1982/35
φ3

100840

BUREAU OF MINERAL RESOURCES, GEOLOGY AND GEOPHYSICS

RECORD

RECORD 1982/35

BMR MAGNETOTELLURIC SYSTEM SOFTWARE, 1980.

by

A. SPENCE & D.W. KERR.

RECORD 1982/35

BMR MAGNETOTELLURIC SYSTEM SOFTWARE, 1980.

by

A. SPENCE & D.W. KERR.

CONTENTS

FIGURES

SUMMARY

This record describes the magnetotelluric software used in field operations during 1980. Because of the resignation from BMR of the software designer (D.W. Kerr) it is regarded as important to document the concepts behind the data acquisition and processing system that was operating at that time.

INTRODUCTION

The data acquisition and processing system used by BMR for magnetotelluric investigations has evolved over a number of years. However the design concepts employed have never been documented. This Record has been written to help overcome the problem of poor documentation. It is foreseen that the magnetotelluric system will continue to evolve as computer hardware and software improve. However, it is instructive to describe the software system for the 1980 field season because until then, the system used Hewlett-Packard RTE-2 compatible software.

The magnetotelluric system may be conveniently divided into four parts; data acquisition, data processing, data manipulation and data interpretation. This paper deals with the programs used in data acquisition and data processing.

DATA ACQUISITION

The 1980 MT data acquisition system comprised a number of programs, which are indicated on the program flow chart No. 1 (Fig. 1). The two programs which interfaced the user to the system were MAGTL and SWTCH. MAGTL started the data acquisition from the Phoenix analogue-to-digital converter and also the digital-to-analogue output. This program prompted the user for information such as the electrode wire lengths, the digital-to-analogue conversion gain, the header information, number of points required, digitising interval, etc. Facilities were built in to record several files with incrementing file numbers and the same headers. It also looked after such incidental information as recording the start time for each file. Program SWTCH was the second program which communicated with the user of the system. Its basic function was to record information about the coils in use and to record and take into account the switch positions on the recording system pre-amplifiers and post-amplifiers. It demultiplexed the digitally encoded switch positions, converted them into poles and zeroes and gain factors, and recorded these on the data file. The program had editing features that allowed corrections to any misreading of the switch positions caused by hardware malfunction. The program was designed to operate with the BMR-produced post-amplifiers, the original Geotronics pre-amplifiers and two different sets of Geotronics coils. These are sets 1014/1015/1016 and 1005/1006/1007. Programs MAGTL and SWTCH are labelled as Layer 1 software on Program Flowchart No. 1 (Fig 1). They simply interfaced the user to the computer acquisition system.

The second layer of software, Layer 2 (Fig 1), consisted of programs MTTSTT and T2. These programs were concerned with the data acquisition itself though they did not need to handle the nitty-gritty details of word and bit manipulation. Program T2 with the help of Layer 3 software took data direct from the analogue-to-digital (A-to-D) converter, directed it to the digital-to-analogue (D-to-A) converter and also buffered it onto the disc. Two buffers (labelled respectively ABUF and BBUF, each of 640 words) received the data continuously from the Phoenix analogue-to-digital converter. This was done in a ping-pong fashion, ABUF filled up first, then BBUF and then back to ABUF. This cyclic buffering will be described further in the Layer 3 documentation.

Program T2 checked the state of the buffers and as each became full, put the contents onto disc in consecutive fashion. This program checked for certain error conditions, one of which was labelled A-to-D sink error. This could occur if the disc for some reason was not available and the buffer currently being transferred to the disc started to be refilled from the A-to-D converter; error messages were printed on the system console.

Program MTSTT had two basic functions. The first function was to force a choice of the second direct memory access (DMA) channel, DMA channel 7, for the Phoenix A-to-D converter. This left channel 6 free for the disc. If this was not done, then disc overrun would occur. The second function of MTSTT was to initiate the Phoenix A-to-D conversion and the cyclic buffering. An EXEC call to the Phoneix A-to-D converter, which was logical unit No. 11 initiated, the A-to-D conversion. This request would normally never terminate because the cyclic buffering continued indefinitely. However, if this request did finish, it would be noticed in MTSTT and the error message "high speed A-to-D termination" would issue. This could happen if, for instance one of the time-code signals from the time-base generator failed, so that the DMA switching did not take place.

The third layer of software consisted of three drivers, two of which were privileged; firstly DVR57, the Phoenix A-to-D converter driver, then the two privileged drivers. The (DVP57) was the DMA synchronising driver which handled the cyclic buffering for DVR57. It received timing signals externally from the time base generator and these timing signals caused a switch of the DMA buffer address and buffer length at the end of each buffer. This had the effect of reinitialising DMA each time one buffer became full. The third driver DVR61 was also a privileged driver; this took data from the cyclic buffers which were filled up by DVR57 and DVP57, demultiplexed the data and transferred than to the display device, via the 8 channel D-to-A converter (BMR board). Like DVP57, DVR61 received its timing signals from the time base generator board; again these signals came in externally in the form of interrupts. The data acquisition rate of the Phoenix A-to-D converter was set at 1 kilohertz, i.e. a digitising interval of 1 millisecond. This was regardless of the actual digitising interval requested by the user, so that only selected points were placed in the buffer. For instance, if a 10-millisecond digitising interval was desired then data were still digitised at the 1 kilohertz rate however only every 10th reading was placed in the buffer. DVR61 operated 10 times slower, i.e. at 100 hertz or a 10 millisecond timing interval.

$ JP57 (Fig. 4) was a standard driver trap cell routine which went with DVR57 to provide base page addressing; similarily $ JP61 was a trap cell routine that went with DVR61 to provide the same information.

There are a few general observations to be made about the MT data acquisition software. 1) There was a limitation to the maximum digitising interval of about 6 seconds. This limitation occurred because of the 16-bit internal word size; however, it could be got round using a double integer word for the time. 2) The buffer sizes, i.e. 640 words in ABUF and BBUF, were carefully chosen to cater for worst case disc access times. 3) Programs in the Layer 1 software, i.e. MAGTL and SWTCH, were both foreground real-time disc resident programs running at a priority of 50. This meant that they could be swapped out. During the actual data acquisition time both of these programs were either dormant or in suspension, allowing other use of foreground disc resident area. 4) Programs MTSST and T2 were both background memory resident and ran at priority 1 and 2 respectively. By being memory resident the background disc resident area was free for other uses, e.g. processing and interpretation of data. 5) There were other pieces of software that went to making up the software system for the MT data acquisition, e.g. the driver for the zeta potter, which was DVR10.

