS
        RETURN
;
;       READ MASK OR APMAE OR MAE
;
;       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 Ø-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 #1777ØØ,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 #17776Ø,R2          ;CLEAR ALL BUT MAE OR APMAE
        MOV R2,@2(R5)           ;RETURN REGISTER VALUE
        RETURN
;
;
;
;       END
;C
;
;
;
TABLE:  Ø
        FPS+11Ø         ;SWR
        FPS+112         ;FN
        FPS+114         ;LITES
        FPS+1Ø6         ;APMA
```

```
        FPS+1Ø2         ;HMA
        FPS+1ØØ         ;WC
        FPS+1Ø4         ;CTRL
        FPS+Ø           ;FMTH
        FPS+2           ;FMTL
        FPS+116         ;RESET
;
;
;
;/***** APOUT = WRITE TO AN AP-12ØB INTERFACE REGISTER = REL 2.Ø  , NOV 77 ****
;C
;       SUBROUTINE APOUT(DATA,NUM)
;       INTEGER DATA,NUM
;C
;C      PUT THE CONTENTS OF 'DATA' INTO INTERFACE REGISTER NUMBER 'NUM'.
;C
;C      PARAMETERS:
;C      DATA    - DATA TO BE PUT INTO AN INTERFACE REGISTER
;C      NUM     - NUMBER OF THE DESTINATION INTERFACE REGISTER, SEE 'APIN'
;C                FOR THE NUMBERING
;C
;C      ROUTINES USED: NONE
;C
;C-------NOTE: THIS ROUTINE WILL TYPICALLY BE IN ASSEMBLY CODE.
;C
;       RETURN
;
;
;
APOUT:  MOV @4(R5),R1
        MOV R1,R2               ;CHECK FOR EXTENDED MEMORY REGISTER WRITE
        SUB #11.,R2              ;LOOK FOR 12,13 OR 14
        BLE APOUT1              ;IF LESS THAN 12
        SUB #4,R2
        BLT OMASK               ;IF EXTENDED MEMORY REGISTER WRITE
;
APOUT1: ASL R1                  ;CONVERT TO BYTES
        MOV @2(R5),@TABLE(R1)   ;STORE INTO APPROPRIATE REGISTER
        RETURN
;
;       WRITE MASK OR APMAE OR MAE
;
;       SEE COMMENTS IN APIN FOR EXTENDED MEMORY REGISTERS(MASK,APMAE,MAE)
;
OMASK:  MOV @#ABRT,R3           ;READ MASK,APMAE,MAE
        MOV @2(R5),R2           ;FETCH REGISTER VALUE
        CMP #12.,R1              ;CHECK FOR MASK
        BNE OAPMAE              ;IF NOT MASK
        BIC #374ØØ,R3           ;CLEAR MASK, KEEP APMAE AND MAE
        SWAB R2                 ;POSITION MASK TO BITS 8-13
        BIS R2,R3               ;ADD NEW MASK TO OLD APMAE AND OLD MAE
        MOV R3,@#LITES          ;WRITE TO AP
        RETURN
;
OAPMAE: CMP #13.,R1              ;CHECK FOR APMAE
        BNE OMAE                ;IF MAE
        BIC #36Ø,R3             ;CLEAR APMAE, KEEP MASK AND MAE
        ASL R2                  ;POSITION APMAE TO BITS 4-7
        ASL R2
        ASL R2
        ASL R2
        BIS R2,R3               ;ADD NEW APMAE TO OLD MASK AND OLD MAE
        MOV R3,@#LITES          ;WRITE TO AP
        RETURN
```

```
;
OMAE:   BIC #17,R3              ;CLEAR MAE KEEP APMAE AND MASK
        BIS R2,R3               ;ADD NEW MAE TO OLD APMAE AND OLD MASK
        MOV R3,@#LITES          ;WRITE TO AP
        RETURN
;
        .END
```

```
.TITLE  RKØ5  VØ3-Ø1

.IDENT  /VØ3.Ø1/

; RT-11 DISK (RK11) HANDLER
;
; DEC-11-ORTSB-A
;
; EF/ABC/RGB/DV/JD
;
; COPYRIGHT (C) 1977
;
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS Ø1754
;
; 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 TIM$IT,TIM$IT=Ø
.IIF NDF MMG$T, MMG$T=Ø
.IIF NDF ERL$G, ERL$G=Ø
.QELDF
.GLOBL  DPSYS,  DSSYS,  DXSYS, DMSYS
.GLOBL  RKSYS,  RFSYS,  DTSYS
DTSYS   = Ø
DSSYS   = Ø
DXSYS   = Ø
DPSYS   = Ø
RFSYS   = Ø
DMSYS   = Ø
RKDS    = 1774ØØ
RKER    = 1774Ø2
RKCS    = 1774Ø4
RKWC    = 1774Ø6
RKBA    = 17741Ø
RKDA    = 177412
RKCNT   = 1Ø
RKSTS   = 1ØØØØØ
RKDSIZ  = 113ØØ
RKIDEN  = Ø
RKIDS   = 377
RKRCNT  = 4ØØØ
RKNREG  = 7
RKREGA  = 1774ØØ
        .DRBEG  RK,22Ø,RKDSIZ,RKSTS
.IF EQ MMG$T
.IFTF
        MOV     #RKCNT,(PC)+
RETRY:  Ø
        MOV     RKCQE,R5
        MOV     @R5,R2
        MOV     2(R5),R4
        ASR     R4
        ASR     R4
        ASR     R4
        SWAB    R4
        BIC     #^C<16ØØØØ>,R4
        BR      2$
1$:     ADD     R2,R4
        ASR     R2
        ASR     R2
        ADD     R3,R2
2$:     MOV     R2,R3
        BIC     #17776Ø,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     #1Ø3,R3
        MOV     #RKDA,R4
        MOV     (PC)+,@R4
DISKAD: Ø
        CMP     (R5)+,(R5)+
```

```
.IFT
        MOV     (R5)+,-(R4)
.IFF
        JSR     PC,@$MPPTR
        MOV     (SP)+,-(R4)
.IFTF
        MOV     (R5)+,-(R4)
        BEQ     7$
        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 ERL$G
        BIT     #62340,R4
        BNE     RKERR
        MOV     PC,R5
        ADD     #RKRBUF-.,R5
        MOV     R5,R2
        MOV     #RKREGA,R3
        MOV     #RKNREG,R4
RKRREG: MOV     (R3)+,(R5)+
        DEC     R4
        BNE     RKRREG
        MOV     #RKNREG,R3
        ADD     #RKRCNT,R3
        MOV     RKCQE,R5
        MOVB    RETRY,R4
        DEC     R4
        JSR     PC,@$ELPTR
        MOV     #RKER,R5
        MOV     (R5)+,R4
.ENDC
RKERR:  MOV     #1,@R5
3$:     TSTB    @R5
        BPL     3$
        DECB    RETRY
        BEQ     HERROR
```

```
        BIT     #11ØØØØ,R4
        BEQ     RKRETR
        MOV     DISKAD,@#RKDA
        MOV     #115,@R5
        BIS     #1ØØØØØ,RETRY
RTSPC:  RTS     PC
HERROR: MOV     RKCQE,R5
        BIS     #1,@-(R5)
.IF NE  ERL$G
        BR      RKEXIT
DONE:   .FORK   RKFBLK
        MOV     #RKIDS,R4
        MOV     RKCQE,R5
        JSR     PC,@$ELPTR
.IFF
DONE:
.ENDC
RKEXIT: CLR     RETRY
        .DRFIN  RK
.ENDC
RKFBLK: .WORD   Ø,Ø,Ø,Ø
.IF NE ERL$G
RKRBUF: .BLKW   RKNREG
.ENDC
        .DREND  RK
.END
```

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

USPØ=17764Ø
AP=176ØØØ
WC=AP+1ØØ
HMA=AP+1Ø2
CTRL=AP+1Ø4
APMA=AP+1Ø6
LITES=AP+114

APGAD:  MOV     @#USPØ,RØ               ;GET PAR Ø BLOCK OFFSET
        ASL     RØ
        ASL     RØ                      ;GET HIGHBITS INTO PLACE
        ASL     RØ
        ASL     RØ
        MOV     RØ,HIGHBT               ;SAVE INTO HIGHBITS
        BIC     #37777,HIGHBT           ;CLEAR OTHER BITS
        ASL     RØ                      ;SET UP 16 BIT PART
        ASL     RØ
        BIC     #1777ØØ,R1              ;CLEAR UNWANTED BITS
        BIC     #77,RØ
        BIS     R1,RØ                   ;FORM 16 BIT PART
        MOV     RØ,LOWBIT               ;AND SAVE IT
        MOV     HIGHBT,RØ               ;RETURN FUNCTION VALUES
        MOV     LOWBIT,R1               ;IN RØ + R1
        RTS     PC                      ;RETURN

APPUTA: MOV     8.(R5),RØ               ;GET ADDRESS OF STORE
        MOV     (RØ)+,HIGHBT            ;GET HIGHBITS FROM STORE
        MOV     (RØ),LOWBIT             ;GET LOWBITS
APPUTX: MOV     R5,-(SP)                ;SAVE R5
        JSR     PC,APWD                 ;CALL APWD
        MOV     (SP)+,R5                ;RESTORE R5
        MOV     #193.,RØ                ;SET UP CONTROL WORD
        BR      COMMON                  ;FOR A PUT AND GOTO COMMON

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

APGETX: MOV     R5,-(SP)                ;SAVE R5
        JSR     PC,APWD                 ;CALL APWD
        MOV     (SP)+,R5                ;RESTORE R5
        MOV     #225.,RØ                ;SET UP CONTROL WORD
                                        ;FOR A GET
COMMON: MOV     @2(R5),@#APMA           ;SET AP ADDRESS
        MOV     @4(R5),R1               ;GET WORD COUNT
        ADD     @6(R5),RØ               ;SET UP THE FORMAT
        ADD     @6(R5),RØ               ;FOR THE TRANSFER
        BIT     RØ,#4                   ;AND SEE
        BNE     1$                      ;IF THE WORD COUNT
        BIT     RØ,#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
                                        ;OF PDP11 ADDRESS
        BIC     #174ØØØ,RØ              ;DISENABLE INTERRUPTS
        MOV     RØ,@#CTRL               ;SETUP CTRL TO START PROCESS
        RTS     PC


LOWBIT: .WORD   Ø
HIGHBT: .WORD   Ø
        .END
```

```
        .MCALL  .READC,.WRITC,.EXIT,.PRINT
        .GLOBL  ADGET,IREADX,IWRITX,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
        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 Routines

Demultiplex Microcode

CALL VBINSC(A1,I1,A2,I2,G1,IG1,N)

Purpose:- To binary scale an input vector using a vector of binary gain values

A1.......AP address of input vector

I1.......AP address increment for each element

A2.......AP address of output vector

A2.......AP address increment for each output element

G1.......AP address of gain vector

IG1......AP address increment for gain values

N........Number of vector elements to apply the gains to

CALL DMXA(A1,A2,I2,A3,A4,N)

Purpose:- To demultiplex and reformat a frame of SEG-A field data

A1.......AP address of running gains vector

A2.......AP address of Field data

I2.......AP address increment per data element

A3.......AP address of submultiplexed gain check

A4.......AP address of Gain switch direction to use next

N........Number of points to demultiplex form the frame

Tri-Diagonal Matrix equation solver

CALL FACTOR(A1,I1,A2,I2,A3,I3,A4,I4,A5,I5,N)

Purpose:- To factorise a tri-diagonal matrix

A1.......AP address of the 1st diagonal, vector A

I1.......AP increment of the above

A2.......AP address of the major diagonal, vector B

I2.......AP increment of the above

A3.......AP address of the 3rd diagonal, vector C

I3.......AP increment of the above

A4.......AP address of the output L vector

I4.......AP increment of the above

A5.......AP address of the output U vector

I5.......AP increment of the above

N........Number of elements in the major diagonal

CALL SOLVE(A1,I1,A2,I2,A3,I3,A4,I4,A5,I5,A6,I6,N)

Purpose:- to solve the Tri-diagonal matrix equation given the factorised input

A1.......AP address of the Factorised L vector

I1.......AP increment of the above

A2.......AP address of the Factorised U vector

I2.......AP increment of the above

A3.......AP address of the RHS vector

I3.......AP increment of the above

A4.......AP address of the C vector

I4.......AP increment of the above

A5.......AP address of workspace vector

I5.......AP increment of the above

A6.......AP address of X result vector

I6.......AP increment of the above

N........Number of elements in the major diagonal

