

Some pages of this thesis may have been removed for copyright restrictions.

If you have discovered material in Aston Research Explorer which is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please read our [Takedown policy](#) and contact the service immediately (openaccess@aston.ac.uk)

ROBERT ALAN CHARD

THE DEVELOPMENT OF A COMPUTER BASED
DATA ACQUISITION AND PROCESSING
SYSTEM.

MASTER OF PHILOSOPHY

OCTOBER 1973

4 dec 73 167675

THESIS
001.64
CHA

ML

SUMMARY

This thesis is a description of the development and use of BASOON an interactive, on-line computer software package. BASOON allows experimental equipment to be accessed by a digital computer via a data logger. All user programming is in BASIC using a series of subroutines to perform the data acquisition.

The relevant literature is surveyed and the equipment fully described. The necessary programming languages are described in sufficient detail to allow readers with limited computer knowledge to understand the system. A brief account of the development work is presented, followed by a summary of the system. A detailed description of the BASOON subroutines and the linking of the various programming languages used is presented to enable the system to be fully understood and implemented on similar equipment elsewhere. Program examples including test routines are included and discussed with recommendations to users.

CONTENTS

- (i) Introduction
 - 1. Literature Survey
 - 2. Equipment -
 - 2.1. The Double Effect Evaporator
 - 2.2. The Honeywell H-316 Digital Computer
 - 2.3. The MDP - 200 Data Logger
 - 3. Programming Languages
 - 3.1. BASIC
 - 3.2. FORTRAN
 - 3.3. DAP - 16 Language - ASSEMBLER
 - 3.4. Relevant on-line Programming techniques
 - 3.5. Program Linking
 - 3.6. Interrupts
 - 3.7. Error Detection
 - 4.1. The Continuous, Steerable Scanning Program (CSP)
 - 4.2. Operation of the Command Interpreter Part of Program
 - 5.1. BASOON - Introduction and Development
 - 5.2. The BASOON Subroutines - General
 - 5.3. The BASOON Subroutines - Detailed Description
 - 5.4. BASOON Initialisation
 - 6. The Use of the BASOON System for Data Acquisition
 - 7.1. Results and Discussion
 - 7.2. The Demonstration Program
 - 7.3. The Distribution Analysis Program
 - 7.4. The Accuracy of Incoming Data
 - 8. Conclusion
 - APPENDIX 1. BASOON ON-LINE FROM BASIC (Instruction Manual)
 - APPENDIX 2. Honeywell H-316 Instruction Complement
 - APPENDIX 3. Preparation of a DAP-16 Program
 - APPENDIX 4. FORTRAN DAP-16 Communication
 - APPENDIX 5. BASOON Programs
 - APPENDIX 6. Assembly Listing of Modifications to BASIC Compiler to Revise Initialisation
- 9. Bibliography.

INTRODUCTION

Computers are now accepted as part of everyday life in most research applications. However, one of the most tedious tasks is often data preparation when there are perhaps thousands of data points to be processed. For the study of some systems, for example dynamic problems, the data must be sampled at specific points in time, and there can be difficulties in physically reading the information at the required times, for example, thermometers and pressure gauges in several places. Data Loggers are available which will log large amounts of data, which is often punched on tape for computer processing. This thesis is concerned with the direct linking of a digital computer to a data logger, to provide on-line capability. The system is made completely flexible by using the interactive high level language BASIC for programming, although FORTRAN can be used.

The name BASOON for the package developed, is made up from BASIC and ON-LINE, suggesting both features. The development of BASOON may be traced through a series of steps, beginning at the first data logging program, written entirely in ASSEMBLER (machine code), which was capable of transmitting a value (of one channel of facility) from the data logger into the computer. The number was still in BCD form (see text), and required conversion to binary before it could be used. This was followed by a larger program which performed more functions, but even at this stage the programming was still mainly ASSEMBLER. The first major development was to write the data logging programs as a series of self contained subroutines, calling these from a FORTRAN main program. There were severe difficulties to be overcome at this stage in the transfer of data between the different languages. However, when these were solved a set of subroutines were available which could perform the data logging functions without any direct ASSEMBLER programming. This had two disadvantages, firstly it was still necessary for the user to understand how the subroutines worked to make efficient use of them, secondly, when a mistake was made it was necessary to start correction by rewriting the FORTRAN source program. This meant re-compile-

ing and reloading into the computer, a task which requires a minimum of 25 minutes.

1.2. INTER The final development was the linking of a new set of subroutines to BASIC, which enabled the main program to be changed instantly. Throughout these stages, more sophisticated subroutines were developed, leading to an overall package for data logging and interrupt handling. Finally a comprehensive error diagnostic system has been built in, which detects errors at any programming level and gives a message on the teletype using similar mnemonics to BASIC.

1.1. SURVEY OF THE RELEVANT LITERATURE

1.2. INTERACTIVE COMPUTING

During the last few years there has been a substantial amount of published work concerning interactive computing. This covers a wide range of applications including data processing, graphic displays and literature searches. Many authors have described systems using graphic display terminals, (1,2,3,4). Feldmann (5), described one such system in which large and small chemical structures could be plotted and manipulated. This was done by a set of programs operating on a time sharing basis. A system with a similar purpose was reported by Meyer (6), but using mass spectral data to build up a 3-D colour display. A rapidly expanding area where interactive techniques find application is in searches. Heller (7), reported on an interactive program developed to search a large file of mass spectral data. This used a peak reduction method developed by Biemann (8), and enabled the file to be examined for single peaks or groups of peaks.

Another application of searches is in the literature, where rapid advances may be expected in the near future. Heller (9 & 10), described a system for this purpose.

Neilsen (11), gave details of a set of interactive programs written in FORTRAN for the teaching of chemical equilibrium and kinetics. To use these the student describes a system to the program which then calculates the equilibrium state and time development of the system. The student may then change the system and observe its behavior.

All programs examined in the literature had one thing in common to their disadvantage. This was that the user could not alter the program readily, but only the input data. Interactive FORTRAN is not mentioned, but would require a disc for storage and some kind of operating system, implying a fairly large computer. Some authors appear to be interpreting "hands on" computer usage of a developed program as interactive working

1.3. INTERFACING

According to Peddie (12), recording and reducing large amounts of data reliably can be a problem for many researches, the two most common methods of handling very large amounts of data being data loggers and dedicated computers. He then describes an interface for linking a mass spectrometer (or other equipment), to a computer terminal. The only disadvantage pointed out by the author was that the data rate was limited by the bandwidth of the telephone lines used. He did not mention what happens during periods of heavy computer usage, when response times on time shared machines can be very slow.

A very good description of the function and use of an interface is given by Mazda (13), in a very readable paper. An interface for coupling four measuring microscopes to a Minsk 2 computer was described by Ammosov (14). The device permitted real time exchange of co-ordinate information between the measuring instruments and the computer whilst measuring photographs of bubble and spark chambers. The components were mounted on standard plug in circuit boards in the computer.

1.4. ON-LINE DATA ACQUISITION

The results of a survey of industrial on-line computer users were published by Kompass (15). He found that the "average" on-line computer installation had 284 analogue inputs and 15.9 of 16 bit word storage. The most common input/output device was the teletype, with the CRT visual display in second place.

The general requirements of an on-line system have been discussed by Burke, (16), and others. Burke concluded that BASIC was the ideal language for programming a data acquisition system, although the system he described used FORTRAN as the user language.

Lord and Macleod, (17), listed their ideas of the necessary computer for on-line work as having 4 to 16 K of storage with a word length of 12 to 24 bits. There should be fast fixed point arithmetic and logic together

with a flexible input/output system. One or more direct memory access channels should be provided and a multi-level interrupt facility. They also suggested that most of the programming be done in ASSEMBLER because care was needed with the timing, but conceded that FORTRAN might be used at the start of setting up an experiment. They give a good description of program organisation.

For non computer personnel requiring a knowledge of on-line time sharing the description given by McCullough (18), is to be recommended. This includes a brief description of the hardware requirements, data flow etc., for a gas chromatography set up.

Perhaps the two fields in which there is most work published are mass spectral data collection and gas chromatography. One such system is well described by Bowen, (19), this uses an Argus 500 computer with 8 K of 24 bit words. External logic circuits are used for timing and conversion of data into computer compatible form. The number of samples taken per spectrum varies from 64 000 to 2 048 000. For fast transfer direct memory access is provided to 2 K of the store, this eliminates the need for using an interrupt system. In a later description Bowen and Fish, (20), presented details of a system for the collection of up to 100 000 samples per second, again from a mass spectrometer. This requires a minimum of 16 K storage. FORTRAN was the high level language used and a short example program is given. This indicates that a dummy call is made to a FORTRAN subroutine to allow transfer to the actual data acquisition program. Landowne's system, (21), for the collection of data from up to 8 gas chromatographs used FORTRAN for the processing, but the main bulk of the programs were written in ASSEMBLER, including the input/output and job sequencing. The experimenter entered his instructions in the form of 8 digits via a communicator box. These were transmitted to the computer when a start button was pressed. The author says, "Only in a few cases does the software check the validity of the data entered, if incorrect a red "error" light is given". This implies that experienced users would not know of mistakes until the results were obtained.

A basic form of acquisition program was described by Desiderio, (22), here, an assembler program was used to collect a lot of data and store it on disc for later off-line processing by a FORTRAN program. However the paper is primarily concerned with the data and its use. This criticism cannot however be made of Bliselius, (23), who gives a very useful description of a PDP 9 installation used for on-line calculation of mean values, variances and amplitudes. Programming is in ASSEMBLER, and all arithmetic done in unsigned integer representation. Program examples and storage allocations are given, with diagrams of timing intervals and program organisation.

An impressive system is installed at the University of North Carolina, Chemistry Department, (24). Each input station is provided with 6 analogue inputs, one multiranging analogue input, 4 analogue outputs, 4 interrupt lines, one 16 bit input channel, one 16 bit output channel and teletype plug in. 26 stations are planned, and to date 10 are in operation.

1.5. MEDICAL APPLICATION

There are many published works concerning the use of on-line data acquisition, these concentrate mainly upon the medical aspects rather than the computer system. (25,26,27).

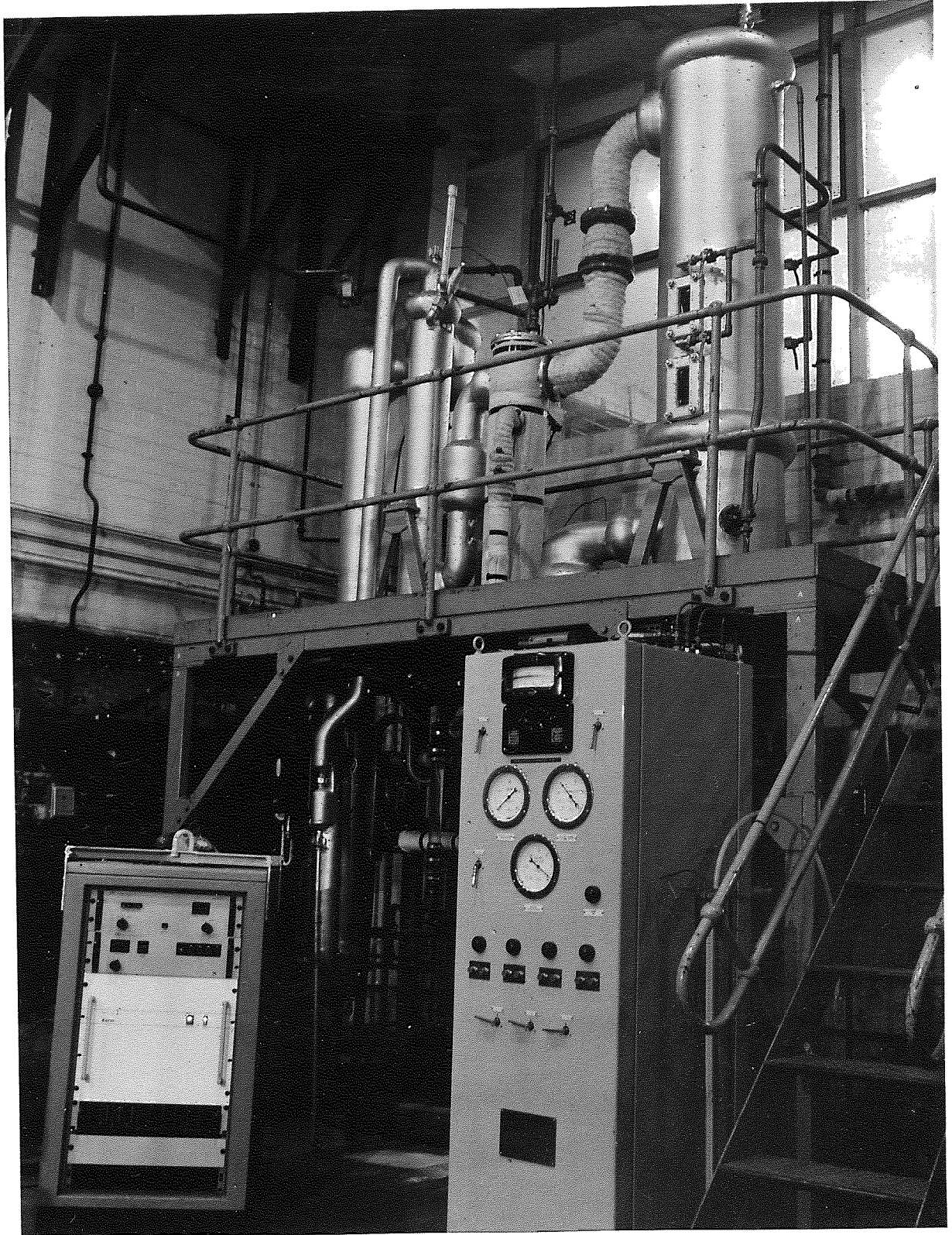
1.6. CONCLUSION

Of the systems examined all use an ASSEMBLER language in such a way that system users must either become conversant with this, or trust their programming to someone else. In the latter case there could be a difficulty of communication between persons of widely different disciplines. BASIC has been suggested as the best programming language, but so far no one has linked this to a data acquisition system.

With very large computer installations such as the one described above, one should consider the response time of the whole. Undoubtedly the necessary large operating

system programs must take up a large proportion of available core, and reduce response time considerably. Secondly it should be asked just how much of the operating system and peripheral equipment is there only to provide time sharing. In many cases would not this capital have been better spent on providing multiple dedicated computers for individual experimenters. For time sharing to be really effective it should only be implemented where there are several experiments running which produce data at fairly slow rates. The various users of a time sharing system could be considered as a mixture of gases in an enclosure, they can co-exist, but each can easily fill the space if allowed to do so.

When a computer is being used for experimental work there becomes a rapid interaction between operator and machine. This can lead to entirely new avenues of thought and fresh approaches to old problems.



PHOTOGRAPH 1

THE DOUBLE EFFECT EVAPORATOR

2.1. Equipment

The data logging system described in this thesis is connected to a laboratory size double effect evaporator. Several projects studying this equipment are in progress, so it is appropriate to describe the equipment here.

The Double Effect Evaporator (Photograph 1)

The evaporator consists of two stages, a climbing film first stage followed by a forced circulation second stage. (See Fig (1)). Feed enters from a header tank and passes through a preheater⁽²⁾, here it is heated by vapour coming from the first effect⁽¹⁾. After preheating the feed enters the bottom of the first effect⁽¹⁾, where it passes vertically upwards through a 1.125 inch (28.6mm) O.D., 16 swg (0.065 inch wall), drawn copper tube. This is heated on its outer surface by condensing steam. As the liquid rises in the tube it is heated, until at some point it begins to boil, the vapour produced forms slugs which rise up the tube. Eventually the slugs break up and a film of liquid is left on the wall, which is carried upwards by the vapour core. A mixture of vapour and boiling liquid is ejected from the top of this stage, this enters a separator⁽³⁾, which produces vapour and liquid streams. The liquid goes forward as feed to the second effect^(6,7&8). The vapour gives up some of its heat as it passes through the shellside of its preheater; it is then passed to the shellside of the second effect calandria⁽⁶⁾. In the second effect liquid is drawn from the large separator vessel⁽⁸⁾, and pumped through an external calandria⁽⁶⁾ back to the separator. The resulting vapour is condensed in a vertical shell and tube condenser⁽⁴⁾, the condensate being removed by the vacuum pump⁽⁵⁾. Any vapour from the first effect which remains uncondensed after passing through both preheater and second effect calandria is also passed to the condenser.

2.1.1. Instrumentation

A diagram of the instrumentation is given in Fig (2). There are eight thermocouple, four pressure transducers,

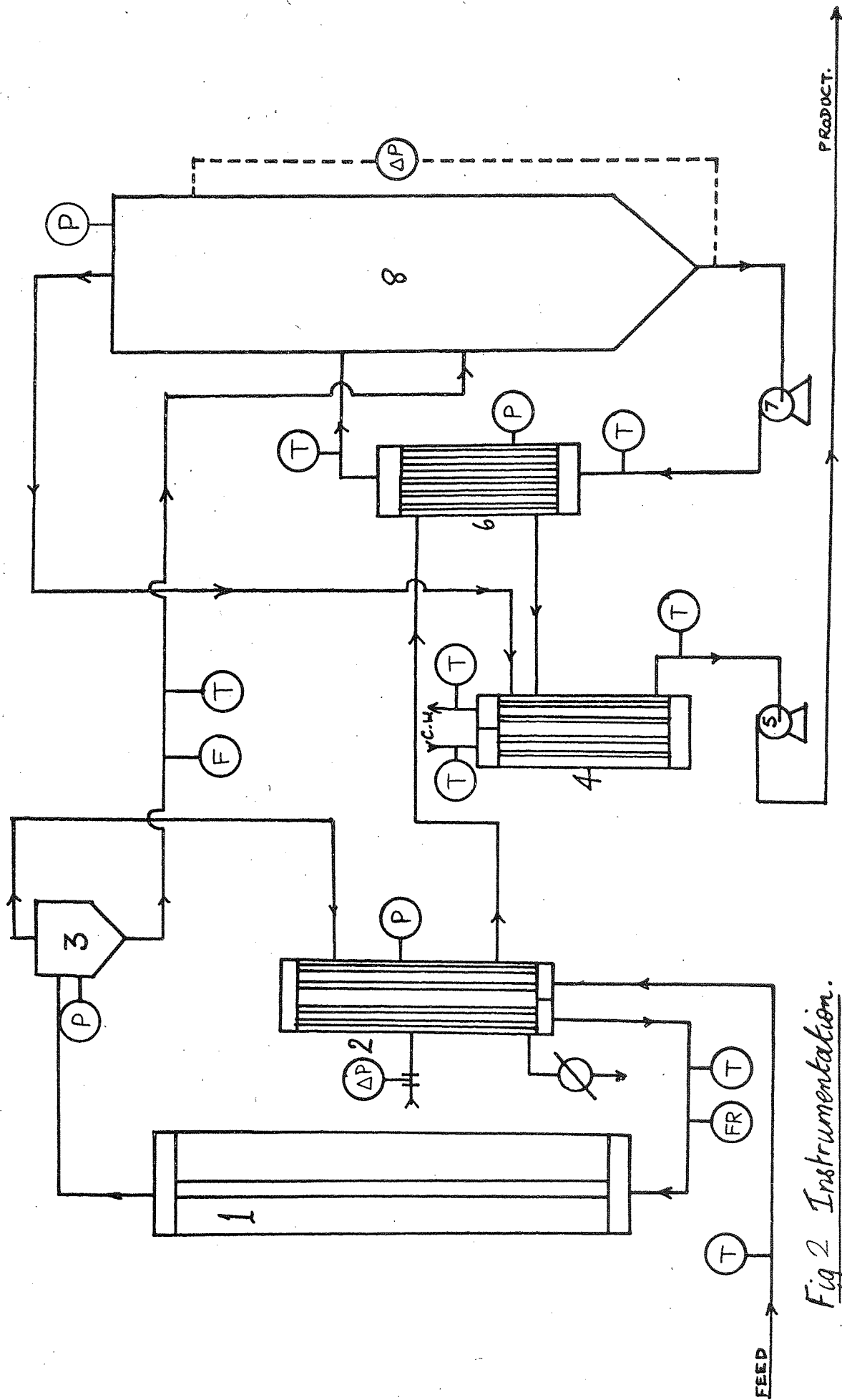
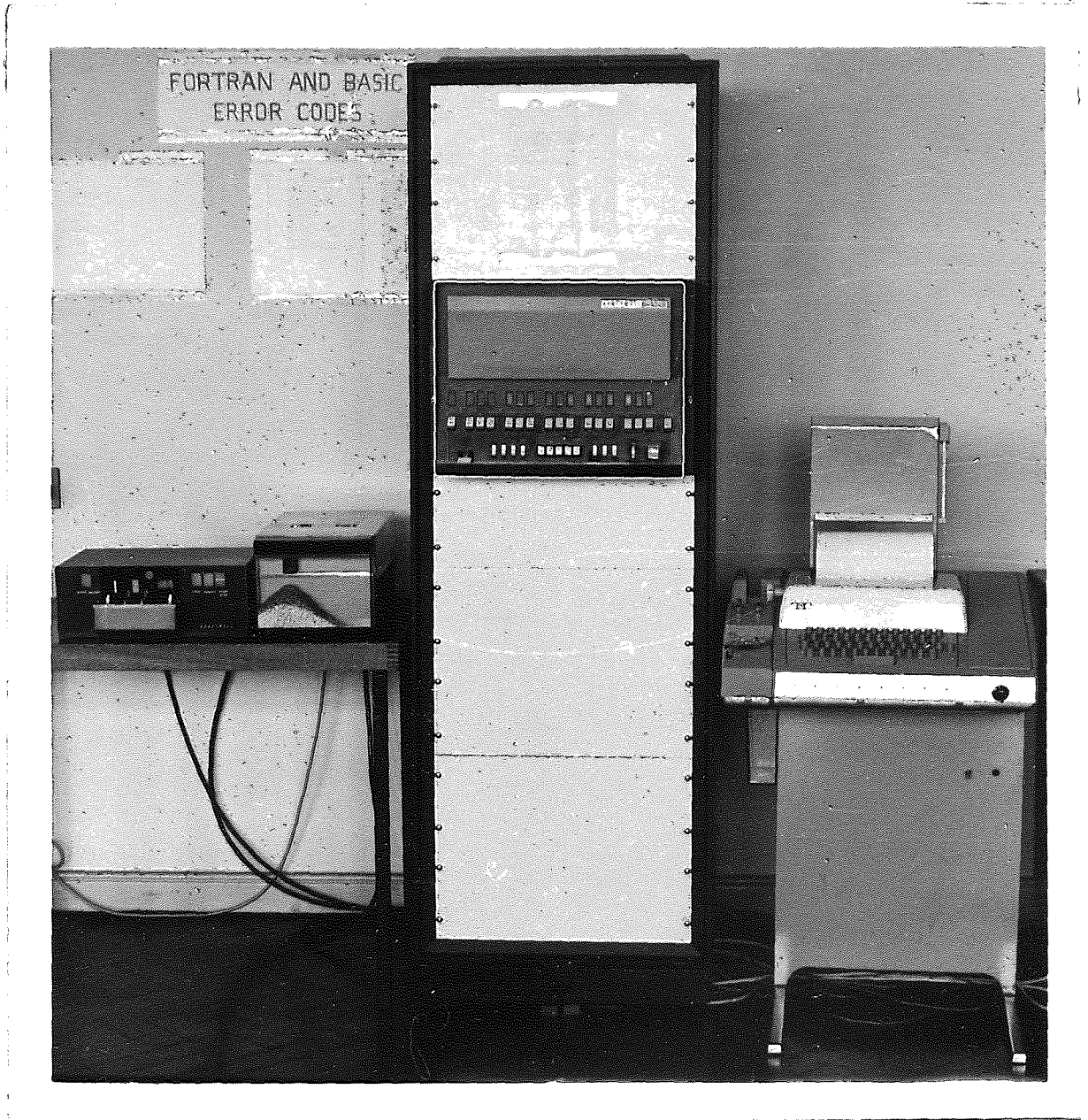


Fig 2 Instrumentation.

two flowmeters and two differential pressure transmitters. These all produce electrical analogue signals which are input to the mobile cabinet of the data logger. A flow meter is required to measure the recirculation rate in the second effect, this is currently under consideration and will be fitted in the near future.

2.1.2. Data Channels assigned to inputs

<u>Channels</u>	<u>Input</u>
12 to 15	Pressure transducers
16,17	Flowmeters
18,19	Differential pressure
20 to 27	Thermocouples



PHOTOGRAPH 2

HONEYWELL H-316 COMPUTER AND PERIPHERALS

The Honeywell H-316 Digital ComputerHardware Description (See also photographs 2 & 5)

The computer system may be conveniently divided into two parts:-

- (i) The 'mainframe', that is the actual central processor and memory.
- (ii) The peripherals, the devices necessary to send information and instructions into the mainframe, and receive answers back.

See Fig. 3

Photograph 2 shows the computer installation. In the centre is the mainframe, housed in a tall rack, the computer itself is the dark coloured panel, the remainder of the rack is for future additions. On the right stands the ASR-33 Teletype, this is a typewriter linked directly to the machine, and can be used to input and output information or instructions. The table on the left supports the high speed paper tape reader (extreme left), and high speed punch. These two devices are used for reading or punching tape at about 150 characters per second, about 15 times faster than the teletype. All the items are free standing, and no special air conditioning is needed.

Closer examination of the front of the mainframe reveals the controls. The top row of small erect oblongs are the indicator lights, (fully described later), below these, the white press buttons (17) for setting and clearing the lights, On the bottom row, reading from left to right, the ON/OFF switch, 4 sense switches, these can be set on or off and used to control programs. In the middle are 5 selector switches to examine the machine registers, then 3 switches which the operator can use to fetch or store information at any particular memory location. The last two

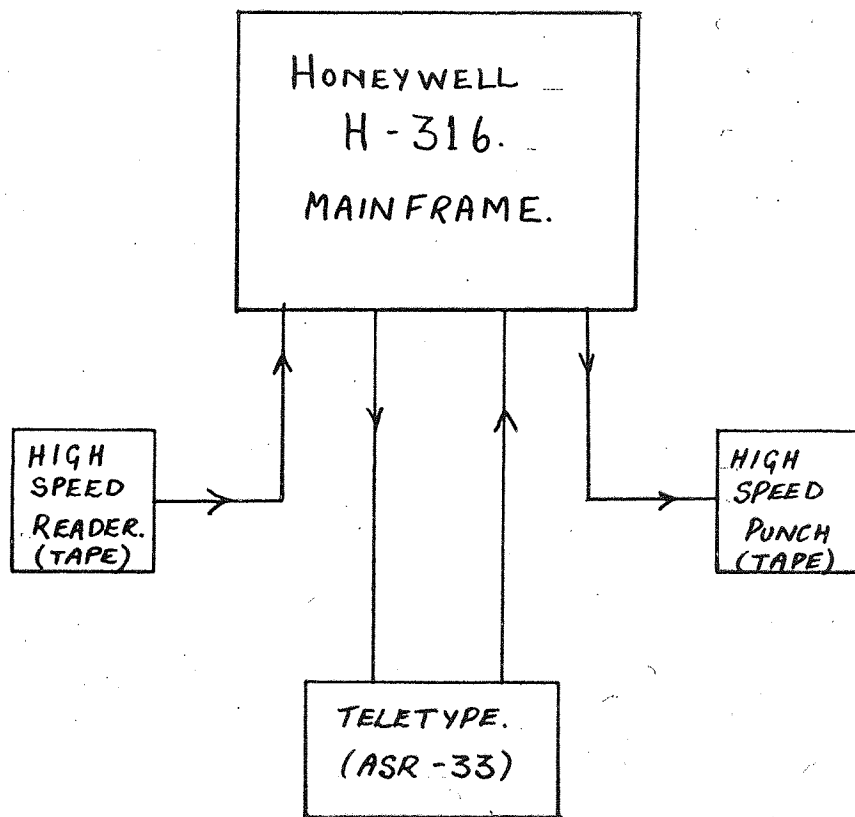


Fig. 3 Computer Configuration.

controls are the mode switch, for selecting the RUN, STOP, or DIRECT ACCESS TO MEMORY modes, and the start button.

The controls are arranged so that it is impossible to damage the machine by operating them, the worst that can happen is that a program could be corrupted.