DETAILED DESCRIPTION OF MAGTL

Program MAGTL firstly printed a header message on the system console. It then started the analogue-to-digital and digital-to-analogue conversion procedure by running program T2. At this stage it was checked that the Phoenix was converting data and the display device was outputting data. The program then allocated working disc storage. These were global tracks and several contiguous tracks were allocated. This comfortably allowed the recording of 8192 sectors of information where each sector contained 5 digitised points. System disc tracks were used in preference to file manager disc tracks because the global disc tracks would always start at the beginning of a track. This could not be guaranteed with the file manager tracks and could cause problems with disc accesses if the file manager tracks were used. This meant that after the recording of each set of data it was necessary to transfer from the global tracks into the file-manager file itself. After allocating the scratch disc base, the program asked for the starting file number. This would normally be a number from zero to 9999; once the program started recording data it simply incremented the file number given. The user had the option of selecting his own file number to avoid conflict with previously recorded data. Facilities existed for setting the digital-to-analogue converter output gain in the software; however, the next stage in MAGTL was to set this gain to 1. The program next asked you whether you would like to process your data on line. There was the facility in MAGTL to schedule the processing program directly as each file was digitised; however, this particular facility was normally only of use when recording data that took a long time to acquire, i.e. the time taken for the acquisition exceeded the time taken to process the data. MAGTL then went ahead to gather further information for the MT data file header. The user was prompted for the following information; the basic header which would usually include a brief description of the site, e.g. Kangaroo Creek, then the azimuth, the angle of the Ex wire measured clockwise from 0°, e.g. 36°. Next it asked for the EX wirelength, then the EY wirelength; these were stored in the header. Then it asked for the number of digitised points; this had to be a binary number 512, 1024, 2048, 4096, or 8192. At this point the program calculated the number of blocks required in the eventual file manager file. The program then asked how many files the operator would like to record. This gave him the option to record automatically any number of files. At this stage the user was asked whether he would like the computer to read the various switch settings and put the pole/zero information into the header. This question was omitted the first time so that the user was forced to read the switches the first time the program was run. This ensured that there was always pole-zero information recorded in the data file. The user was asked to type the frequency band and then program SWTCH was scheduled.(SWTCH will be described in detail in the next section). MAGTL next printed out the current file number and then suspended itself awaiting operator intervention to acquire data; when the user typed"GO" a binary file type 1, of the correct number of blocks was created with security code 50 on disc cartridge 45. The digital-to-analogue gain was reset to 1; the system, time and date were obtained and written into the file header. The start time was printed on the system console for later reference. The header and pole/zero information from the global scratch tracks were then written into the file manager file and the data acquisition commenced. At the end of the data acquisition period there was a short delay while recorded data were transferred from global disc tracks to the file-manager file. At this stage program MAGTL performed certain housekeeping functions, such as checking to see if there were any more files of data to be recorded automatically, and checking if processing of data was required. Various shortcuts were provided so that the user did not need to read the poles and zeroes again if he didn't want to, and the same with the other header information. At any stage while MAGTL was running the user could, in response to a question from the program, type "control bell bell". This allowed the user to go back to the start in case he had made a mistake. "Control bell T" (for terminate) allowed him to get out of MAGTL completely.

In use, MAGTL tended to be self explanatory; it prompted you with questions and if you answered incorrectly then it would reprompt you. The file header for each MT file comprised 128 words of which only 43 words were used. The format for the header was as follows: the first 13 words contained the basic alphanumeric header information about the site. Then followed AZ and the azimuth two word numerical field indicating the azimuth in degrees. Then there were seven words containing the band of interest, for instance, 2 to 20 Hertz; then the time and date followed by the file number of the format MTNNNN where NNNN was a number from 0 to 9999; then the EX wirelength (one word binary) the EY wirelength (one word binary), the digitising interval (one word binary), then the number of points acquired in the file (one word binary). The following 85 words were not used. The pole/zero information occupied the next 512 words in the file following the header, i.e. 8 sectors.

DETAILED DESCRIPTION OF PROGRAM SWTCH

Program SWTCH picked up two parameters when scheduled from program MAGTL; the first of these was the disc logical unit, the second parameter was the scratch track. These parameters were required to define the global scratch area used by MAGTL for the data acquisition. Program SWTCH first asked for the coils in use; these could be some combination of 1014, 1015, 1016, 1005, 1006, 1007. For instance, it was possible, though not recommended, to use a set of coils 1014, 1016 and 1006. The program prompted for each coil and checked the legality of the reply; it obtained from a table, the poles and zeroes and relevant gain factors and put these into a buffer; the program then read and demultiplexed all gains, low pass and high pass filters and other switch settings from the post-amplifiers and pre-amplifiers. It formatted these in a readable way and displayed the result on the system teletype; it asked the user if they were correct, and gave the user the opportunity to modify them by editing or changing switch positions; again the prompts and editing sequences requested were self explanatory. Poles, zeroes and gains were stored in floating point format, i.e. internal HP floating point format, and the table at the end of the program gave a comprehensive listing of these with some brief comments to indicate what the poles and zeroes belong to. The poles, zeroes and gains were obtained in their polar form from the Geotronics manual and the poles and zeroes of the BMR equipment were derived by Mr Ben Liu of the D & D Group. Access to the table of poles, zeroes and gains was via an indirect table lookup approach. Insertion losses had to be taken into account for various notch and low pass filters and these were not really described adequately in the Geotronics manuals. Again dynamic code modification was used in this program. The pole, zero information was stored by program SWTCH via buffer memory directly on to the global disc tracks to enable easy access by program MAGTL. The data acquisition system accessed the switch settings on the pre-amplifiers and post-amplifiers so infrequently that it was considered a waste of memory to incorporate special drivers to do this; therefore program SWTCH accessed the switch settings directly via the multiplexer by means of calls to the privileged routines, $ LIBR and $ LIBX. An example of this was to be found in the internal subroutine GW in program SWTCH. The format for the poles, zeroes and gains produced by program SWTCH could be found from a detailed examination of this program. It should be noticed that index instructions were used in this program thereby prohibiting the use of this software by anything other than HP's latest 1000 series computers, e.g. the software will not run on HP2100, 2116 and other similar computers. Furthermore the software was written specifically for the combination of Bureau built post-amplifiers and Geotronics pre-amplifiers. The software was not compatible with the Geotronics post-amplifiers as was an earlier version of the software which ran under BCS (basic control system.)