```
        $TITLE 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    $EQU Ø
        DBASE   $EQU 1
        DINC    $EQU 2
        GCNT    $EQU 3
        DIR     $EQU 4
        N1      $EQU 5
        DADR    $EQU 6
        DIRCK   $EQU 7
        DATA    $EQU 7
        IADR    $EQU 1Ø
        TEMP    $EQU 11
        N       $EQU 12
        N27     $EQU 13
        SYNC    $EQU 14
        BIAS    $EQU 15
        TWO     $EQU 16
        MASK    $EQU 17
        ERR     $EQU 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=2ØØ
        MOV GAIN,GAIN;SETMA                              "SET UP SAVE ON DPY
        SUB DINC,DBASE;LDDPA;DB=1.                       "SET UP BASE ADDRESSES
        LDSPI N;DB=3Ø.
        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.                              "HERE EVERY 3Ø TIMES
SKIP:   INC IADR;SETMA
        LDSPI N;DB=3Ø.
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                                           "TURN -1 TO +1 AND VV
        INC DIR
        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
        $END

        $TITLE VBINSC
        $ENTRY VBINSC,7
" THIS IS A PROGRAM WHICH
" REDUCES THE EXPONENT OF A
" FLOATING POINT NUMBER BY A
" SPECIFIED AMOUNT
"
"     S-PAD DEFINITIONS
        A $EQU Ø            "VECTOR BASE ADDRESS
        I $EQU 1            "INC OF VECTOR A
        C $EQU 2            "BASE ADDRESS OF RESULT
        K $EQU 3            "INC OF VECTOR C
        G $EQU 4            "GAIN ADDRESS
        J $EQU 5            "GAIN INCREMENT
        N $EQU 6            "NO OF VECTOR ELEMENTS
        FACT $EQU 7         "GAIN VALUE
        RES $EQU 8          "NEW EXPONENT
VBINSC: MOV G,G;SETMA;FADD ZERO,ZERO       "GET GAIN AND INIT FADDER
        MOV A,A;SETMA;FADD                 "GET A(Ø)
        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 ACCESSTO 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
        $END
```

```
        $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       $EQU    Ø
        AINC    $EQU    1
        B       $EQU    2
        BINC    $EQU    3
        C       $EQU    4
        CINC    $EQU    5
        L       $EQU    6
        LINC    $EQU    7
        U       $EQU    1Ø
        UINC    $EQU    11
        N       $EQU    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
$END
```

```
        $TITLE SOLVE
        $ENTRY SOLVE,15
        $EXT DIV
        $EXT SPUFLT
" MPSOLV.APS
" THIS IS A ROUTINE TO SOLVE A TRIDIAGONAL
" MARIX 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    Ø
        LINC    $EQU    1
        U       $EQU    2
        UINC    $EQU    3
        RHS     $EQU    4
        RHSINC  $EQU    5
        C       $EQU    6
        CINC    $EQU    7
        Y       $EQU    1Ø
        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
```

Plotting subroutines

CONSYS

Consys was implemented from the version resident on MTS at Newcastle, and full documentation is available from this source. Basically it is a general contouring subroutine.

```
CALL CONTUR(X,IX,Y,IY,Z,IDX,CZ,NC,PTR,SWCHES,MINDIS,
            NXL,XLOC,NYL,YLOC)
```

Arguments:-

```
X.....Floating point...Grid positions in X direction
IX....Integer..........Number of X grid points
Y.....Floating point...Grid positions in Y direction
IY....Integer..........Number of Y grid points
Z.....Floating Point...Virtual array containing values to be
                        contoured
                       Z(I,J)= Function of (X(I),Y(J))
IDX...Integer..........Declaration of column size for Z array
                        Z(IDX,IDY)
CZ....Floating point...Values at which to have contour lines
NC....Integer..........Number of contour values
PTR...Floating point...Work array of size NC
SWCHES.Logical.........5 element array of logical switches
                       1 - draw XB,YB XT,YB border
                       2 - draw XB,YB XB,YT border
                       3 - draw XB,YT XT,YT border
```

4 - draw XT,YT XT,YB border

5 - label contours

MINDIS.Floating point..Minimum distance between contour labels

NXL...Integer..........Number of constant X points for labels

XLOC..Floating point...Constant X positions for labels

NYL...Integer..........Number of constant Y values for labels

YLOC..Floating point...Constant Y positions for labels

Rasterising Interception

The rasterising interception program MPRASM picks up the active vector plot file from the system disc and then is fully interactive for the remaining options. The user is asked if output is to disk or tape. If it is to disc he is then asked for an output file name, or if it is to tape the drive number. The plot is then rasterised and saved to the chosen medium. At the end of the program the total number of raster lines generated is written out for later use with the merge programs.

```
      SUBROUTINE 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
      VIRTUAL Z(IDX,IY)
      REAL Z, X(IX), Y(IY), CZ(NC)
      REAL MINDIS, XLOC(NXL), YLOC(NYL)
      LOGICAL*1 SORTED, SWCHES(5)
      LOGICAL*1 LBLLOC, DISTOK
      INTEGER*2 PARERR, CONCNT, CCP1, NTEMP
      INTEGER*2 PL, PH
      INTEGER*2 PTR(NC)
C
      COMMON /CONCOM/ LOWER, UPPER, XL, XR, XC, YL, YU, YC,
     +                ZLL, ZUL, ZLR, ZUR
      INTEGER*2 LOWER, UPPER,ERRUNT
      DATA IOMAX/16/,ERRUNT/7/
C
C
C
C
C  CHECK FOR OBVIOUS ERRORS IN THE PARAMETERS.
C
C
      PARERR = 0
      IF( IX .GT. IDX )  PARERR = PARERR + 1
      IF( IX .LT. 2 )    PARERR = PARERR + 1
      IF( IY .LT. 2 )    PARERR = PARERR + 1
      IF( IDX .LT. 2 )   PARERR = PARERR + 1
      IF( NC .LT. 1 )    PARERR = PARERR + 1
      IF( PARERR .NE. 0 )  GO TO 986
      IF(.NOT. SWCHES(5))GOTO 1994
      IF(MINDIS.LT.0) PARERR = PARERR+1
      IF(NXL .LT. 0)PARERR=PARERR+1
      IF(NYL .LT. 0)PARERR=PARERR+1
      IF(NXL+NYL .LT .1)PARERR=PARERR+1
      IF(PARERR.NE.0)GOTO 1993
```

```
C
C  CHECK X AND Y TO ENSURE THEY ARE IN STRICTLY ASCENDING
C  ORDER.  NOTE THAT THE ERROR MESSAGE WHICH IS WRITTEN IF THEY
C  ARE NOT SCARES THE USER INTO CHECKING BOTH ARRAYS, EVEN
C  THOUGH THE CODE DOESN'T CHECK Y IF X IS BAD.
C
1994  DO 88Ø I = 2, IX
          IF( X(I) .LE. X(I-1) )  GO TO 988
88Ø       CONTINUE
      DO 881 I = 2, IY
          IF( Y(I) .LE. Y(I-1) )  GO TO 988
881       CONTINUE
C
C  SORT THE ARRAY OF CONTOUR VALUES.
C
      IF( NC .EQ. 1 )  GO TO 5
          DO 1 M = 1, NC
              PTR(M) = M
1             CONTINUE
          M = NC-1
2             CONTINUE
              SORTED = .TRUE.
              DO 4 K = 1, M
                  PL = PTR(K)
                  PH = PTR(1+K)
                  IF( CZ(PH) .GT. CZ(PL) )  GO TO 3
                      PTR(K) = PH
                      PTR(1+K) = PL
                      SORTED = .FALSE.
3                 CONTINUE
4                 CONTINUE
              IF( SORTED )  GO TO 6
              M = M - 1
              IF( M .GE. 1 )  GO TO 2
          GO TO 6
5     CONTINUE
          OFS = Ø
          PTR(1) = 1
          GO TO 6
6     CONTINUE
      CZMAX = CZ(PTR(NC))
      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
      IYM1 = IY - 1
```