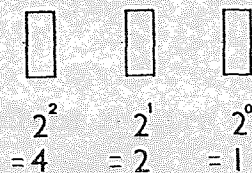
Storage

Numbers, and programs are both stored in the computer's memory. The same memory location may be used for either, it is the interpretation of the word which is important. Each word consists of 16 bits of information, and may be fetched and displayed on the 16 indicator lights on the control panel. The 16 bits are arranged in five groups of three, and one odd bit on the left hand end. The machine used had 12K (1K=1024 words) of storage.

Numbers

If we consider the word represents a number, the left hand bit indicates the sign, if it is set (lighted) the number is negative, otherwise positive. The five groups of 3 represent five digits. The storage is binary, each digit being an OCTAL number. This is because with 3 binary bits it is only possible to count up to 7.

Consider a group of 3



The digit is made up by adding together the bits which are set. Thus

$$IOI = 5$$

$$III = 7$$

Each group of three is interpreted in this way, building up a five digit number. To remind us that these are octal, an apostrophe is placed before the number. Thus, five octal digits can express numbers from '00000 to '77777. An octal count would go:-

I, 2, 3, 4, 5, 6, 7, '10, '11, '12 - - -

'17, '20, '21 - - - '776, '777, '1000 etc.

(NB I='1 etc up to 7='7) Digits 8,9 are never used.

2.2.2

Software Word Formats

Integers require one location for storage, and are held in the form of a sign bit, plus 15 magnitude bits.

eg. 24 = '000030

-24 = 'I77750

Real numbers require two memory locations, and are stored as a sign bit, an 8 bit characteristic and 23 floating point bits.

eg. 0.I = '037346, '063146

503.25 = '042 375, '150 000

-0.I = 'I4043I, '0I4632

-503.25 = 'I35402, '630 000

Double Precision numbers require three memory locations and are stored as sign bit and 39 floating bits and an 8 bit characteristic

0.I = '037 346, '063146, '063 I46

-0.I = 'I4043I, 'II463I, 'II4632

Machine Registers

These are used for controlling the computer, i.e., telling it where to start, for transferring information in and out of the memory, to and from the peripherals. The contents of a register may be displayed on the indicator lights by the push button selectors.

The 'A' Register

This is a 16 bit register, used as the primary arithmetic register. This means that all the arithmetic is done by bringing numbers into the A register, performing the operations, and then storing the answer back in the memory. For example, consider adding two numbers together. One number is loaded into the A register, and the other added to it, the sum may be stored or other operations performed on it.

Use of A Register for Data Transfer To/From Peripherals

Information is transferred to and from the peripherals via the A register. Eight binary bits are used to represent each of the letters of the alphabet, punctuation marks and numerals, hence one 16 bit word can hold two. The method used is to transfer the right hand eight bits from the A register to the punch or teletype. At the punch, holes are made in the tape (8 track), and at the teletype the pattern is interpreted as a character and typed. The right and left halves of the register are then exchanged (because only the RHS can be used for input/output) and the process repeated. Similarly data can be input from the tape reader or teletype.

B Register

Again a 16 bit register, but used as the secondary arithmetic register. This is when floating point or double precision numbers are being used as

they require two 16 bit words for storage. In this case the computer is put into the double precision mode, and treats the A+B registers as one 32 bit word.

P Register - The Program Counter

When a program is in the memory, it has a starting place, i.e., the first instruction to be executed, and an end. Between these are a series of instructions, all stored in consecutive memory locations, which tell the central processor what to do. Hence, the machine must be told where to start and the P register is used for this purpose. During the program's running this register contains the location of the next instruction to be executed. Thus, to start a program, its starting address is put in the P register, where it answers the role of 'next instruction to be executed' and the start button pressed, this causes the contents of the location to be treated as instructions. Under normal conditions the P register is incremented by one during the execution of each instruction, so the program continues until a halt is encountered. However, certain instructions cause the register to be altered by more than one, causing branches in the program, for example a jump, or a comparison, as in an IF statement.

Index Register

This 16 bit register is used for the modification of addresses. When indexing is specified the contents of the index register are added to the address referred to by the instruction, before execution. By adding one onto the index register within a loop the next location must be used. This saves writing out repetitive manipulations. Index register programming is only really necessary at Assembler level.

Sense Switches

These are the four switches on the control panel marked I to 4. They may be on, referred to as 'set', or off, 'reset'. During a program these can be tested, and action taken depending upon what state is detected. A useful example is to consider input of data, the program may be written so that when sense switch I (say) is set the data may be typed in, and when reset, input is by paper tape.

2.2.3

Indirect Addressing

If bit I of a memory reference instruction is set indirect addressing takes place. In this case the effective address of the operation is assumed to be the contents of the location specified by the direct address. This is best illustrated.

If there is an ADD command in sector 2, and this is flagged for indirect addressing (by placing an asterisk after the op. code).

```
ADD* '444
```

This means ADD the contents of the location whose address is in location '444

```
if '444 contains '0623I
```

This is sector '06 location '23I. Therefore the contents of location '23I of sector '06 will be added to the A register.

Since the address field of the indirect address ('444 in this case) is I6 bits, up to I6K of memory may be addressed in this mode. Indirect addressing adds a cycle to the execution time of an instruction.

Multi-level Indirect Addressing

If Bit I of the indirect address is set, another level of indirect address-

ing takes place.

ex. if '444 had contained 'I 0623I then the address specified would have been referenced. This chaining continues until an address is found in which the flag is not set.

2.2.4

Hardware Word Formats

(Honeywell 316 computer).

This refers to the method of interpreting the 16 bits of each word.

Words can represent (i) Data

(ii) Instructions.

Data - this is stored in binary form. Single precision data words include 15 magnitude bits, plus a sign bit, and so represent data in the range $\pm 2^{15}-1$. ie. ± 32767



Double precision data words are made up of two 16 bit words, each having 15 magnitude bits. The first word contains the sign bit, and the 15 most significant bits of the data. In the second word the sign bit is always zero, plus the 15 least significant bits of the data. Double precision data words represent numbers in the range $\pm 2^{30}-1$. (± 1073741823).

Instructions

There are four types on instruction,

(i) memory reference

(ii) input/output

(iii) shift

(iv) generic

each will be dealt with separately.

I. Memory Reference (MR)

These are words which access the memory, this is divided up into SECTORS, each containing 1000 memory locations, (each holding one 16 bit word), these are numbered from '000 to '777.

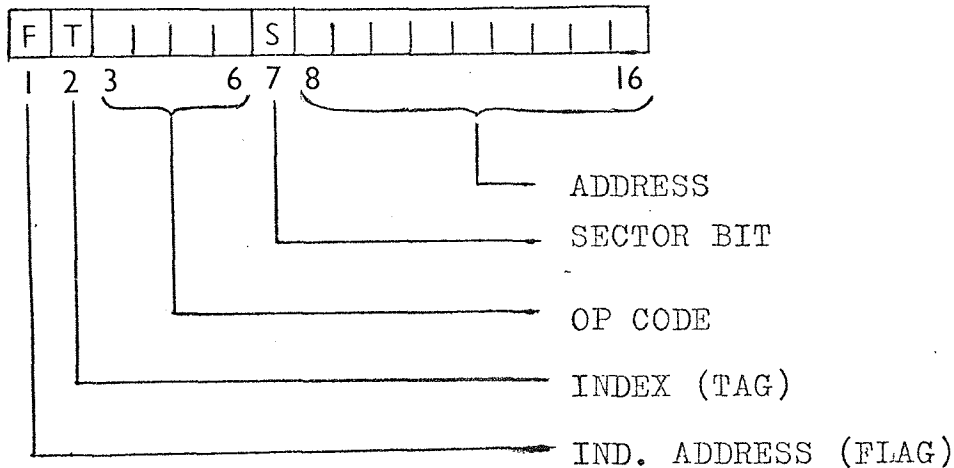


FIG. 4

The 16 bits are divided up as shown in fig (4).

Bit 1: Specifies indirect addressing when set.

Bit 2: Specifies indexing.

Bits 3 to 6: The operation code, ie. what command is being performed.

eg. ADD 0110 ('06)

SUBTRACT 0111 ('07)

COMPARE 1001 ('11)

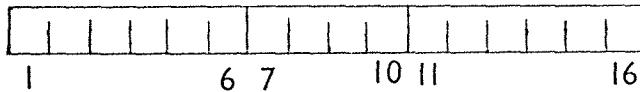
Bit 7: the sector bit, if this is set the memory location being referenced is in the same sector as the instruction, if bit 7 is reset (=0) the

reference is to sector zero. ('00)

Bits 8 to 16: The address. This is the address within the sector. Here it can be seen that because only 9 bits are used to represent the address it is only possible to access addresses up to '777 with an MR type instruction as it stands. This is overcome by using Indirect Addressing.

2. Generic (G) These instructions occupy the whole 16 bits of a word, and perform certain set functions. For example IAB (Interchange the A and B registers), CRA (clear the A register), SSS(Skip the next instruction if any sense switch is set).

3. Input-output (I/O).

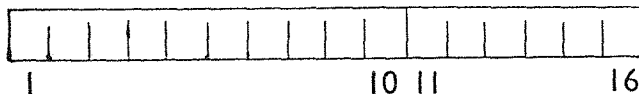


Bits 1 to 6 define the operation which is to be performed.

Bits 7 to 10 define the function which the operation is going to perform.

Bits 11 to 16 give the device being used.

4. Shifts (S)



Shifts are used for moving bit patterns across the registers, eg by Left shifting from B to A, or moving parts of bit patterns. See BCDO in subroutine in continuous scanning program (Section 4) for a good example of shifting uses.

Bits 1 to 10 contain the operation code.

Bits 11 to 16 the number of places which the data is to be shifted, (in two's complement form (ie negative)).

(See Fig. 5 for complete data logging system)

2.3. The MDP - 200 Data Logger

This is the system which presents data, for access by the computer, it comprises the following:-

- (a) Mobile cabinet (photograph 1 and 3)
- (b) Fixed cabinet (photographs 4 and 5)
- (c) Interconnecting cables

The cables run from the computer room to 5 of the departments laboratories. Here, there is a junction box into which the mobile cabinet is plugged. Thus it is possible to use the system over the whole building, the furthest point being about 500 ft from the computer room. The junction boxes also have a connection for the teletype, which enables the computer to be used anywhere within the laboratories.

2.3.1. The Mobile Cabinet

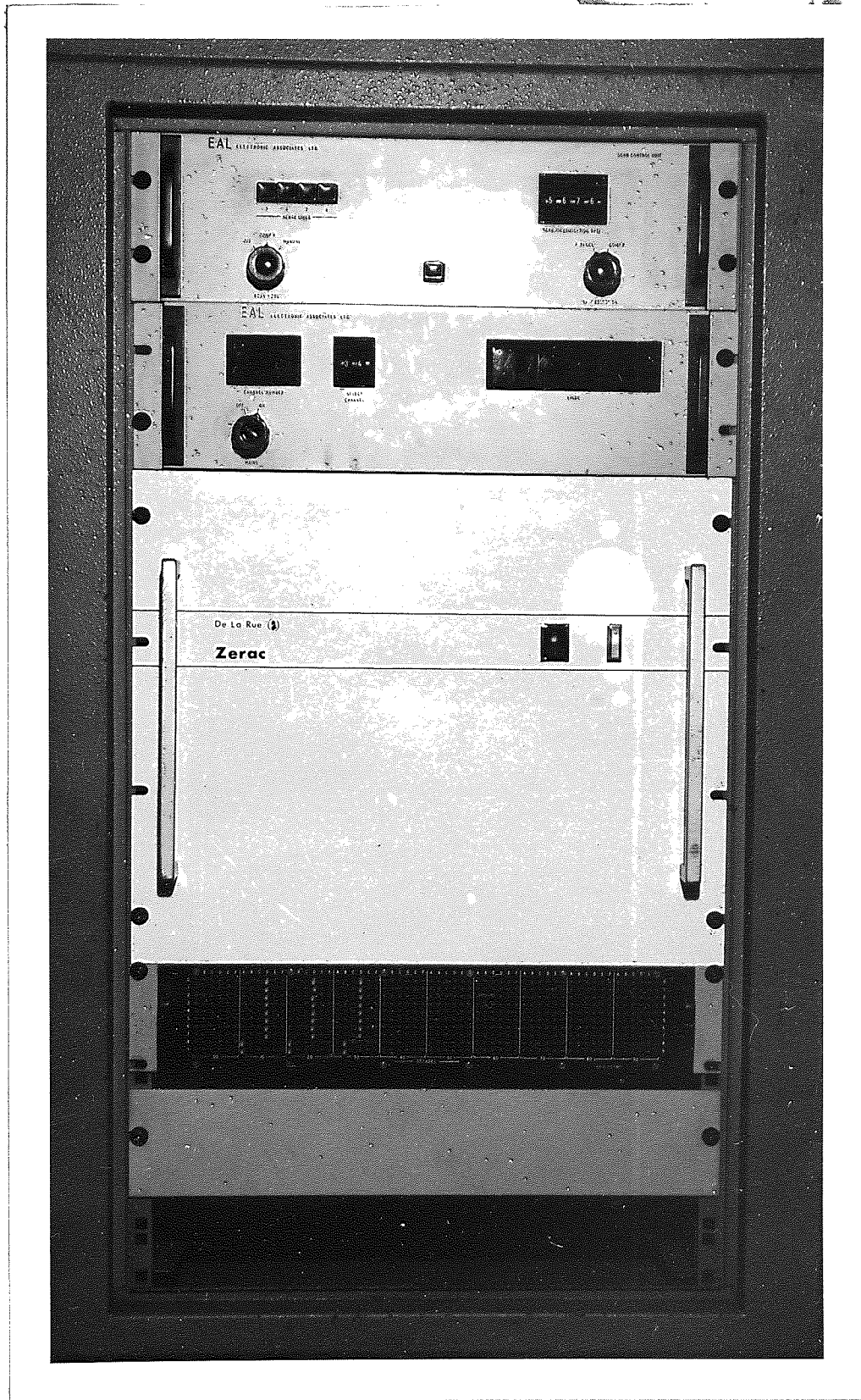
The electrical analogue inputs are connected into the data logging system at the back of this cabinet. On the front are the controls, and displays.

2.3.1.1. Sense Switches

These are located on the top left hand side, in the form of four push buttons. These are pushed in for set and pushed again to reset. Whilst set they are illuminated from the rear. These are duplicates of the switches on the computer's control panel.

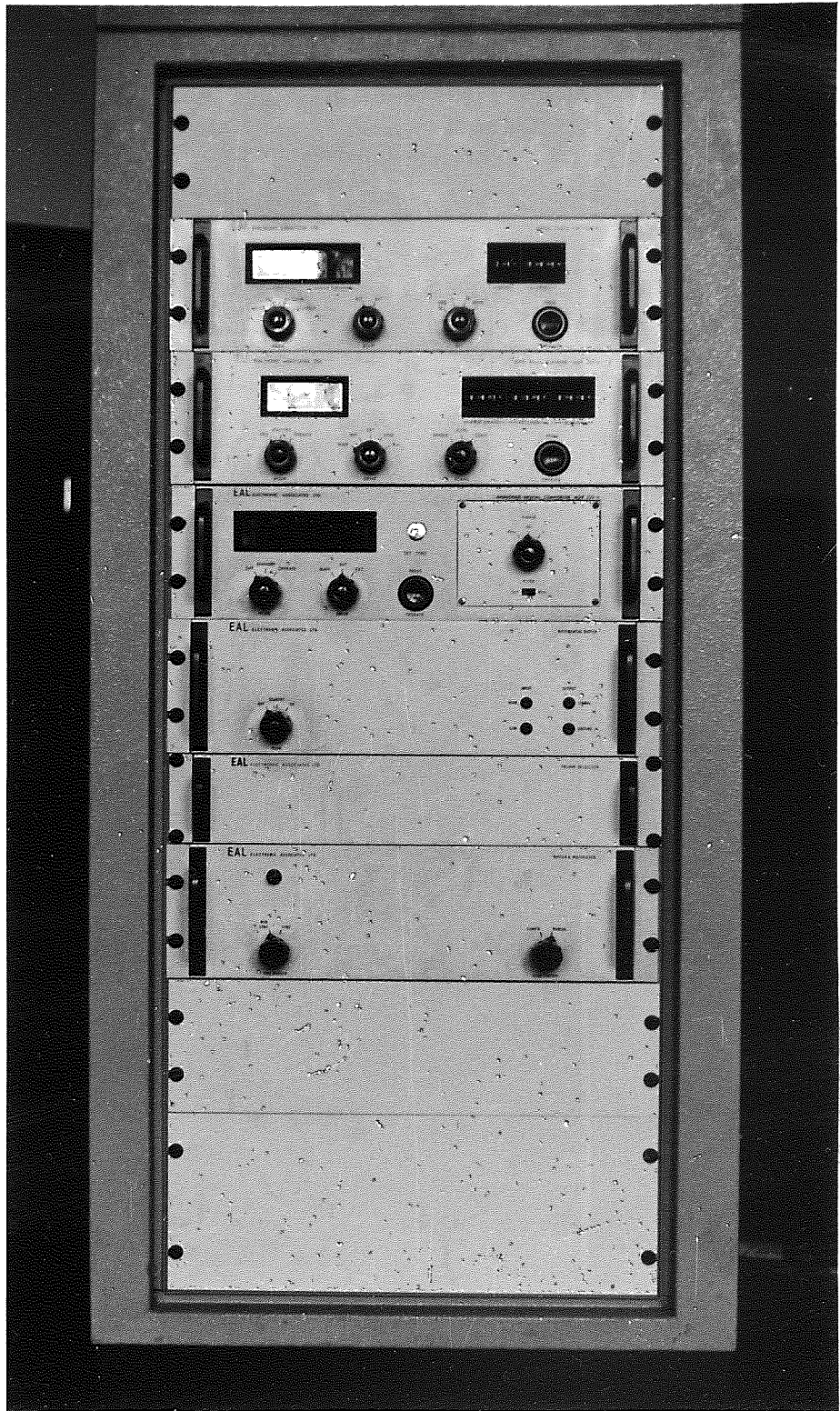
2.3.1.2. Thumb switches

Labelled Scan Identification Data, these are used to input integer values, for example test run numbers. In the photograph (3) They are set at 5676.



PHOTOGRAPH 3

MDP 200 DATA LOGGER, MOBILE CABINET



PHOTOGRAPH 4

MDP 200 DATA LOGGER, FIXED CABINET

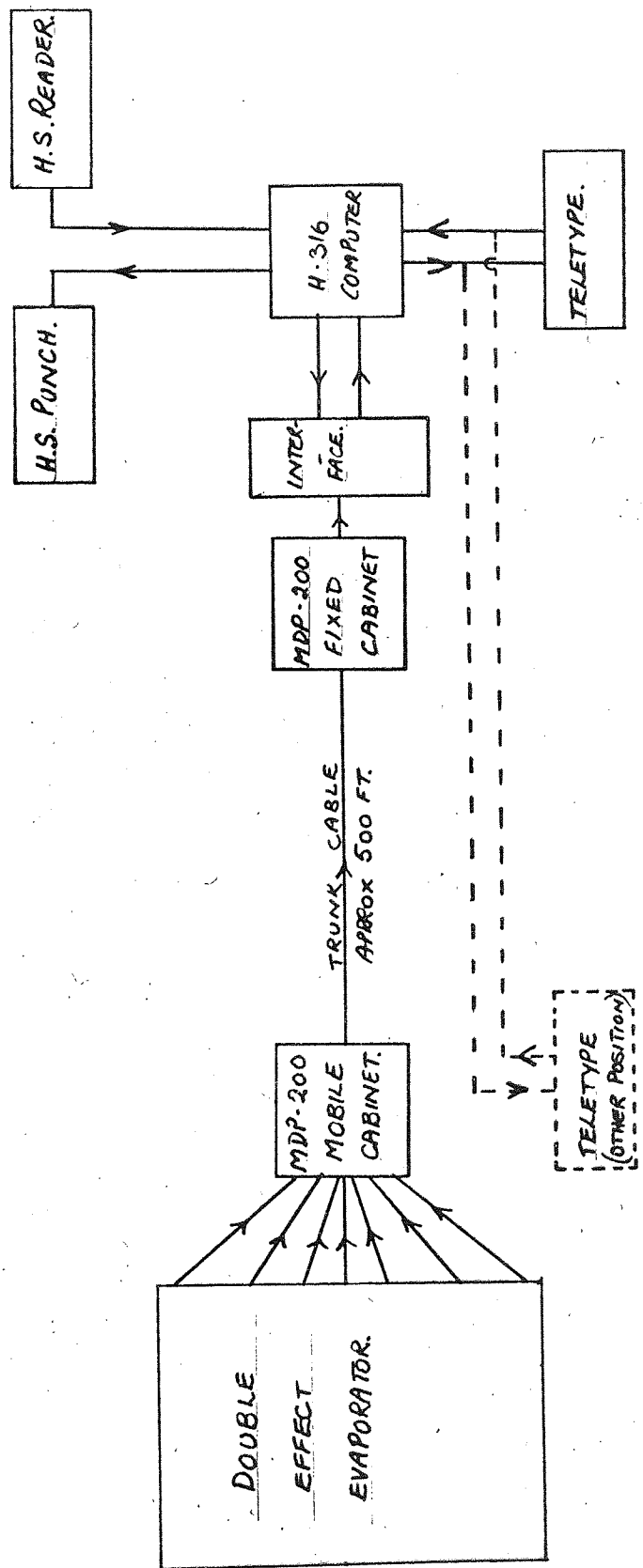


FIG 5. DATA LOGGING SYSTEM.

2.3.1.3. Remote Visual display

Any one of the data channels may be displayed on this indicator, by setting its number on the selector switch. The channel number is then shown on the left, and the value on the right hand digital displays. It should be noted that the value is only changed when the computer inputs a value from this channel, so the value shown is the last input which was made.

2.3.1.4. Channel selector and amplifier

The channel selector unit receives a signal from the fixed cabinet which tells it which of the inputs is required. This is then connected to a high gain amplifier, whose gain is set by pins on a pegboard, although computer gain selection may be used. The input signal, in the range 0-10, 0-100 or 0-1000 millivolts is amplified to the range 0-10 volts and transmitted back to the main cabinet by banks of reed relays.

2.3.2. Fixed Cabinet

This contains

- (a) the clock
- (b) reed relay scanner
- (c) Analogue to digital converter
- (d) Differential buffer
- (e) Trunk selector
- (f) Buffer multiplexer

2.3.2.1. MDP - 240 Clock

Situated at the top of the mobile cabinet this unit provides the time in hours, minutes, seconds and tenths of seconds. Hours and minutes are shown on digital displays. Two modes of operation are possible:- (a) 24 hours, repeating. Thus when the time reaches 23 59 59 99 it starts again from 00 00 00 00.

(b) 30 hour, in which elapsed time is measured from zero, in this mode the clock stops when it reaches 29 59 59 99.

The time from this clock may be read by the computer, this is achieved in two halves, one being the hours and minutes, then the seconds.

2.3.2.2. Reed Relay Scanner

Here, the appropriate reed relays are selected to connect the required channel to the amplifier in the mobile cabinet. The current channel being scanned is shown on a two digit display on the left hand side of this unit.

2.3.3.1. Analogue to Digital Converter

The name is virtually self explanatory. An input electrical analogue signal in the range 0 to 10 volts is converted into its digital equivalent. This has two forms, one the actual digital decimal value, which is displayed on a four digit indicator situated on the LHS of the unit, two the BCD, (Binary Coded Decimal) number presented to the computer.

2.3.3.2. Binary Coded Decimal

Four binary bits are used to represent each digit, thus 8 4 2 1, the digit being the sum of the "set" bits. In this way four bits can be used to represent numbers from 0 to 15, but only 0 to 9 are needed for this application.

2.3.4. Differential Buffer

This unit eliminates the differences in ground voltage which may occur between the fixed and mobile cabinets.

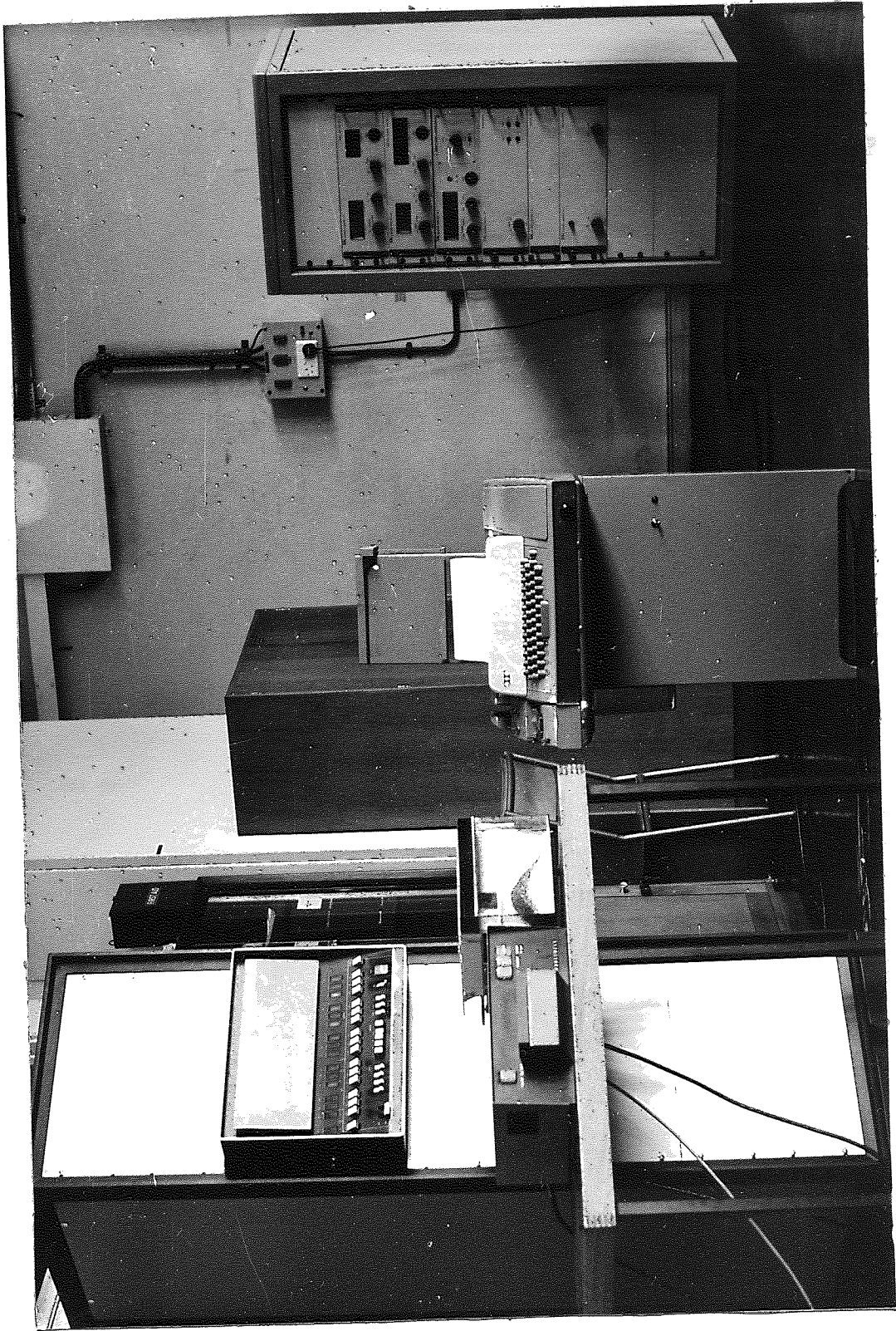
2.3.5 Trunk Selector - The unit which selects the trunk cable over which the signal will be coming from the mobile cabinet.

This cable selected will be different for each of the five junction boxes.

2.3.6. Buffer Multiplexer - Accepts the input command from the computer and decodes it into instructions for the data logger, then, when the data has been gathered it is placed in the buffer ready for transmission back to the computer.

2.3.7. General When the system is in operation, the following layout has been found best. The operator places the teletype, computer and fixed cabinet to form three sides of a square. Sitting at the typewriter the user then has the computer controls within easy access on his left, and the data logger similarly placed on his right.

The wires connecting the fixed cabinet to the computer may be seen across the floor in photograph 5.