DETAILED DESCRIPTION OF PROGRAM MTSTT

As previously mentioned this program had two main functions, firstly it forced a choice of direct memory access (DMA) channel 6. This was done in the first part of the program simply by checking word 1654 octal to see if DMA channel 6 was in use. If it was not in use then a dummy word was placed in this position by means of the privileged library routines $ LIBR, and $ LIBX. If DMA 6 was in use the program simply looped until DMA 6 became free. Once DMA channel 6 was in use and effectively allocated to this program by means of the dummy word, which is decimal 1280, a two buffer read was initiated on the Phoenix A-to-D converter via logical unit 11. An exec call was passed after the address of the double cyclic buffer and the length of the two buffers which was decimal 1280. Once the analogue-to-digital conversion process started this call should never be completed; it could only be completed if the DMA word count were exhausted and this could only occur in one of two ways: firstly if the time base signal for reinitiallising DMA failed or secondly if there was some other hardware fault in the machine. If this call was completed then the error message "high-speed A-to-D termination" was printed on the system console; if this message occurred then the user needed to take some remedial action.

DETAILED DESCRIPTION OF DVR57

DVR57 was the Phoenix analogue-to-digital conversion driver. It was written for RTE2 and 3 systems, as far as possible as a standard driver.

The only deviation from normal was the inclusion of an extra entry point labelled $$SVF. This was the entry point to a small panel routine which reset the DMA buffer address and the DMA length. When this driver was called via its entry point the driver checked to see if DMA 6 contained the special word placed there by program MTSTT, which was decimal 1280. If it did contain this word then the driver cleared it, thus making DMA channel 6 available to other users. It was essential that this happen so that DVR32, the disc driver, could have access to DMA channel 6. Inclusion of this piece of code plus the special code at entry point $SPF did not in any way preclude the driver from being used by the normal user who did not want to make use of the extra facilities. The driver provided capabilities for random or sequential scanning and also for external or internal clocks. It should be noted that all the facilities of this driver were not necessarily used by the MT acquisition system, therefore not all the facilities available in the driver were fully tested. The documentation of DRV57 was quite comprehensive; standard documentation for this type of driver could be found in HP's RTE descriptions and information peculiar to the Phoenix A-to-D converter could be found in the documentation for that instrument.

DETAILED DESCRIPTION OF DRIVER DVP57

Program DVP57 was a privileged high-speed processor that worked in conjunction with the standard driver, DVR57, for the Phoenix A-to-D cpmverter. There were three calling sequences which were: read, write and control. The driver was privileged and it allowed high-speed cyclic buffering. This driver was interrupt-driven at a 128 millisecond rate via an external line from the time base generator. One set of five readings or one vector was digitised by DVR57 each millisecond. Each buffer filled up in 128 msec and at this time a hardware interrupt occured to DVP57 which then switched the buffer address and DMA buffer length so as to fill up the next buffer. The 'control' request to this driver was used to stop the cyclic buffering at the next interrupt. The 'read' request, each time it was satisfied, returned or indicated that one buffer of information had been completed and the 'write' request was simply a synchronising request that started the cyclic buffering process. Various error conditions such as overrun were checked in the driver and an error return code was returned to the user. It should be noted that the interrupt response time of this program had to be less than 1 millisecond, which was the time interval in between taking readings used by DVR57. If this 1 millisecond in response time was not met then the data integrity would be suspect and the timing incorrect.

DETAILED DESCRIPTION OF DVR61

DVR61 was the privileged D/A driver to the BMR 8-channel multiplexed D/A converter board. This driver was interrupt driven from a 100 Hertz signal externally derived from the time-base generator card. It ran synchronously with the DVR57. Every 10th running buffered by DVR57 was buffered out to the D-to-A card via DVR61. DVR61 also provided the necessary demultiplexing of the 5 channels of data. This driver was written to do a number of things. Firstly it could run synchronously with DVR57 on a continuous basis. It could be used asynchronously in an independent environment to simply output the number of channels of digital-to-analogue conversion data. The driver could be called to several optional parameters; one of these allowed you to output from 1 to 8 channels of data; the second parameter allowed variable spacing between outputs, in other words the driver could be clocked at a rate different to 100 Hertz if desired. In the MT system it was usually called with a spacing factor of 10 which meant that each 10th digitised point or set of points was output from the driver. The driver could also handle variable buffer lengths either in the synchronous or asynchronous mode. The driver was called with a write sequence, which was as indicated in the internal documentation, in order to output D-to-A points. The control request cleared the driver from further output and reinitialised it. This latter-mentioned function was sometimes useful if the synchronisation was disturbed between DVR61 and DVR57 when running in the sychronous mode. The driver had a special calling sequence, using the control request to set the gain of the D-to-A process. Gain factors were calculated in the software; they applied only to integer numbers and they applied to all channels simultaneously.

MT DATA PROCESSING (STAGES 1 AND 2)

The Mt data processing had historically been divided into two parts. Initially this was because of the restraints of available memory but with the availability of more memory in the RTE2 system it would have been possible to put the two parts together. However, Stage 1 and Stage 2 processing tended to form distancing tendencies in themselves due to the fact that the initial stages of the processing involved large amounts of number crunching with very little or no output apart from standard output data files. Stage 2 of the processing involved a fair amount of operator interaction. It was therefore convenient if these two processes could continue simultaneously in the computer. The most desirable situation was for the number crunching program to reside in memory in the computer, using up all spare CPU time, while a high-priority stage 2 processing program produced the rotated-tensor resistivities and other similar results which are more human oriented and involve considerably more output but not too much in the way of CPU time.

Stage 1 of the MT data processing, demultiplexed the raw data which was stored in the MT files produced by the data acquisition system. It detrended and removed the mean from the raw data; it then transformed the data from the time domain into the frequency domain using the Fast Fourier Transform algorithm.

The frequency domain data were then screened for noise and corrected for the inverse transfer functions of the pre-amplifiers, coils and post-amplifiers involved in the data acquisition progress. This was carried out by means of the pole, zero and gain corrections that were stored in the header record of each file recorded by program MAGTL in the data acquisition system. The large amount of data in the corrected transforms in the frequency domain was reduced by cross and auto power spectral averaging. Cross and auto power spectra were generated from all combinations of the five corrected spectra and then were averaged on a logarithmic basis. These averaged results were stored on an output file along with the copy of the five corrected raw transforms.