```
ØØ78           YU = Y(1)
ØØ79           DO 38 J = 1, IYM1
ØØ8Ø              YL = YU
ØØ81              YU = Y(J+1)
ØØ82              YC = Ø.5*(YL + YU)
ØØ83              XR = X(1)
ØØ84              ZLR = Z(1, J)
ØØ85              ZUR = Z(1, J+1)
ØØ86              DO 37 I = 2, IX
ØØ87                 XL = XR
ØØ88                 XR = X(I)
ØØ89                 ZLL = ZLR
ØØ9Ø                 ZUL = ZUR
ØØ91                 ZLR = Z(I, J)
ØØ92                 ZUR = Z(I, J+1)
ØØ93                 ZMIN = AMIN1(ZLL, ZUL, ZLR, ZUR)
ØØ94                 IF( ZMIN .GT. CZMAX )  GO TO 37
ØØ96                 ZMAX = AMAX1(ZLL, ZUL, ZLR, ZUR)
ØØ97                 IF( ZMAX .LT. CZMIN )  GO TO 37
ØØ99                 IF( ZMAX .EQ. ZMIN )  GO TO 37
Ø1Ø1                    DO 12 CONCNT = 1, NC
Ø1Ø2                       ZØ = CZ(PTR(CONCNT))
Ø1Ø3                       IF( ZØ .LT. ZMIN )  GO TO 12
Ø1Ø5                          IF( ZØ .GT. ZMAX )  GO TO 37
Ø1Ø7                             LOWER = CONCNT
Ø1Ø8                             UPPER = LOWER
Ø1Ø9                             IF( UPPER .EQ. NC )  GO TO 14
Ø111                                CCP1 = CONCNT + 1
Ø112                                DO 11 II = CCP1, NC
Ø113                                   IF( ZMAX .LT. CZ(PTR(II)) )
        +                                                          GO TO 14
Ø115                                   UPPER = II
Ø116      11                           CONTINUE
Ø117                                GO TO 14
Ø118      12                       CONTINUE
Ø119      14                    CALL CTQQ(CZ,PTR,NC)
      C
      C  IF THE USER SPECIFIED VIA SWCHES(5) THAT LABELS ARE TO BE
      C  DRAWN, FIND OUT HERE IF THIS CELL IS A CANDIDATE FOR
      C  LABELING, AND IF IT IS, CALL LABELR TO DRAW THE LABEL.
      C
Ø12Ø                    IF( .NOT. SWCHES(5) )  GO TO 29
Ø122                       LBLLOC = .FALSE.
Ø123                       IF( NXL .EQ. Ø )  GO TO 23
Ø125                          DO 22 M = 1, NXL
Ø126                             IF( .NOT.(XL .LE. XLOC(M) .AND.
        +                                   XLOC(M) .LT. XR) )  GO TO 22
Ø128                                LBLLOC = .TRUE.
Ø129                                GO TO 27
Ø13Ø      22                     CONTINUE
Ø131      23               CONTINUE
Ø132                       IF( NYL .EQ. Ø )  GO TO 27
Ø134                          DO 26 M = 1, NYL
Ø135                             IF( .NOT.(YL .LE. YLOC(M) .AND.
```

```
     +                                   YLOC(M) .LT. YU) )  GO TO 26
                                  LBLLOC = .TRUE.
                                  GO TO 27
      26                       CONTINUE
      27                 CONTINUE
                         IF( .NOT. LBLLOC )  GO TO 28
                            IF( DISTOK(XC, YC,MINDIS) )
     +                                CALL NUMBER(XC, YC,Ø.1, ZØ,Ø.Ø,4)
      28                 CONTINUE
      29              CONTINUE
      37           CONTINUE
      38        CONTINUE
C  DRAW BORDER LINES, IF THE USER INDICATED VIA THE SWCHES
C  VECTOR THAT THEY ARE WANTED.
C
             XL = X(1)
             XR = X(IX)
             YL = Y(1)
             YU = Y(IY)
             IF( .NOT. SWCHES(1) )  GO TO 42
                CALL PLOT(XR, YL,+3)
                CALL PLOT(XL, YL,+2)
      42     CONTINUE
             IF( .NOT. SWCHES(2) )  GO TO 44
                CALL PLOT(XL, YL,+3)
                CALL PLOT(XL, YU,+2)
      44     CONTINUE
             IF( .NOT. SWCHES(3) )  GO TO 46
                CALL PLOT(XL, YU,+3)
                CALL PLOT(XR, YU,+2)
      46     CONTINUE
             IF( .NOT. SWCHES(4) )  GO TO 48
                CALL PLOT(XR, YU,+3)
                CALL PLOT(XR, YL,+2)
      48     CONTINUE
             RETURN
C
C
C  HANDLE BAD PARAMETERS IN THE CONTUR CALL HERE.
C
C
      995    CONTINUE
                WRITE(ERRUNT, 997)  PARERR, IX, IY, IDX, NC
                WRITE(ERRUNT, 999)
             CONTINUE
             STOP
C
      998    CONTINUE
                WRITE(ERRUNT, 994)
                WRITE(ERRUNT, 999)
             CONTINUE
             STOP
C
C  FORMAT STATEMENTS FOR CONTUR ERROR COMMENTS.
```

```
C
 994    FORMAT(' **** ERROR: SUBROUTINE "CONTUR" HAS BEEN CALLED',
     +          'WITH EITHER THE X OR',/,6X,'THE Y VECTOR (OR ',
     +          'BOTH) NOT IN STRICTLY ASCENDING ORDER.')
 997    FORMAT(' **** THERE ARE ',I1,' ERROR(S) IN THE PARA',
     +          'METERS IN A CALL',/,6X,'TO SUBROUTINE "CONTUR".',
     +          '  DIAGNOSTIC INFORMATION FOLLOWS:',/,6X,'IX = ',
     +          I15,'   IY = ',I15,/,6X,'IDX = ',I15,'   NC = ',
     +          I15)
 999    FORMAT(' **** DUE TO THE ABOVE ERROR, CONTUR WILL NOT ',
     +          'DRAW A CONTOUR MAP,',/,6X,'BUT WILL INSTEAD ',
     +          'IGNORE THE CALL.')
C
C
C
C
C
C
C   HANDLE ERRORS IN PARAMETERS TO CONLBL HERE.
C
C
 1993   CONTINUE
           WRITE(ERRUNT, 1998)  PARERR, MINDIS, NXL, NYL
           WRITE(ERRUNT, 1999)
        CONTINUE
        STOP
C
C   FORMAT STATEMENTS FOR CONLBL ERROR COMMENTS.
C
 1997   FORMAT(' **** ERROR: SUBROUTINE "CONLBL" HAS BEEN CALLED',
     +          ' WITH NO',/,6X,'PRECEDING INITIALIZATION CALL ',
     +          'TO SUBROUTINE "CONSET".')
 1998   FORMAT(' **** THERE ARE ',I1,' ERROR(S) IN THE ',
     +          'PARAMETERS IN A CALL TO',/,6X,'SUBROUTINE ',
     +          '"CONLBL".  DIAGNOSTIC INFORMATION FOLLOWS:',/,6X,
     +          'MINDIS = ',G13.6,/,6X,'NXL = ',I15,'  NYL = ',I15)
 1999   FORMAT(' **** DUE TO THE ABOVE ERROR, CONLBL WILL NOT ',
     +          'INITIALIZE THE',/,6X,'CONTOUR LABELING ROUTINES,',
     +          ' BUT WILL INSTEAD IGNORE THE CALL.')
        END
```

```
ØØØ1          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
ØØØ2          REAL X, Y, MINDIS
ØØØ3          REAL COORD(1ØØ)
ØØØ4          INTEGER*2 CURLEN
ØØØ5          DATA CURLEN/Ø/,NCOORD/1ØØ/
        C
ØØØ6          DISTOK = .FALSE.
ØØØ7          IF( CURLEN .LT. 1 )  GO TO 9
ØØØ9          IF(CURLEN.GE.NCOORD)RETURN
ØØ11             DO 8 I = 1, CURLEN, 2
ØØ12                XI = COORD(I)
ØØ13                IF( ABS(X-XI) .GT. MINDIS )  GO TO 5
ØØ15                   YI = COORD(1+I)
ØØ16                   IF( SQRT((X-XI)**2+(Y-YI)**2) .LE. MINDIS )
                                                                   RETURN
ØØ18    5           CONTINUE
ØØ19    8           CONTINUE
ØØ2Ø    9     CONTINUE
        C
ØØ21          DISTOK = .TRUE.
ØØ22          COORD(1+CURLEN) = X
ØØ23          COORD(2+CURLEN) = Y
ØØ24          CURLEN = CURLEN + 2
ØØ25          RETURN
ØØ26          END
```

```
ØØØ1          SUBROUTINE CTQQ(CZ,PTR,NC)
        C  CTQQ IS CALLED FROM CONTUR TO PRODUCE CONTOUR COORDINATE
        C  VALUES FOR THE GRID CELL BOUNDED BY XL AND XR, YL AND YU, AND
        C  ZLL, ZUL, ZUR, AND ZLR.  COORDINATE PAIRS THUS PRODUCED ARE
        C  DISPOSED OF VIA CALLS TO THE USER-SPECIFIED ROUTINES PLOT
        C  AND CKVA.  THE Z VALUES TO BE CONTOURED ARE STORED IN
        C  CZ(PTR(LOWER)),...,CZ(PTR(UPPER)).
        C
        C  THE ALGORITHM FOR THIS ROUTINE WAS TAKEN FROM:
        C    G. W. HARTWIG, "CONTUR - A FORTRAN IV SUBROUTINE FOR
        C    PLOTTING CONTOUR LINES," BALLISTIC RESEARCH LABORATORIES
        C    MEMORANDUM REPORT # 2282, ABERDEEN PROVING GROUND,
        C    MARYLAND, MARCH, 1973. (NTIS ACCESSION NUMBER AD-76Ø 437).
        C
        C  THE FOLLOWING CODE IS FOR VERSION 1.2 OF CONSYS, PRODUCED
        C  15 MAY, 1976.  GNC
        C
ØØØ2          REAL CZ(NC)
ØØØ3          LOGICAL KCHK(8), CENTER
ØØØ4          REAL PX(8), PY(8), PTEMP
ØØØ5          EQUIVALENCE (PX(1), PX1), (PX(2), PX2), (PX(3), PX3),
             +            (PX(4), PX4), (PX(5), PX5), (PX(6), PX6),
             +            (PX(7), PX7), (PX(8), PX8)
ØØØ6          EQUIVALENCE (PY(1), PY1), (PY(2), PY2), (PY(3), PY3),
             +            (PY(4), PY4), (PY(5), PY5), (PY(6), PY6),
             +            (PY(7), PY7), (PY(8), PY8)
ØØØ7          INTEGER*2 PTR(NC)
        C
ØØØ8          COMMON /CONCOM/ LOWER, UPPER, XL, XR, XC, YL, YU, YC,
             +                ZLL, ZUL, ZLR, ZUR
ØØØ9          INTEGER*2 LOWER, UPPER
        C
        C
        C
        C
ØØ1Ø          XC = Ø.5*(XL + XR)
ØØ11          XLMXR = XL - XR
ØØ12          XLMXC = XL - XC
ØØ13          XRMXC = XR - XC
ØØ14          YLMYC = YL - YC
ØØ15          YLMYU = YL - YU
ØØ16          YUMYC = YU - YC
ØØ17          ZC = Ø.25*(ZLL + ZUL + ZLR + ZUR)
ØØ18          DO 12Ø LEVEL = LOWER, UPPER
ØØ19             ZØ = CZ(PTR(LEVEL))
ØØ2Ø             TLL = ZLL - ZØ
ØØ21             TUL = ZUL - ZØ
ØØ22             TLR = ZLR - ZØ
ØØ23             TUR = ZUR - ZØ
ØØ24             TC  = ZC  - ZØ
        C
ØØ25             IC = Ø
ØØ26             CENTER = .FALSE.
ØØ27             DO 11 M = 1, 8
```