PHOTOGRAPH 5 - GENERAL VIEW OF COMPUTER AND DATA LOGGER

3.1.1. Basic

Introduction

BASIC - 16 is a Honeywell version of the high level programming language BASIC which is in wide use on a variety of digital computers. BASIC is however simpler than FORTRAN or ALGOL to which it has a family resemblance.

3.1.2. Instructions, statements, variables and expressions

A BASIC program consists of a sequence of instructions written according to the rules of the language. A typical instruction is:-

LET v = e

where LET is a BASIC statement, v represents a variable and e is an expression. (N.B. It is not necessary to include the LET).

An example of a variable is a simple variable which is either (1) a letter e.g. A X Y

or (2) a letter followed by a digit e.g. A4 X5 Y9

An expression is in general a mathematical formula in proper form. It may also be a variable or a numerical constant. Examples of expressions are:-

10 -5 .6

D Q N9

D + 10 -(A + B) *C/D

The general form of an expression is a string of variables and constants separated by operators e.g. + - * (). The rules for writing and evaluating an expression are similar to those for FORTRAN or ALGOL expressions:-

- (1) Expressions in parentheses are evaluated first.
- (2) Operators are evaluated in the order:-
 - (i) Exponentiation (\uparrow).
 - (ii) Multiplication (*) and division (/).
 - (iii) Addition (+) and subtraction (-).
 - (iv) Operators of equal precedence are evaluated from left to right in the expression.

The effect of the instruction `LET v = e` is to evaluate the expression `e` and set the variable `v` equal to the value of the expression. For example, the formula $y = ax^2 + bx + c$ is evaluated by the instruction `LET Y = A*X2 + B*X + C`. All variables in the expression must, of course, have been previously assigned numerical values.

The second type of BASIC variable is a subscripted variable which has the form letter (expression, expression,.....)

e.g. `A(1) B(1) X(J+1) A(1,5) B(1,J) Y(J+2,K+3)`
`A(1,2,3) B(1,J,K) Y(J+2,K+3,L+5)`

- NB. (i) Subscripts are allowed to assume zero but not negative values.
- (ii) If a subscript is an expression, the expression will be evaluated and truncated to obtain an integer result
- e.g. `X(0.3) X(0), A(1.9) A(1)`.
- (iii) Any number of subscripts is allowed.

3.1.3. Input/output

The statement INPUT A causes the numerical value entered by the user to be assigned to the variable A. Similarly the statement PRINT X causes the current value of the variable X to be printed out.

Using these statements a program for the evaluation of a quadratic is:-

```
INPUT A
INPUT B
INPUT C
INPUT X
LET Y = A*X2 + B*X + C
PRINT A
PRINT B
PRINT C
PRINT X
PRINT Y
```

The above program may be written in a more compact form by the use of commas in the input/output instructions:-

```
INPUT A,B,C,X
PRINT A,B,C,X,A*X2 + B*X + C
```

Note also the use of an expression in the PRINT instruction.

3.1.4. BASIC operation

Most input/output operations in BASIC-16 are carried out via the ASR 33 teletype. After the BASIC compiler has been loaded and entered and an initialisation sequence completed, the teletype will print a question mark (?) which is a request for input from the user.

BASIC-16 operates in two modes:-

- (i) Step-by-step mode
- and (ii) Stored program mode.

In the step-by-step mode the user may enter a BASIC instruction of the type described above (terminated by 'return'), which if valid will be executed immediately. If the instruction is invalid an error message will be printed.

In the stored program mode BASIC instructions are entered preceded by a statement number and terminated by 'return'. i.e. a carriage return on the teletype. This has the effect of storing the instructions in the computer. When the program is complete it may be executed by typing RUN (return). Thus the quadratic program described above might be written in stored program form as:-

```
10 INPUT A,B,C,X
20 PRINT A,B,C,X,A*X^2 + B*X + C
```

Note that each line of the program has a statement number which must be in the range 1 to 9999. The statement numbers control the order in which the instructions are stored in the computer. However, (as indicated above), they need not be consecutive and it is advantageous to use non-consecutive statement numbers in case it is desired to add lines to the program later.

Each BASIC statement normally appears on a new line. However, more than one statement may appear on one line by use of the colon (:) as a statement delimiter. Thus the above program could be further condensed to:-

```
10 INPUT A,B,C,X: PRINT A,B,C,X, A*X^2 + C
```

The resulting print-out for the program in Fig.1 is shown in Fig.2.

```
?10 INPUT A,B,C,X
?20 PRINT A,B,C,X,A*X^2+B*X+C
?RUN
!1
!2
!3
!2
1 2 3 2 11
0 EXIT
?
```

Note that when the computer encounters an INPUT instruction it prints an exclamation mark (!). The user then enters the required value terminated by 'return'. The message 0 EXIT indicates that the calculation has been completed.

The error diagnostic messages which may be printed during program execution are listed in section 3.1.13.

3.1.5. Details of BASIC 16

In the following sections:-

v, v1, v2 etc. denote variables

e, e1, e2 etc. denote expressions

n, n1, n2 etc. denote statement numbers

a1, a2, etc. denote list items which may be constants, variables, expressions, labels or tabs depending on context.

op denotes a comparison operator

3.1.6. BASIC statements

(a) LET v = e (Alternative form: v = e, ie. the "LET" may be omitted), causes v to be set equal to the current value of e

(b) INPUT v1, v2, causes an exclamation mark (!) to be printed and the program to wait until the user types in the data value followed by 'return'. If more than one data value is required the items must be separated by commas.

(c) PRINT a1, a2 causes the list items (a1, a2,....) to be printed on the teletype. A list item may be a variable, an expression, a label or a tab statement. A label is a string of any keyboard characters enclosed in quotation marks ("). On execution the label will be printed and in this way titles and headings may be printed on the output. For example the instructions:-

```
=====
A = 2
B = A^2 + A
PRINT "A = ", A, "B = ", B
=====
```

cause the following output:-

```
A= 2 B= 6
```

The list items will be printed at character positions 0, 15, 29, 43 and 57, the last item being followed by 'return' 'line feed'. If the list items are separated by a semi-colon (;) tabbing to the above-mentioned character positions is suppressed and a closer packed output results. A comma or semi-colon after the list item suppresses the 'return' 'line feed' to the character position corresponding to the (truncated integer) value of e provided the current head position is less than this value.

If $e > 70$ the head tabs to position 70. By combination of the various list items an informative and pleasing output can be obtained.

```
(d)      READ      v1, v2 . . .  
          DATA     a1, a2 . . .  
  
          RESTORE
```

When a READ statement is executed its list items (v1, v2, . . .) are set equal to the next available elements in the DATA statement list. Thus the instructions:-

```
      READ      A,B,C  
      =====  
      READ      A,B,C  
      =====  
      DATA     1,2,3,4,5,6
```

cause A,B and C to be set equal to 1, 2 and 3 respectively on execution of first READ statement and to 4, 5 and 6 respectively on execution of the second READ statement.

Variables and expressions as well as constants may appear as items in the DATA statement list. All variables used must have been previously assigned numerical values.

If insufficient unused data items remain, an error message is printed. The instruction RESTORE returns the data 'pointer' to the first item in the first DATA list.

e) STOP

END

cause the statement number and the word EXIT to be printed and the computer to await input from the user. A program need not have STOP or END statements but conventionally STOP is used at the logical end(s) of the program and END at the physical end of the program.

f) REM

All characters following REM (remark) up to a 'return' are recorded in the program but cause no action. Remarks are used to identify sections of a program for the convenience of the user.

g) DIM

A DIM (dimension) statement is required if the maximum value of a subscript of a subscripted variable exceeds 10. Typical DIM statements are:-

```
DIM A (20)
```

```
DIM X (50, 50)
```

h) FOR v = e1 TO e2 STEP e3

(Alternative form: FOR v1 = e1, e2, e3)

NEXT v

When the FOR statement is encountered the simple variable v is set equal to e1 and the program proceeds until the NEXT v statement is encountered when the FOR statement is returned to and v is set equal to e1 + e3.

Thus these instructions provide a means for 'looping' through a set of instructions a prescribed number of times, v being increased by e_3 each time. The number of loops is $(e_2 - e_1) / e_3 + 1$ and after the last loop the program proceeds to the instruction after the NEXT v statement. Care must be taken that the values of e_1 , e_2 and e_3 are compatible with each other. (N.B. e_3 may be negative) but the loop will be executed at least once (i.e. for $v = e_1$). Loops may be nested to any extent but inner loops must be 'closed' before outer loops. If $e_3 = 1$ then the forms

FOR $v = e_1$ TO e_2 (or FOR $v = 31, e_2$)

may be used.

(i) GO TO n

IF e, n_1, n_2, n_3

IF e_1 op e_2 THEN n (or IF e_1 op e_2 GO TO n)

ON e GO TO $n_1, n_2, n_3 \dots n_k$

GOSUB n

RETURN

Normally programs are executed in increasing order of statement number. The FOR statement interrupts this order by looping and the above statements enable the programmer to change the execution order of the program either conditionally or unconditionally.

(a) GO TO n causes transfer to statement number n in the program

(b) IF e, n_1, n_2, n_3 causes transfer to statement number

n_1 if $e < 0$, to n_2 if $e = 0$ or to
 n_3 if $e > 0$.

(c) i)

IF e_1 op e_2 THEN n

(Alternative form: IF e_1 op e_2 GO TO n)

causes transfer to statement number n if the

comparison e_1 op e_2 is true, otherwise the

program continues to the next instruction.

(ii) IF e1 op e2 THEN S causes the BASIC statement S to be executed if the comparison e1 op e2 is true, otherwise control passes directly to the next line of the program. In the form

IF e1 op e2 THEN S1 : S2

if e1 op e2 is false S1 and S2 are not executed and control passes to the next line (i.e. the next numbered statement).

The comparison operator (op) is one of the forms:-

= equal to
<> not equal to
> greater than
>= greater than or equal to
< less than
<= less than or equal to

(d) ON e GO TO n1, n2, n3, . . . nk causes transfer to statement number n1 if the value of e truncated to the greatest integer < e is 1 e.g. if e = 3 transfer is to statement number n3. N.B. if e < 0 or >> k+1 an error exit will be caused.

(e) GOSUB n

RETURN

GOSUB causes transfer to statement number n which is the first instruction of a subroutine. The subroutine must be terminated by a RETURN statement which returns the program to the instruction immediately following the GOSUB statement.

Subroutines may be 'nested'.

3.1.7. Standard Functions

The following standard functions are provided by BASIC and may be included in any expression:-

LOG (e) - \log_e

EXP (e) - exponential

ABS (e) - absolute value

SQR (e) - square root

INT (e) - integer value of the expression (truncated)

SIN (e) - sine (argument in radians)

COS (e) - cosine (argument in radians)

TAN (e) - tangent (argument in radians)

ATN (e) - arctangent (result in radians)

SGN (e) - result = -1 if e < 0

= 0 if e = 0

= 1 if e > 0

RND (e) - generates a random number in the range 0 - 1. This function is used to generate a sequence of pseudo-random numbers. The sequence is pseudo-random in the sense that it repeats itself after an integer multiple of 128 numbers have been generated. The sequence is initialised by setting e \neq 0 when the value of RND (e) will be e if e is positive or a positive number in the range 0 - 1 if e is negative.

The random numbers are then generated by calling RND (e) with e = 0. The same sequence is always produced for the same initial value of e. The length of the sequence depends on the initial value of e. At present the longest known sequence is 8192 numbers which is obtained, for instance, by setting e = 1000.

3.1.8. User defined functions

The statement DEF FN a (v) = e enables the user to define a function of a single variable which may then be called anywhere in the program. Up to 26 different functions denoted by a letter (a) may be defined. For example, the instructions:-

```
DEF FNQ (X) = 3*X^2 + 2*X + 1
```

```
Y = FNQ (Z)
```

```
W = FNQ (3)
```

causes Y to be set equal to $3*Z^2 + 2*Z + 1$ and W to be set equal to 34. Note that in the DEF FN a (v) statement v is a 'dummy' variable.

3.1.9. BASIC commands

A command is not a statement belonging to the BASIC language but a direction to the BASIC compiler causing certain action.

RUN (see above) is an example of a command. All commands are terminated by 'return'.

(a) RUN

causes the program to be executed from its first statement.

RUN n

causes the program to be executed from statement number n. A program may be interrupted during execution by setting sense switch 1 on the computer (see CONTINUE below).

(b) LIST

causes the entire program to be printed on the teletype.

LIST n

causes the program to be listed from statement number n to the end.

LIST n1, n2

causes the program to be listed from statement number n1 to n2. If $n1 = n2 = n$ only line n is printed.

LIST, n

causes the program to be listed from the beginning to statement number n.

(c) PUNCH

causes the entire program to be punched out on paper tape.

Segments of program may be punched by PUNCH commands similar to the LIST commands described above.

(d) LOAD

causes a program to be read from paper tape via the high-speed reader

(e) JOB

deletes all stored programs and variables from the computer.

(f) CLEAR

clears all the variables from the computer but does not affect stored programs.

(g) QUIT

causes the computer to become idle. Press 'START' button on computer to re-enter BASIC compiler.

(h) CONTINUE

If sense switch 1 on the computer is set during program execution the program will halt after completion of the instruction being executed.

The message n BREAK will be printed on the teletype where n is the statement number of the last instruction executed. The program may be re-entered at that point by the command CONTINUE (N.B. it may be necessary to enter the command CONTINUE twice to clear a BREAK). Alternatively any other command may be entered.

3.1.10. Input/output of program and data

- (a) To remove an instruction from a stored program enter statement number followed by 'return'.
- (b) To amend a stored instruction enter the statement number and the new instruction.
- (c) To amend the last n characters entered (before a 'return') enter n times and continue. (N.B. NOT A 'CONTINUE' STATEMENT).
- (d) To delete a whole line (before a 'return' enter and continue.
- (e) To punch a program off-line on a teletype, type 'return' 'line feed' at end of each instruction and extra 'return' 'line feed' after the last line. 'X-OFF' 'RUB OUT' should follow each 'line feed' if the tape is to be input through the teletype.
- (f) Numerical constants and data may be entered in normal decimal form or in exponential notation e.g. 100
100.0 0.1L3 10.0E1 are equivalent forms.
- (g) Numbers with absolute values between 0.1 and 9999 will be output in the format XXXX.YYYYYY or - XXXX.YYYYYY with leading and trailing zeros suppressed. The decimal point is suppressed for integer values. Numbers outside the above range will be printed in the format .XXXXXXE⁺ ZZ or
- .XXXXXXE⁺ -ZZ.

(h) The character 'space' may be used freely in order to improve the appearance of a program.

3.1.11. Compilers

Three versions of the BASIC compiler are available. One uses the teletype for all input/output. The second uses the high-speed reader and punch for the commands LOAD and PUNCH and the teletype for all other input/output. The compiler "BASOON" which contains special subroutines to access the MDP 200 data logger directly by ordinary BASIC statements.

3.1.12. Honeywell BASIC Manual

Further details of BASIC 16 are contained in the Honeywell Manual 'Models 316 and 516 BASIC Language'.

(Honeywell Doc.No. 70130072 543, M-449).

3.1.13.

DIAGNOSTICS (ORDINARY BASIC)

<u>Error code</u>	<u>Meaning</u>
AS	Array subscript out of bounds
DA	Attempt to READ more data than available
DF	Attempt to use a function deleted during initialisation.
DL	Statement terminator error
DP	Two decimal points in a number
DV	Dummy variable in DEF statement is subscripted
DZ	Divide by zero
FD	Invalid delimiter in FOR statement
FN	Characters FN misplaced in DEF statement
GS	GO SUBs nested more than eight deep
IC	Condition in IF statement is incorrect
ID	General error
IV	Index variable in FOR statement is subscripted
LG	Negative logarithmic function argument
MO	Memory overflow
M,	Missing or misplaced comma
M=	Missing or misplaced equals sign
M(Missing or misplaced left parenthesis
NO	Numerical overflow
NU	Numerical underflow
NX	NEXT statement has no matching FOR
ON	Expression in ON statement is non-positive, or the GO TO is missing
PD	Strange item delimiter in PRINT statement
RT	RETURN statement not in subroutine
SN	Statement number error (range 1-9999)
SQ	Negative square root function argument
SS	Subroutine selector in CALL out of range (1-10) or subroutine is missing
TH	THEN left out of IF statement
TX	No end of quote
UF	Undefined function
UM	Unitary minus error
US	Undefined statement number
UV	Undefined variable

3.2.

FORTRAN

The Honeywell 316 is provided with a Fortran IV compiler with some useful additions. For example, the sense switches may be tested, and specific values of variables may be loaded into machine registers.

Fortran program development is slow, because the installation does not contain any disc or magnetic tape storage. Hence all the development must be done manually via punched tape.

3.2.1. Fortran Preparation

Running a program in Fortran involves the following steps:-

- (a) Typing program onto tape (source tape).
- (b) Loading compiler into computer.
- (c) Using compiler to convert source tape into machine code (object tape).
- (d) Loading "loader" program into computer.
- (e) Using loader in computer to load object program.
- (f) Using "loader" to load user subroutines (if any).
- (g) Using "loader" to scan system library tapes.
- (h) Starting execution of the users program.
- (i) Making a tape of the whole for future use.

Only a brief description of each stage will be given in general terms, because this only applies to Honeywell machines with this particular configuration.

3.2.2. (a) Typing source program

The tape must be punched as if it were cards, thus each statement must be on a separate line, separated from each other by a "carriage return" and "line feed". Fig (7) shows a small test example. It is important to put spaces in the unused character position 1 to 6 inclusive or numerous errors will result.

(b) Loading compiler into computer

The compiler is a self-loading tape, meaning that the program can be put in the tape reader and after pressing the start button, will be completely loaded without operator action. Most programs are not self loading, and must be taken into the computer by a "loader".

(c) Compiling

The instructions of a program are written, for our convenience, in a "high level language", be it Algol, Fortram or Basic, but these same instructions are, as they are written, quite meaningless to the computer. Compilers are the link between high level and machine code language. Compilers convert one line of "source program" (ie as written) into a whole series of machine code instructions, whilst assemblers in general convert one line of source into one machine code instruction.

In effect the compiler "reads" the Fortram and converts each line into a series of machine code instructions which will perform the requires tasks. Whilst doing this, an error detection is carried out for syntax errors, which are indicated on a teletype print out. The "object" program is then produced by the high speed punch.

(d) "Loading the loader"

Having our object tape, we have finished with the compiler, and now load the "loader" into the computer. This is the program which will accept our object tape and put it into the memory.

(e) Loading the object tape.

The object tape is now loaded by the loader program, whilst loading, the loader makes a table of all the subroutines called, e.g. user subroutines or standard subroutines like SQRT, SIN, etc.

(f) Loading subroutines (user written) When the loading terminates, the message MR, (for more routines), is printed, indicating that loading is not yet complete.

If there are any user written subroutines (i.e. in Fortram) the object code versions of these are now entered simply by putting them into the tape reader and pressing the start button. The loading program packs them into the memory following the main program.

(g) Scanning the library tapes. The method of compilation used by Honeywell is to break each Fortram statement down into a series of subroutine calls. Hence each of these, together with the standard functions and input outputs must be obtained from a library tape. Again, the procedure is to simply put the tapes into the reader and press the start button. The loader examines each program name, (at the start of each program) and compares it with a table of required routines, thus, only the ones which are actually needed are read in and stored, the others are skipped over.

These library tapes are arranged so that it is not necessary to scan them all, for example, if the program does not use double precision or complex numbers there is no need to scan these tapes.

Eventually, the message LC (Loading Complete) is typed. This indicates that the users program is ready for execution.

(h) Execution and explanation which will enable the reader to understand the listings. For full description refer to the Fortran Programmer's Reference Manual. Pressing the start button causes the loader to "jump" program control to the start of the program it has just loaded (1000 for Fortran programs). The input data is read from the device jack specified and calculation done.

3.2.3. Making a self loading tape.

The procedure described above is obviously tedious and time consuming, so one does not want to have to do it each time the same program is to be run. For program development, where changes have to be made there is no alternative, but when this is complete a "self loading system tape" can be made.

The "SLST" contains all the user programs and Fortran subroutines on one tape. This is done by punching out the tape before program execution, using another special utility program (PAL-AP). Each time this same calculation is required the SLST can be loaded and used immediately.

3.3 DAP-16 Language - Assembler

This is not intended to be a full discription of the assembler, but an explanation which will enable the reader to understand the listings, for full discription refer to the Honeywell Programmers Reference Manual.

DAP-16 is a symbolic assembly program which translates a symbolic (source) program into a machine code (object code). Each line of assembler program generally represents one machine code instruction. Initially the DAP-16 tape is loaded into the computer, this is a "self loading tape" in that it loads itself into the correct part of the memory. The symbolic source tape is then read by the assembler. There are two modes of operation.

i). Two pass assembly. The source tape is read twice by the assembler. The first pass allows the assembler to set up a table of all the symbols used. The second pass assembles the object program by reference to this table.

ii). One pass assembly. The symbol table and object assembly are done in the same pass. The difficulty here is that symbols will be referred to before they have been defined. In this case DAP-16 flags these symbols with a double asterisk (**) and assigns each symbol an internal symbol number which is output with the instruction in which the symbol occurs. When this object program is loaded, the loader maintains a table of symbol numbers and their use. When the value of a symbol becomes known, DAP-16 outputs a value along with the object program, which the loader uses to fill in the references to that symbol. The object program from one pass assembly is longer than for two pass, because of the extra information the tape must contain.

DAP-16 functions by assembling a word of machine code from a series of "fields". These are:

- a. LOCATION
- b. OPERATION
- c. VARIABLE
- d. COMMENTS

Location Field Character positions, 1 to 5 of source record.

The location field is used to assign a "label" to a statement, which will be referred to by some other instruction. The label can be up to 4 characters long, and one character must be non-numeric.

Operation Field Character positions, 6 to 11.

This is the operation which is going to be performed by this line, it will be a mnemonic of three or four letters representing one of the standard instructions (see instruction appendix) or one of the DAP-16 Pseudo Operations.

Indirect addressing is specified in this field by placing an asterisk immediately after the mnemonic.

Variable Field Character position 12 until a space or position 72.

This field is used to specify an address and index register for the instruction. When used with pseudo operations the significance depends upon the operation.

Comments Field

The comments are used to make notes or remind the programmer of what is going on. They do not have any effect on the assembly, but are printed on the listing.

Asterisks

These are used extensively in the following ways:

- * in column 1 causes the whole line to be treated as comment.
- * immediately after an instruction mnemonic causes the indirect addressing flag to be set.
- * as an element in the variable field, - the current value of the location counter. (See "expressions").
- ** as an address, - put zeros in the address, because it will be modified by another instruction.
- *** as an operation code, - put zeros in operation code as it will be modified by another instruction.

Symbols

Symbols consist of one to four characters taken from the alphabet, the 10 digits and the dollar sign (§). At least one of the characters must be non-numeric, and the § should

not be used in column 1 as it is used by the operating system as a control character.

A symbol is defined by putting it in the location field, this gives it a symbolic address. The assembler keeps track of the instructions by stepping a location counter by one for each instruction. When a symbol appears in the location field it is normally assigned the current value of the location counter. Any subsequent occurrence in the location field causes an error printout. Any undefined symbols are defined at the end of the program, and may be manually altered after the program is loaded. The following are all acceptable symbols:

FRST, LAST, BGN, STRT,
S2, A3, B5, NEXT,
LØØP

Expressions

Expressions may appear only in the variable field, and may consist of a single element (Simple) or have two or more elements separated by operators. (Compound).

When a single asterisk appears in an expression it designates an address equal to the current value of the location counter. Thus $*+1$ means "this location plus one". Operators are used to separate elements in compound expressions and may be $+$ or $-$. To specify indexing, a comma and an integer l are placed after an expression. Any permissible expression may be written and indexed for an address portion of an instruction, arithmetic is octal.

eg. acceptable expression
for easy reading.

A+6

B+A+C-D

BGN+3

BGN+6,1 (with indexing)

Literals (Constants)

Reference may be made to constants by defining them as in the following example:

LDA A load

—
—
—
A DEC 2 A defined as decimal 2

Alternatively this could have been written,

—
—
—
LDA =2

This would have caused the assembler to assign a location the value 2, and make the instruction access this word. Octal numbers may be set by placing an apostrophe before the number.

LDA n, MESSAGE

where n specifies that there are 2n characters to be converted, and MESSAGE represents the character.

eg. LDA = '20

ISO code may also be generated by the following:

DAC Define Address Constant. This causes DAP-16 to generate a 16 bit binary word which will be used by a flag of memory reference instruction, to access an operand in any action.

LDA=AJK

where J and K are the two letters to be converted. If only one letter is given the other is assumed to be a space.

Assembly Listing

The printout from DAP-16 is called an assembly listing, and gives the symbolic instructions in the order in which they appear. There is also an octal representation of the binary code which has been generated, which is grouped for easy reading.

- Column 1. Contains a decimal line count.
- Column 2. Memory location of each instruction.
- Column 3. Octal representation of machine code.

Pseudo Operations

These enable program linking and assembly control.

REL Relocatable. Tells the assembler to assemble the following instructions in a relocatable mode, so that they may be loaded at any place in the memory.

END Terminate this assembly pass. This must be the last statement in a source program.

BCI Binary Coded Information. Used to generate ISO code direct from alpha numeric data. Provides easy conversion where messages are to be output. The instruction takes the form

`BCI n, MESSAGE`

where `n` specifies that there are `2n` characters to be converted, and `MESSAGE` represents the character.

Block Starting with Symbol.

DAC Define Address Constant. This causes DAP-16 to generate a 16 bit binary word which will be used by a flagged memory reference instruction, to access an operand in any sector.

eg. `B DAC LAST`

would cause the address of `LAST` to be stored in this location and called `B`. A flagged instruction could then access `LAST` via `B`.

eg. `LDA* B`

would load the contents of the address specified in `B` ie `LAST`. This is also used for transferring data from a main program to a subroutine. eg.

Output

`JST ØUT` `ØUT DAC**`

`DAC A` `LDA* ØUT`

`IRS ØUT`

`JMP* ØUT`

DEC Decimal

Causes DAP-16 to generate binary data to represent decimal data. If more than one number is to be converted it may be

done by the same DEC by separating it from the others by a comma.

OCT Octal. Causes DAP-16 to generate binary data words from octal data. One or more words may be converted, provided they are separated by commas.

CALL For calling a subroutine. DAP-16 generates a Jump and store instruction to the symbolic address.

SUBR Directs DAP-16 to a name for external reference. The name may be 1 to 6 characters. Only the first 4 of which are used as the internal name. This internal name may be different, and is then defined by putting a comma followed by this name after the external name.

BSS Block Starting with Symbol. Allocates storage for variables.

BSZ Block starting symbol, initially zeros. Defines a block of zeros.

COMN Common. Used to assign absolute storage at the top end of memory. This may be used for communication of data from one assembler program to another, or from an assembler program to a Fortran program.

There is also a line address bus which decodes instructions for setting up the interfaces. A good example of how the buffer functions is during an interrupt caused by the teletype or other input device. Suppose an interrupt is generated by a printer (section 4.1.) This will cause the machine to stop what it is doing and go to the service routine. The letter 6 is transferred only as far as the buffer, and remains there until the service routine has stored the previous contents of the register into the

3.4. Relevant On Line Programming Techniques

"On line" refers to the computer being directly connected to some external piece of equipment, being able to input or output data to that device. Hence, the teletype is referred to as an "on line device", similarly the high speed reader and punch. Usually, when referring to normal peripherals the "on line" is dropped from the title, and reserved for those devices which are not normally connected in this way, e.g. the data logger.

In this section we are concerned with the machine code programming necessary to fetch data from external devices.

3.4.1. The Input/Output buffer

All information going into or out of the computer must pass through the A register, but, between this and the device is the buffer. It is a means of giving each of the 16 bits of the register a wire for input/output, and so consists of 16 bits. Information flows through the buffer from external devices, for example, when data is available on the wires it is in the buffer, but not the A register. A command (INA) is necessary to perform the transfer. There is also a ten line address bus which decodes instructions for setting up the interfaces. A good example of how the buffer functions is during an interrupt caused by the teletype or other input device. Suppose an interrupt is generated by typing a C, (section 4.1.) this will cause the machine to stop what it is doing and go to the service routine. The letter C is transferred only as far as the buffer, and remains there until the service routine has stored the previous contents of the A register and asks for the input. The letter is then brought into the register.

3.4.2. Techniques for the MDP 200 data logger

Data acquisition takes place in two separate stages.