Stage 1 of this processing was basically one segmented program and consisted of program FFOUR. This was the main segment and it had a number of subroutines, which were as follows:

1) Cosin; this routine formed a sine-cos lookup table and speeded up the Fast Fourier Transform algorithm considerably.

2) Fpasc; this subroutine performed floating point to ASCII conversion. Its use meant that the formatter need not be included with the program, thus saving valuable memory space.

Subroutine CONTL, shown as the main segment subroutine on the MT data processing program flow chart No. 2 (Fig. 5), looked after the job of scheduling various segment subroutines. Normal Hewlett Packard Fortran did not allow Fortran programs to call Fortran segments and then return to the main program. The subroutine CONTL included in the system circumvented that problem. Program FFOUR called various entry points; there were five of them in subroutine CONTL. Subroutine CONTL looked after the scheduling of the relevant segments, checking to see whether or not they were in memory, returning from the segment and then returning to FFOUR. The segment suboutines are briefly described in program flow chart No. 2 (Fig. 5). Segments were used to save memory space. Approximately three times the memory space actually used would be required if these segments were not used.

In an RTE4 system with an F series processor this whole processing stage could be one program. A brief description of the segments is as follows:

EXAMN. This segment allowed the user to examine transforms, and compute ratios of transforms, and was generally used during the testing phases of, for instance, new post-amplifiers.

NORM1. This segment was used to normalise spectra. Spectra were normalised on the basis of EX, both for phase and amplitude. At the same time the data which apparently had bad signal-to-noise ratios were removed from the spectrum.

XFQRM. This segment carried out the Fast Fourier Transform, and as mentioned earlier used subroutine cosin, the sin-cos lookup table, to speed up the implementation.

POZ. This segment carried out the inverse filter corrections to the transformed data using the pole/zero values stored in the header of the data file.

PWRS. This segment computed cross and auto power spectra and called subroutine AVRGE which averaged spectra and stored them on the output disc file.

FACILITIES OF FFOUR

FFOUR has a number of useful facilities. When a magnetotelluric site is set up problems can occur, e.g. connections to the coils may be reversed, connections to the electrodes may be reversed or the amplifiers may be incorrectly connected together. It would be unfortunate if data collected while these irregularities existed could not be used, so facilities were provided in FFOUR to invert any channel or set of channels which may have been wired up incorrectly.

Facilities also existed for changing the data channel order. For instance data were normally multiplexed onto the disc in the following order EX, EY, HZ, HX, HY. It was possible in the program to specify an alternate channel order, e.g. EX, HZ, HX, EY, HY. In like manner the poles and zeroes recorded with a particular data channel could become associated with the wrong channel. Facilities existed in the software so that, for instance, the poles and zeroes recorded with channel EX could be applied to any other channel and so on with all the other recorded channels. The correction of the transforms with the inverted transfer functions could also be negated at this software stage. This latter capability was useful, not only for test purposes, but could not be used in the future if, for instance, rubidium, caesium, or proton magnetometers were used. These would normally have a flat transfer function and the normal pole/zero sets would not apply. Facility also existed in FFOUR to print the averaged spectra on the system teletype device. The various aforementioned facilities were selected via the scheduling parameters; for program FFOUR the standard parameters were as follows:

Parameter No. 1 contained the number of the file to be processed by FFOUR. Data files produced by program MAGTL are of the form MT nnnn where nnnn was a number from 0 to 9999. Parameter 1 then was a number from 0 to 9999 and the default value was 0.

Parameter No. 2 contained the logical unit number of the print device for standard default processing; this was a positive number from 0 upwards, the default value was +1. If this logical unit number was specified as a negative number, e.g. the negative of the desired logical unit number, then program FFOUR would prompt the user with various questions, asking him whether he would like to reverse the phase of any particular channel, whether he would like to print out the averaged spectra, whether he would like to change the data channel order, whether he would like to change the poles and zeroes to be associated with particular channels, and whether or not he would like the transforms to be corrected. For instance, typing -1 as parameter 2 meant that the program responded to logical unit 1 and prompted the user with the type of dialogue just mentioned.

Parameter No. 3 contained the number of files that the user would like to process. The default value was 1. For instance, specifying the number 5 for parameter No. 3 would take the file specified in parameter No. 1, process that, and then the next four consecutive files following on from that. In the default mode all that program FFOUR printed out was the current file being processed. Program FFOUR produced a single output file but there was some resemblance to the input file; the output file name was the same as the input file name with the exception that the first letter M was replaced by F, e.g. MT 1234 would become a new file FT1234; file MT 1234 was untouched by the program. If the user, in response to prompting said that he would like to print averaged spectra in FFOUR, then he was further prompted for a file name and channel number. For the user frequency the program listed, the amplitude and phase for a particular data channel of a particular file averaged on a logarithmic basis as described earlier. This facility could be used to check the transfer functions of various amplifier filter combinations, e.g. a record of data could be processed with FFOUR and the uncorrected transform printed out. The same set of data could be processed by FFOUR with the corrections applied and again the averaged spectrum printed out. The ratio in the complex plane of the two spectra printed out, gave the transfer function or the inverse transfer function, depending on which way it was done, of the filter-amplifier combination. The above could be used to check that amplifiers and filters were performing to specification. Quite obviously, if memory space permitted (and this may well be the case in the future) this facility could be expanded; for instance it would be nice to generate the transfer function immediately and fit a polynomial to it to produce pole/zero combinations for the filters and amplifiers involved.

GENERAL COMMENTS ABOUT FFOUR, ITS SUBROUTINES AND SEGMENTS

The main problem in writing program FFOUR occurred with memory availability. Several large buffers were used for the demultiplexing and the Fast Fourier Transform routines. For these reasons there was no room left in the maximum background partition in which to use the formatter supplied by Hewlett Packard Standard Fortran I/O. All I/O was handled by program CONTL written in assembly language. Furthermore, discrete functions of the program were built into the five segments previously mentioned; this meant that it was quite easy to add additional functions to the program without increasing its size in any way, simply by adding more segments. The calls from FFOUR always passed through subroutine CONTL via subroutine entry points. Subroutine CONTL then called in the required segment if it was not already in memory and upon completing a segment returned to subroutine CONTL by means of various entry points defined E1, E2, E3, E4, E5. CONTL then returned control to program FFOUR for further processing. Subroutine FPASC written in assembler by D. Downie of BMR, provided the floating point-to-integer conversion required by the program.