```
              KCHK(M) = .FALSE.
  11           CONTINUE
C
C   SEGMENT 1:
C
           IF( TLL*TLR .GT. Ø. )  GO TO 19
              KCHK(1) = .TRUE.
              IF( TLL*TLR .EQ. Ø. )  GO TO 12
                 IC = IC + 1
                 PX(IC) = TLL * XLMXR/(ZLR-ZLL) + XL
                 PY(IC) = YL
                 GO TO 18
  12          CONTINUE
                 IF( TLL .EQ. Ø. )  GO TO 15
                    IC = IC + 1
                    PX(IC) = XR
                    PY(IC) = YL
                    GO TO 17
  15             CONTINUE
                    IC = IC + 1
                    PX(IC) = XL
                    PY(IC) = YL
                    IF( TLR .NE. Ø. )  GO TO 16
                       IC = IC + 1
                       PX(IC) = XR
                       PY(IC) = YL
  16                CONTINUE
                    GO TO 17
  17             CONTINUE
                 GO TO 18
  18          CONTINUE
  19       CONTINUE
C
C   SEGMENT 2:
C
           IF( TLL*TC .GT. Ø. )  GO TO 29
              KCHK(2) = .TRUE.
              IF( TLL*TC .EQ. Ø. )  GO TO 22
                 IC = IC + 1
                 FAC = TLL/(ZC - ZLL)
                 PX(IC) = XLMXC*FAC + XL
                 PY(IC) = YLMYC*FAC + YL
                 GO TO 28
  22          CONTINUE
                 IF( TC .NE. Ø. )  GO TO 25
                    CENTER = .TRUE.
                    IC = IC + 1
                    PX(IC) = XC
                    PY(IC) = YC
  25             CONTINUE
                 GO TO 28
  28          CONTINUE
  29       CONTINUE
C
```

```
      C   SEGMENT 3:
      C
ØØ82            IF( TUL*TLL .GT. Ø. )  GO TO 39
ØØ84               KCHK(3) = .TRUE.
ØØ85               IF( TUL*TLL .EQ. Ø. )  GO TO 32
ØØ87                  IC = IC + 1
ØØ88                  PX(IC) = XL
ØØ89                  PY(IC) = TLL * YLMYU/(ZUL-ZLL) + YL
ØØ9Ø                  GO TO 38
ØØ91  32           CONTINUE
ØØ92                  IF( TUL .NE. Ø. )  GO TO 35
ØØ94                     IC = IC + 1
ØØ95                     PX(IC) = XL
ØØ96                     PY(IC) = YU
ØØ97  35              CONTINUE
ØØ98                  GO TO 38
ØØ99  38           CONTINUE
Ø1ØØ  39        CONTINUE
      C
      C   SEGMENT 4:
      C
Ø1Ø1            IF( TUL*TC .GE. Ø. )  GO TO 49
Ø1Ø3               KCHK(4) = .TRUE.
Ø1Ø4               IC = IC + 1
Ø1Ø5               FAC = TUL/(ZC - ZUL)
Ø1Ø6               PX(IC) = XLMXC*FAC + XL
Ø1Ø7               PY(IC) = YUMYC*FAC + YU
Ø1Ø8  49        CONTINUE
      C
      C   SEGMENT 5:
      C
Ø1Ø9            IF( TUL*TUR .GT. Ø. )  GO TO 59
Ø111               KCHK(5) = .TRUE.
Ø112               IF( TUL*TUR .EQ. Ø. )  GO TO 52
Ø114                  IC = IC + 1
Ø115                  PX(IC) = TUL * XLMXR/(ZUR-ZUL) + XL
Ø116                  PY(IC) = YU
Ø117                  GO TO 58
Ø118  52           CONTINUE
Ø119                  IF( TUR .NE. Ø. )  GO TO 55
Ø121                     IC = IC + 1
Ø122                     PX(IC) = XR
Ø123                     PY(IC) = YU
Ø124  55              CONTINUE
Ø125                  GO TO 58
Ø126  58           CONTINUE
Ø127  59        CONTINUE
      C
      C   SEGMENT 6:
      C
Ø128            IF( TUR*TC .GE. Ø. )  GO TO 69
Ø13Ø               KCHK(6) = .TRUE.
Ø131               IC = IC + 1
Ø132               FAC = TUR/(ZC - ZUR)
```

```
                PX(IC) = XRMXC*FAC + XR
                PY(IC) = YUMYC*FAC + YU
   69        CONTINUE
C
C  SEGMENT 7:
C
             IF( TLR*TUR .GE. 0. )  GO TO 79
                KCHK(7) = .TRUE.
                IC = IC + 1
                PX(IC) = XR
                PY(IC) = TUR * YLMYU/(ZUR-ZLR) + YU
   79        CONTINUE
C
C  SEGMENT 8:
C
             IF( TLR*TC .GE. 0. )  GO TO 89
                KCHK(8) = .TRUE.
                IC = IC + 1
                FAC = TC/(ZC - ZLR)
                PX(IC) = XRMXC*FAC + XC
                PY(IC) = YLMYC*FAC + YC
   89        CONTINUE
C
C  NOW DERIVE THE LINE SEGMENTS TO BE DRAWN FROM THE CONTENTS
C  OF THE PX AND PY ARRAYS.
C
             IF( IC .LE. 1 )  GO TO 117
                IF( IC .GE. 6 .OR.
     +             (IC .EQ. 5 .AND. CENTER) )  GO TO 100
                   IF( .NOT. KCHK(8) )  GO TO 95
                      DO 94 L = 1, 7
                    LS=L
                         IF( .NOT. KCHK(L) )  GO TO 93
                            PX(IC+1) = PX(1)
                            PY(IC+1) = PY(1)
                            DO 92 M = 1, IC
                               PX(M) = PX(M+1)
                               PY(M) = PY(M+1)
   92                          CONTINUE
                            IF( MOD(LS, 2) .EQ. 1 )  GO TO 95
   93                    CONTINUE
   94                    CONTINUE
                      GO TO 97
   95              IF( .NOT. CENTER .OR. KCHK(1) )  GO TO 97
                      PTEMP = PX1
                      PX1 = PX2
                      PX2 = PTEMP
                      PTEMP = PY1
                      PY1 = PY2
                      PY2 = PTEMP
                      GO TO 97
   97              CONTINUE
                   CALL PLOT(PX1, PY1,+3)
                   DO 98 M = 2, IC
```

```
0184                  PXM = PX(M)
0185                  PYM = PY(M)
0186                  CALL PLOT(PXM, PYM,+2)
0187   98             CONTINUE
0188               GO TO 109
0189   100      CONTINUE
0190               IF( IC .GE. 6 )  GO TO 104
0192                  PX6 = PX5
0193                  PY6 = PY5
0194                  PX5 = PX2
0195                  PY5 = PY2
0196   104         CONTINUE
0197               IF( KCHK(2) )  GO TO 105
0199                  CALL PLOT(PX5, PY5,+3)
0200                  CALL PLOT(PX6, PY6,+2)
0201                  CALL PLOT(PX1, PY1,+2)
0202                  CALL PLOT(PX2, PY2,+3)
0203                  CALL PLOT(PX3, PY3,+2)
0204                  CALL PLOT(PX4, PY4,+2)
0205                  GO TO 108
0206   105         CONTINUE
0207                  CALL PLOT(PX1, PY1,+3)
0208                  CALL PLOT(PX2, PY2,+2)
0209                  CALL PLOT(PX3, PY3,+2)
0210                  CALL PLOT(PX4, PY4,+3)
0211                  CALL PLOT(PX5, PY5,+2)
0212                  CALL PLOT(PX6, PY6,+2)
0213                  GO TO 108
0214   108         CONTINUE
0215               GO TO 109
0216   109      CONTINUE
0217   117   CONTINUE
0218   120   CONTINUE
0219      RETURN
       C
0220      END
```

```
C-T  PROGRAM RASM
C
C-F  MAIN PROGRAM FOR VECTOR TO RASTER CONVERSION - MAPPED ALGORITHM
C
C  COPYRIGHT C1976,  VERSATEC INC., SANTA CLARA, CALIFORNIA  95Ø51
C
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
C        CALLS: DOPEN,DREAD,DWAIT,MWAIT,PREAD,INVECT,CYCLER,IRZERO
C        CALLED BY: -NONE-
C        COMMON VARIBLES USED: /PPEP2/ ISCAN,NWORD,LYNEND,IQ,MAXQ,NEPL,
C                                      MSGLVL,LOST,IXØ,IY1,NDLTX,IY2,
C                                      IBUFG,I2FLG,IR2,MXSTEP,IM
C                              /IOCOM/ LBLK,NBLK,LREC,JUNIT,LUNIT,IPARM,
C                                      MUNIT
C
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
C-P  5ØØ3Ø-2Ø5Ø3  REV. A          - PART NUMBER
C-S  RT-11                        - OPERATING SYSTEM
C
C    AUTHOR: M.D. DOBERVICH         Ø7/Ø7/76
C
C    PROGRAM RASTER
C
      COMMON /PPEP2/ISCAN,NWORD,LYNEND,IQ,MAXQ,NEPL,
     1              NBITS,IBT(16),KBT(16),JBT(16),MSGLVL,LOST,
     2              IXØ,IY1,NDLTX,IY2,
     3              NDW,NDB,NRUN,ISUM,NDLTY,
     4              IBUFG,I2FLG,IR2,MXSTEP,
     N              IM(35ØØ)
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/35ØØ/,INBUFD/512/,MAPEND/6ØØ/
C
      DATA IREC/Ø/,IOLD/1/,MAPKEY/1Ø2/
C
    1 FORMAT (25H FILE/ALGORITHM MISMATCH ,I6)
C-D 2 FORMAT (21H IXØ,IY1,NDLTX,IY2 = 4(1XI6))
    2 FORMAT (21H IXØ,IY1,NDLTX,IY2 = 4(1XI6))
    3 FORMAT (44H IM OR INBUF ARRAY NOT PROPERLY DIMENSIONED )
    4 FORMAT (21H MAP BUFFERS EXCEEDED)
    6 FORMAT (1XI6,13H VECTORS LOST  /
     1        1XI6,18H ACTIVE LINES USED  /)
C
C...  ATTACH THE MATRIX TO THIS JOB.
C...  FORM FEED AT PLOT/FRAME START.
C
C...  OPEN MAP/PARAMETER FILE.
      CALL DOPEN (IPARM,-1,1)
      CALL MTXSET
      CALL MTX(IM(IBUFG),Ø,2)
C
C...  CHECK ALGORITHM KEY.
```