- (a) The logger must be told what is required, and obtain this.
- (b) The data is returned to the computer.

The "Interface"

An interface in this context is a complicated electronic device which allows the signals to pass between the computer and logger.

It is necessary because they work at entirely different voltage levels, and at vastly differing speeds. During the system description reference will be made to setting up the interface, this means telling it which direction data is going to flow. It may be thought of as setting the points on a railway track to allow a train to pass. If the points are set wrongly the train is de-railed, similarly, if the interface is set for input, and an output command is given the program comes to a step which cannot be executed and "hangs-up".

Data Acquisition

Suppose we wish to "read" channel 22, to obtain the value the following procedure is necessary.

- (a) A command word for the data logger is assembled in the A register.
- (b) The interface is set to output mode.
- (c) The command is sent to the logger.
- (d) Because the logger is so much slower than the computer, the latter must wait for a signal to say the data is ready.
- (e) Set interface to input mode.
- (f) Accept data and transfer it into the A register.

This is a simplified version of what takes place, the full series of operations follows, each machine codes instruction is explained as fully as possible. The explanation is lengthy, but fundamental in this work.

The sequence begins when the appropriate command has been assembled in the A register.

(a) OCP '130

Output Control Pulse. '130

This command sets the interface into the output mode, ready to send the command out.

(b) SKS '130

Skip if not ready.

The interface is tested to see that the logger is ready to accept the command.

If it is ready the next instruction is executed, if not ready the next instruction is skipped.

(c) SKP

(c) SKP

Unconditionally skip the next instruction.

i.e. go to instruction (e)

(d) JMP *-2

Jump back two instructions.

i.e. to SKS '130. Considering

instructions (b), (c) and (d) together,

it can be seen that:- if the logger is

ready, control passes to instruction (e)

via (c). If it is not ready the computer

keeps on testing it. This will continue

until a ready state is detected and

control passes to instruction (e).

(e) OTA '1030

Output the A register to device '1030.

Once the interface and logger are ready, this instruction causes the contents of the A register to be sent through the buffer, interface and into the data logger. In addition, this instruction checks that output has been done, and skips the next instruction when this is true. i.e. goes to (g).

(f) JMP *-1

Jump back to previous instruction. If output has not been made by instruction (e), (f), is executed, this in effect says go back and try again. Hence the program cannot continue until a satisfactory output has been achieved.

(g) SKS '130

Skip if interface now ready.

This is to test that the interface has returned to its ready status after being used for output. This is necessary because the computer works so fast it could be trying to use the interface again before it was ready.

(h) JMP *-1

Jump back to previous instruction. If the interface was not ready this instruction is executed, and the device is tested again, this will continue until a ready status is detected.

(i) SKS '630 Skip when data logger ready.

(j) JMP *-1 Jump to previous instruction.

Taking these two together they say wait. (i) tests the data logger to see if it has the requested information ready for return. If the data is not ready, (j) is executed, causing the test to be done until the ready is true and (j) is skipped.

(k) OCP '30 Output Control Pulse '30.

Set interface into the input mode, ready to receive data from the logger.

(M) SKS '130 Skip

(N) SKP is together they are exactly the same as

(P) JMP *-2 (b) to (d), i.e. a test is made to see that the interface is ready to transmit data, and only when this is true does

control pass to (q)

(Q) INA '1030 Input the data from device '1030.

(R) JMP *-1 Jump to previous instruction.

The input instruction transfers the data from the logger to the A register, via the interface and buffer.

As with the OTA (e), if transfer is not executed the next instruction is executed, thus causing a "wait", until the input has been achieved.

The data requested is now in the computer's A register in BCD format. (See Section 2, 3).

Sign Transfer and BCD to binary conversion

Having the data back in the computer is only half way to completion, it still requires the sign to be set correctly and conversion to true binary form.

Conversion is achieved by a subroutine, written within the department, (28) which when called converts the number in the A register from BCD to binary. A similar routine is available for the reverse process.

The sign is tested by a separate instruction, at machine code level.

(u) SKS '530 Skip if sign of input is + ve. i.e.
 Skip the next instruction if the input
 is + ve, execute it if - ve.

(v) IRS NF Increment NF.
 This adds one to the location which has
 been designated NF, the negative flag.

A negative flag must be used here because the number is still in BCD format, and could not be negated by two's complementing.

Furthermore, the MDP 200 uses 16 digit bits, and a separate sign bit, this must therefore be reduced to 15 magnitude bits and one sign bit.

The sign can only be tested at this point, when the interface is set in the input mode and the data is on the input lines.

When the input has been converted to binary NF is tested and if required the number converted.

The instructions described above are performed each time an input is made from the data logger, however, because they are performed by a subroutine the user does not have to concern himself with them.

and these programming routines are all to be performed by the computer as a result of communicating between the user and the computer. The difficulty arises when data is to be transferred between the computer and the user. There are two methods:-

- (i) Argument Transfer - where the information is transferred to the subroutine immediately after the call statement and is accessible to different languages.
- (ii) Argument Transfer - where the information is transferred to the subroutine immediately after the call statement.

Storage in BASIC

each variable as it is encountered during the execution of the program. This means that only the original lower address of a variable is stored and if the program is changed by the addition of new lines the storage will change. The call statement is used to pass the location on to subroutines.

1.5.2. Common Storage

In FORTRAN the method of transferring data between main and sub programs is by defining it as "COMMON". When this is done, space is reserved for the common variables at the "top end" of the core store where it can be accessed by any of the programs.

3.5.1. Program Linking

There are certain tasks which can only be performed at assembler level and others which are best done in FORTRAN or BASIC. If the computer and these programming techniques are to be used fully then some way of communicating between the three languages is necessary. The difficulty arises when data is to be transferred between languages because storage methods differ, as will the names given to variables.

There are two methods:-

i) Common storage - where variables

are stored in a common area and accessible to different languages.

ii) Argument Transfer - where the

information is transferred to the subroutine immediately after the call statement.

Storage in BASIC When using BASIC, a storage location is given to each variable as it is encountered during the execution of the program. This means that only the compiler knows where each variable is stored and if the program is changed by the addition of new lines the storage will change. The call statement is used to pass the location on to subroutines.

3.5.2. Common Storage

In FORTRAN the method of transferring data between Main and Sub programs is by defining it as "COMMON". When this has been done, space is reserved for the common variables at the "top end" of the core store where it can be accessed by any of the programs.

If we have a detailed knowledge of the order in which the FORTRAN compiler allocates this storage it is possible to write an assembler program which will be able to use this data. This method was used by the Continuous Steerable Program. However, it is tedious and difficult to implement. This is because the FORTRAN compiler and DAP-16 assembler allocate common storage in entirely different ways, starting at different places in the core. FORTRAN common is set in reverse order to that in which it is declared, whilst DAP-16 is in the declared order. FORTRAN also has rules governing which order of declaration the different types of variables take. Appendix (4) gives a full explanation.

Comments on Common Storage method

The method requires extreme care and attention to detail; even following the directions given by the manufacturers of the machine it was very difficult to use this method.

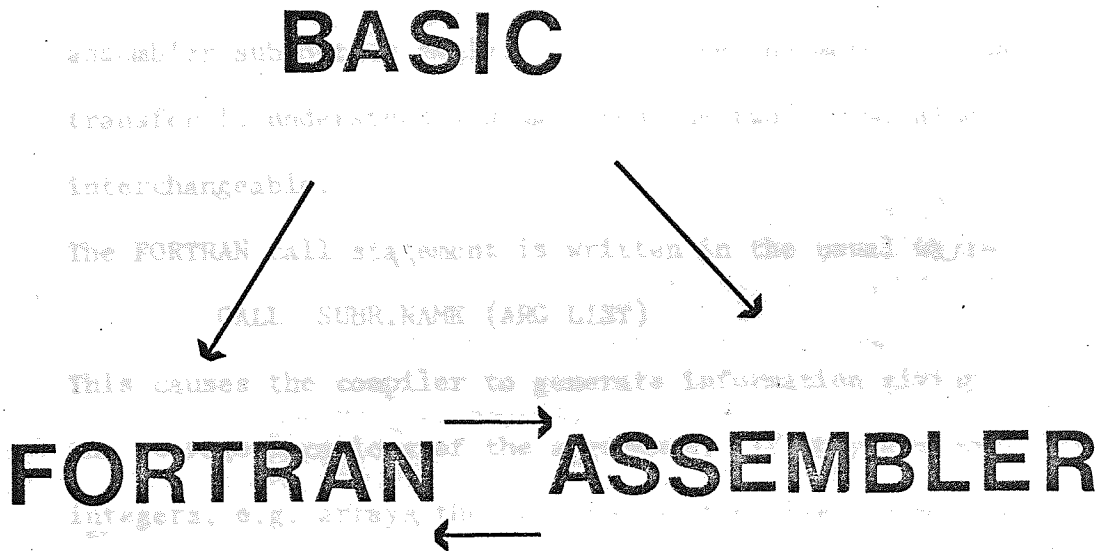
3.5.3. Argument Transfer

This method is the same as that used by the FORTRAN compiler and other Honeywell subroutines. In brief, it is not the argument itself which is transferred to the subroutine but the location at which the argument is stored. This enables the same transfer routine to be used for arguments which may be integer, floating point or double precision, because whatever type, their location will only require the transfer of one address.

A detailed description of how the argument transfer method works is given in section 3.5 as the author is certain many readers will be particularly interested in this.

Calling Methods

Programs in one language may call subroutines written in others according to the following diagram:-



BASIC/FORTRAN

This is done in the usual way; a BASIC statement of the form (Line number) CALL (N, ARG LIST) is used.

N is the subroutine number

ARG LIST the arguments

If the FORTRAN subroutine uses integers, a dummy subroutine is used to convert the required numbers from floating point, then calls the actual subroutine required.

BASIC/ASSEMBLER

Again the normal BASIC CALL is used, in the same form as above. When the assembler routine takes program control it calls the argument transfer subroutine which transfers the locations of the arguments.

FORTRAN/ASSEMBLER

These two languages are completely compatible because the FORTRAN compiler converts the source program into a series of assembler subroutine calls. Hence, once the method of data transfer is understood and mastered the two become almost interchangeable.

The FORTRAN call statement is written in the usual way:-

CALL SUBR.NAME (ARG LIST)

This causes the compiler to generate information giving the storage locations of the arguments. If they are not integers, e.g. arrays, the location of the first element is given. When the assembler routine takes over it uses the argument transfer subroutine to fetch the locations at which the arguments are stored. These can then be accessed by indirect addressing. For a full explanation see 2.2.3

ASSEMBLER/FORTRAN

This is the reverse of the above procedure, so the assembler routine must specify where it has stored the arguments immediately following the CALL.

```

Thus CALL NG T
      DAC ARG 1
      DAC ARG 2
      DAC ARG 3
      OCT 0

```

return made here

The octal zero is to tell the transfer subroutine that all the arguments have been transferred and that a return is to be made to the next instruction after execution of the subroutine.

Comments on Argument Transfer

This is the most convenient method of sending information between programs, because the programmer does not need to know the actual place where the arguments are stored. Programs written in this way were found to work first attempt, whereas similar programs using the common storage method had taken many attempts.

Writing up Interrupts

The program which is going to be interrupted must not contain instructions to allow the interruption. First it must specify a class of office or devices may interrupt it. It must then specify the action to be taken when the interrupt occurs. Finally it must enable the interrupt. If the interrupt is not enabled, the program will not be interrupted.

3.6.

Interrupts

An interrupt is a means of causing the computer to stop what it is doing, and go to do a different thing, independent of the program. Interrupts may be caused by,

- (i) Punch
- (ii) Reader
- (iii) teletype
- (iv) real time clock
- (v) power failure

To illustrate the idea, consider a power failure. The computer has a mechanism which detects the power has failed and causes an interrupt. This makes the computer stop executing the current program and go to a power failure routine. This makes a note of what the program was doing and where it was, it also sets up a return. When the power is restored the operator then only has to press the start button and the original program execution continues.

Setting up Interrupts

The program which is going to be interrupted must set up certain instructions to allow the interruption. First it must specify which device or devices may interrupt it. It must then specify where the execution is to continue when the interrupt occurs. Finally it must enable the interrupt. If the interrupt is not enabled it causes no action, but is stored until it is enabled, then it causes its action. When the interrupt occurs, execution of the first program ceases, and execution is transferred to a preselected point. Here the new program tests to see

which device has interrupted. The programmer can arrange a priority structure so that if a device of low priority interrupts, all the interrupts of higher priority are reset. In this way the execution of one interrupt can itself be interrupted while a higher priority device is serviced. Fig (7) shows a possible structure.

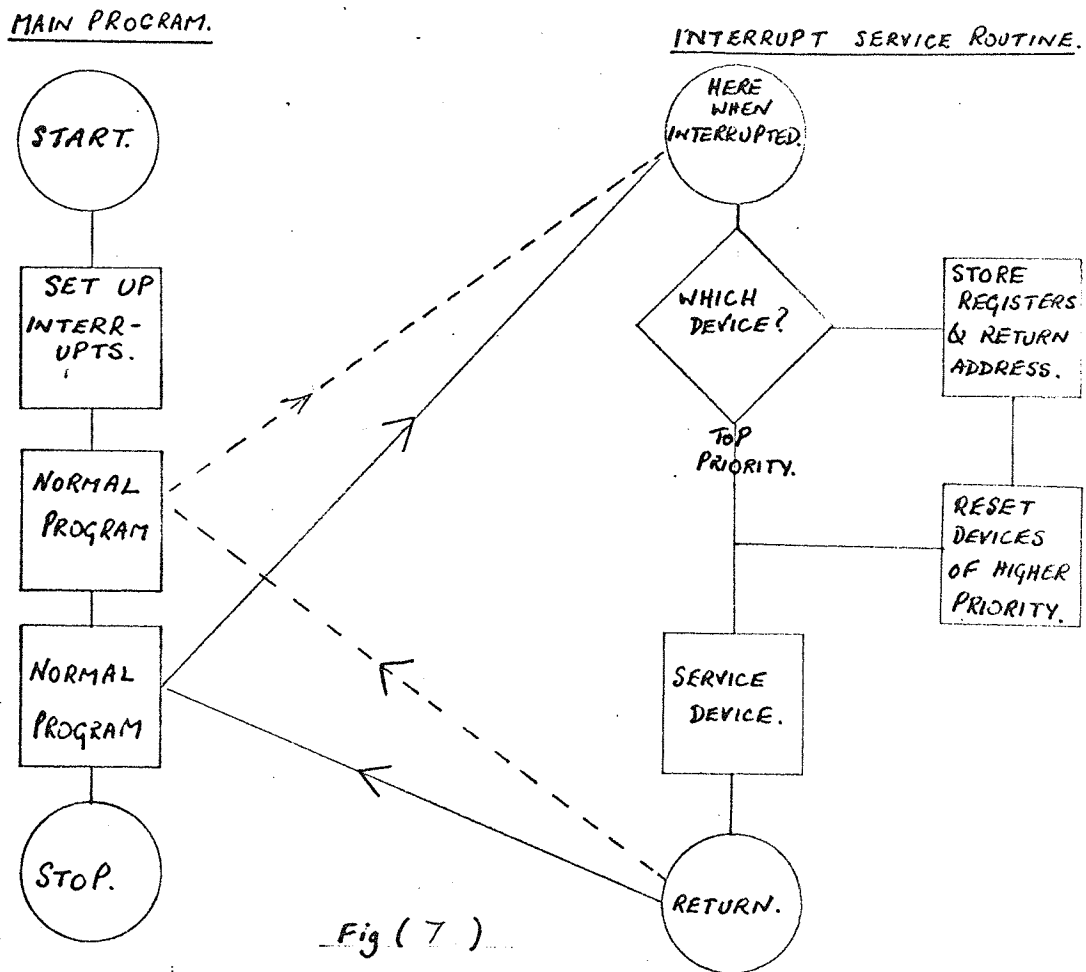


Fig (7)

Reader/Punch Interrupts

Can be used when a large amount of data is to be read or punched, with only a small amount of computation. Calculation occurs whilst the tape is moving between character positions. As soon as the character is in position an interrupt occurs and the character is transmitted.

Real Time Clock

The clock can be used in two ways:

(i) to cause an interrupt after a certain length of time.

Eg. in case a program has gone into a never ending loop

(ii) to time the running of a program.

The clock can be started and stopped by special instructions. Its method of operation is to pass one onto a dedicated memory location ('61) every 20 milliseconds. (50 pulses/second). An interrupt is generated when the location changes over from '177777 to '000000. The incrementing does not affect any other program operation.

To time a program

Times of up to $2^{15}/50$ seconds can be recorded. The dedicated location is set to zero at the start of the program and the clock started. At the end, the clock is stopped, and the time printed out.

To cause interrupt

In this case the length of time required is converted into a number of pulses by multiplying by 50, the number is then negated and put in the dedicated location. The number must be negative because the interrupt occurs when the number in the dedicated location changes from -1 to zero. The address where execution is to continue after interruption is put in the interrupt address, and the interrupt enabled.

When the interrupt occurs execution will continue from the specified place.

Teletype Interrupt

The teletype can cause an interrupt when any key is pressed, provided the interrupt is set up. The character which is pressed is transmitted

not as far as the computer's A register, but as far as the input buffer. The interrupt handling routine can then store the contents of the A register before transferring the transmitted character into this register. Pressing the key has therefore caused two actions, firstly it has caused the program to be interrupted, and secondly the character has been transmitted. This method is used for giving commands to the continuous scanning program, and in the BASOON interactive/on-line program for immediate attention to commands.

3.7. ERROR DETECTION

Three programming languages are in use in this system, each of which may detect an error condition. However, in each case different actions follow.

At the BASIC level an error causes a printout, giving the error type and program line number where it occurred. Program control is then returned to the operator so that he can take appropriate action. Similarly, FORTRAN errors generate an error message, but this is only a two letter mnemonic, giving a cryptic clue to the cause and stopping computer operation.

At machine code level the programmer must devise his own error detection mechanisms and build these into each program.

BASIC has been used throughout as the written language, the user having no control over the execution of the FORTRAN or machine code subroutines, once they are called, hence all the error detections must be via that part of the BASIC compiler which processes errors. The FORTRAN error routine was rewritten to give its normal two letter message followed by a direct transfer of control to the BASIC error handler. Thus when an error is detected in a FORTRAN subroutine (e.g. a "number overflow") this is indicated and control transferred to the BASIC error processor which gives the line number. Similarly, all the possible error conditions have been allowed for at the machine code level, here however, a two letter message is passed to the BASIC compiler which indicates to the user exactly what error has been made. For example, if a non-existent channel is requested from the data logger a message would be typed, and control return to the operator. Error processing is one of the most important features of any language, because the computer itself does not know that an error has been made. It is the error detection mechanisms which find mistakes then cause some action to be taken. If left to its own devices, the computer would carry on processing until a halt instruction was found. In the early part of BASOON development, this was a frequent occurrence.

4.1. The Continuous, Steerable Scanning Program (CSP)

This is a system developed to access a single channel of the data logger at variable time intervals. Facilities are available for changing the channel and time interval. Steering is by commands typed on the teletype; this allows the program to be controlled from anywhere in the building. All the subroutines used by the CSP, with the exception of the time interval input, are written in machine code via the assembler language, DAP-16.

This does limit the program, but the lessons learnt during its development were used to produce the Basoon system which supersedes it.

Figure (8), a flowchart, shows how the command interpreter and subroutines are organised.

A complete assembly listing is also provided in Figure (L2).

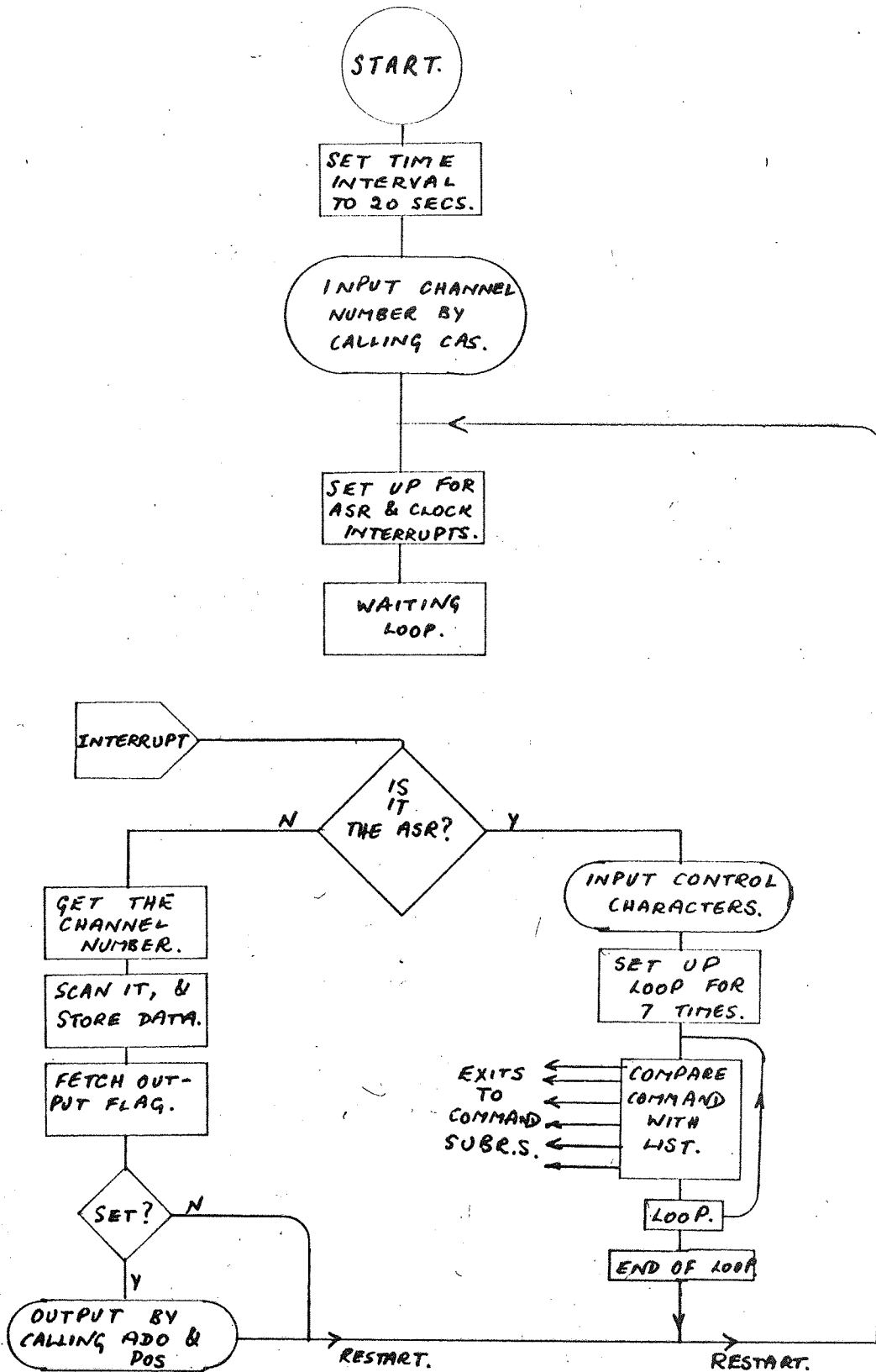


Fig (8). Continuous Steerable Scanning Program.

```

* CONTINUOUS SCANNER
0001 * CONTINUOUS SCANNER
0002 *
0003 * USES COMMON STORAGE.
0004 * FOR TRANSFER OF DATA.
0005 *
0006 REL
0007 SETC '26700
0008 * SET COMMON TO THE SAME.
0009 * AS FOR IRAM
0010 026677 TIMC COMN 1
0011 026676 COM COMN 1
0012 026675 PF COMN 1
0013 026674 LW COMN 1
0014 *
0015 00000 0 02 00136 SFT LDA IMC
0016 00001 0 04 26677 SIA TIMC
0017 00002 101000 BRN NOP
0018 00003 101000 NOP
0019 * SPACE FOR ALTERATIONS.
0020 00004 0 02 00025 LDA IA
0021 00005 0 04 00063 STA '63
0022 00006 0 02 26677 LDA TIMC
0023 00007 0 04 00061 STA '61
0024 00010 0 02 00155 LDA ='41
0025 00011 74 0020 SMK '20
0026 00012 14 0004 OCP '4
0027 00013 14 0020 OCP '20
0028 00014 000401 ENB
0029 *

```

0030	00015	140040		CBA	
0031	00016	0 35 00154	IRL	LD:	=-10000
0032	00017	101000		NOP	
0033	00020	101000		NOP	
0034	00021	0 12 00000		HRS	0
0035	00022	0 01 00017		JMP	*-3
0036	00023	141206		AOA	
0037	00024	0 01 00016		JMP	IRL
0038			*		
0039			*	TIME LOOP.	
0040			*		
0041	00025	0 000026	IA	DAC	++1
0042	00026	0 00 00000	IH	***	**
0043	00027	101000		NOP	
0044	00030	34 0404		SKS	'404
0045	00031	100000		SKP	
0046	00032	0 01 00066		JMP	CL0
0047	00033	54 1004		INA	'1004
0048	00034	0 01 00033		JMP	*-1
0049	00035	0 11 00153		CAS	= '212
0050	00036	100000		SKP	
0051	00037	0 01 00002		JMP	BGN
0052	00040	0 11 00152		CAS	= '215
0053	00041	100000		SKP	
0054	00042	0 01 00002		JMP	BGN
0055	00043	0 11 00151		CAS	= '240
0056	00044	100000		SKP	
0057	00045	0 01 00066		JMP	CL00

* CONTINUOUS SCANNER

```

0058 00046 141340 ICA
0059 00047 0 04 00070 STA HALF
0060 00050 0 10 00137 JSI INA4
0061 00051 0 06 00070 ADD HALF
0062 00052 0 04 00071 STA CONC
0063 00053 0 02 00072 LDA M7
0064 00054 0 04 00000 STA 0
0065 *
0066 * PROCESS COMMAND.
0067 *
0068 00055 0 02 00071 TEST LDA CONC
0069 00056 1 11 00102 CAS CC+7,1
0070 00057 100000 SKP
0071 00060 -1 01 00111 JMP* CPT+7,1
0072 00061 0 12 00000 IRS 0
0073 00062 0 01 00055 JMP TEST
0074 00063 14 0004 OCP '4
0075 00064 000401 ENB
0076 00065 -0 01 00026 JMP* IH
0077 *
0078 *
0079 00066 0 10 00000 CLO CALL CLOK
0080 00067 0 01 00002 JMP BGN
0081 *
0082 *
0083 00070 HALF BSS 1
0084 00071 CONC BSS 1
0085 00072 177771 M7 DEC -7
0086 *
0087 00073 150301 CC BCI 1,PA

```

0088	00074	141701		BCI	1,CA
0089	00075	141724		BCI	1,C1
0090	00076	150317		BCI	1,P0
0091	00077	151303		BCI	1,RC
0092	00100	152323		BCI	1,TS
0093	00101	147317		BCI	1,NO
0094			*		
0095	00102	0 000111	CPI	DAC	PAA
0096	00103	0 000113		DAC	CAA
0097	00104	0 000115		DAC	CIA
0098	00105	0 000120		DAC	POA
0099	00106	0 000122		DAC	RCA
0100	00107	0 000125		DAC	ISA
0101	00110	0 000133		DAC	NOA
0102			*		
0103			*		
0104			*	COMMAND AREA.	
0105			*		
0106			*		
0107	00111	000000	PAA	HLT	TEMP. STOP.
0108	00112	0 01 00002		JMP	BGN
0109	00113	0 10 00000	CAA	CALL	CAS CHANGE ADDRESS.
0110	00114	0 01 00002		JMP	BGN
0111	00115	0 10 00000	CTA	CALL	CTS CHANGE TIME INTERVAL.
0112	00116	0 10 00000		CALL	CRLF
0113	00117	0 01 00002		JMP	BGN
0114	00120	0 12 26675	POA	IRS	PF ALLOW PRINTING.