Because of memory limitations program FFOUR was limited to the processing of data files containing 2048 channels or vectors each containing 5 discrete digitised data points. Data were supplied to the program in integer format, 16 bits wide in 2's complement.

After the initial demultiplexing, all calculations in this program were carried out in floating point, single precision, 32 bit. There was no requirement to expand the capability of FFOUR to process data files containing more than 2048 digitised sets of data. This was because the hardware filters designed for the BMR equipment were designed with an appropriate band width to suit a maximum of 2048 points. Obviously the more points that were transformed at a time, the wider was the band-width being viewed at a particular time. The capability of the program. could have been enhanced in a number of ways. One way would have been to increase the buffer sizes for the transform; a second way would have been to implement some fancy techniques that are now in vogue with Fast Fourier Transforms for actually splitting the transform and doing it in several parts and then recombining at the end.

In earlier Hewlett Packard machines cos and sin functions were generated entirely in the software. This took a considerable time, sometimes as much as 30 milliseconds per sin or cos. For this reason subroutine COSIN, a small Fortran routine, was written to generate a cosin-sin lookup table. In a 2048 point Fas Fourier Transform only 1023 sins or cosins are required; subroutine COSIN went through these sins or cosins once, normalised the result to make maximum use of a 16-bit integer word and stored the values in array ISINE in subroutine COSIN. When looking up the table the reverse process occurred and the results were normalised back to the correct value. This gave a precision of 4 to 5 digits for the sins and cosins which was quite sufficient for this type of FFT. Hewlett Packard's actual software sin-cos routines can in fact be quite inaccurate and for this reason the DSIN function was used in generating the look-up table. With recent changes to HP library software and the implementation of firmware sin-cos functions in the F series processor, this look-up table will become redundant. Use of the table saved approximately 70 to 80% of the sin computation time in the Fast Fourier Transform. This was a fair proportion of the actual time for the FFT.

Segment NORM1 normalised phases of each channel, based on EX which was assumed to be noise free. Any data points in a particular channel which were less than approximately 4% of the local mean were disregarded and all points in that particular vector were set to 0.

Segment subroutine XFORM carried out the Fast Fourier Transform one array at a time, i.e. it would do channel EX, then channel EY, then channel HZ, then channel HX, and finally channel HY. This therefore took five calls to XFORM. Initially the raw time-series data were detrended and the mean removed before doing the Fast Fourier Transform; however, changes were made so that the mean was removed from the transform, i.e. the average value of all the particular time-dependent values comes out at 0. However, instead of detrending the data, a Hanning window was applied. The Hanning window applied made use of the sin-cos look-up table via subroutine COSIN. A number of windows were tried on the MT data. The Hanning window gave the most accurate and best results. The transform data were corrected for the inverse transfer function of the filters and also for various gain factors after the Fourier Transform was performed; thus the final output was a true scaled representation of the input to the MT system for the particular channel concerned. It should be noted that corrections were done in the complex domain. This improved the speed of computation over the use of polar coordinates.

Segment subroutine POZ took the poles and zeroes and converted them into the correct complex form for segment subroutine XFORM.

Segment PWRS performed auto and cross power calculations and produced all combinations of auto and cross powers of the five channels. Computations of the auto and cross power spectra were then averaged on a logarithmic basis as described earlier. Subroutine AVRGE was used to average the data. This was done to produce five bands per decade of period and there was zero overlap between the bands.

STAGE 2 OF DATA PROCESSING PROGRAMS: TENSC AND TENSR

Stage 2 of the processing was unsegmented and ran in the background area of RTE2. In general the theory involved in subroutine TENSR, which was the main numerical part of stage 2 of the processing, was derived from Vozoff's, (1972) paper entitled "The magneto-telluric method in the exploration of sedimentary basins". Reference was also made to the paper of William E. Sims and F.X. Bostick Jnr. (1969), entitled "Methods of magneto-telluric analysis". The actual equations used were derived by D. Kerr following through the theory described in these two papers; in some instances errors were detected, especially in the Bostick article, and the final formulations used were believed to be free of error. They have been extensively checked using synthetic and real MT data.

Subroutine TENSR and its associated subroutines were originally written in Fortran 2 before Fortran 4 became available on HP's RTE systems. In general therefore complex arithmetic was not used and computations were performed on real and imaginary parts separately. This resulted in considerable savings in memory utilisation and CPU time and also simplified some of the complex expressions; however, in some cases the expressions become considerably more difficult to manage algebraically.

Subroutine TENSC was the main control program for the tensor analysis. It set up the data in a form suitable for processing by subroutine TENSR. It handled the user interface and also allowed the printing out of raw data, filter settings, header information, etc. Program TENSC also allowed the user to specify various options for output data; for instance the normal output from Stage 2 of the processing was two files, one named ROTTEN and the other TIPPER. ROTTEN contained normally; rotated tensor data; however, it could also contain unrotated tensor information, or Cagniard resistivities if these were desired. Other information such as skew, coherency, phase, resistivity, were included as well as the period range and the actual period at which the values were taken. Various printer outputs could be generated, or entirely suppressed if desired. The minimum output from the program was the header of each file as it was processed. Options were available to output all coherencies, power spectra, Cagniard resistivities, rotated tensors, unrotated tensors and tipper information.

Program TENSC had as its input the FT nnnn file generated by program FFOUR. If the program was run without parameters it would process one file with various default settings. There were four scheduling parameters for program TENSC and these were as follows:

Parameter 1 was the first file to be processed, this was a number ranging from 0 to 9999; the default value was 0.

Parameter 2 was the number of files to process; the default was one file.

Parameter 3 was a control word which would normally be specified in octal, e.g. 17654B. Each of the 16 bits in this control word had a specific meaning and was used to control various functions in subroutine TENSR. Functions of the various bits in this word are described later on.

Parameter 4 was the output logical unit on which the output from TENSR was to appear; default value is 6.

The values and functions of the various bits detailed in Parameter 3 above were as follows:

Bit 1 controlled the printout of gains, poles, and zeroes. Setting of this bit would output poles and zeroes in their complex form as well as gains.