```
      CALL PREAD (KEY,1)
      IF (KEY.EQ.MAPKEY) GO TO 7Ø1Ø
      WRITE (MUNIT,1) KEY
      STOP
C
 7Ø1Ø 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.Ø.AND.IBLKSZ.LE.INBUFD.AND.IWORD.LE.IMD)
     1 GO TO 7Ø11
      WRITE (MUNIT,3)
      STOP
C
 7Ø11 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.Ø) MAPEND = (IWORD-(NEPL*LYNES) - (I2FLG*ISCAN))
 7Ø2Ø MAPSZ = 1
 7Ø3Ø CALL PREAD (IM(MAPSZ),2)
C
C...  END OF MAP FOR THIS PLOT?
      IF (IM(MAPSZ).LT.Ø) GO TO 7Ø4Ø
      MAPSZ = MAPSZ + 2
C
C...  DOES MAP SIZE FIT WITHIN THE MAXIMUM MAP ALLOCATION AREA?
      IF (MAPSZ.LT.MAPEND)   GO TO 7Ø3Ø
C
C...  MAP BUFFERS EXCEEDED.
      WRITE (MUNIT,4)
      STOP
C
C...  CHECK FOR END OF ALL PLOTTING.
 7Ø4Ø IF (MAPSZ.EQ.1) GO TO 77ØØ
      MMAXX = IM(MAPSZ+1)
      IQNDX = MAPEND
C
C...  IF VECTOR QUEUING IS USED THEN MAP ALLOCATION BECOMES DYNAMIC.
      IF (LYNES.NE.Ø) 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 = Ø
      IXEND =  NSCAN-1
      IXSTRP = IXSTR
      NBANDS = Ø
C
C...  READ INITIAL BUFFER
      CALL DREAD (JUNIT,INBUF(IOLD),JREC)
      CALL IRZERO (IM(IBUFG))
C
C...  WAIT FOR LAST READ OPERATION COMPLETE
 71ØØ CALL DWAIT (JUNIT,IERR)
C
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C...  SEARCH MAP FOR VALID VECTOR BLOCK TO READ.
C
      IJMP = 1
      GO TO 712Ø
C
 711Ø NDXFTR = NDXFTR - 2
      NREC = NREC - 1
C
C...  HAS ENTIRE MAP BEEN SEARCHED?
 712Ø IF (NDXFTR.LT.Ø)             GO TO 713Ø
C
C...  DOES MAXIMUM(VECTOR) START BEFORE CURRENT BAND?
      IF (IM(NDXFTR).LT.IXSTR)   GO TO 711Ø
C
C...  DOES MINIMUM (VECTOR) START AFTER CURRENT BAND?
      IF (IM(NDXFTR+1).GT.IXEND) GO TO 711Ø
C
C...  VALID VECTOR DATA FOUND. (INCREMENT MAP AND BLOCK INDEX)
      NDXPRS = NDXFTR
      NDXFTR = NDXFTR - 2
      JREC = NREC + IREC
      NREC = NREC - 1
      GO TO 715Ø
C
C...  END OF BAND. (RESET MAP AND BLOCK INDEX)
 713Ø NDXFTR = MAPSZ -2
      NREC = ((MAPSZ+1)/2) - 1
      NBANDS = NBANDS + 1
      IJMP = Ø
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 712Ø
```

```
      IJMP = -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 (MSGLVL.GE.6) WRITE(LUNIT,2) IX0,IY1,NDLTX,IY2
      IF (MSGLVL.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 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C...  END OF CURRENT VECTOR INPUT BLOCK.
C...  IS STATUS - END OF PLOT,END OF BAND,BAND NOT COMPLETE?
C
C                EOP, EOB, BNC
      IF (IJMP) 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?
C                FFO,  NA, IFF
      IF (IOUT) 7370,7380,7360
C
```

```
C...   CYCLE IOUT TIMES TO MOVE PLOT/FRAME PAST TONER.
 736Ø CALL CYCLER (IOUT,IM(IQNDX))
C
C...   FORM FEED AT END OF PLOT/FRAME:
 737Ø CALL MTX (IM(IBUFG),Ø,2)
 738Ø MAXQ = MAXQ/NEPL
      IF (MSGLVL.GE.1) WRITE (MUNIT,6) LOST,MAXQ
      LOST = Ø
      MAXQ = Ø
C
C...   ADJUST CUMULATIVE PLOT (RECORD) POINTER.
      IREC = IREC + ((MAPSZ+1)/2) - 1
      CALL MWAIT
C
C...   CHECK FOR NEXT PLOT
      GO TO 7Ø2Ø
C
C...   END OF BAND
 74ØØ CALL CYCLER (NBANDS,IM(IQNDX))
      NBANDS = Ø
      IXSTRP = IXSTR
C
C...   SWAP INPUT BUFFERS
 75ØØ IS = INEW
      INEW = IOLD
      IOLD = IS
      GO TO 71ØØ
C
C...   END OF ALL PLOTTING
 77ØØ CALL MTX (IM(IBUFG),-1,2)
      STOP
      END
```

```
FORTRAN IV        VØ2.Ø4      THU Ø8-JAN-81 ØØ:15:37                 PAGE ØØ1

      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
ØØØ1            SUBROUTINE MWAIT
      C
      C DUMMY ROUTINE
      C
ØØØ2            RETURN
ØØØ3            END
```

```
FORTRAN IV        VØ2.Ø4      THU Ø8-JAN-81 ØØ:15:47                 PAGE ØØ1

ØØØ1            SUBROUTINE MTXSET
      C
      C SET UP ROUTINE
      C
ØØØ2            REAL*8 FSPECR
ØØØ3            REAL*4 FBUF(3)
ØØØ4            LOGICAL*1 ITPDRW
ØØØ5            COMMON /MPMTX/ICH,IOFLG,IBLK,IWD,ITPDRW
      C
      C GET DATA PARAMETERS
      C
ØØØ6            IBLK=1
ØØØ7            IWD=1
ØØØ8            WRITE(7,1ØØØ)
ØØØ9      1ØØØ FORMAT(' ENTER A Ø FOR TAPE OUTPUT,',/,
               %' OR A 1 FOR DISC OUTPUT:',$)
ØØ1Ø            READ(5,1ØØ1)IOFLG
ØØ11      1ØØ1 FORMAT(I1)
      C
      C GET FILE NAME IF REQD
      C
ØØ12            IF(IOFLG.EQ.Ø)GOTO 1Ø
ØØ14            WRITE(7,1ØØ2)
ØØ15      1ØØ2 FORMAT(' ENTER FILE NAME FOR OUTPUT:',$)
ØØ16            READ(5,1ØØ3)FBUF
ØØ17      1ØØ3 FORMAT(3A4)
ØØ18            CALL IRAD5Ø(12,FBUF,FSPECR)
ØØ19            ICH=IGETC()
ØØ2Ø            IF(IENTER(ICH,FSPECR,-1).LT.Ø)STOP'ENTER ERROR'
ØØ22            RETURN
      C
      C GET TAPE NO
      C
ØØ23        1Ø WRITE(7,1ØØ4)
ØØ24      1ØØ4 FORMAT(' ENTER TAPE DRIVE NUMBER:',$)
ØØ25            READ(5,1ØØ1)ITPDRW
ØØ26            RETURN
ØØ27            END
```

```
      SUBROUTINE MTX(IPBUF,NWDS,IFLAG)
C
C MAIN TRANSFER ROUTINE
C
      INTEGER*2 IBUF(2048),IPBUF(1),BUF(2304)
      LOGICAL*1 ISTAT,ITPDRW
      COMMON /MPMTX/ICH,IOFLG,IBLK,IWD,ITPDRW
      EQUIVALENCE (BUF(257),IBUF(1))
      DATA NBYTBF/4608/,NBLKBF/9/,ICOUNT/0/
      DO 5 J=1,10
    5 BUF(J)=0
C
C CHECK FOR OPERATION TYPE
C
      IF(NWDS.EQ.0)RETURN
      ICOUNT=ICOUNT+1
      IF(NWDS.LT.0)GOTO 500
C
C SWOP DATA TO OUTPUT BUFFER
C
      DO 10 J=1,NWDS
      IBUF(IWD)=IPBUF(J)
      IWD=IWD+1
      IF(IWD.LE.2048)GOTO 10
      IF(IOFLG.EQ.0)GOTO 20
C
C DISC OUTPUT
C
      IF(IWRITW(2048,IBUF,IBLK,ICH).LT.0)STOP'WRITE ERROR'
      IBLK=IBLK+8
      IWD=1
      GOTO 10
C
C TAPE OUTPUT
C
   20 IFLEN=NBYTBF
      NBYT=NBYTBF
      IEOTW=0
      CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
      IWD=1
C
C CHECK FOR ERRORS
C
      IF(IEOTW.GE.0)GOTO 30
      WRITE(7,1000)ITPDRW,ICOUNT
 1000 FORMAT(' EOT ON WRITE DRIVE:',I2,' BUFFER NUMBER:',I5)
      WRITE(7,1001)
 1001 FORMAT(' ENTER NEW WRITE DRIVE NUMBER:',$)
      READ(5,1002)ITPDRW
 1002 FORMAT(I1)
      IEOTW=0
      IF(ITPDRW.GT.2)STOP' EOT TERMINATE'
   30 IF(ISTAT.GE.0)GOTO 10
      WRITE(7,1003)ICOUNT
```