* CONTINUOUS SCANNING

Fig L2

5 of 6

0115	00121	0 01 00002	JMP	BGN	
0116	00122	0 10 00000	RCA	CALL	RCS CLOCK READ
0117	00123	0 10 00000	CALL	CRLF	
0118	00124	0 01 00002	JMP	BGN	
0119	00125	0 10 00000	PSA	CALL	CRLF
0120	00126	0 10 00000	CALL	RCS	
0121	00127	0 10 00000	CALL	ADD	
0122	00130	0 10 00000	CALL	POS	
0123	00131	0 10 00000	CALL	CRLF	
0124	00132	0 01 00002	JMP	BGN	
0125	00133	140040	NOA	CRA	INHIBIT OUTPUT.
0126	00134	0 04 26675	SIA	PF	
0127	00135	0 01 00002	JMP	BGN	
0128			*		
0129	00136	176030	IMC	DEC	-1000
0130			*		
0131			*		
0132	00137	0 000000	INA4	DAC	**
0133	00140	34 0104	SKS	'104	
0134	00141	0 01 00140	JMP	*-1	
0135	00142	14 0004	OCP	'4	
0136	00143	54 1004	INA	'1004	
0137	00144	0 01 00143	JMP	*-1	
0138	00145	34 0104	SKS	'104	
0139	00146	0 01 00145	JMP	*-1	
0140	00147	-0 01 00137	JMP*	INA4	
0141	00150	000000	HLT		

Fig L2
PAGE 6 of 6.

0142 *
0143 * THAT'S ALL.
0144 *

0145 00151 000240 END
00152 000215
00153 000212
00154 154360
00155 000041

BGN	000002	CAA	000113	CC	000073	CLP	000066
CONC	000071	COV	026676A	CPT	000102	CIA	000115
HALF	000070	IA	000025	IB	000026	IMC	000136
INA4	000137	IB	026674A	M7	000072	NOA	000133
PAA	000111	PF	026675A	POA	000120	NCA	000122
STI	000000	TEST	000055	TIMC	026677A	ISA	000125
TAL	000016						

0000 WARNING OR ERROR FLAGS

DAP-16 MOD 2 REV. C 01-26-71

AC

System description

The program runs continuously, at pre-determined time intervals the channel being studied is scanned and the result printed. The timing and immediate attention to commands is achieved by using the interrupt facilities of the H316, as described in section 3. Commands may be typed at any time (except when results are being output), and cause an interrupt - thus they receive immediate attention.

Commands

These take the form of two letters which are typed by the operator:-

PA Pause: This stops computer operation, but does not alter anything - to re-start the start button is pressed.

CA Change Channel: A ! is typed by the program and the machine waits for a two-digit number to be input.

CT Change Time: A message is typed asking for a revised time interval between scans to be given. This should be in the range 1 to 655 seconds.

PO Print Out: Sets a flag so that the result of the scan will be printed.

NO No Output: Re-sets the printing flag so that no printing takes place.

RC Read Clock: Prints the time in hours and minutes, as taken from the clock in the MDP 200 data logger. This is printed even if No Output mode is selected.

TS Immediate Printout: This causes an immediate printout to occur, as if the time interval had finished, and printing was due.

The method of interpreting the commands is described fully in section 4.2.

Initialisation

When the program tape is loaded and started an initial time delay of 20 seconds between readings is assumed, a ! is output, requesting the user to input the channel he requires to examine. This should be a two-digit integer. Mistakes may be corrected by inputting an @ which will cause the ! to be re-typed. It is assumed that output is required. Processing then begins, and the channel number, followed by the value of the input is typed every 20 seconds. These are modified as necessary by using the commands.

4.1.2. Subroutines Used

(a) Figure (L14) shows the FORTRAN subroutine used to ask for a new time interval. The message printed is:-

INPUT A NEW TIME INTERVAL, (I3)

I3 indicates that the expected number will be a three-digit integer.

Hence, the leading zero is significant.

010 = 10 seconds

100 = 100 seconds

```
C      CHANGE TIME INTERVAL
      SUBROUTINE CIS
      COMMON ITIME
      WRITE (1,1)
1     FORMAT(37H INPUT NEW TIME INTERVAL IN SECONDS,13)
      READ (1,2) A
2     FORMAT (F10.4)
      ITIME=IFIX((A*50.)+0.5)
      RETURN
      END
SO
```

Fig 2 14.

(b) ADO - Address Output. Figure (L15).

When called, this routine types the number of the channel currently being examined. The number to be output is stored in BCD form. Only the last two digits are required, so the first two are discarded by clearing the left-hand half of the A register. This leaves the proper digits in BCD. Output is achieved by moving the right-hand digit into the B register using a ROTATE instruction, then adding '260 to the BCD digit, converting it to ISO code. This can then be directly output to the teletype. The A register is then cleared before rotating the other digit back from the B register and following the same output procedure. This method of output is very simple, extremely fast, saving the trouble of converting the number from BCD to binary and using a FORTRAN output routine.

(c) BCDO - BCD Output. Figure (L16)

This subroutine outputs the current value of the channel being examined. As with ADO, a machine code output is used to save time and bulky programming. The main part of the subroutine is similar to ADO in that '260 is added to each of the four digits in turn and the result output to the teletype. Figure (9) shows the process diagrammatically.

The number to be output is initially in the A register, it is shifted into the B register and the A cleared. Each digit is brought in turn into the right-hand bits of the A register and '260 added before being output.

```

0001 * CHANNEL ADDRESS OUTPUT.
0002 *
0003 SUBR ADO
0004 REL
0005 SETC '26700
0006 026677 TIMC COMN 1
0007 026676 COW COMN 1
0008 00000 0 000000 ADO DAC **
0009 00001 140040 CRA
0010 00002 000201 IAB
0011 00003 0 10 00000 CALL CRLF
0012 00004 0 02 26676 LDA COW
0013 00005 141050 CAL
0014 00006 0400 74 LRL 4
0015 00007 0 06 00022 ADD = '260
0016 00010 0 10 00000 CALL OIA
0017 00011 140040 CRA
0018 00012 0410 74 LLL 4
0019 00013 0 06 00022 ADD = '260
0020 00014 0 10 00000 CALL OIA
0021 00015 0 02 00021 LDA = '240
0022 00016 0 10 00000 CALL OIA
0023 00017 0 10 00000 CALL OIA
0024 00020 -0 01 00000 JMP* ADO
0025 00021 000240 END
00022 000260
    
```

ADO 000000 COW 026676A TIMC 026677A

Fig L15

```

0001          * ECL OUTPUT.
0002          SUBI  ECDO
0003          REL
0004 00000    0 000000  ECDO DAC  **
0005 00001    000201      IAB
0006 00002    140040      CRA
0007 00003    0 15 00015  STX  SIR
0008 00004    0 35 00017  LDX  =- 4
0009 00005    140040      NXT  CRA
0010 00006    0412 74      LLR  4
0011 00007    0 06 00016  ADD  =' 260
0012 00010    0 10 00000  CALL  OTA
0013 00011    0 12 00000  IRS  0
0014 00012    0 01 00005  JMP  NXT
0015 00013    0 35 00015  LDX  SIR
0016 00014    -0 01 00000  JMP*  ECDO
0017 00015          SIR  BSS  1
0018 00016    000260      END
          00017    177774

```

```

ECDO  000000  NXT  000005  SIR  000015

```

0000 WARNING OR ERROR FLAGS

DAF-16 MOD 2 REV. C 01-26-71

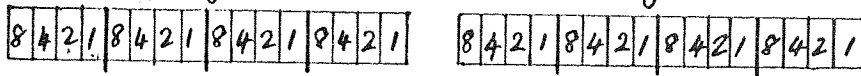
AC

Fig L.16.

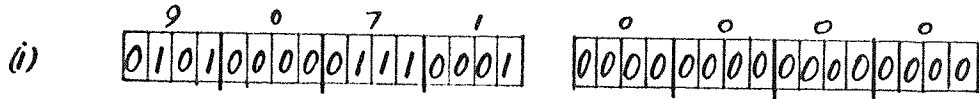
Control Program for the...

A register.

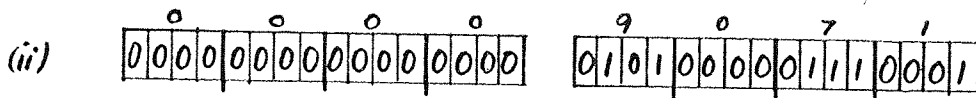
B. register.



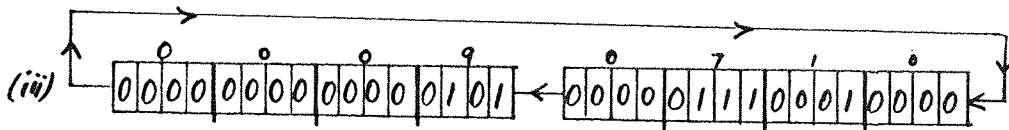
4 BCD digits, each comprising 4 binary bits.
Eg. BCD 9071. In the A register, this is:- (i).



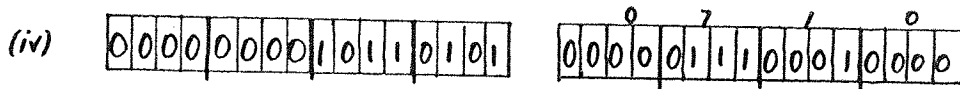
The A and B registers now have their contents interchanged, giving (ii)



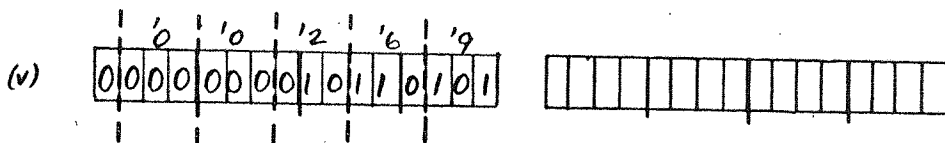
The two registers are now "rotated" left 4 places.



This puts the first digit, (9) in the A register, where octal 260 is added. (010 110 000 binary)



This is then "regrouped", ie read as an octal number, (A register only)



The three right hand digits are '269', the ISO code, which when output to the teletype will produce the digit 9.

The same procedure is followed for the remaining digits.

Fig (9).

Operation of Subroutine BCDO.

(d) CAS - Change Address Subroutine - Figure (L17)

Used to change the channel being scanned. Operation is the reverse of BCDO, i.e. a digit is input, the '260 subtracted from it, leaving the binary bits for that digit. (All these subroutines depend on the fact that the bit patterns of the digits 0 to 9 in BCD are the same as the code used for input/output if '260 is added/subtracted as appropriate).

Hence, the address is made up by inputting the first digit, chopping '260 off it, shifting it to the left and storing this "first half" of the address. The second half is then input and chopped, and the first half added, giving the two-digit address. This has '10000 added to give the full code for output to the data logger.

(e) CLOK - Clock read - Figure (L18)

After the pre-determined time, a clock interrupt is generated by the computer. This transfers program control to the subroutine CLOK which gives the full output, consisting of:-

time - from the MDP 200 clock
channel number - by calling ADO
Channel value - by calling a FORTRAN output
subroutine (POS).


```

* CHANGE ADDRESS.
0001
* CHANGE ADDRESS.
0002          SUBR  CAS
0003          REL
0004          SEIC  '26700
0005          026677  TIME  COMN  1
0006          026676  COM  COMN  1
0007 00000  0 000000  CAS  DAC  **
0008 00001  0 10 00000  CALL  CRLF
0009 00002  0 02 00034  LDA   ='241
0010 00003  0 10 00000  CALL  01A
0011 00004  0 10 00000  CALL  INA
0012 00005  0 07 00033  SUB   ='260
0013 00006  0414 74     LBL   4
0014 00007  0 04 00027  STA  HALF
0015 00010  0 10 00000  CALL  INA
0016 00011  0 07 00033  SUB   ='260
0017 00012  0 06 00027  ADD  HALF
0018 00013  0 06 00032  ADD  ='10000
0019 00014  0 04 26676  STA  COM
0020 00015  0 10 00000 AI  CALL  INA
0021 00016  0 11 00031  CAS  ='300 P
0022 00017  100000     SKP
0023 00020  0 01 00001  JMP  CAS+1
0024 00021  0 11 00030  CAS  ='215
0025 00022  100000     SKP
0026 00023  100000     SKP
0027 00024  0 01 00015  JMP  AI

```

Fig L17
PAGE 2 of 2.

0028 00025 0 10 00000 CALL CRLF
0029 00026 -0 01 00000 JMP# CAS
0030 00027 HALF BSS 1
0031 00030 000215 END
00031 000300
00032 010000
00033 000260
00034 000241

AI 000015 CAS 000000 COL 026676A HALF 000027
TIMC 026677A
0000 WARNING OR ERROR FLAGS
DAP-16 MOD 2 REV. C 01-26-71

AC

```

0001      * CLOCK
0002      SUBR  CLOK
0003      REL
0004      SETC  *26700
0005      026677  TIMC COMN 1
0006      026676  COL  COMN 1
0007      026675  PF   COMN 1
0008      026674  IW   COMN 1
0009      *
0010 00000  0 000000  CLOK DAC  **
0011 00001  0 02 26676  LDA  COW
0012 00002  0 10 00000  CALL FECH
0013 00003  0 04 26674  STA  IW
0014 00004  0 02 26675  LDA  PF
0015 00005  101040  SNZ
0016 00006  -0 01 00000  JMP* CLOK
0017 00007  0 10 00000  CALL ACS
0018 00010  0 10 00000  CALL ADD
0019 00011  0 10 00000  CALL POS
0020 00012  0 10 00000  CALL CRLF
0021 00013  -0 01 00000  JMP* CLOK

```

0022 END

CLOK 000000 COW 026676A IW 026674A PF 026675A

TIMC 026677A

0000 WARNING OR ERROR FLAGS

DAP-16 MOD 2 REV. C 01-26-71

AC

Fig L18

The subroutines used are:-

RCS - Fig. (L19)

ADO - Fig. (L15)

POS - Fig. (L20)

CRLF - Fig. (L21)

OTA - Fig. (L22)

(f) POS - Printout Subroutine - Figure (L20)

This is a FORTRAN subroutine which is called from a machine code main program to output a decimal number.

(g) CRLF - Carriage return - line feed - Figure (L22)

Outputs one new line of the teletype - used for producing neat printouts.

4.1.3. Conclusion on CSP

This program system worked satisfactorily, but was very limited in application because the majority of the programming was in assembler. Expansion of this system would have been possible, but only along restricted lines. Thus, the next development would have been to allow access to more than one channel. It was realised, however, that some much more flexible system was required. This led to the use of FORTRAN as the main language, and almost simultaneously, BASIC, giving the Basoon system.

Fig L19
PAGE 1/2

* READ CLOCK SUBR.

```
0001 * READ CLOCK SUBR.
0002 * THE HOURS & MINS ARE PRINTED,
0003 * & IF SS1 IS SET SECS & TENTHS
0004 * ARE ALSO PRINTED.
0005          SUBR  RCS
0006          REL
0007 00000  0 000000  RCS  DAC  **
0008 00001  0 02 00030  LDA  ='240
0009 00002  0 10 00000  CALL  OIA
0010 00003  0 10 00000  CALL  OIA
0011 00004  0 02 00027  LDA  ='130000
0012 00005  0 10 00000  CALL  FECD
0013 00006  0 04 00024  STA  HM
0014 00007  101020      SS1
0015 00010  0 01 00014  JMP  HMP
0016 00011  0 02 00026  LDA  ='110000
0017 00012  0 10 00000  CALL  FECD
0018 00013  0 04 00025  STA  SECS
0019 00014  0 02 00024 HMP LDA  HM
0020 00015  0 10 00000  CALL  BCDD
0021 00016  101020      SS1
0022 00017  0 01 00022  JMP  FIN
0023 00020  0 02 00025  LDA  SECS
0024 00021  0 10 00000  CALL  BCDD
0025 00022  140040      FIN  CRA
0026 00023  -0 01 00000      JMP*  RCS
0027          *
0028 00024          HM  BSS  1
0029 00025          SECS BSS  1
0030 00026  110000      END
```

Fig L19
PAGE 2 of 2.

00027 130000

00030 000240

FIN 000022 HY 000024 HYP 000014 ROS 000000

SECS 000025

0000 WARNING OR ERROR FLAGS

DAP-16 MOD 2 REV. C 01-26-71

AC

```
C SUBROUTINE PRINT OUT.  
SUBROUTINE POS  
COMMON IW, ICOW, IPF, ITIMC  
W=FLOAT(IW)  
WRITE (1,1) W  
1 FORMAT (F10.4)  
RETURN  
END
```

```
$0.0000 0.10 00000  
0.0000 0.02 00000  
0.0004 0.10 00000 CALL 011  
0.0005 -0.01 00000 JPF* CALL  
0.0006 0.00018 END  
0.0007 0.00015
```

Fig 120

000000

ENDING ON ERROR CLAP

Fig 121

* CRLF

PAGE

1

0001

* CRLF

0002

*CARRIAGE RETURN,

0003

* LINE FEED.

0004

*

0005

SUBR CRLF

0006

REL

0007 00000

0 000000

CRLF DAC **

0008 00001

0 02 00007

LDA ='215

0009 00002

0 10 00000

CALL 01A

0010 00003

0 02 00006

LDA ='212

0011 00004

0 10 00000

CALL 01A

0012 00005

-0 01 00000

JMP* CRLF

0013 00006

000212

END

00007

000215

CRLF 000000

0000 WARNING OR ERROR FLAGS

DAP-16 MOD 2 REV. C 01-26-71

AC

Fig L21

* OUTPUTER.

PAGE

1

```
0001 * OUTPUTER.  
0002 SUBR OIA  
0003 REL  
0004 00000 0 000000 OIA DAC **  
0005 00001 14 0104 OCP '104  
0006 00002 74 0004 OIA '4  
0007 00003 0 01 00002 JMP *-1  
0008 00004 34 0104 SKS '104  
0009 00005 0 01 00004 JMP *-1  
0010 00006 -0 01 00000 JMP* OIA  
0011 END
```

OIA 000000

0000 WARNING OR ERROR FLAGS

DAP-16 MOD 2 REV. C 01-26-71

AC

Fig L 22 .

```

0001      * INP01 SUBR. INA
0002      *
0003      SUBR  INA
0004      REL
0005 00000  0 000000  INA  DAC  **
0006 00001  34 0104      SKS  '104
0007 00002  0 01 00001  JMP  *-1
0008 00003  14 0004      CCP  4
0009 00004  54 1004      INA  '1004
0010 00005  0 01 00004  JMP  *-1
0011 00006  34 0104      SKS  '104
0012 00007  0 01 00006  JMP  *-1
0013 00010  -0 01 00000  JMP* INA
0014      END

```

INA 000000

0000 WARNING OR ERROR FLAGS

DAP-16 MOD 2 REV. C 01-26-71

AC

Fig L 23.

4.2. Operation of Command Interpreter part of program

This is the section of program which examines two letters, and exits to various places depending what these letters are. The two letters are stored in ISO code, each being represented by 8 binary bits, the two letters thus form one complete 16 bit word. This word is compared in turn with the 16 bit words formed by the commands, these are stored in an array called CC. This is done by loading -7 into the index register (because there are 7 commands to be tested), and using an indexed instruction inside a loop. That is, the contents of the index register are added to the address before execution of the command. The command CAS CC+7,1 does this. CC+7 is the last word of the array, and ,1 specifies indexing. Hence, the first time through, when -7 is in the index register, this is added to CC+7 giving (CC+7-7) which is CC, the first command. The index register is then incremented by +1 giving -6, and the next time through the command is compared with CC+7-6 = CC+1 ie the next word.

(See program lines 68 to 73. Fig. (L2))

If no match is found after comparison with all seven commands in the array CC then no action is taken. However, when a match is found an exit must be made. The instruction CAS is a "Compare and Skip", which has the following action, eg. CAS NØM

JMP GR

JMP EQ

JMP LE

If the A register is greater than NØM the next instruction will be executed, ie a jump will be made to GR. If the A register is equal to NØM the next instruction will be skipped, in this case a jump to EQ will occur. Finally when the A register is less than NØM the next two instructions are skipped and in this case the jump to LE would occur. In the actual program the instructions are:

```
TEST LDA    CØNC
      CAS    CC+7,1
      SKP
      JMP*   CPT+7,1
```

IRS 0
JMP TEST

The only case we are interested in is when the command given is equal to the command in the array. Hence, the other two options must be considered and blocked. In both cases all that is required is that the next loop should be executed. So, putting a SKIP after the CAS makes both greater than and less than option increment the index register and return for the next loop. Only when the command and array are equal will the jump instruction be executed.

The 7 addresses of the subroutines represented by the commands are stored in the array CPT. When the jump is made the index register still contains the correct command number, which can be added to CPT+7 to give the correct subroutine starting address. The * after the JMP means indirect addressing, see section (Appendix 1). In the case where the input command is not equal to any of the commands in the array, the program returns to the point where it was before being interrupted.

BASOON - INTRODUCTION & DEVELOPMENT

5.1. Given the systems described in section 2, the problem was to develop software which would make all the functions of the MDP 200 available to the computer. At the initial stages it was recognised that people with varying programming skills would be using the system, and that basically the same operations would be required by each of them. Some means of timing was desirable so that samples could be taken at regular time intervals. The H 316 real time clock was chosen for this because it is within the computer and could cause an interrupt when necessary.

System Description

The programming is done in the standard form of Honeywell BASIC, using a suite of subroutines to perform the data logging and timing operations. There are five subroutines called by the Basic statement CALL (----) where the arguments in the bracket are used to specify which subroutine is being called and transfer data to and from the main program.

Calling sequences

To call a subroutine the Basic instruction is:-

```
CALL (N,A,B,C,D - - - -M)
```

N,A,B,C,D etc are referred to as the arguments.

N is the subroutine number, from 1 to 9 and specifies which subroutine is being called, this is instead of a name.

The remaining letters represent data which is being transferred from Basic to the subroutine, and/or back again. There need not be any arguments except N.

Calling by value is also possible, but not recommended. Thus

```
10 A=10  
20 CALL(2,A,B)
```

could be written

```
10 CALL(2,10,B)
```

and would have the same effect. It must be noted that if values are used for arguments which are set by the subroutine, eg answers, then these will be lost, because there is no way of knowing where they are stored.

As a single digit is used as the subroutine reference, up to 9 different ones may be called. In Basoon, five are used, they are:-

1. Timing and delay
2. Inputs a value from a channel.
3. Reads MDP 200 clock, and digit switches.
4. Tests a specified sense switch.
5. Audible warning.

Program linking

Using a normal high level language such as ALGOL or FORTRAN it is a simple matter to call a subroutine which will perform some task, and then return control to the main program. However, with the system being described this is not readily possible, because the actual data acquisition and interrupt handling must be done by machine code programs. Furthermore

the incoming data, and outgoing commands to the data logger must all be in BCD number format, this necessitates a conversion for which no high level language could be used. The problem is complicated by the fact that Basic uses only floating point numbers, each of which require two words of storage, whilst the values from the data logger will be integer even after conversion from BCD to binary. These problems were overcome by using an intermediate stage between Basic and Machine code to perform the necessary number conversions (Integer floating point). Thus the complete cycle would involve transferring data from Basic through Fortran to machine code, here it would be used as a command to the data logger, or interrupt system, and the result passed back.

5.2. THE BASOON SUBROUTINES - GENERAL

Each of the subroutines is described with its calling sequence, operation and error diagnostics. (The order is of increasing complexity).

Audible Warning

This subroutine is used to draw the operator's attention by ringing the bell on the teletype - no other action is performed.

Calling sequence: CALL (5)

When the call is executed there are no arguments to be transferred, so it is a direct call from Basic to machine code. At this level the appropriate code is output to the teletype, and a return made to Basic. This is the

simplest subroutine because it does not involve any data transfer, and no errors are possible.

Sense Switch Test.

This operation tests any one of the four sense switches on the computer control panel, or their duplicates which are wired in parallel on the MDP 200 mobile cabinet. No data is transferred from Basic, but an answer is brought back.

Calling Sequence CALL (4,N,R)

4 is the subroutine number

N the number of the switch to be tested, and R the result of the test. If the switch is set R becomes equal to 1 otherwise R equals 2.

This is a standard Honeywell library subroutine written in Fortran, so no special programming was called for. It should be noted that sense switch one is used to cause a program break in Basic, and is therefore not available for user options.

MDP 200 Clock and Digital Switch Read

This routine performs the dual function of reading the time from the MDP clock, and inputting a value in the range 0 to 9999 from the digit switches on the mobile cabinet. These switches take the form of thumb wheels which are turned until the required digit shows in a small window.

Calling Sequence CALL(3,H,S,D)

3 is the subroutine number

When the call is executed, control passes to a Fortran subroutine which itself calls a machine code subroutine to access the data logger. At this level, a data word is sent out from the computer instructing the logger to read the hours and minutes from its clock. When the computer receives a signal to say this is ready, it accepts the value, converts it from BCD to a binary number and stores it. This procedure is then repeated for the seconds and tenths of seconds followed by the digit switches. The three words of information are now in the computer, but only available at the machine code level, as integers. These are transferred back to the Fortran level, and here converted to the floating point numbers which are taken back to the Basic level. Here it is set equal to a four digit number representing the hours and minutes, between 0000 and 2359. Similarly, S is the seconds and tenths in the range 0000 to 5999 and D the digit switch value 0 to 9999.

In this subroutine data is only being transferred in one direction, from the logger to the computer.

Channel input

Each of the 39 channels can be 'read' by the computer. The system is capable of expansion to 99 channels in steps of 10 by adding modules. Any type of electrical analogue input may be used, the ones currently in operation with this machine are :-

- Thermocouples (Cr-Al)
- Pressure transducers
- Differential pressure transmitters
- Flow meters

Calling Sequence CALL(2,C,V,)

2 is the subroutine number

Execution of this call causes V to be set equal to the value of the input on channel C. This involves passing the value C, in the range 1 to 39, to the logger, and returning with the input value. The process is as follows. A Fortran subroutine is called which converts the value of C from floating point to integer. At this point a machine code subroutine is called to access the data logger. This does the actual data logging, and passes the input value back to the Basic program via the Fortran level. So, by one simple call statement the whole series of data logging instructions at machine code level have been performed. This enables a person with very little knowledge of the system to use it effectively.

Error Diagnostics

In this subroutine there are two possible error conditions. One that the channel number specified is less than or equal to zero, two, channel number greater than 39. These are indicated by error messages in the usual Basic format, that is, two letters representing the type of error, followed by the line number at which the error occurs.

Thus, CU indicates 'Channel Under flow' ($C \leq 0$)

CO 'Channel Overflow' ($C > 39$)

At first sight this may not appear necessary, but it was found that meaningless answers were produced if non-existent channels were interrogated.

Timing and delay, Subroutine No. I.

This is the most complex subroutine, and contains programming to handle the computer's real time clock, together with simultaneous interrupt from the teletype and real time clock. During timing a delay may be called for, in which case the delay time is incorporated in the elapsed time without loss.

Calling Sequence CALL (1,T,D,E)

1 is the subroutine number.

This routine is called in three different ways; to start and stop timing and to cause a delay.

The argument T specifies which function is being performed by this call

i.e. T=1 for a delay
T=2 to start timing
T=3 to stop timing

D is the required delay, in seconds. This may be in the range 0.02 to 655.0. (The computer can handle numbers in the range $\pm 2^{15}$ -1, $655 \times 50 = 32750$ \therefore this is the biggest possible delay).

T is the elapsed time (when timing).

Although both D and T are not used at the same time, both must be used in the call statement for completeness.

Timing

When called for timing the action of the subroutine is to start the computer's real time clock (T=2), then return to normal execution of the BASIC program. To retrieve the elapsed time another call is made, with T=3. This causes the time, in seconds, to be allocated the name T. Time is counted in 50ths of a second, so very accurate timing is possible using this mechanism.

Delay

A delay is called for by putting T=1, and D equal to the required delay. In operation the delay is set on the real time clock, while the computer goes into a waiting program. Computation may be continued, but the usual reason for a delay is to keep the machine running awaiting further action, e.g. a sample, is to occur. At the end of the delay the clock causes an interrupt, which brings the computer out of the waiting loop and returns

to the Basic program. At an early stage it was found that once a delay was entered the operator was incapable of taking any action to control his program. In this situation the teletype interrupt mechanism provided an ideal solution. The delay program has been written so that during the delay the operator may cause action by typing commands on the teletype. The length of the delay is unaffected, because the real time clock within the computer is running 'under' the program. Typing a C causes program control to be returned directly to the operator, ie. the 'command mode' of Basic. An 'R' (for 'return') causes the delay to be terminated, but program execution to be continued as if the delay had run its full period. If timing was in progress as well as a delay, and an R is typed the elapsed time will be the actual elapsed time, and not that which would have included the full delay time. Any character other than C or R will be ignored, having no effect on the timing or delay.