Bit 2 allowed the output of the real and imaginary parts of transforms for each of the 5 channels. These were not averaged and a considerably quantity of data was produced. Setting this bit 'on' produced output.

Bit 3.Setting this bit 'on' produced a printout of the average spectra.

Bit 4. This bit was not used.

Bit 5. Setting this bit enabled printing out of the raw coherencies.

Bit 6. Setting this bit with bit 7 clear enabled TENSR to output Cagniard resistivities and coherencies to file ROTTEN. Bit 6 was overriden by bit 7. In other words if bit 7 is set then bit 6 may be either set or clear; it has no function in this latter case.

Bit 7. Setting this bit 'on', enabled TENSR to produce a file of rotated tensor resistivities, rotation angles, phase angles and frequencies, again in the file ROTTEN. Setting of this bit overrides any set or clear of bit 6. This bit had to be clear to enable bit 6 to be set.

Bit 8. Setting this bit enabled the printout of resistivities, phases and appropriate coherencies for each particular frequency or period.

Bit 9. Setting this bit 'on' inhibited the printing out of rotated tensor results. Thus clearing of this but enabled the printing out of rotated of rotated tensor results. This was the default situation.

Bit 10. Setting this bit 'on' produced a printout of the unrotated tensor analysis.

Bit 11. This bit was not implemented.

Bit 12. Setting this bit 'on', inhibited output of tipping vector results. The default setting of this bit, equal to '0', produced tipper information.

It should be noted that in program TENSR that there was a difference between the coherencies calculated for the unrotated tensor analysis and those calculated for the rotated tensors. The rotated tensors were used to generate what are called the rotated coherency or the coherency in the rotated frame of reference in the rotated tensor analysis. It was found that there is no documentation available in the literature for the derivation of these equations. However, it was basically straightforward algebraic manipulation to obtain them. The equations were derived by Kerr and Jupp working together in 1977 at BMR. No record of this work was produced.

Most of the tipping vector analysis will be found in Vozoff (1972) mentioned earlier. There were a few slight changes to the equations shown in that paper. Definitions of tipper magnitude and tipper phase were modified slightly to reflect more realistically real earth situations. Reference to this is contained in Jupp and Vozoff (1976).

Various subroutines were required for subroutine TENSR. Most of these were included in the source code of TENSR and could be found in the appropriate file. An external routine, CMUL was required. This performed complex multiplications using real and imaginary parts. Other external subroutines not supplied by Hewlett Packard are as follows:

Subroutine IMPED calculated the impedance from the cross and auto power spectra.

Subroutine PRED calculated what is called the predictability of a particular field component in the analysis. For instance, it is possible to predict an E field in the X direction from the knowledge of the E field in the Y direction and the two horizontal magnetic components. This predicted component could be compared with the actual measured component to give an idea of the noise or lack of it in the measured MT components. Subroutine PTRF calculated predictability of EX and EY components for both the rotated and unrotated cases. The unrotated case was simply a more specific case of the rotated case with beta set to 0.

Subroutine CALC. This subroutine simply contained frequently used code for the PRED routine. It was used to save memory.

Subroutine TRES calculated tensor resistivities and phases from the tensor impedance components.

Subroutine ATAN3 computed arctan for a given numerator and denominator for all four quadrants. It also checked for near 0 denominators.

File output formats from TENSR were as described by R.F. Moore (1976, 1977). Data at the output stage of this program were frequently plotted and then put into screening, averaging and various other data manipulation programs. There were various versions of the TENSR subroutine available; the one described was in file TENSRS on the magnetotelluric cartridge 45. There were basically two variations on the standard program; one variation simply allowed different formatting of the output for use on narrow 80 column printers such as the NCR260 or Texas thermal printers. The second variation contained various experimental code. This version could be found under file NEWTEN on the MT master cartridge. The impedances Zxx and Zxy, Zyx and Zyy could be derived in any of up to 6 different ways. A reference to this can be found in Vozoff (1972, p. 111), or in Sims and Bostick (1969) paper. All six methods of solving these equations were tried with the MT system and typical data; it was found in practice that the set of equations described in Vozoff (1972) was by far the best. This set of equations was the set used in the standard version of TENSR. Code for the method of GOUBAU, GAMBLE, CLARKE from Berkerley University was shown in the version NEWTEN; this technique expressed auto powers in terms of cross powers, thus eliminating all auto powers from the derivation of the impedances. One of the problems with using auto power is that any errors are cumulative,whereas for cross powers the errors tend to be noncumulative and cancel out. Thus this method should provide an unbiassed estimate of magnetotelluric impedance. This was tried in practice; however, it was found that typical BMR results were not appreciably biassed and the resulting improvement in bias was not justified because the noise increased several times.

REFERENCES

GOUBAU, W.M., GAMBLE, T.D., & CLARKE, J., 1978 - Magnetotelluric data analysis: removal of bias. Geophysics 43 (6), 1157-1166.

JUPP, D.L.B., & VOZOFF, K., 1976 - Discussion on "The magnetotelluric method in the exploration of sedimentary basins" by K. Vozoff. Geophysics No. 41 (2), 325-328.

MOORE, R.F., 1976 - Graphical presentation of magneto-telluric data. Bureau of Mineral Resources, Australia, Record 1976/97 (unpublished).