```
 1003 FORMAT(' FATAL ERROR ON WRITING BUFFER NUMBER:',I5)
      STOP' FATAL WRITE ERROR'
   10 CONTINUE
      RETURN
C
C COME HERE TO FLUSH BUFFERS
C
  500 CONTINUE
      DO 40 J=NWD,2048
   40 IBUF(J)=0
      IF(IOFLG.EQ.0)GOTO 50
      IF(IWRITW(2048,IBUF,IBLK,ICH).LT.0)STOP' WRITE ERROR'
      CALL CLOSEC(ICH)
      WRITE(7,1010)ICOUNT
      RETURN
C
C FLUSH TO TAPE
C
   50 IFLEN=NBLKBF
      NBYT=NBYTBF
      IEOTW=0
      CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
      IF(IEOTW.LT.0)GOTO 60
C
C AS ONLY FLUSHING BUFFERS IGNORE ERRORS
C
      BUF(1)='E'
      BUF(2)='O'
      BUF(3)='D'
      CALL TAPSUB(1,ITPDRW,ISTAT,IFLEN,ICOUNT,BUF,NBYT,IEOTW)
   60 WRITE(7,1010)ICOUNT
 1010 FORMAT(' BUFFERS WRITTEN=',I5)
      RETURN
      END
```

Utilities

File Save/Restore Utility:- MPTPSV

This program is completely interactive and the user is promted 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 promted.

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 198Ø
C  TAPE/DISC SAVE RESTORE PROGRAM
C  VERSION 1
C
C
      REAL*8 FSPECR,FSPECW,FSPEC(1ØØØ),DATE(1ØØØ),NBUF,DBUF,FSBUF,
     %EOTCOD,QSPEC(2Ø),TPNUM,TODATE
      REAL*4 FBUF(3),COM(4),CBUF
      INTEGER*2 VNUM(1ØØØ),SIZE(1ØØØ),VBUF,SBUF,
     %IVNUM(1Ø),IFNUM(1Ø),QNO(2Ø),BUF(1Ø24)
      LOGICAL*1 EOT,EOR,HBLK(2Ø),ANS,YES,NO,IDRV,ISTAT
      EQUIVALENCE (HBLK(1),NBUF),(HBLK(9),DBUF),(HBLK(17),VBUF),
     %(HBLK(19),SBUF)
      COMMON/SERCH/FSPEC,DATE,VNUM,SIZE,HBLK,IFILE
      DATA DEV/3RRK /,FSBUF/12RDKØMPTPSVDAT/,YES/'Y'/,NO/'N'/
      DATA COM(1)/'SAVE'/,COM(2)/'REST'/,COM(3)/'STOP'/,COM(4)/'TDIR'/
      DATA EOTCOD/12REOTEOTEOTEOT/
C
C SET UP I/O CHANNELS
C
      IF(ICDFN(25).NE.Ø)STOP'CHAN OVERFLOW'
      IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERR'
      ITR=IGETC()
      IRD=2Ø
      IWRT=21
      IFILE=Ø
      IQNUM=Ø
      EOR=.FALSE.
      EOT=.FALSE.
C
C GET TODAYS DATE
C
      CALL GDATE(TODATE)
      TYPE 1Ø
   1Ø FORMAT('  TAPE SAVE/RESTORE PROGRAM VERSION 1')
      TYPE 15
   15 FORMAT(' ENTER TAPE NAME AND DRIVE NUMBER')
      ACCEPT 16,TPNUM,IDRV
   16 FORMAT(A8,I2)
      TYPE 17
   17 FORMAT(' IS THE TAPE TO BE INITIALISED Y/N:',$)
      ACCEPT 18,ANS
   18 FORMAT(A1)
      IF(ANS.EQ.NO)GOTO 19
C
C TAPE INITIALISATION CODE
C
      NBUF=TPNUM
      DBUF=TODATE
      ILEN=1
      CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,2Ø)
C
C WRITE EOT BLOCK AND REWIND TO BEGINNING OF BLOCK
```

```
C
      NBUF=EOTCOD
      ILEN=1
      CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,2Ø)
      CALL TPHAND(6,IDRV,ISTAT,2,ILEN,IFILE, , )
      EOT=.TRUE.
      GOTO 1
C
C NORMAL INTRO TO A SESSION
C
   1Ø CALL TPHAND(2,IDRV,ISTAT, ,ILEN,IFILE,HBLK,2Ø)
      IF(TPNUM.NE.NBUF)TYPE 11
   11 FORMAT(' WARNING TAPE NAME DIFFERENT TO INIT VALUE!!!')
      CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
C
C START OF MAIN SOFTWARE LOOP
C
    1 TYPE 2Ø
   2Ø FORMAT(' ?',$)
      ACCEPT 3Ø,CBUF
   3Ø FORMAT(A4)
C
C AFTER COMMAND RECEIVED DECODE IT
C
      DO 4Ø I=1,4
   4Ø IF(CBUF.EQ.COM(I))NCOM=I
      GOTO (1ØØØ,2ØØØ,3ØØØ,4ØØØ)NCOM
C
C CODE FOR SAVEING A FILE
C
 1ØØØ TYPE 1Ø1Ø
 1Ø1Ø FORMAT(' ENTER FILENAME TO BE SAVED,WITH VERSION NUMBER')
      ACCEPT 1Ø2Ø,FBUF,NVNO
 1Ø2Ø FORMAT(3A4,I2)
      CALL IRAD5Ø(12,FBUF,FSPECR)
C
C GET TO END OF TAPE TO SAVE THE NEW FILE
C
      IF(.NOT.EOT)CALL FSERCH(EOTCOD,IDRV,EOT)
      EOT=.TRUE.
C
C OPEN FILE TO BE SAVED
C
      ISIZE=LOOKUP(IWRT,FSPECR)
      IF(ISIZE.LE.Ø)GOTO 9ØØ1
      IF(ISIZE.GT.4) GOTO 1Ø5Ø
      NWDS=ISIZE*256
      NBYTS=NWDS*2
      IF(IREADW(NWDS,BUF,Ø,IWRT).LT.Ø)GOTO 9ØØ2
C
C SET UP INTERNAL DIRECTORY
C
 1Ø5Ø IFILE=IFILE+1
      FSPEC(IFILE)=FSPECR
```

```
      DATE(IFILE)=TODATE
      VNUM(IFILE)=NVNO
      SIZE(IFILE)=ISIZE
C
C SET UP HEADER BLOCK
C
      NBUF=FSPECR
      DBUF=TODATE
      VBUF=NVNO
      SBUF=ISIZE
C
C WRITE HEADER BLOCK
C
      ILEN=1
      CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
C
C WRITE OUT THE FILE
C
      ILEN=ISIZE
      IF(ILEN.GT.4)CALL TPHAND(3,IDRV,ISTAT, ,ILEN,IFILE, , )
      IF(ILEN.LE.4)CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,BUF,NBYTS)
C
C SEE IF MORE SAVES RTO BE DONE
C
      CALL CLOSEC(IWRT)
      TYPE 1030
 1030 FORMAT(' MORE FILES TO BE SAVED Y/N ?',$)
      ACCEPT 1040,ANS
 1040 FORMAT(A1)
      IF(ANS.EQ.YES)GOTO 1000
C
C WRITE EOT FILE IF NO MORE
C
      NBUF=EOTCOD
      ILEN=1
      CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
C
C REWIND TO BEGINNING OF EOT FILE
C
      CALL TPHAND(6,IDRV,ISTAT,2, ,IFILE, , )
      GOTO 1
C
C CODE FOR A RESTORE
C
 2000 TYPE 2010
 2010 FORMAT(' ENTER FILE TO BE RESTORED')
      ACCEPT 2020,FBUF
 2020 FORMAT(3A4)
      CALL IRAD50(12,FBUF,FSPECW)
      IVER=0
C
C SEARCH TO SEE IF FILE ALREADY PASSED
C
      DO 2030 I=1,IFILE
```

```
      IF(FSPECW.NE.FSPEC(I))GOTO 2030
      IVER=IVER+1
      IFNUM(IVER)=I
      IVNUM(IVER)=VNUM(I)
 2030 CONTINUE
      IF(IVER.LE.1)GOTO 2040
C
C IF MORE THAN ONE VERSION FIND WHICH ONE REQUIRED
C
      TYPE 2050
 2050 FORMAT(' ENTER VERSION NUMBER REQUIRED:',$)
      ACCEPT 2060,NVNO
 2060 FORMAT(I2)
      DO 2065 I=1,IVER
 2065 IF(NVNO.EQ.IVNUM(I))IFNUM(1)=IFNUM(I)
C
C FIND OUT IF NEED TO QUEUE OR CONTINUE
C
 2040 IF(IVER.NE.0)GOTO 2070
      TYPE 2050
      ACCEPT 2060,NVNO
 2042 CALL FSERCH(FSPECW,IDRV,EOT)
      IF(EOT)GOTO 2999
      CALL TPHAND(2,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
      CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
      IF(NVNO.NE.VBUF)CALL TPHAND(5,IDRV,ISTAT,2,ILEN,IFILE, , )
      IF(NVNO.NE.VBUF)GOTO 2042
      ISIZE=SBUF+16
      IF(IENTER(IRD,FSBUF,ISIZE).LT.0)GOTO 9003
      CALL TPHAND(1,IDRV,ISTAT, ,ILEN,IFILE, , )
      CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
      CALL R50ASC(12,FSPECW,FBUF)
      TYPE 2041,FBUF
 2041 FORMAT(' ENTER NEW NAME FOR TAPE FILE: ',3A4)
      ACCEPT 2020,FBUF
      CALL IRAD50(12,FBUF,FSPECW)
      ISIZE=SBUF
      IF(IENTER(ITR,FSPECW,ISIZE).LT.0)GOTO 9003
      IBLK=0
      DO 2045 I=1,ISIZE
      IF(IREADW(256,BUF,IBLK,IRD).LT.0)GOTO 9004
      IF(IWRITW(256,BUF,IBLK,ITR).LT.0)GOTO 9005
      IBLK=IBLK+1
 2045 CONTINUE
      CALL CLOSEC(IRD)
      CALL CLOSEC(ITR)
      GOTO 1
C
C PUT NAME IN THE QUEUE
C
 2070 IQNUM=IQNUM+1
      QSPEC(IQNUM)=FSPECW
      QNO(IQNUM)=IFNUM(1)
 2080 IF(.NOT.EOR)GOTO 1
```

```
C
C COME HERE AT END OF RUN
C
      CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
      IQ=0
      IFILE=1
      CALL TPHAND(5,IDRV,ISTAT,2, ,IFILE, , )
      N2=1
C SORT QUEUE INTO ORDER
C
      IGAP=IQNUM
 2081 IF(IGAP.LE.1)GOTO 2100
      IGAP=IGAP/2
      IMAX=IQNUM-IGAP
 2085 IEX=0
      DO 2086 I=1,IMAX
      IPLUSG=I+IGAP
      IF(QNO(I).LE.QNO(IPLUSG))GOTO 2086
      ISAVE=QNO(I)
      FSPECW=QSPEC(I)
      QNO(I)=QNO(IPLUSG)
      QSPEC(I)=QSPEC(IPLUSG)
      QNO(IPLUSG)=ISAVE
      QSPEC(IPLUSG)=FSPECW
      IEX=IEX+1
 2086 CONTINUE
      IF(IEX.GT.0)GOTO 2085
      GOTO 2081
C
C MAIN RESTORE LOOP
C
 2100 CONTINUE
      IQ=IQ+1
      N1=N2
      N2=QNO(IQ)
      NSEP=N2-N1
      IFILE=IFILE+NSEP
      NFILE=NSEP*4
C
C WIND FOWARD TO CORRECT FILE
C
      IF(NFILE.GT.0)CALL TPHAND(5,IDRV,ISTAT,NFILE,ILEN,IFILE, , )
C
C READ HEADER BLOCK AND CHECK IF FOUND CORRECT FILE
C
      CALL TPHAND(2,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
      IF(NBUF.NE.QSPEC(IQ))GOTO 2800
      CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
      FSPECW=NBUF
      ISIZE=SBUF+16
C
C READ ONTO TEMPORARY FILE
C
      IF(IENTER(IRD,FSBUF,ISIZE).LT.0)GOTO 9003
```

```
              CALL TPHAND(1,IDRV,ISTAT, ,ILEN,IFILE, , )
              CALL TPHAND(5,IDRV,ISTAT,1,ILEN,IFILE, , )
              CALL R5ØASC(12,FSPECW,FBUF)
              TYPE 2Ø41,FBUF
              ACCEPT 2Ø2Ø,FBUF
              CALL IRAD5Ø(12,FBUF,FSPECW)
              IF(IENTER(ITR,FSPECW,SBUF).LT.Ø)GOTO 9ØØ3
      C
      C TRANSFER TO APPROPRIATE PERMANENT FILE
      C
              IBLK=Ø
              DO 21Ø1 I=1,SBUF
              IF(IREADW(256,BUF,IBLK,IRD).LT.Ø)GOTO 9ØØ4
              IF(IWRITW(256,BUF,IBLK,ITR).LT.Ø)GOTO 9ØØ5
              IBLK=IBLK+1
        21Ø1  CONTINUE
              CALL CLOSEC(IRD)
              CALL CLOSEC(ITR)
              IQNUM=IQNUM-1
              N2=N2+1
              IFILE=IFILE+1
              IF(IQNUM.NE.Ø)GOTO 21ØØ
              CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
              STOP' NORMAL TERMINATION'
      C
      C WRONG FILE FOUND
      C
        28ØØ  CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
              CALL R5ØASC(12,NBUF,FBUF)
              TYPE 281Ø,FBUF
        281Ø  FORMAT(' FILE FOUND ON TAPE= ',3A4)
              CALL R5ØASC(12,QSPEC(IQ),FBUF)
              TYPE 282Ø,FBUF
        282Ø  FORMAT(' FILE REQUIRED = ',3A4)
              STOP' WRONG FILE FOUND FOR RESTORE'
      C
      C FILE NOT FOUND
      C
        2999  TYPE 29ØØ
        29ØØ  FORMAT(' FILE NOT FOUND')
              GOTO 1
      C
      C CODE FOR STOP COMMAND
      C
        3ØØØ  EOR=.TRUE.
              IF(IQNUM.NE.Ø)GOTO 2Ø8Ø
              CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
              CALL CLOSEC(IRD)
              CALL CLOSEC(IWRT)
              CALL CLOSEC(ITR)
              STOP' NORMAL TERMINATION'
      C
      C CODE FOR DIRECTORY
      C
```

```
      4000 IF(.NOT.EOT)CALL FSERCH(EOTCOD,IDRV,EOT)
           EOT=.TRUE.
           TYPE 4070
      4070 FORMAT(' DIR ON TERMINAL Y/N:',$)
           ACCEPT 4080,ANS
      4080 FORMAT(A1)
           LUNIT=6
           IF(ANS.EQ.YES)LUNIT=7
      4010 WRITE(LUNIT,4020)TPNUM,IDRV
      4020 FORMAT('1 TAPE NO= ',A8,' DRIVE NO',I3,' FILE DIRECTORY')
           WRITE(LUNIT,4030)
      4030 FORMAT(' FILE NAME ',5X,' DATE SAVED ',5X,' VERSION ',
          X5X,' SIZE')
           DO 4040 I=1,IFILE
           CALL R50ASC(12,FSPEC(I),FBUF)
      4040 WRITE(LUNIT,4050)FBUF,DATE(I),VNUM(I),SIZE(I)
      4050 FORMAT(1X,3A4,3X,A8,14X,I2,10X,I5)
           WRITE(LUNIT,4060)IFILE
      4060 FORMAT(' TOTAL NUMBER OF FILES ON TAPE=',I3)
           GOTO 1
C
C ERROR FINISHES
C
      9001 NBUF=EOTCOD
           ILEN=1
           CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
           CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
           CALL CLOSEC(IRD)
           CALL CLOSEC(IWRT)
           CALL CLOSEC(ITR)
           STOP'LOOKUP ERROR'
      9002 NBUF=EOTCOD
           ILEN=1
           CALL TPHAND(4,IDRV,ISTAT, ,ILEN,IFILE,HBLK,20)
           CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
           CALL CLOSEC(IRD)
           CALL CLOSEC(IWRT)
           CALL CLOSEC(ITR)
           STOP'READ ERR FOR SAVE'
      9003 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
           CALL CLOSEC(IRD)
           CALL CLOSEC(IWRT)
           CALL CLOSEC(ITR)
           STOP' ENTER ERR FOR RESTORE '
      9004 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
           CALL CLOSEC(IRD)
           CALL CLOSEC(IWRT)
           CALL CLOSEC(ITR)
           STOP' READ ERR ON RESTORE TRANSFER'
      9005 CALL TPHAND(7,IDRV,ISTAT, , ,IFILE, , )
           CALL CLOSEC(IRD)
           CALL CLOSEC(IWRT)
           CALL CLOSEC(ITR)
           STOP' WRITE ERR ON RESTORE TRANSFER'
```



```
           END
```

```
ØØØ1          SUBROUTINE FSERCH(FNAME,IDRV,EOT)
        C
        C SUBROUTINE WHICH SEARCHES FOWARD TO SPECIFIED
        C FILE NAME ON THE TAPE
        C
ØØØ2          REAL*8 FNAME,FSPEC(1ØØØ),DATE(1ØØØ),NBUF,DBUF,EOTCOD
ØØØ3          INTEGER*2 SIZE(1ØØØ),VNO(1ØØØ),VBUF,SBUF
ØØØ4          LOGICAL*1 HBLK(2Ø),IDRV,EOT,ISTAT
ØØØ5          COMMON/SERCH/ FSPEC,DATE,VNO,SIZE,HBLK,IFILE
ØØØ6          EQUIVALENCE(HBLK(1),NBUF),(HBLK(9),DBUF),(HBLK(17),VBUF),
             X(HBLK(19),SBUF)
ØØØ7          DATA EOTCOD/12REOTEOTEOTEOT/
ØØØ8       1Ø CALL TPHAND(2,IDRV,ISTAT, ,ILEN,IFILE,HBLK,2Ø)
ØØØ9          IF(NBUF.EQ.EOTCOD)GOTO 15
        C
        C STICK HEADER INFO IN DIRECTORY
        C
ØØ11          IFILE=IFILE+1
ØØ12          FSPEC(IFILE)=NBUF
ØØ13          DATE(IFILE)=DBUF
ØØ14          VNO(IFILE)=VBUF
ØØ15          SIZE(IFILE)=SBUF
ØØ16          IF(NBUF.EQ.FNAME)GOTO 2Ø
        C
        C POSITION TAPE NEXT TO NEXT HEADER
        C
ØØ18          CALL TPHAND(5,IDRV,ISTAT,3,ILEN,IFILE, , )
ØØ19          GOTO 1Ø
        C
        C REWIND TO BEGINNING OF FOUND HEADER
        C
ØØ2Ø       15 EOT=.TRUE.
ØØ21       2Ø CALL TPHAND(6,IDRV,ISTAT,1,ILEN,IFILE, , )
ØØ22          RETURN
ØØ23          END
```

```
      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
      INTEGER*2 MASK(8),ESTATI
      LOGICAL*1 ISTAT,COM(4),SDSCOM(8),IDRV,ITLEN,ECOM(4),
     %IFLEN,ESTAT,BUF(1)
      DATA MASK/"1,"2,"4,"10,"20,"40,"100,"200/
      DATA SDSCOM/"0,"1,"2,"3,"4,"5,"6,"7/
      ITRY=0
      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
 1000 ITRY=ITRY+1
C
C SET UP COMMAND FOR READ
C
      COM(1)=SDSCOM(4)
      COM(2)=1
      COM(3)=IDRV
      COM(4)=-1
      CALL SDS10(COM,ISTAT,ITLEN,ILEN)
      IF(ISTAT.EQ.0)RETURN
C
C ERROR DETECTED ON READ
C
      ISTATI=ISTAT
      GOTO 40
C
C READ FROM TAPE TO MEMORY
C
 2000 NBUF=NBYT
      CALL TREAD(BUF,NBUF,ISTAT,IDRV)
      ITRY=ITRY+1
      IF(ISTAT.EQ.0)RETURN
```

```
      ISTATI=ISTAT
      GOTO 4Ø
C
C IF SHORT RECORD FOUND REREAD TAPE
C
   5Ø ITMP=ISTATI.AND.MASK(6)
      IF(ITMP.NE.Ø)GOTO (1ØØØ,2ØØØ)ICOM
      ITMP=ISTATI.AND.MASK(2)
      IF(ITMP.EQ.Ø)RETURN
C
C IF CRC ERROR FOUND REWIND TAPE AND RETRY
C
      TYPE 2Ø1Ø,IFNUM
 2Ø1Ø FORMAT(' FILE NO ',I4,' CRC ERROR REWINDING')
      IF(ITRY.GE.2)GOTO 13Ø
      ECOM(1)=SDSCOM(6)
      ECOM(2)=1
      ECOM(3)=IDRV
      ECOM(4)=Ø
      CALL SDS1Ø(ECOM,ESTAT, , )
      GOTO (1ØØØ,2ØØØ)ICOM
C
C WRITE SECTION
C
 3ØØØ ITRY=ITRY+1
      IF(ITRY.GT.2)GOTO 13Ø
      COM(1)=SDSCOM(7)
      IFLEN=(ILEN+3)/4
      COM(2)=IFLEN
      COM(3)=IDRV
      COM(4)=1
      CALL SDS1Ø(COM,ISTAT, , )
      IF(ISTAT.EQ.Ø)RETURN
C
C WRITE ERROR DETECTED
C
      ISTATI=ISTAT
      GOTO 4Ø
C
C MEMORY TO TAPE WRITE
C
 4ØØØ CONTINUE
      NBUF=NBYT
      IFLEN=(ILEN+3)/4
      IF(IFLEN.LT.2)IFLEN=2
      IPAD=(IFLEN*2Ø48)-NBUF
      CALL TWRIT(BUF,NBUF,ISTAT,IPAD,IFLEN,IDRV)
      IF(ISTAT.EQ.Ø)RETURN
      ISTATI=ISTAT
      GOTO 4Ø
C
C ERROR RETURN POSITION
C
   7Ø ITMP=ISTATI.AND.MASK(6)
```

```
           ITMPI=ISTATI.AND.MASK(2)
           IF(ITMP.EQ.Ø.AND.ITMPI.EQ.Ø)RETURN
C
C REPORT AND RETRY
C
           TYPE 2Ø2Ø,IFNUM
     2Ø2Ø  FORMAT(' FILE NO ',I4,' WRITE CRC ERR ')
           RETURN
C
C WIND FOWARD ONE FILE
C
     5ØØØ  IF(ICOM.EQ.5)COM(1)=SDSCOM(5)
           IF(ICOM.EQ.6)COM(1)=SDSCOM(6)
           IF(ICOM.EQ.7)COM(1)=SDSCOM(2)
           COM(2)=ITLEN
           COM(3)=IDRV
           COM(4)=Ø
           CALL SDS1Ø(COM,ISTAT, , )
C
C CLEAR IRRELEVANT BITS FROM ERROR BYTE
C
           ISTAT=ISTAT.AND..NOT.MASK(6)
           ISTAT=ISTAT.AND..NOT.MASK(2)
           IF(ISTAT.EQ.Ø)RETURN
           ISTATI=ISTAT
           GOTO 4Ø
       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
       4Ø  TYPE 1Ø1Ø,ISTATI,IFNUM
     1Ø1Ø  FORMAT(' STATUS=',I3,' FILE NO=',I4)
           ISTATI=ISTATI.AND..NOT.MASK(4)
           ITMP=ISTATI.AND.MASK(5)
           IF(ITMP.EQ.Ø)GOTO 8Ø
       9Ø  ECOM(1)=SDSCOM(1)
           ECOM(2)=Ø
           ECOM(3)=IDRV
           ECOM(4)=Ø
           CALL SDS1Ø(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.Ø)GOTO 9Ø
           ITMP=ESTATI.AND.MASK(4)
           IF(ITMP.EQ.Ø.AND.ICOM.EQ.7)GOTO 9Ø
           IF(ICOM.EQ.7)GOTO 35
```

```
      GOTO (1000,2000,3000,4000,5000,5000,5000)ICOM
C
C TAPE OFFLINE
C
   60 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
      GOTO (1000,2000,3000,4000,5000,5000,5000)ICOM
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)
      IDRV=3
      RETURN
  120 GOTO(50,50,70,70,35,35,35)ICOM
C
C ERROR EXIT RETURN
C
  130 ISTAT=-1
      RETURN
      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