Timing and Delay

The same clock, that in the computer, is used for both timing and delay. However, it is possible to use it for both 'simultaneously'. Hence, even though timing is in progress, a delay may be used. This is done at the machine code level.

Error diagnostics

There are four error conditions possible in this subroutine, each of which is identified by its own message, these are:-

RU 'Running' - this indicates that a call has been made to start the clock timing, whilst timing is already in progress.

NR 'Not running' - the user has tried to stop timing before he has started.

TW 'Type wrong' - argument I (T in the example) is not a 1, 2 or 3.

TU 'Time underflow' - a delay of less than 0.02 seconds has been re-
requested, which is out of the permitted range.

If a delay greater than 655 seconds is called for an error occurs in the
Fortran number conversion, which again results in detection and a printout
from Basic.

Program Examples

To illustrate the use of the Basoon system the following examples have been
chosen, they are written in Honeywell Basic.

Example 1 To input the values of channels 1 to 25 then ring the bell.

```
10 DIM K(25)
20 FOR S=1, 25
30 CALL (2,S,K(S))
40 NEXT S
50 CALL (5)
60 STOP
```

Example 2 To read the digit switch on the MDP 200 mobile cabinet.

```
10 CALL (3,A,B,C)
20 PRINT C
30 STOP
```

Example 3 To read the time from the MDP 200 clock.

```
10 CALL (3,A,B,C)
20 PRINT A; 'HOURS/MINS'; B; 'SECS, SECS/10'
30 STOP
```

Example 4 To test the states of all sense switches.

```
10 PRINT 'SENSE SWITCH STATES ARE----'
20 FOR F=1,4
```

```
30 M=0
40 CALL (4,F,M)
50 IF M=1 THEN PRINT F; "SET"; GO TO 70
60 PRINT F; "RESET"
70 STOP
```

Example 5 To time a calculation

```
5 A=0
10 CALL (1,2,A,A)
20 REM A IS A DUMMY ARGUMENT
30 FOR H=1, 1000
40 G= H+1
50 NEXT H CALL (1,3,A,T)
60 CALL (1,3,A,T)
70 REM A IS A DUMMY
80 REM T THE ELAPSED TIME
90 PRINT "THAT TOOK"; T; "SECONDS"
100 STOP
```

For the above example the time will be about 6.06 seconds.

Example 6 To cause a delay.

```
5 D=0: A=0
10 CALL (1,1,D,A)
20 REM D IS THE DELAY
30 REM A IS A DUMMY
40 STOP
```

Example 7 To scan a channel every N seconds.

```
10 INPUT N,C
20 REM N IS TIME BETWEEN SCANS, CHANNEL C
30 CALL (1,2,A,A): REM START TIMING
40 CALL (2,C,V)
50 PRINT V
60 CALL (1,3,A,T)
70 D=N-T
80 CALL (1,1,D,A)
90 GOTO 30
```

5.3. The Basoon subroutines - detailed description

This section comprises a detailed description of the operation of each of the assembler subroutines, together with its calling sequence through the different programming languages.

5.3.1. Subroutine 1. Timing and Delay

This is the largest and most important subroutine in the series.

Figure FS1 is a flowchart and Figure L3 a complete assembly listing.

The calling sequence begins with the BASIC statement:-

```
CALL (1,ARG1,ARG2,ARG3)
```

This passes control to the FORTRAN level, where the program FTIME takes over. This is listed in Figure L4.

FTIME does the following:-

(TYPE OF CALL) ARG1 converted from floating point to integer.

(TIME DELAY) ARG2 the number of seconds in the delay - this is multiplied by 50, to give the number of clock increments and converted to a negative integer.

(ELAPSED TIME) ARG3 Set equal to zero.

N.B. On the return from the assembler subroutine the number of clock increments is divided by 50 converted from integer to floating point and stored in ARG3.

FTIME then transfers control to the assembler level by calling

TIM by the FORTRAN statement:-

```
CALL TIM (ARG1,ARG2,ARG3)
```

Program control is then passed to the assembler level where the required action takes place.

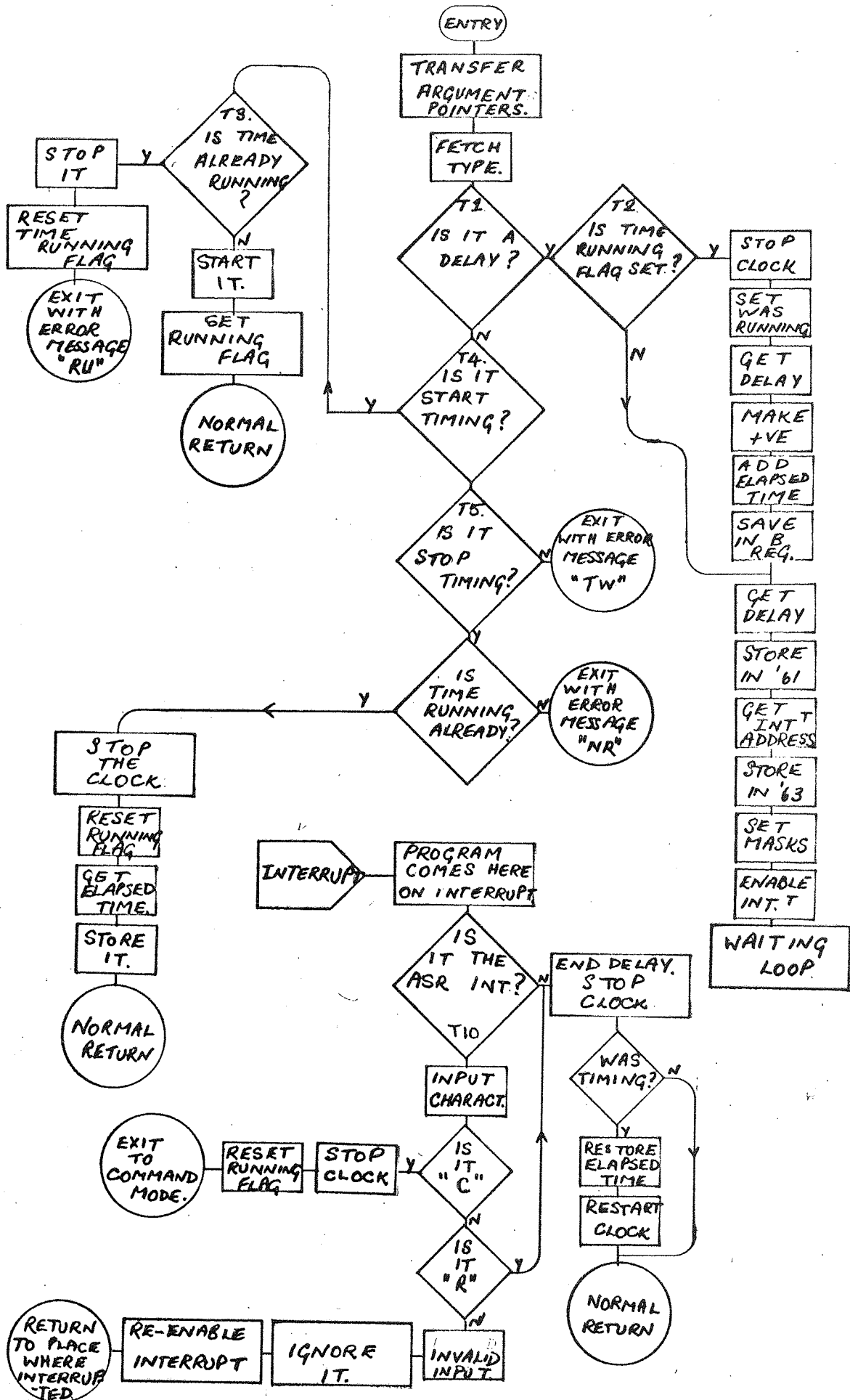


Fig F51 FLOWSHEET FOR TIMER/DELAY SUBROUTINE.

* TIME DELAY/MARKER FOR BASIC.

Fig L3
PAGE 1 of 8.

```

0001      * TIME DELAY/MARKER FOR BASIC.
0002      *
0003      * CALLED FROM FORTRAN BY:-
0004      * CALL TIM(IA,IB,IC)
0005      *
0006      * IA CODE NO.
0007      *           1 DELAY
0008      *           2 START TIME
0009      *           3 STOP TIMING.
0010      * IB TIME DELAY IN SECONDS*(C-50)
0011      * IC TIME USED IN SECONDS.
0012      *
0013      * ERROR MESSAGES, VIA BASIC.
0014      * I# TYPE WRONG.
0015      * NR CLOCK NOT RUNNING.
0016      * RD CLOCK ALREADY RUNNING.
0017      * TD TIME UNDERFLOW.
0018      *
0019      *
0020      *           SOBR   TIM
0021      *           REL
0022 00000  0 000000  TIM  DAC  **
0023 00001  0 10 00000  CALL  FSAI
0024 00002  000003  OCT  3
0025 00003  000000  FP  OCT  0
0026 00004  000000  DP  OCT  0
0027 00005  000000  RI  OCT  0
0028 00006  101000  NOP
0029      *
    
```

0030	00007	-0 02 00003	LDA*	FP
0031	00010	0 11 00162 11	CAS	DI
0032	00011	100000	SKP	
0033	00012	0 01 00043	JMP	IR2
0034	00013	0 11 00163 14	CAS	IS2
0035	00014	100000	SKP	
0036	00015	0 01 00033	JMP	IR1
0037	00016	0 11 00164 15	CAS	IE
0038	00017	100000	SKP	
0039	00020	100000	SKP	
0040	00021	0 01 00147	JMP	ERS
0041	00022	0 02 00165	LDA	IRF
0042	00023	101040	SNZ	
0043	00024	0 01 00151	JMP	ERNR
0044		*		
0045	00025	14 0220 A	OCF	'220
0046	00026	140040	CRA	
0047	00027	0 04 00165	SIA	IRF
0048	00030	0 02 00061	LDA	'61
0049	00031	-0 04 00005	SIA*	RI
0050	00032	-0 01 00000	JMP*	IIM FIRST RETURN.
0051		* 13		
0052	00033	0 02 00165 1RI	LDA	IRF

* TIME DELAY/MARKER FOR BASIC.

0053	00034	100040		SZE	
0054	00035	0 01 00153		JMP	ERR 1
0055			*		
0056	00036	140040	B	CRA	
0057	00037	0 04 00061		STA	'61
0058	00040	14 0020		OCP	'20
0059	00041	0 12 00165		IRS	IRF
0060	00042	-0 01 00000		JMP*	TIM SECOND RETURN.
0061			*	16	
0062	00043	0 02 00165	IR2	LDA	IRF
0063	00044	101040	12	SNZ	
0064	00045	0 01 00054		JMP	DLY
0065			*		
0066	00046	14 0220	C	OCP	'220
0067	00047	0 12 00166		IRS	IRF
0068	00050	-0 02 00004		LDA*	DP
0069	00051	140407		ICA	
0070	00052	0 06 00061		ADD	'61
0071	00053	000201		IAB	
0072			*		
0073	00054	-0 02 00004	DLY	LDA*	DP
0074	00055	0 11 00171		CAS	M1
0075	00056	101000		NOP	
0076	00057	0 01 00160		JMP	ERR 10
0077	00060	0 04 00061		STA	'61
0078	00061	0 02 00072		LDA	IA

0079	00062	0 04 00063	STA	'63
0080	00063	0 02 00174	LDA	'41
0081	00064	74 0020	SKP	'20
0082	00065	14 0004	OCP	'4
0083	00066	14 0020	OCP	'20
0084	00067	000401	ENB	
0085	00070	101000	WAIT NOP	
0086	00071	0 01 00070	JMP	*-1
0087			*	
0088			* WAITING LOOP.	
0089			*	
0090	00072	0 000073	IA DAC	++1
0091	00073	0 00 00000	IB ***	**
0092	00074	101000	NOP	
0093	00075	101000	NOP	
0094	00076	34 0404	SKS	'404
0095	00077	100000	SKP	
0096	00100	0 01 00125	JMP	CLO
0097	00101	54 1004	INA	'1004
0098	00102	0 01 00101	JMP	*-1
0099	00103	0 11 00173	CAS	'322 R
0100	00104	100000	SKP	
0101	00105	0 01 00125	JMP	CLO
0102	00106	0 11 00172	CAS	'303 C
0103	00107	100000	SKP	
0104	00110	0 01 00120	JMP	CMOD

* TIME DELAY MARKER FOR BASIC.

Fig L3
PAGE 5 of 8

0105	00111	101000	NOB	
0106	00112	0 02 00174	LDA	'41
0107	00113	74 0020	SMK	'20
0108	00114	14 0004	OCP	'4
0109	00115	14 0020	OCP	'20
0110	00116	000401	ENB	
0111	00117	-0 01 00073	JMP*	IN GO BACK HERE TO 1 HERE
0112			*	
0113	00120	14 0220	CMOD OCP	'220
0114	00121	140040	CRA	
0115	00122	0 04 00165	STA	IRF
0116	00123	74 0020	SMK	'20
0117	00124	-0 01 00167	JMP*	CM
0118			*	
0119			*	EXIT TO COMMAND MODE.
0120			*	
0121	00125	101000	CLO	NOB
0122	00126	14 0104	OCP	'104 RESET ASR FOR INPUT!
0123	00127	14 0220	OCP	'220
0124	00130	0 02 00166	LDA	TWP
0125	00131	101040	T7	SNZ
0126	00132	0 01 00144	JMP	RT7
0127			*	RETURN AFTER DELAY

0128	00133	140040	CRA	
0129	00134	0 04 00165	STA	TRF
0130	00135	000201	IAB	
0131	00136	0 06 00061	ADD	'61
0132	00137	0 04 00061	STA	'61
0133	00140	140040	CRA	
0134	00141	74 0020	SMK	'20
0135	00142	14 0020	OCF	'20
0136	00143	-0 01 00000	JMP*	TIM RETURN,
0137				* AFTER DELAY TIMING.
0138				*RT7 RETURN WITH SMK'20
0139	00144	140040	RT7	CRA
0140	00145	74 0020	SMK	'20
0141	00146	-0 01 00000	JMP*	TIM
0142			*	
0143			*	
0144			*	ERROR MESSAGES
0145			*	
0146	00147	-0 10 00170	EX5	JST* ERR
0147	00150	152327	BCI	1, TW
0148				*TYPE WRONG IN CALL
0149			*	
0150	00151	-0 10 00170	ERNR	JST* ERR
0151	00152	147322	BCI	1, NR CLOCK NOT RUNNING
0152			*	
0153	00153	14 0220	ERR0	OCF '220
0154	00154	140040	CRA	
0155	00155	0 04 00165	STA	TRF AND CLEAR FLAG.
0156				* BEFORE YOU EXIT

* TIME DELAY/MARKER FOR BASIC.

0157 00156 -0 10 00170 JST* ERR
0158 00157 151325 BCI 1, RU CLOCK RUNNING
0159 *
0160 00160 -0 10 00170 ERTU JST* ERR
0161 00161 152325 BCI 1, TU TIME UNDERFLOW
0162 *
0163 * DATA
0164 *
0165 00162 000001 DT OCT 1
0166 00163 000002 TSW OCT 2
0167 00164 000003 TE OCT 3
0168 *
0169 0165 0000 TRF BSZ 1
0170 00166 000000 TRP BSZ 1
0171 *
0172 00167 001000 CM OCT 1000
0173 00170 005243 ERR OCT 5243
0174 *
0175 00171 177777 M1 DEC -1
0176 *

0177 *ALL FOR NLS
0178 *R. A. CHARD 11.7.72 ILY(6) 687.1.
0179 *
0180 00172 000303 END
00173 000322
00174 000041

NO ERRORS IN ABOVE ASSEMBLY.

DAP-16 REV. E

The first thing done is to call the appropriate transfer
statement. This causes the program to enter the appropriate routine
to be transferred to the appropriate time in figure 13.

```
SUBROUTINE FTIME(TYPE, I, RT)
  IT=IFIX(T*(-50.))+0.5)
  IRT=0
  ITYPE=IFIX(TYPE)
  CALL TIM(ITYPE, IT, IRT)
  IF(IRT.GT.0) GOTO 1
  RETURN
1  RT=FLOAT(IRT)/50.
  RETURN
END
```

50

Fig L 4

The first thing done is to call the argument transfer subroutine FSAT. This causes the locations at which the arguments are stored to be transferred to the subroutine lines 23 to 27 in Figure L3. The arguments themselves are accessed by indirect addressing through FP, DP and RT.

TYPE 1

The processing part of TIM begins by fetching the first argument, LDA*FP - line 30 Figure L3, and testing this to determine what action is required. (see Figure FS1, block T1). A true result indicates that a delay has been requested, this causes TIM to test whether it has previously been called for timing (block T2). This is necessary because the same clock must be used for timing and delays. If the clock is not being used, it is clear for use and a clock interrupt is set up for the required time delay. If, however, the clock was being used it must be handled with care or the timing part will be lost. The clock is first stopped and a flag set to indicate that timing was in progress when the delay was entered. The elapsed time is fetched, and added to the requested delay, and the result stored in the B register, leaving the clock free and the total time saved. A clock interrupt is now set up in the same way as it is when the clock is initially free. During the delay the machine executes a time-wasting program; this must be done to keep it running. It should be noted that if there is any other computation to be performed the delay time could be used for this.

While the clock is in operation it increments location '61 every 20 ms (50 times/second) and when this location changes from -1 to zero ('177777 to '000000) it causes an interrupt to be generated. This is the reason for using a negative integer for the time delay.

Teletype action during delay

To permit operator action during the delay a teletype interrupt handling routine is built into TIM. This enables interrupts to be generated by the teletype as well as the clock, whilst a delay is in progress.

The TIM interrupt handler tests whether the clock or teletype interrupted and takes appropriate action. In the case of the clock, this indicates the end of the delay, before a return to the FORTRAN level, the flag is tested to check whether timing was in progress when the delay began; if so, the elapsed time must be restored and the clock re-started before returning. The teletype causes a different action, there are three possibilities:-

- i) an "R" for RETURN - this instructs the program to terminate the delay immediately, as if it had run its full time. A correction is made when timing is also in progress, so that the recorded time is the actual time elapsed, and not that which would have elapsed if the delay had run its full course.

- ii) a "C" for COMMAND - this causes all the necessary initialisations to be performed and an exit made direct to the COMMAND MODE of BASIC. This can be very useful if an extremely long delay is entered accidentally.
- iii) Any other character - this causes an interrupt, but when the character is tested and found to be invalid, a return is made to the delay.

It should be noted that the clock is kept running during these service routines so that the delay is exact, no time being lost or gained no matter how many invalid characters are typed.

TYPE 2 (start timing)

If the type call was not a 1, the type argument digit is tested for being a 2, (figure FS1, block T4). If this is true, TIM tests whether it is already timing a program by examining the "running flag". The flag is set whilst timing is in progress, so if it is found set the user has tried to start the clock whilst already running. This causes an exit from the subroutine via the error processor of BASIC, producing the message RU (RUNNING). The running flag is re-set whilst the error is processed. Normally the clock is not running, in this case the running flag is set and the clock started. Control is then returned to the BASIC level via FORTRAN.

TYPE 3 (stop timing)

Having found that argument 1 is not a 1 or 2 TIM tests it for being a 3, (figure FS1, block T5). If the argument is not a 3 a wrong type call has been made and an exit is made giving the message TW at the BASIC level.

A true result causes the running flag to be tested; this indicates an error if not set, as a stop clock command has been given whilst the clock is stopped. This results in the message NR for NOT RUNNING.

Correct calling results in the clock being stopped, the running flag being re-set and the elapsed time fetched ready for conversion to seconds. The time is in clock increments and must be divided by 50 to give seconds. This is done at the FORTRAN level.

5.3.2. Subroutine 2 - Channel Fetcher.

Calling begins at the BASIC level with the statement CALL (2,ARG1,ARG2), control passes to the FORTRAN level, where the intermediate subroutine FCHANL takes over (Figure L10). FCHANL performs the conversion of the channel number (ARG1) from floating point to integer, and on the return converts the input value from integer to floating point. The actual data logging routine which operates at the ASSEMBLER level is FET. (Figure FS2 is a flowchart; Figure L1 an assembly listing), at the FORTRAN level the call is

```
CALL FET (ARG1,ARG2)
```

FET

Upon entry FET calls the argument transfer routine FSAT, which transfers the locations of the arguments into the subroutine. In the data logger there are 39 inputs, numbered serially from 1 to 39, hence any call outside this range will result in useless data being fetched. To avoid this, the first argument, (the channel number) is tested first to see if it is ≤ 0 and then > 39 , blocks T1 and T2 of Figure FS2. The appropriate error printouts are:-

```
CU - CHANNEL UNDERFLOW if ARG1  $\leq$  0
```

```
CO - CHANNEL OVERFLOW if ARG1  $>$  39
```

Having passed the range test, the number is converted to BCD and put into the appropriate format for output to the data logger.

This takes the form IONM,

I is used to tell the data logger that the amplifier gains are pre-set on a patch panel. N and M are the two digits of the channel number.


```
C      SUBROUTINE CHANL
      SUBROUTINE CHANL(C,A)
      IC=IFIX(C+0.5)
      IA=0
      CALL FET(IC,IA)
      A=FLOAT(IA)
      RETURN
      END
```

50

Fig L10

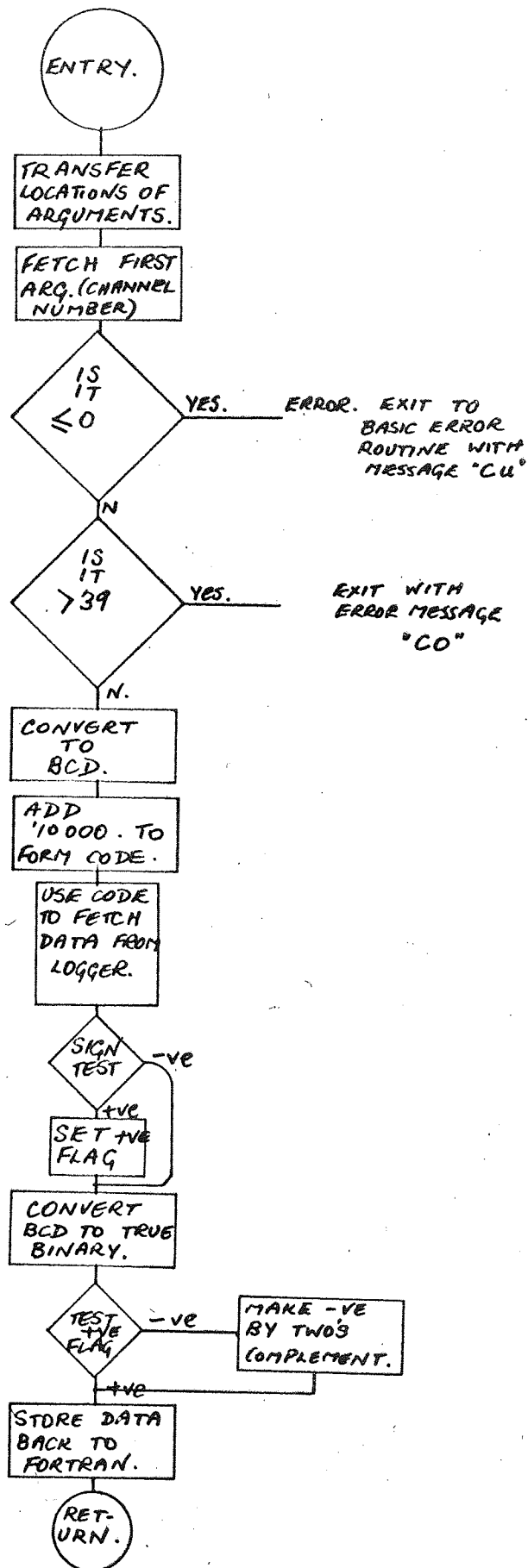


Fig. FS2 Operation of Subroutine FET.

* FET. CALLED FROM BASIC VIA FORTRAN.

Fig 21
PAGE 1 of 4

```
0001      * FET. CALLED FROM BASIC VIA FORTRAN.
0002      *
0003      * USES ARGUMENT TRANSFER.
0004      *
0005              SUBR  FET
0006      *
0007      * ERROR MESSAGES.
0008      *
0009      * CO CHANNEL UNDERFLOW.
0010      * CO CHANNEL OVERFLOW.
0011      *
0012      * THESE INDICATE A CHANNEL
0013      * OUT OF THE RANGE 1 TO 39.
0014      *
0015              REL
0016 00000  0 000000  FET  DAC  **
0017 00001  0 10 00000      CALL  FSAT
0018 00002  000002      OCT  2
0019 00003  000000  OPP  OCT  0
0020 00004  000000  RWP  OCT  0
0021 00005  -0 02 00003      LDA*  OPP
0022 00006  0 11 00070      CAS  = '0
0023 00007  0 01 00012      JMP  *+3
0024 00010  101000      NOP
0025 00011  0 01 00062      JMP  ERCU
0026 00012  0 11 00061      CAS  D39
```

0027	00013	0 01 00064	JMP	EXCD
0028	00014	101000	NOP	
0029	00015	101000	NOP	
0030	00016	0 10 00000	CALL	BINBCD
0031	00017	101000	NOP	
0032	00020	0 06 00067	ADD	= '10000
0033	00021	14 0130	OCP	'130
0034	00022	34 0130	SKS	'130
0035	00023	100000	SKP	
0036	00024	0 01 00022	JMP	*-2
0037	00025	74 1030	OIA	'1030
0038	00026	0 01 00025	JMP	*-1
0039	00027	34 0630	SKS	'630
0040	00030	0 01 00027	JMP	*-1
0041	00031	140040	CRA	
0042	00032	0 04 00060	STA	NF
0043	00033	14 0030	OCP	'30
0044	00034	34 0130	SKS	'130
0045	00035	100000	SKP	
0046	00036	0 01 00034	JMP	*-2
0047	00037	54 1030	INA	'1030
0048	00040	0 01 00037	JMP	*-1
0049	00041	34 0530	SKS	'530
0050	00042	0 12 00060	IRS	NF
0051	00043	0 10 00000	CALL	BCDBIN
0052	00044	0 04 00057	STA	COV

Fig L1
PAGE 2 of 4.

Fig L1
PAGE 3 of 4.

* FET. CALLED FROM BASIC VIA FORTRAN.

0053	00045	0 02 00060	LDA	NF	
0054	00046	101040	SNZ		
0055	00047	0 01 00053	JMP	RTN	
0056	00050	0 02 00057	LDA	COV	
0057	00051	-0 04 00004	SIA*	RWP	
0058	00052	-0 01 00000	JMP*	FET	
0059	00053	0 02 00057	RTN LDA	COV	
0060	00054	140407	ICA		
0061	00055	-0 04 00004	SIA*	RWP	
0062	00056	-0 01 00000	JMP*	FET	
0063			*		
0064			*		
0065	00057		COV BSS	1	
0066	00060		NF BSS	1	
0067			*		
0068	00061	000047	D39 DEC	39	
0069			*		
0070	00062	-0 10 00066	ERCU JST*	ERR	
0071	00063	141725	BCI	1, CU CHANL UNDERF.	

0072 *
0073 00064 -0 10 00066 ERCO JST* ERR
0074 00065 141717 BCI 1500 CHANL OFLO.
0075 *
0076 00066 005243 ERR OCT 5243
0077 *
0078 00067 010000 END
00070 000000

Fig L1
PAGE 4 of 4.

NO ERRORS IN ABOVE ASSEMBLY.

DAP-16 REV. E

AC

This word is assembled in the A register and transmitted to the data logger which examines the specified channel and returns the input value to the computer. The sign is tested before conversion from BCD to binary, and a flag set if the result is positive. (Negation is by twos complement which would give a meaningless answer when applied to a BCD number, this led to some extraordinary results during early development work).

After conversion reference is made to this flag to determine whether negation is required. The result is now in integer format and is passed back to the FORTRAN level for conversion to floating point and return to BASIC.

5.3.3. Subroutine 3 - Reading the MDP 200 Clock and data switches.

Two functions are performed by subroutine 3; firstly the time is read from the MDP clock, secondly the scan data switches (integer digital data) are read. Figure FS3 shows the flowchart, figure L5 the assembly listing.