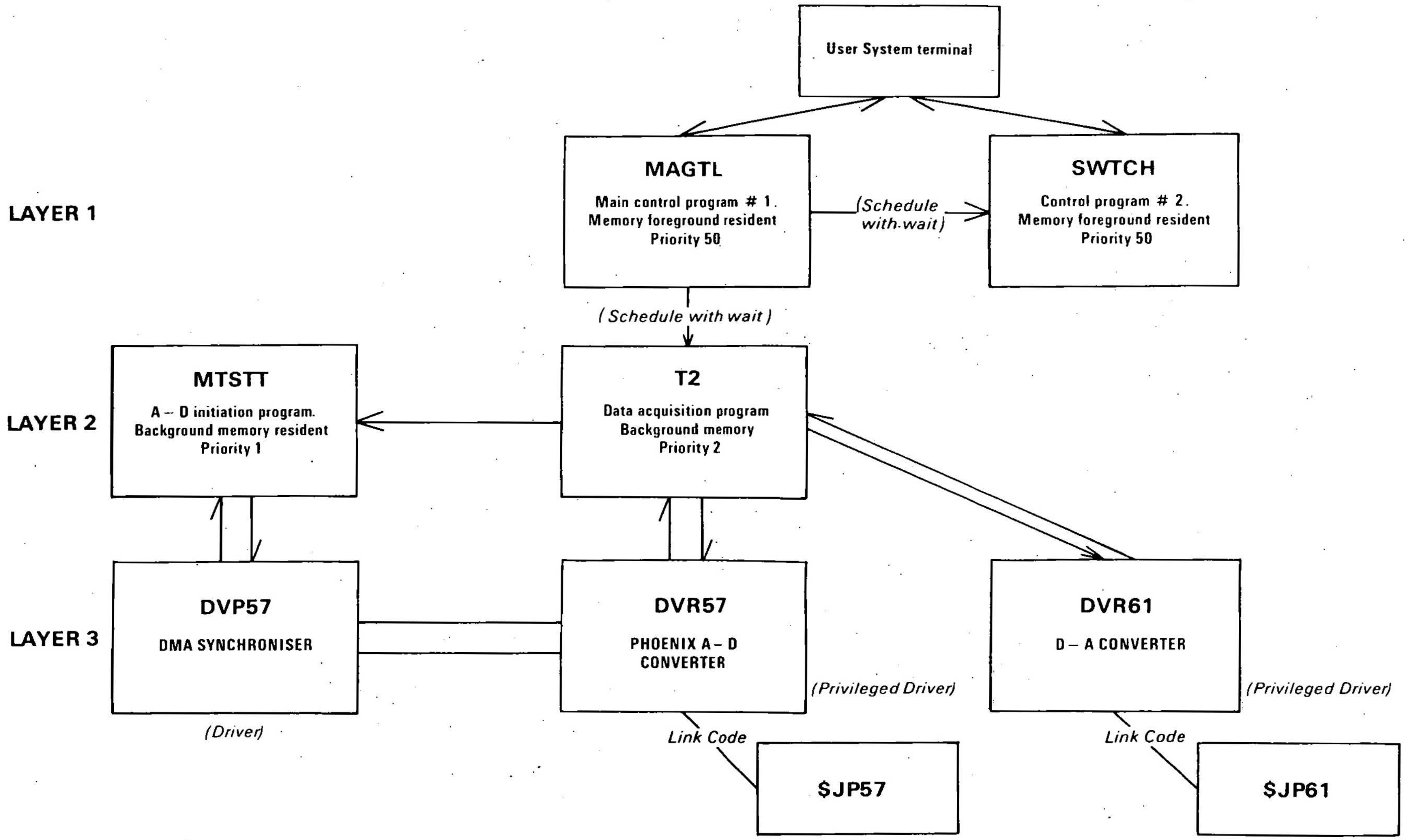
MOORE, R.F., 1977 - Screening and averaging magneto-telluric data prior to one-dimensional inversion. Bureau of Mineral Resources, Australia Record 1977/8 (unpublished).

SIMS, W.E., & BOSTICK, F.X., (Jr), 1969 - Methods of magnetotelluric analysis. Technical Report No. 58. Electronics Research Centre, University of Texas, at Austin.

VOZOFF, K., 1972 - The magnetotelluric method in the exploration of sedimentary basins. Geophysics, 37 (1), 98-114.

Fig 1 PROGRAM FLOWCHART No 1 — DATA ACQUISITION

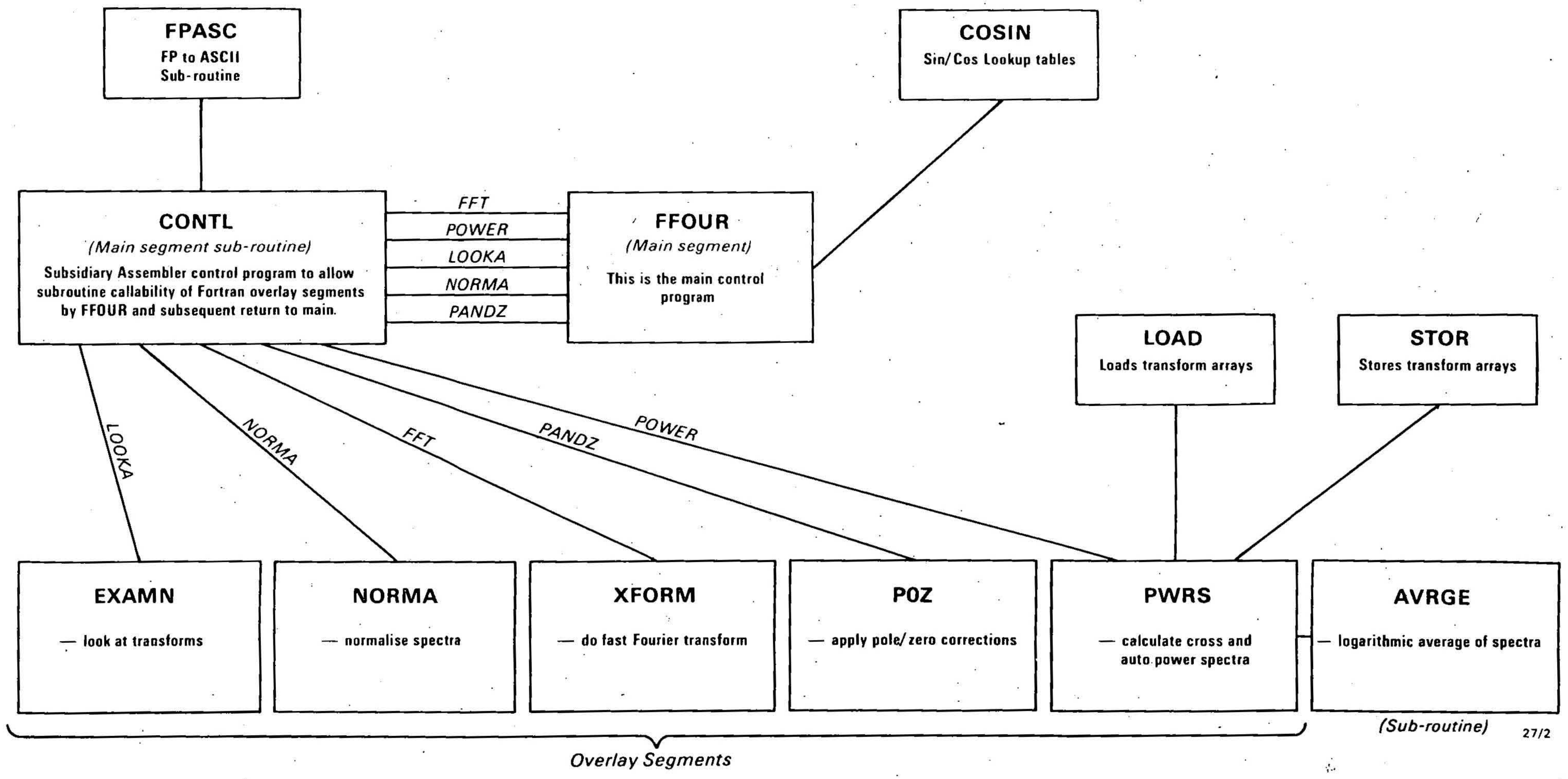
(Acquires MT data, filter and gain information and stores it on random access disc files)