```
      REAL*8 FSPEC
      REAL*4 DEVNAM,FMBUF(3)
      LOGICAL*1 SDSCOM(4),COM(8),LUN,HBLK(1Ø),FNUM,ITYP
      INTEGER*2 STATUS,TLEN,FLEN
      DATA COM/"Ø,"1,"2,"3,"4,"5,"6,"7/
      DATA DEVNAM/3RRK /
      IF(IFETCH(DEVNAM).NE.Ø)STOP'FETCH ERROR'
      IF(ICDFN(3Ø).NE.Ø)STOP'CDFN ERROR'
    1 CONTINUE
      TYPE 1Ø
   1Ø FORMAT(' ENTER TEST OPTION  ',/,
     %' Ø FOR A READ',/,
     %' 1 FOR A WRITE',/,
     %' 2 FOR UTILITY COMMANDS',/,
     %' 3 TO LEAVE THE PROGRAM :',$)
      ACCEPT 2Ø ,IOPT
   2Ø FORMAT(I1)
      IF(IOPT.EQ.3)STOP'EXECUTION TERMINATED'
      IF(IOPT.EQ.2)GO TO 6Ø
      TYPE 3Ø
   3Ø FORMAT(' ENTER THE FILE SPEC,NO COLONS OR DOTS',/,
     %' BUT WITH IMPLIED BLANKS INCLUDED:',$)
      ACCEPT 4Ø,FMBUF
   4Ø FORMAT(3A4)
      CALL IRAD5Ø(12,FMBUF,FSPEC)
      IF(IOPT.GT.Ø)GOTO 5Ø
C
C READ
C
      IENT=IENTER(2Ø,FSPEC,4ØØ)
      IF(IENT.LT.Ø)TYPE 45,IENT
   45 FORMAT(' ENTER ERROR IENT=',I3)
      IF(IENT.LT.Ø)STOP
      TYPE 7Ø
   7Ø FORMAT(' ENTER TAPE UNIT TO BE READ FROM :',$)
      ACCEPT 2Ø,LUN
      TYPE 71
   71 FORMAT(' ENTER Ø FOR FIELD TAPE 1 FOR INTERNAL:',$)
      ACCEPT 2Ø,ITYP
      SDSCOM(1)=COM(4)
      SDSCOM(2)=ITYP
      SDSCOM(3)=LUN
      SDSCOM(4)="2ØØ
      CALL SDS1Ø(SDSCOM,STATUS,TLEN,FLEN)
      TYPE 35,STATUS,FLEN
   35 FORMAT(' STATUS EQUALS...',I3,'FILE LENGTH= ',I3)
      IF(ITYP.EQ.1)GOTO 91
      IF(IREADW(5,HBLK,Ø,2Ø).LT.Ø)STOP'READ HBLK ERROR'
      DO 8Ø I=1,1Ø
   8Ø HBLK(I)=HBLK(I)+"6Ø
      TYPE 9Ø,HBLK(2),HBLK(1),HBLK(4),HBLK(3),HBLK(6),HBLK(5),
     %HBLK(8),HBLK(7),HBLK(1Ø),TLEN
   9Ø FORMAT(' TAPE HEADER INFO',/,
     %' FILE NO = ',3A1,/,
```