The calling sequence at the BASIC level is:-

```
CALL (3,ARG1, ARG2, ARG3)
```

Execution of this transfers control to the FORTRAN level, where the program FCLTS operates, (Figure L6), allocating storage for the data, and initialising this to zeros. Transfer to the assembler level is achieved by the FORTRAN statement:-

```
CALL CLTS (ARG1,ARG2,ARG3)
```

As before the argument locations are transferred before any other operation. The required information is retrieved by loading the appropriate code into the A register and calling a subroutine FECO (Figure L7). This is similar to FET, but its function is only to output the contents of the A register and exit with the corresponding input in the A register. No sign test is required because the time and data will always be positive. The input value is converted to binary and passed back to the FORTRAN level before the second and third arguments are processed. Control is returned to FORTRAN for floating the inputs and final transfer back to BASIC.

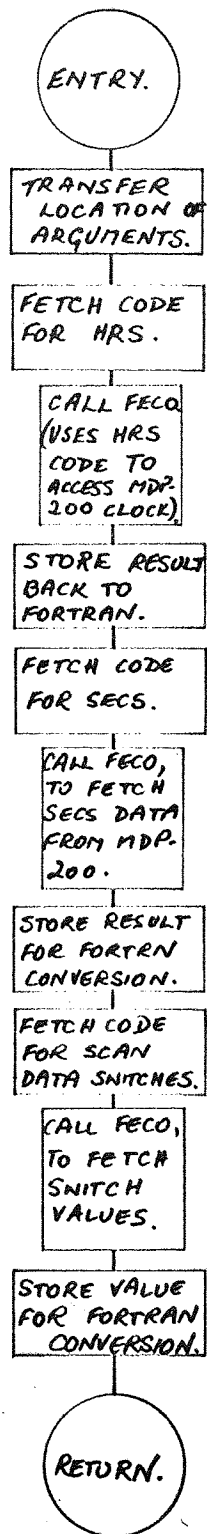


Fig FS3. Operation of CLTS Subroutine.

* CLOCK & THUMB SWS.

```
0001 * CLOCK & THUMB SWS.
0002 SUBR CLTS
0003 IEL
0004 00000 0 000000 CLTS DAC **
0005 00001 0 10 00000 CALL FSAT
0006 00002 000003 OCT 3
0007 00003 000000 HF OCT 0
0008 00004 000000 SP OCT 0
0009 00005 000000 VP OCT 0
0010 00006 0 02 00025 LDA = '130000
0011 00007 0 10 00000 CALL FEEO
0012 00010 0 10 00000 CALL BCDBIN
0013 00011 -0 04 00003 STA* HP
0014 00012 0 02 00024 LDA = '110000
0015 00013 0 10 00000 CALL FEEO
0016 00014 0 10 00000 CALL BCDBIN
0017 00015 -0 04 00004 STA* SP
0018 00016 0 02 00023 LDA = '120000
0019 00017 0 10 00000 CALL FEEO
0020 00020 0 10 00000 CALL BCDBIN
0021 00021 -0 04 00005 STA* VP
0022 00022 -0 01 00000 JMP* CLTS
0023 00023 120000 END
00024 110000
00025 130000
```

NO ERRORS IN ABOVE ASSEMBLY.

DAP-16 REV. E

Fig L5

2001

PROGRAM FOR INVERTING PERCENTAGE

2001

2001

```
SUBROUTINE FCLTS(HRS, SECS, TSV)
```

```
IHRS=0
```

```
I SECS=0
```

```
ITSV=0
```

```
CALL CLTS(IHRS, I SECS, ITSU)
```

```
HRS=FLOAT(IHRS)
```

```
SECS=FLOAT(I SECS)
```

```
TSV=FLOAT(ITSU)
```

```
RETURN
```

```
END
```

\$0

Fig L6

* FEEO. ORDINARY FETCHED.

0001			* FEEO. ORDINARY FETCHED.
0002			*
0003			SUB: FEEO
0004			BEL
0005	00000	0 000000	FEEO DAC **
0006	00001	14 0130	OCP '130
0007	00002	34 0130	SKS '130
0008	00003	100000	SKP
0009	00004	0 01 00002	JMP *-2
0010	00005	74 1030	OTA '1030
0011	00006	0 01 00005	JMP *-1
0012	00007	34 0130	SKS '130
0013	00010	0 01 00007	JMP *-1
0014	00011	34 0630	SKS '630
0015	00012	0 01 00011	JMP *-1
0016			* END OF OUTPUT.
0017	00013	101000	NOP
0018	00014	14 0030	OCP '30
0019	00015	34 0130	SKS '130
0020	00016	100000	SKP
0021	00017	0 01 00015	JMP *-2
0022	00020	54 1030	INA '1030
0023	00021	0 01 00020	JMP *-1
0024	00022	14 0230	OCP '230
0025	00023	34 0130	SKS '130
0026	00024	0 01 00023	JMP *-1
0027	00025	101000	NOP
0028	00026	-0 01 00000	JMP* FEEO
0029			END

Fig L7

5.3.4. Subroutine 4 - Sense Switch Test

This is a Honeywell extension of FORTRAN IV (Figure L8 - listing) which allows the user to test the sense switches. The program is called as a FORTRAN subroutine directly from BASIC:-

```
CALL (4,N,M)
```

Switch N is tested, and M made equal to 1 if the switch is set and 2 otherwise. It should be noted that switch one is used to cause a program break in BASIC precluding its use for options.

5.2.5. Subroutine TSW - Test Sense Switches

This is a direct transfer from BASIC to assembly code.

Number of arguments: 2 (S, V) (S, V are transferred).

```
C   SUBROUTINE TSW
C   TO TEST THE SENSE SWITCHES
SUBROUTINE TSW(S,V)
  IS=IFIX(S+0.5)
  IV=0
  CALL SSETCH(IS,IV)
  V=FLOAT(IV)
  RETURN
END
```

\$0

Fig 4.8.

5.3.5. Subroutine 5 - Audible Warning

This is a direct transfer from BASIC to ASSEMBLER and back again, because no arguments are to be converted and/or transferred.

Figure FS4 shows a flowchart and Figure L9 an assembly listing.

The routine operates simply, in that it loads the appropriate code for the teletype BELL into the A register ('207) and outputs this to the teletype. Before returning to BASIC the routine sets the teletype interface back into the input mode, this is necessary because the BASIC compiler operates with the interface set for input, only setting it to output as necessary.

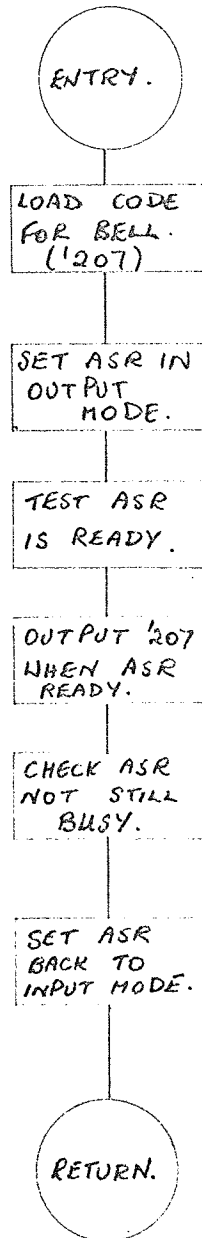


Fig FSA Operation of "Bell" subroutine.

* SUBR BELL

```
0001          * SUBR BELL
0002          * TO RING BELL
0003          * 6.7.72
0004          *
0005          SUBR BELL
0006          REL
0007 00000 0 000000 BELL DAC **
0008 00001 101000     NOP
0009 00002 0 02 00013 LDA = '207
0010 00003 34 0104     SKS '104
0011 00004 0 01 00003 JMP *-1
0012 00005 14 0104     OCP '104
0013 00006 74 0004     OTA '4
0014 00007 0 01 00006 JMP *-1
0015 00010 34 0104     SKS '104
0016 00011 0 01 00010 JMP *-1
0017 00012 -0 01 00000 JMP* BELL
0018 00013 000207     END
```

NO ERRORS IN ABOVE ASSEMBLY.

DAP-16 REV. E

Fig L9.

5.3.6. FORTRAN ERROR ROUTINE - F~~S~~ER

Throughout BASOON, FORTRAN subroutines are intermingled freely with ASSEMBLER routines and linked to the BASIC compiler; however, it must not be forgotten that these use their own error detection mechanism F~~S~~ER. This is a Honeywell subroutine which is called whenever an error condition is detected, its function is to print a two-letter error mnemonic and halt. Thus program execution is stopped, with no satisfactory means of recovery back to the calling level (BASIC). During the development of BASOON it was necessary to know not only what error had occurred, but also in which of the FORTRAN subroutines the occurrence took place - with this in mind, F~~S~~ER was re-written. The new routine retains the name F~~S~~ER, to save re-writing the whole FORTRAN package, and performs the following functions:-

- i) the two-letter FORTRAN mnemonic is printed
- ii) the octal location from which the error routine was called is printed.
- iii) an exit is made to the error processor in the BASIC compiler.

An error printout from the FORTRAN level would therefore have the following form:-

```
XX  a two-letter error MNEMONIC from FORTRAN
NNNNN the octal location from which the call was
      made, together with a "MEMORY MAP" - this
      enables the faulty routine to be located.
><  is a BASIC message to tell the user that
      an error occurred at the FORTRAN level.
ZZZ  is the line number in the BASIC program
      which was being executed.
```

This special F~~S~~ER is listed in Fig. L 11

* SPECIAL FSER

Fig 2 11
PAGE 1 of 4.

0001 * SPECIAL FSER
0002 *
0003 SOB R FSER, ERR
0004 *
0005 * THIS HAS BEEN WRITTEN BASED
0006 * ON THE HONEYWELL FSER, BUT
0007 * NOW DOES NOT HALT.
0008 * WHEN CALLED AT THE FORTRAN
0009 * LEVEL THE NORMAL ERROR
0010 * MESSAGE IS PRINTED, FOLLOWED
0011 * BY THE OCTAL LOCATION IN THE
0012 * CORE FROM WHICH THE CALL WAS
0013 * MADE.
0014 * AN EXIT IS THEN MADE TO THE
0015 * BASIC ERROR ROUTINE WITH THE
0016 * MESSAGE ><.
0017 *
0018 REL
0019 00000 0 000000 ERR DAC **
0020 00001 -0 02 00000 LDA* ERR GET MAS. LOC
0021 00002 0 04 00726 STA '726 STORE IN BASE.
0022 00003 -0 02 00726 LDA* '726 GET MESSAGE.
0023 00004 0 04 00726 STA '726 STORE IT BACK.
0024 00005 34 0104 SKS '104
0025 00006 0 01 00005 JMP *-1
0026 00007 14 0104 OCP '104

Fig L 11
PAGE 2 of 4

0027	00010	0 02 00025	LDA	CRLF	
0028	00011	0 10 00015	JST	OUT NEW LINE.	
0029	00012	0 02 00726	LDA	'726	
0030	00013	0 10 00015	JST	OUT NEW THE MESSAGE.	
0031	00014	0 01 00026	JMP	MINE	
0032			* 10 PRINT	THE LOCATION.	
0033	00015	0 000000	OUT	DAC	**
0034	00016	0416 70	ALR	8	
0035	00017	74 0004	OIA	'4	
0036	00020	0 01 00017	JMP	*-1	
0037	00021	0416 70	ALR	8	
0038	00022	74 0004	OIA	'4	
0039	00023	0 01 00022	JMP	*-1	
0040	00024	-0 01 00015	JMP*	OUT	
0041	00025	106612	CRLF OCT	106612	
0042			*		
0043	00026	0 02 00025	MINE LDA	CRLF	
0044	00027	141340	ICA		
0045	00030	0 10 00055	JST	OIA	
0046	00031	141340	ICA		
0047	00032	0 10 00055	JST	OIA	
0048	00033	0 02 00000	LDA	ERR	
0049	00034	0 07 00067	SUB	= '1	
0050	00035	000201	IAB		
0051			* PUT IN B	READY.	
0052	00036	0 02 00066	LDA	== -4	

* SPECIAL F5ER

```
0053 00037 0 04 00064 SIA W0
0054 00040 140040 CRA
0055 00041 0412 74 LLR 4 FIRST DIGIT.
0056 00042 0 06 00065 ADD ='260 PAD IT.
0057 00043 0 10 00055 JST OTA & OP IT
0058 00044 140040 NXL CRA
0059 00045 0412 75 LLR 3
0060 00046 0 06 00065 ADD ='260
0061 00047 0 10 00055 JST OTA
0062 00050 0 12 00064 IRS W0
0063 00051 0 01 00044 JMP NXL LOOP
0064 00052 -0 10 00054 JST* BERR
0065 * EXIT TI BACIS ERROR
0066 00053 137274 BCI 1,><
0067 * WITH YOUR MESSAGE!
0068 *
0069 00054 005243 BERR OCT 5243
0070 * THAT'S WHERE I GO!
0071 00055 0 000000 OTA DAC **
```

Fig L11
PAGE 4 of 4

```
0072 00056 34 0104 SKS '104
0073 00057 0 01 00056 JMP *-1
0074 00060 14 0104 OCP '104
0075 00061 74 0004 OIA '4
0076 00062 0 01 00061 JMP *-1
0077 00063 -0 01 00055 JMP* OIA
0078 *
0079 00064 50 BSS 1
0080 *
0081 * R. A. CHARD
0082 *
0083 00065 000260 END
      00066 177774
      00067 000001
```

NO ERRORS IN ABOVE ASSEMBLY.

DAP-16 REV. E

AC

5.4. BASOON INITIALISATION

The entire package is contained in one self loading tape. When loaded into the machine and execution started an initialisation routine is performed. A printout is provided below, Fig. L 13 for reference. The ! signifies that the compiler is requesting a reply from the user.

The special modifications necessary to the compiler were done at ASSEMBLER level, and are listed in appendix 6 (BASIC INIT. MODS).

BASOON INITIALISATION

FIG. L 13

BASOON ON-LINE FROM BASIC

DO YOU WISH TO DELETE LIBRARY FUNCTION ATN? (ANSWER YES OR NO)

!YES

DO YOU WISH TO DELETE LIBRARY FUNCTIONS SIN, COS, TAN ?

!YES

DO YOU WISH TO DELETE LIBRARY FUNCTION SQR?

!NO

THE HIGH OCTAL ADDRESS IS SET TO 24777

7192 LOCATIONS FOR USER STORAGE AND TABLES

?

6. The use of the Basoon system for data acquisition

The data acquisition system may be used in a variety of ways, the most usual are:

- (i) Monitoring - Scanning channels looking for abnormal reading and taking some action if these are found.
- (ii) Simple data acquisition - When the data is not used immediately, but is processed at a later time, perhaps on a different computer.
- (iii) As (ii) but at a specified time interval.
- (iv) Data input followed by immediate processing.

In (iii) and (iv) two possibilities exist, these are:

- a) That the time between samples will be greater than the acquisition and output or processing time, in which case a delay will be called for until the next sample.
- b) Time between samples less than output or processing time, here some extra programming is called for to ensure that the samples are taken on time.

6.1. Monitoring only

Here we are concerned with inputting a value and seeing whether it is within predetermined limits.

Consider all 39 channels are to be monitored. The limits would be determined and held in an array, say L. As each value is input it can be compared with the upper and lower limits and appropriate action taken.

```
eg.   10 DIM L(2,39): T=0
      20 FOR A=1, 39
      30 CALL (2,A,I)
      40 IF I>L(1,A) THEN GOSUB 110
      50 IF I<L(2,A) THEN GOSUB 110
      60 NEXT A
      70 CALL (4,4,T): REM TEST SS4
      80 IF T=1 GOTO 100:REM EXIT IF SET
      90 GOTO 20
     100 STOP
     110 PRINT "CHANNEL" ; A; "IS"; I; "LIMITS ARE";
```

```
115 PRINT L(1,A); "TO" ; L(2,A)
120 RETURN
```

This example will test all the channels, and print the message eg.

```
CHANNEL 24 IS 705 LIMITS ARE 1000 TO 1020
```

After each scan a test of sense switch 4 is performed to see if further monitoring is required. This type of program could easily be built into a more sophisticated data logging routine.

6.2. Simple Data Acquisition

Consider the case of examining 10 temperatures during an experiment in which the operator must decide at what point to take his reading. Here only the data is required, with no processing. A typical program would be:

```
10 DIM R(25)
20 FOR C = 15, 24:REM CHANNELS 15 TO 24
30 CALL (2,C,R(C))
40 NEXT C
50 PRINT:PRINT:REM SPACING
60 FOR C=15,24
70 PRINT R(C)
80 NEXT C
90 STOP
```

Typing "RUN" would cause channels 15 to 24 to be scanned and the input data stored in the array R. The data is then printed. A data tape could be obtained at the same time as the printing by switching on the teletype punch.

Alternatively if a Basic program is going to be used for the processing a special program could be written which would output the information in the form of data statements.

6.3. Input and Printing at regular time intervals

This uses the Basoon timing and delay subroutines. The length of time taken to fetch the data and print it is measured, and a delay entered for the remainder of the time

to the next sample. The case where the printing time is greater than the sampling time is dealt with in section 6.4

For an example consider taking readings from channels 1 to 10 inclusive every 2 minutes, and printing the results.

```
10 DIM R(10) : A=0 : T=0
20 INPUT S : REM TIME BETWEEN SAMPLES
30 CALL (1,2,A,A):REM START TIMING
40 FOR C=1, 10
50 CALL (2,C,R(C))
60 NEXT C
70 PRINT:REM SPACING
80 FOR C=1, 10
90 PRINT R(C)
100 NEXT C
110 CALL (1,3,A,T,):REM STOP TIMING
120 REM T=ELAPSED TIME
130 D=S-T
140 REM D=REQUIRED DELAY
150 IF D=0 THEN GOTO 180
160 CALL (1,1,D,A) : REM CALLING DELAY
170 GOTO 30:REM RESTART
180 PRINT "SAMPLE TIME TOO SHORT!"
190 GOTO 30
```

The above program would examine the required channels every S seconds and print the results, if the printing took longer than S seconds a message would be printed and the next sample taken. For most applications this overshoot would not be a serious problem.

6.4. Input and Processing at regular time intervals

For some applications the overshoot mentioned in section 6.3. would not be acceptable, for example, when the samples are required to be taken at exact time intervals. The difficulty can be overcome by some extra programming, forming an operating system.

The method consists of having a small subroutine which keeps track of elapsed time and returns

MAIN PROGRAM.

F FLAG IS SET
WHILST THERE IS
A DATA SET STORED
AND AWAITING
PROCESSING.

P FLAG IS SET
WHILST DATA
PROCESSING IS IN
PROGRESS.

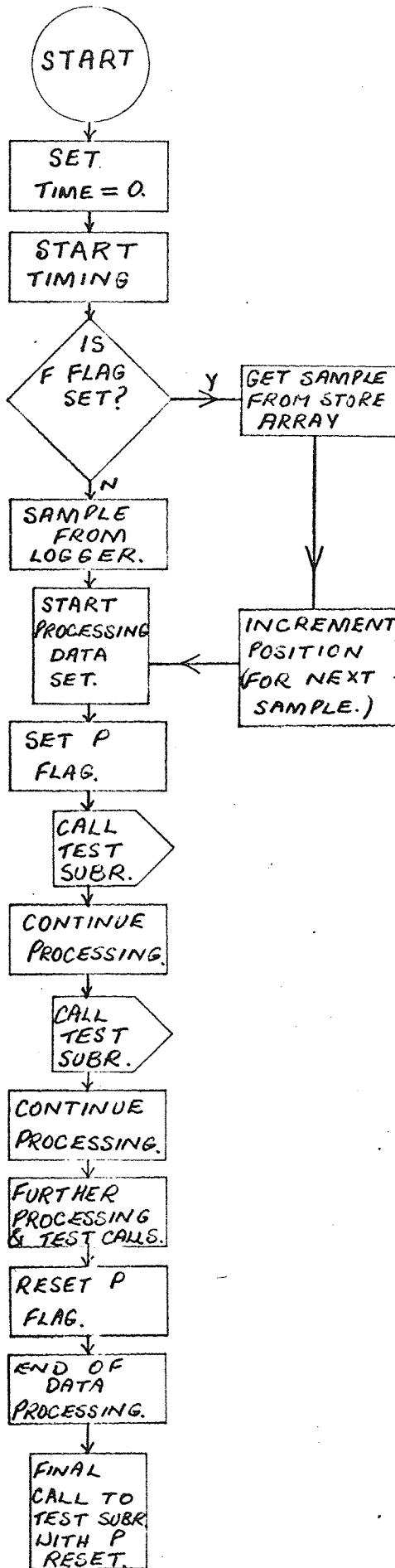


FIG. F55.

TEST SUBROUTINE.

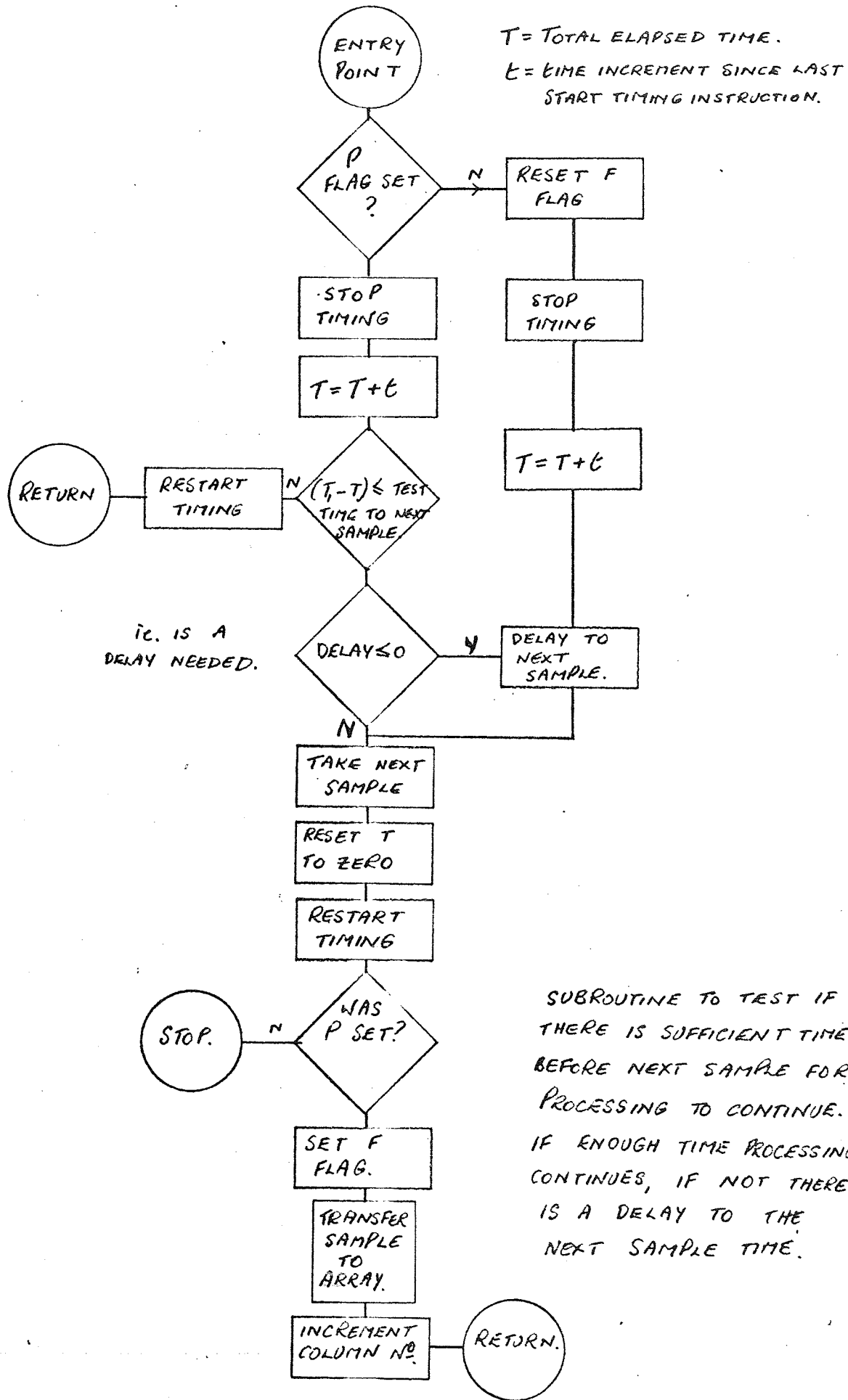


FIG. F55. CONCLUDED.

to the processing, or, as the next sample time approaches the subroutine calls the sample at the appropriate time then returns to processing. A possible flow chart is given in FIG. (FS)5.

The test subroutine is called at various convenient places during the processing part of the program, and at the end of processing a test is made to see if data processing was in progress, and in the NO case a delay is caused until the next sample is required. For YES, the elapsed time is found by stopping timing and adding the time onto the running time total, if there is sufficient time for more processing a return is made to the processing program. Insufficient time leads to a short delay until the exact time the next sample is due. The incoming data is put into temporary storage, then if processing was in progress it is packed into an array column to await its own processing turn, otherwise it is transferred to the current data array and processing started immediately.

This programming procedure allows samples to be taken at very short time intervals and processed as the machine is ready. The number of samples which can be taken is limited only by the storage capacity of the computer. If a very large number of samples was required a further refinement could be incorporated so that as the stored data array was processed the columns which had been used could be overwritten by further incoming data. Eventually however the stage would be reached where the array had been completely filled and filled again up to the data set currently being processed, and at this point no more data could be input.

7.1. RESULTS AND DISCUSSION

The intention of this research project has been to investigate the possibilities of developing a flexible user orientated computer package to link a high speed digital computer and existing data logger. This has been found possible, and fully implemented, to this extent the software package is a major result. Some aspects of the use of the system have been studied, and the results of these studies are reported here to enable users to make the best use of available functions.

7.2. DEMONSTRATION PROGRAMS

This program provides the user with a complete demonstration of the capabilities of the package. Each channel is read, timed, and sense switches tested. The program is listed in Table 7.1. and provides a reference for users. A sample execution of this program is given in Table 7.2.

7.3.1. THE DISTRIBUTION ANALYSIS PROGRAM

This program was intended to show the distribution about its mean point of the incoming data. The program itself is listed in Appendix 5. A sample printout is provided in Table 7.3. which shows an analysis of data taken from channel 22, 200 samples were taken and a very good distribution obtained. The printout shows the distribution first in a numerical table and then in the form of a histogram to give a graphical representation. The data, as shown by this printout is very accurate, so this program does not show a great deal and was superseded by the Ensemble test.

7.3.2. DATA CARRY-OVER FROM PREVIOUS SCANS

When the data logger is operating there could be a tendency for the incoming signal to be retained in the digital voltmeter and buffer and affect the next results.

If this happened it would be most apparent when changing channels, in which case the first scan of any channel would be the most likely to be inaccurate. To investigate this possibility the sampling test was written. (The program is listed in Appendix 5).

This program takes 50 scans of a channel consecutively, followed by 50 scans non-consecutively. To obtain the non-consecutive samples, scanning takes place from the channel below the one on test to the one above. For example, when testing channel 20, one scan of channels 19, 20 and 21 would be performed with the scan of channel 20 being the required input. Table 7.4. shows the output from this program. Each of the 100 input samples are printed together with some analysis. This test was performed on all the available input channels and the results are included for reference.

7.4.1. ACCURACY OF INCOMING DATA

As we are concerned with a system for high speed data acquisition it is of prime importance that these data are accurate. The programs that have been written and used were intended to test the accuracy of the system and enable recommendations to be made to future users.

The factors influencing accuracy are:

- a) Data carry-over from previous scans - dealt with above.
- b) The rate of scanning the channels.
- c) The number of consecutive scans which are averaged to give an input sample, (ideally this should be 1).

7.4.2. THE RATE OF SCANNING & THE NUMBER OF SCANS PER SAMPLE

It was decided that both of these could be investigated by one test program. This has been titled The Ensemble Test Program. (See listing in Appendix 5). The Ensemble is defined as the number of scans in any given

sample. Thus if an Ensemble of 5 is used the channel is scanned 5 times, and the average of the 5 scans is taken as the sample.