27/1

Fig 2 PROGRAM FLOWCHART No 2 DATA PROCESSING Stage 1

(Produces corrected auto and cross power spectra)

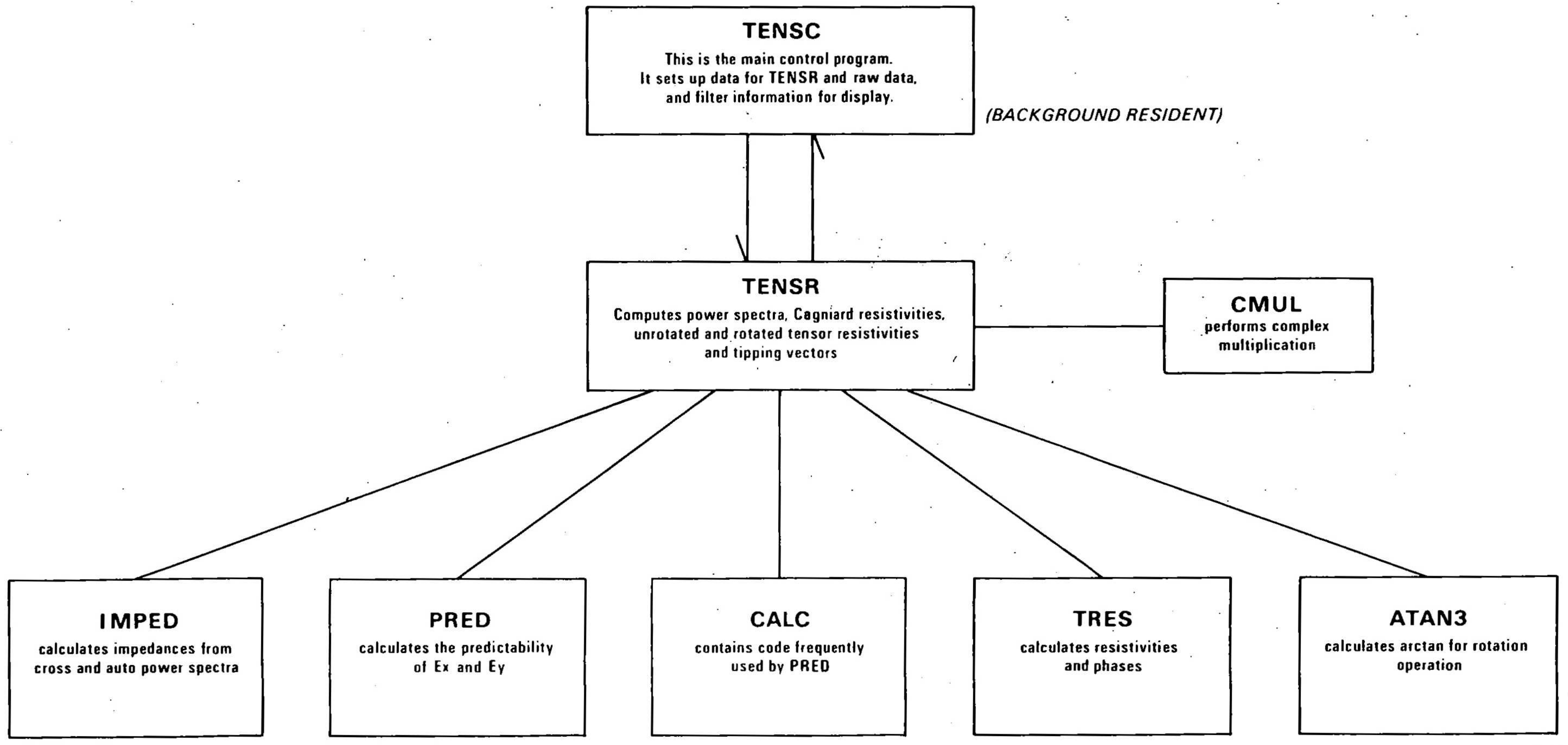


FPASC
FP to ASCII Sub-routine
COSIN
Sin/Cos Lookup tables
CONTL
(Main segment sub-routine)
Subsidiary Assembler control program to allow subroutine callability of Fortran overlay segments by FFOUR and subsequent return to main.
FFT
POWER
LOOKA
NORMA
PANDZ
FFOUR
(Main segment)
This is the main control program
LOAD
Loads transform arrays
STOR
Stores transform arrays
LOOKA
NORMA
FFT
PANDZ
POWER
EXAMN
— look at transforms
NORMA
— normalise spectra
XFORM
— do fast Fourier transform
POZ
— apply pole/zero corrections
PWRS
— calculate cross and auto power spectra
AVRGE
— logarithmic average of spectra
(Sub-routine)
Overlay Segments

27/2

Fig 3 PROGRAM FLOWCHART No 3 —— DATA PROCESSING Stage 2

(Produces Cagniard resistivities, unrotated and rotated tensor resistivities and tipping vector outputs)


TENSC
This is the main control program. It sets up data for TENSR and raw data, and filter information for display.
(BACKGROUND RESIDENT)
TENSR
Computes power spectra, Cagniard resistivities, unrotated and rotated tensor resistivities and tipping vectors
CMUL
performs complex multiplication
IMPED
calculates impedances from cross and auto power spectra
PRED
calculates the predictability of Ex and Ey
CALC
contains code frequently used by PRED
TRES
calculates resistivities and phases
ATAN3
calculates arctan for rotation operation

27/3