```
     %' DATA CONSTANTS = ',5A1,/,
     %' SAMPLING PERIOD = ',A1,/,
     %' DATA TIME LENGTH = ',I2)
   91 CALL CLOSEC(20)
      GO TO 1
C
C WRITE
C
   50 FLEN=LOOKUP(21,FSPEC)
      IF(FLEN.LT.0)STOP'LOOKUP ERROR'
      FLEN=(FLEN+3)/4
      TYPE 35,STATUS,FLEN
      TYPE 95
   95 FORMAT(' ENTER THE TAPE UNIT TO BE WRITTEN TO:',$)
      ACCEPT 20,LUN
      SDSCOM(1)=COM(7)
      SDSCOM(2)=FLEN
      SDSCOM(3)=LUN
      SDSCOM(4)=1
      CALL SDS10(SDSCOM,STATUS,,FLEN)
      TYPE 35,STATUS,FLEN
      CALL CLOSEC(21)
      GOTO 1
C
C WIND
C
   60 FNUM=0
      LNUM=0
      TYPE 100
  100 FORMAT(' UTILITY COMMAND TABLE:',/,
     %' ----------------------',/,
     %' 1:-STATUS',/,
     %' 2:-REWIND',/,
     %' 3:-REWIND OFF LINE',/,
     %' 5:-FORWARD WIND N FILES',/,
     %' 6:-REVERSE WIND N FILES',/,
     %' 8:-RETURN PDP-8E TO OS8',/,
     %'  ENTER YOUR OPTION :',$)
      ACCEPT 20,NOPT
      IF(NOPT.EQ.8)GOTO 110
      TYPE 120
  120 FORMAT(' ENTER TAPE UNIT NO:',$)
      ACCEPT 20,LUN
      IF(NOPT.EQ.5.OR.NOPT.EQ.6)TYPE 130
  130 FORMAT(' ENTER NO OF FILES TO BE WOUND PAST :',$)
      IF(NOPT.EQ.5.OR.NOPT.EQ.6)ACCEPT 20,FNUM
  110 SDSCOM(1)=COM(NOPT)
      SDSCOM(2)=FNUM
      SDSCOM(3)=LUN
      SDSCOM(4)=0
      CALL SDS10(SDSCOM,STATUS,,)
      TYPE 35,STATUS,FLEN
      GOTO 1
      END
```

```
      REAL*8 FSPECR,FSPECW
      REAL*4 FBUF(3),SEISM(2Ø48)
      LOGICAL*1 TPDRR,ISTAT,ITLEN
      DATA DEV/3RRK /
      IF(ICDFN(25).NE.Ø)STOP'CHAN ERR'
      IF(IFETCH(DEV).NE.Ø)STOP'FETCH ERROR'
      CALL ASSIGN(1,'DK2:MPTPDK.DAT',14)
      CALL ASSIGN(2,'DK2:MPTPDK.LOG',14)
      IRD=2Ø
      IWRT=21
      READ(1,1ØØØ)NFILIN,NBLKS,IBLKST,TPDRR
1ØØØ  FORMAT(4I5)
      READ(1,12ØØ)FBUF
12ØØ  FORMAT(3A4)
      CALL IRAD5Ø(12,FBUF,FSPECW)
      READ(1,12ØØ)FBUF
      CALL IRAD5Ø(12,FBUF,FSPECR)
      NWDS=NBLKS*256
      IBLKOT=1
      IBLKSZ=NFILIN*NBLKS+1
      IF(IENTER(IWRT,FSPECW,IBLKSZ).LT.Ø)STOP'ENTER ERR'
      DO 2Ø I=1,NFILIN
      IFIL=1
      IF(IENTER(IRD,FSPECR,3ØØ).LT.Ø)STOP'ENT2 ERR'
      IF(TPDRR.LE.2)GOTO 3Ø
      WRITE(2,14ØØ)IFIL
14ØØ  FORMAT(' FILE NO ',I5,' EOT ')
      CALL CLOSEC(IWRT)
      STOP'EOT'
  3Ø  CALL TAPRED(-1,TPDRR,ISTAT,ITLEN,IFLEN,IFIL)
      IF(ISTAT.LT.Ø)WRITE(2,15ØØ)IFIL
15ØØ  FORMAT(' FILE NO ',I5, 'RETRIES FAILE D LAST READ USED')
      IF(TPDRR.GT.2)GOTO 4Ø
      CALL TAPRED(Ø,TPDRR,ISTAT, , ,IFIL)
  4Ø  CALL CLOSEC(IRD)
      IF(LOOKUP(IRD,FSPECR).LT.Ø)STOP'LOOKUP ERR'
      IF(IREADW(NWDS,SEISM,IBLKST,IRD).LT.Ø)STOP'READ ERR'
      IF(IWRITW(NWDS,SEISM,IBLKOT,IWRT).LT.Ø)STOP' WRITE ERR'
      IBLKOT=IBLKOT+NBLKS
      CALL CLOSEC(IRD)
  2Ø  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