The Ensemble Test Program begins by taking one complete scan of channels 1 to 29, this is timed to determine the scanning rate. The "true mean" of each channel is taken in this instance as being the average of 50 consecutive scans of a channel. This is found for each of the channels used, i.e. 12 to 29 inclusive.

For each of these channels 20 scans are then taken, and the average calculated for an ensemble of 1 to 20. These data are then printed, Table 7.5. gives these data for scanning rates of 7,9,12 and 14 channels per second. It will be noted looking at the 5 pages which each printout occupies that it is somewhat difficult to interpret this data readily. The program was therefore extended to include accuracy codes. These codes are the letters of the alphabet A to H and the asterisk (*), which are printed in the form of an array and represent the deviation of each sample from its true mean. The program has been written in such a way that the percentage bands represented by these accuracy codes can be altered easily, hence the current values assigned to each code are printed before the arrays. The full results considered together show that the accuracy is very high even with fast scanning rates. As might be expected at the fast rates, slight inaccuracies do occur, the equipment was found to require approximately 30 minutes to warm up and a set of results of the Ensemble Test taken during this period are included for reference. (It will be noted from the results that channel 15, one of the pressure transducers, produces erratic results, this indicates that there is a fault somewhere on this line).

7.5. SCANNING RATES

The scanning rate is adjusted by a potentiometer located on one of the printed circuit boards in the data logger. This adjusts the scanning rate by varying the time between the closure of the read relays and the operation

of the analogue to digital converter. The minimum and maximum rates obtainable are 7 and 14 channels per second, respectively. In view of the results of the test programs the system may be confidently used at maximum scanning rate taking only one scan to each sample.

TABLE 7
THE RESULTS OF VARIOUS PROGRAMS

1. The Demonstation Program
2. Sample Printout from Demonstration Program
3. Sample Printout from Distribution Analysis Program
4. Complete Printout of Execution of Sampling Test Program showing 50 consecutive samples and 50 non-consecutive samples for channels 12 to 29 inclusive.
5. Ensemble Test and Accuracy Codes - Printouts at 7,9,12 and 14 channels per second, showing the accuracy of incoming data.
6. Ensemble Test and Accuracy Codes - an execution during the warm up period showing how this affects the results.

LIST

```
1  A=0:T=0: PRINT "DEMONSTRATION PROGRAM"
2  DIM Z(39)
5  X=20
6  PRINT : PRINT : PRINT
10 PRINT "ONE SCAN OF CHANNELS 1-39"
20 PRINT "BELL INDICATES START & END OF SCAN"
25 CALL (5)
26 CALL (1,2,1,1)
30 FOR G=1,39
40 CALL (2,G,Z(G))
50 NEXT G
55 CALL (5)
60 CALL (1,3,A,T)
61 PRINT "THAT TOOK";T;"SECONDS"
62 PRINT : PRINT
63 PRINT "THE VALUES ARE....."
66 FOR Q=1,39
67 PRINT Q,Z(Q)
68 NEXT Q
80 PRINT "NOW A DELAY OF 20 SECONDS"
90 PRINT "RETURN TO COMMAND MODE DURING DELAY BY TYPING C"
100 PRINT "RETURN AND CONTINUE PROGRAM BY TYPING R"
101 PRINT
105 CALL (1,1,X,B)
106 PRINT : PRINT
110 PRINT "MDP CLOCK TIME"
120 CALL (3,A,B,C)
130 PRINT A,B
135 PRINT
140 PRINT "SCAN DATA SWW";C
145 PRINT "SENSE SW STATES ARE..."
150 FOR F=2,4
155 M=0
160 CALL (4,F,M)
170 IF M=1 THEN PRINT F;"SET": GOTO 190
180 PRINT F;"RESET"
190 NEXT F
195 PRINT
196 PRINT
200 PRINT "THAT'S YOUR LOT"
210 PRINT : PRINT : PRINT
220 STOP
```

TABLE 7.1 DEMONSTRATION PROGRAM.

ONE SCAN OF CHANNELS 1-39
FELL INDICATES START & END OF SCAN
THAT TOOK 2.16 SECONDS

THE VALUES ARE.....

1	-9999
2	9999
3	-9999
4	-9999
5	-9999
6	-9999
7	-9999
8	-9999
9	-401
10	-9999
11	9999
12	-9999
13	-9999
14	-9999
15	-9999
16	-9999
17	-9999
18	-9999
19	-4401
20	-9999
21	9999
22	-9999
23	9999
24	-9999
25	-9999
26	-9999
27	-401
28	-9999
29	-7101
30	-9999
31	9999
32	-9999
33	-9999
34	-9999
35	-7999
36	-9999
37	-9999
38	-3799
39	-9999

NOW A DELAY OF 20 SECONDS

RETURN TO COMMAND MODE DURING DELAY BY TYPING C

RETURN AND CONTINUE PROGRAM BY TYPING R

R

MDP CLOCK TIME

1340 3130

SCAN DATA SWW 5676

SENSE SW STATES ARE...

2 RESET

3 SET

4 RESET

TABLE 7.2

THAT'S YOUR LOT

220 EXIT

TABLE 7.3.

RUN
DISTRIBUTION ANALYSIS.
CHANNEL?
122
TIME BETWEEN SAMPLES?
12
SCANS/SAMPLE?
120
SAMPLES?
1200

ANALYSIS OF DATA FROM CHANNEL 22
20 SCANS TO EACH SAMPLE.
SAMPLES TAKEN EVERY 2 SECONDS

ANALYSIS OF SAMPLE MEANS.

LARGEST SAMPLE MEAN 753
SMALLEST 751
LARGEST ST. DEV. 1.28145
SMALLEST 0

THE MEAN OF THE SAMPLES IS 752.055
AND THEIR ST. DEV. .928921

DISTRIBUTION OF SAMPLES
VALUE NO. OF POINTS

751	3
752	183
753	14

751 ***
752 TOO MANY FOR ONE LINE!
753 *****

TABLE 7.4.

Complete Printout of execution of Sampling
Test Program showing 50 consecutive samples
and 50 non-consecutive samples for channels
12 to 29 inclusive.

A. CHARD SAMPLING TEST... PAGE 0.

PURPOSE: TO TEST THE ACCURACY OF DATA BEING
SCANNED FROM A CHANNEL CONSECUTIVELY
AND IN THE MIDST OF OTHER CHANNELS.

TEST BEGINS. STARTING TIME 1048 3860

• • • • • • • • • • • • • • •

CHANNEL 12 50 CONSECUTIVE SCANS.

-4032	-4033	-4032	-4033	-4032
-4033	-4032	-4033	-4032	-4033
-4032	-4033	-4032	-4033	-4032
-4031	-4032	-4031	-4031	-4033
-4032	-4033	-4032	-4033	-4032
-4033	-4032	-4033	-4032	-4033
-4032	-4033	-4031	-4033	-4031
-4033	-4032	-4033	-4032	-4033
-4032	-4033	-4032	-4033	-4032
-4031	-4031	-4033	-4031	-4033

CHANNEL 12 50 INTERRUPTED SCANS.

-4032	-4023	-4030	-4034	-4033
-4030	-4034	-4028	-4033	-4030
-4029	-4033	-4033	-4032	-4034
-4033	-4033	-4027	-4033	-4033
-4033	-4033	-4033	-4033	-4033
-4033	-4033	-4033	-4033	-4033
-4033	-4034	-4032	-4033	-4033
-4032	-4034	-4033	-4033	-4034
-4033	-4033	-4033	-4033	-4033
-4033	-4032	-4034	-4033	-4033

DATA ANALYSIS.

CHANNEL 12

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-4033 TO -4031	-4034 TO -4023
MEAN	-4032.28	-4032.36
ST. DEVN.	.729552	1.99755

DIFFERENCE IN MEAN VALUES --.1E-01 %

SCANNING RATE = 7 CHANNELS / SECOND

CHANNEL 13 50 CONSECUTIVE SCANS

373	371	371	373	373
371	373	371	371	371
371	373	373	373	373
373	373	373	373	373
373	373	373	373	373
373	373	373	373	373
373	373	373	373	373
373	373	373	373	373
373	373	373	373	373
373	373	373	373	373
373	373	371	373	373

CHANNEL 13 50 INTERRUPTED SCANS

371	371	369	369	369
369	369	369	367	367
367	369	369	367	367
367	367	369	369	367
369	369	369	369	367
367	369	369	369	369
367	369	369	369	367
367	369	369	369	369
367	369	369	369	369
367	369	367	369	369
367	367	367	369	369

DATA ANALYSIS

CHANNEL 13

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	371 TO 373	367 TO 371
MEAN	372.68	368.36
ST. DEVN.	.740657	1.10213
DIFFERENCE IN MEAN VALUES	1.15 %	
SCANNING RATE	7 CHANNELS / SECOND	

CHANNEL 14 50 CONSECUTIVE SCANS

9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999

CHANNEL 14 50 INTERRUPTED SCANS

9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999
9999	9999	9999	9999	9999

DATA ANALYSIS.

CHANNEL 14

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	9999 TO 9999	9999 TO 9999
MEAN	9999	9999
ST. DEVN.	0	0

DIFFERENCE IN MEAN VALUES 0 %

SCANNING RATE 7 CHANNELS / SEC QND.

CHANNEL 15 50 CONSECUTIVE SCANS.

16	17	19	19	17
19	19	19	19	19
19	19	19	19	19
19	19	19	19	19
19	19	19	19	19
19	19	19	19	19
19	19	19	19	19
19	19	19	19	19
19	19	19	19	19
19	19	19	19	19

CHANNEL 15 50 INTERRUPTED SCANS.

71	73	65	73	71
73	65	73	71	65
73	71	65	73	71
63	71	71	73	65
73	71	67	73	71
65	71	71	71	69
73	71	63	69	71
71	63	73	71	65
73	71	69	73	71
73	65	73	71	67

DATA ANALYSIS.

CHANNEL 15

	CONSECUTIVE SCANS	INTERRUPTED. SCANS
SAMPLE RANGE	16 TO 19	63 TO 73
MEAN	18.86	69.88
ST. DEVN.	.571786	3.26165
DIFFERENCE IN MEAN VALUES	270.51 %	
SCANNING RATE	7 CHANNELS / SECOND.	

CHANNEL 16 50 CONSECUTIVE SCANS.

-3983	-3983	-3982	-3983	-3982
-3981	-3980	-3983	-3983	-3983
-3982	-3983	-3982	-3981	-3980
-3983	-3981	-3981	-3982	-3983
-3982	-3983	-3982	-3981	-3980
-3983	-3982	-3981	-3982	-3983
-3982	-3981	-3982	-3983	-3980
-3983	-3982	-3983	-3982	-3983
-3982	-3981	-3980	-3983	-3981
-3981	-3982	-3983	-3981	-3981

CHANNEL 16 50 INTERRUPTED SCANS.

-3961	-3963	-3971	-3965	-3972
-3971	-3963	-3962	-3963	-3963
-3965	-3962	-3973	-3966	-3974
-3967	-3961	-3961	-3962	-3963
-3963	-3967	-3966	-3970	-3971
-3963	-3959	-3962	-3962	-3963
-3962	-3971	-3967	-3968	-3973
-3969	-3962	-3961	-3962	-3964
-3964	-3963	-3962	-3969	-3967
-3974	-3970	-3964	-3962	-3961

DATA ANALYSIS.

CHANNEL 16

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-3983 TO -3980	-3974 TO -3959
MEAN	-3981.92	-3965.38
ST. DEVN.	1.00692	4.11513

DIFFERENCE IN MEAN VALUES -42 %

SCANNING RATE 7 CHANNELS / SECOND.

CHANNEL 17 50 CONSECUTIVE SCANS.

-3988	-3987	-3988	-3989	-3986
-3987	-3987	-3987	-3986	-3987
-3986	-3987	-3988	-3987	-3986
-3987	-3986	-3987	-3988	-3987
-3988	-3987	-3986	-3989	-3986
-3987	-3988	-3987	-3986	-3989
-3986	-3989	-3988	-3985	-3986
-3987	-3985	-3989	-3987	-3987
-3988	-3987	-3987	-3989	-3987
-3987	-3988	-3987	-3986	-3987

CHANNEL 17 50 INTERRUPTED SCANS.

-3987	-3987	-3987	-3987	-3987
-3987	-3987	-3987	-3987	-3987
-3987	-3985	-3987	-3985	-3987
-3985	-3987	-3985	-3987	-3987
-3987	-3987	-3987	-3989	-3987
-3987	-3989	-3989	-3989	-3987
-3987	-3987	-3987	-3989	-3987
-3987	-3987	-3987	-3985	-3987
-3987	-3987	-3985	-3987	-3985
-3987	-3985	-3987	-3987	-3987

DATA ANALYSIS.

CHANNEL 17

	CONSECUTIVE SCANS	INTERRUPTED. SCANS
SAMPLE RANGE	-3989 TO -3985	-3989 TO -3985
MEAN	-3987.12	-3986.88
ST. DEVN.	1.023	1.023
DIFFERENCE IN MEAN VALUES	-.1E-01 %	
SCANNING RATE.	7 CHANNELS / SEC OND.	

CHANNEL 18 50 CONSECUTIVE SCANS.

-1288	-1289	-1288	-1287	-1287
-1289	-1288	-1289	-1288	-1289
-1288	-1287	-1287	-1289	-1288
-1289	-1288	-1289	-1288	-1289
-1288	-1289	-1287	-1287	-1288
-1289	-1288	-1289	-1288	-1287
-1287	-1287	-1287	-1289	-1288
-1289	-1288	-1289	-1288	-1289
-1287	-1287	-1287	-1289	-1288
-1289	-1288	-1289	-1288	-1287

CHANNEL 18 50 INTERRUPTED SCANS.

-1289	-1289	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1289
-1289	1	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1289
-1289	-1289	-1289	-1289	-1291

DATA ANALYSIS.

CHANNEL 18

	CONSECUTIVE SCANS.	INTERRUPTED. SCANS.
SAMPLE RANGE	-1289 TO -1287	-1289 TO 1
MEAN	-1288.08	-1263.24
ST. DEVN.	.804071	182.439
DIFFERENCE IN MEAN VALUES		-1.93 %
SCANNING RATE	7 CHANNELS / SECOND.	

CHANNEL 19 50 CONSECUTIVE SCANS.

-3548	-3545	-3545	-3545	-3546
-3547	-3546	-3547	-3546	-3545
-3544	-3545	-3544	-3545	-3545
-3547	-3546	-3547	-3546	-3545
-3545	-3545	-3544	-3545	-3545
-3547	-3546	-3547	-3546	-3545
-3545	-3545	-3544	-3545	-3544
-3545	-3545	-3545	-3546	-3545
-3546	-3545	-3544	-3545	-3544
-3545	-3545	-3545	-3546	-3545

CHANNEL 19 50 INTERRUPTED SCANS.

-3547	-3545	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545
-3545	-3543	-3545	-3545	-3545
-3545	-3545	-3545	-3545	-3545

DATA ANALYSIS.

CHANNEL 19

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-3548 TO -3544	-3547 TO -3543
MEAN	-3545.36	-3545
ST. DEVN.	.942424	.404061

DIFFERENCE IN MEAN VALUES -.2E-01 %

SCANNING RATE 7 CHANNELS / SECOND.

CHANNEL 20 50 CONSECUTIVE SCANS.

-844	-845	-845	-845	-844
-845	-844	-845	-845	-845
-845	-845	-845	-845	-845
-845	-844	-845	-845	-845
-845	-845	-845	-845	-844
-845	-844	-845	-845	-845
-845	-845	-845	-845	-844
-845	-844	-845	-845	-845
-845	-845	-845	-845	-844
-845	-844	-845	-845	-845
-845	-845	-845	-845	-844
-845	-844	-845	-845	-845

CHANNEL 20 50 INTERRUPTED SCANS.

-847	-847	-845	-847	-845
-845	-845	-847	1	-845
-847	-845	-847	-845	-847
-847	-847	-845	-845	-845
-847	-845	-845	-847	-845
-847	-845	-847	-845	-847
-845	-847	-845	-845	-845
-845	-847	-845	-845	-845
-847	-845	-847	-845	-847
-845	-845	-847	-845	-845

DATA ANALYSIS.

CHANNEL 20

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-845 TO -844	-847 TO 1
MEAN	-844.8	-828.88
ST. DEVN.	.404061	119.762
DIFFERENCE IN MEAN VALUES		-1.89 %
SCANNING RATE	7 CHANNELS / SEC OND.	

CHANNEL 21 50 CONSECUTIVE SCANS

-848	-849	-848	-849	-847
-847	-847	-849	-848	-849
-847	-847	-846	-847	-847
-847	-847	-847	-848	-847
-847	-847	-846	-847	-848
-849	-848	-847	-847	-847
-847	-847	-846	-847	-848
-849	-847	-847	-848	-847
-847	-847	-846	-847	-846
-847	-847	-847	-847	-847

CHANNEL 21 50 INTERRUPTED SCANS

-849	-847	-849	-847	-847
-847	-847	-847	-847	-847
-847	-847	-847	-847	-847
-847	-847	-847	-847	-847
-847	-849	-847	-847	-847
-847	-847	-847	-849	-847
-847	-847	-847	-847	-847
-849	-847	-847	-847	-847
-847	-847	-847	-847	-847
-847	-847	-847	-847	-847

DATA ANALYSIS.

CHANNEL 21

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-849 TO -846	-849 TO -847
MEAN	-847.3	-847.2
ST. DEVN.	.814411	.606092
DIFFERENCE IN MEAN VALUES	-.2E-01 %	
SCANNING RATE	7 CHANNELS / SEC OND.	

CHANNEL 22 50 CONSECUTIVE SCANS

-852	-853	-852	-853	-852
-853	-852	-851	-851	-853
-852	-853	-852	-851	-852
-851	-851	-851	-851	-851
-851	-851	-852	-851	-851
-851	-850	-851	-851	-853
-852	-851	-851	-851	-850
-851	-851	-851	-852	-853
-852	-851	-851	-851	-850
-853	-852	-853	-852	-853

CHANNEL 22 50 INTERRUPTED SCANS

-851	-851	-851	-851	-851
-851	-851	-849	-851	-851
-851	-851	-851	-851	-851
-851	-851	-851	-851	-851
-851	-851	-851	-851	-849
-851	-851	-851	-851	-851
-851	-851	-851	-851	-851
-851	-851	-851	-851	-851
-851	-851	-851	-851	-851
-851	-849	-851	-851	-851

DATA ANALYSIS CHANNEL 22

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-853 TO -850	-851 TO -849
MEAN	-851.6	-850.88
ST. DEVN.	.880631	.479796
DIFFERENCE IN MEAN VALUES	-.9E-01 %	
SCANNING RATE	7 CHANNELS / SEC OND.	

CHANNEL 23 50 CONSECUTIVE SCANS.

-828	-829	-828	-829	-828
-827	-827	-829	-828	-829
-828	-829	-828	-827	-826
-827	-827	-829	-828	-829
-828	-827	-826	-827	-828
-829	-828	-827	-827	-827
-826	-827	-827	-827	-828
-829	-828	-827	-827	-827
-827	-827	-828	-829	-828
-827	-827	-827	-827	-827

CHANNEL 23 50 INTERRUPTED SCANS.

-829	-827	-827	-829	-827
-827	-827	-827	-827	-827
-827	-827	-827	-829	-827
-829	-827	-827	-829	-827
-827	-827	-827	-827	-827
-827	-827	-827	-827	-829
-827	-829	-827	-829	-827
-827	-829	-827	-827	-827
-827	-827	-827	-827	-827
-827	-827	-827	-827	-827

DATA ANALYSIS.

CHANNEL 23

	CONSECUTIVE SCANS	INTERRUPTED. SCANS
SAMPLE RANGE	-829 TO -826	-829 TO -827
MEAN	-827.62	-827.36
ST. DEVN.	.878078	.776176

DIFFERENCE IN MEAN VALUES -.4E-01 %

SCANNING RATE 7 CHANNELS / SECOND.

CHANNEL 24 50 CONSECUTIVE SCANS

-860	-859	-858	-859	-858
-859	-860	-859	-858	-859
-858	-859	-858	-859	-858
-859	-858	-859	-858	-859
-858	-859	-858	-859	-858
-859	-858	-859	-858	-859
-859	-859	-859	-859	-858
-859	-858	-859	-859	-859
-860	-859	-858	-859	-858
-859	-858	-859	-858	-859

CHANNEL 24 50 INTERRUPTED SCANS

-859	-861	-861	-859	-859
-859	-859	-861	-859	-861
-859	-859	-861	-859	-861
-859	-859	-859	-859	-859
-859	-859	-861	-859	-859
-859	-859	-859	-859	-859
-859	-859	-859	-859	-861
-859	-859	-859	-859	-861
-859	-861	-859	-859	-861
-859	-859	-859	-859	-861

DATA ANALYSIS

CHANNEL 24

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-860 TO -858	-861 TO -859
MEAN	-858.68	-859.48
ST. DEVN.	.586933	.862838

DIFFERENCE IN MEAN VALUES .01 %

SCANNING RATE 7 CHANNELS / SECOND

CHANNEL 25 50 CONSECUTIVE SCANS.

-860	-859	-858	-859	-858
-861	-860	-861	-860	-859
-858	-859	-858	-859	-858
-861	-860	-859	-858	-859
-858	-861	-860	-861	-860
-859	-858	-859	-858	-861
-860	-861	-860	-859	-858
-859	-858	-859	-859	-859
-858	-859	-860	-859	-858
-859	-858	-859	-858	-859

CHANNEL 25 50 INTERRUPTED SCANS.

-863	-861	-861	-861	-861
-861	-861	-861	-861	-861
-861	-861	-861	-861	-861
-861	-861	-861	-861	-859
-861	-861	-861	-861	-859
-861	-861	-861	-861	-859
-861	-861	-861	-861	-861
-861	-861	-861	-861	-861
-861	-861	-861	-861	-861
-861	-859	-861	-861	-861

DATA ANALYSIS.

CHANNEL 25

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-861 TO -858	-863 TO -859
MEAN	-859.16	-860.88
ST. DEVN.	1.0174	.627271
DIFFERENCE IN MEAN VALUES		-.21 %
SCANNING RATE	7 CHANNELS / SECOND.	

CHANNEL 26 50 CONSECUTIVE SCANS.

-841	-841	-841	-843	-842
-841	-840	-841	-840	-841
-840	-841	-841	-841	-840
-841	-840	-841	-841	-841
-841	-841	-841	-841	-840
-841	-841	-841	-840	-841
-841	-841	-841	-841	-840
-841	-841	-841	-841	-841
-841	-841	-841	-841	-841
-841	-841	-840	-841	-841
-841	-841	-841	-841	-841

CHANNEL 26 50 INTERRUPTED SCANS.

-841	-843	-841	-841	-841
-841	-843	-841	-841	-843
-841	-841	-841	-841	-841
-841	-841	-841	-843	-841
-841	-841	-841	-843	-841
-841	-841	-841	-843	-841
-841	-841	-841	-841	-841
-841	-841	-841	-841	-841
-841	-841	-841	-841	-841
-841	-841	-841	-841	-841
-841	-841	-841	-841	-841

DATA ANALYSIS.

CHANNEL 26

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-843 TO -840	-843 TO -841
MEAN	-840.88	-841.24
ST. DEVN.	.520596	.656521

DIFFERENCE IN MEAN VALUES $-5E-01$ %

SCANNING RATE 7 CHANNELS / SECOND.

CHANNEL 27 50 CONSECUTIVE SCANS.

-829	-831	-830	-831	-830
-831	-830	-831	-831	-831
-830	-829	-828	-831	-830
-831	-830	-829	-830	-829
-829	-831	-828	-831	-830
-831	-830	-831	-829	-829
-828	-831	-830	-831	-830
-831	-830	-829	-828	-831
-829	-831	-830	-831	-830
-829	-830	-831	-829	-829

CHANNEL 27 50 INTERRUPTED SCANS.

-831	-833	-833	-831	-831
-831	-831	-831	-831	-831
-831	-831	-831	-831	-833
-831	-833	-831	-831	-831
-831	-833	-831	-831	-831
-831	-831	-831	-833	-831
-831	-831	-831	-831	-831
-831	-831	-831	-833	-831
-833	-831	-831	-833	-831
-831	-831	-831	-831	-831

DATA ANALYSIS.

CHANNEL 27.

	CONSECUTIVE SCANS	INTERRUPTED. SCANS
SAMPLE RANGE	-831 TO -828	-833 TO -831
MEAN	-829.98	-831.36
ST. DEVN.	.979172	.776176

DIFFERENCE IN MEAN VALUES - .17 %

SCANNING RATE 7. CHANNELS / SECOND.

CHANNEL 28 50 CONSECUTIVE SCANS.

-974	-975	-974	-975	-974
-975	-974	-975	-974	-973
-973	-975	-973	-975	-974
-975	-974	-975	-974	-973
-973	-973	-973	-975	-974
-975	-974	-973	-974	-975
-973	-975	-974	-975	-974
-975	-974	-975	-973	-975
-974	-975	-974	-975	-974
-975	-974	-975	-973	-975

CHANNEL 28 50 INTERRUPTED SCANS.

-977	-975	-975	-977	-975
-977	-977	-975	-977	-977
-977	-977	-977	-977	-975
-975	-975	-977	-975	-977
-977	-975	-977	-975	-977
-977	-975	-977	-975	-975
-975	-977	-977	-977	-977
-975	-977	-975	-977	-977
-975	-975	-975	-977	-975
-977	-977	-977	-977	-975

DATA ANALYSIS.

CHANNEL 28

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-975 TO -973	-977 TO -975
MEAN	-974.2	-976.16
ST. DEVN.	.782461	.997139
DIFFERENCE IN MEAN VALUES		-.21 %
SCANNING RATE	7 CHANNELS / SEC QND.	

CHANNEL 29 50 CONSECUTIVE SCANS.

-958	-959	-958	-959	-958
-959	-957	-957	-957	-959
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-956
-957	-957	-957	-957	-957
-957	-957	-957	-957	-956
-957	-957	-959	-957	-957

CHANNEL 29 50 INTERRUPTED SCANS.

-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957
-957	-957	-957	-957	-957

DATA ANALYSIS.

CHANNEL 29

	CONSECUTIVE SCANS	INTERRUPTED SCANS
SAMPLE RANGE	-959 TO -956	-957 TO -957
MEAN	-957.22	-957
ST. DEVN.	.678834	0

DIFFERENCE IN MEAN VALUES -.3E-01 %

SCANNING RATE 7 CHANNELS / SECOND.

TABLE 7.5
ENSEMBLE TEST

ENSEMBLE TEST
SCANS PER SECOND

TABLE 7.5 PAGE 1

ENSEMBLE TEST SCANS PER SECOND

TABLE 7.5.

Ensemble Test and Accuracy Codes - Printouts
at 7, 9, 12 and 14 channels per second,
showing the accuracy of incoming data.

A. CHARD. ENSEMBLE TEST.
 SCANNING RATE 7 CHANNELS/SECOND.

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 1			
		SAMPLE MEAN. FOR GIVEN NO. OF SCANS.	1	2	3
12	4033.32	4034	4034	4034	4034
13	375.74	371	373	374	374.75
14	9999	9999	9999	9999	9999
15	1132.13	3999	6999	7999	3499
16	2797.34	2798	2797.5	2797.33	2797.25
17	3953.7	3956	3955.5	3955.33	3955.5
18	1276.44	1279	1273	1277.66	1277.5
19	3533.93	3542	3542	3542	3541.75
20	337.82	369	363.5	363.33	363.25
21	333.34	366	366	366	366
22	345.16	372	372	372	372
23	336.8	363	363	363	363.25
24	337.76	364	364	364	364
25	343.54	373	373	373	373
26	349.72	374	374	374	374
27	340.04	363	363	363	363
28	361.06	333	333	333	333
29	354.53	375	374.5	374.66	374.75

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 2			
		SAMPLE MEAN 5	FOR GIVEJ 6	NO. OF 7	SCANS. 3
12	4033.32	4034	4034	4034	4033.87
13	375.74	375.2	375.5	375.71	375.37
14	9999	9999	9999	9999	9999
15	1132.13	8799	7514.16	6433	5673.37
16	2797.34	2797.2	2797.16	2797.14	2797
17	3953.7	3955.6	3955.5	3955.42	3955.25
18	1276.44	1277.39	1277.33	1277.23	1277.25
19	3533.93	3541.3	3541.66	3541.57	3541.5
20	337.32	363.2	363.16	363.14	363.12
21	333.34	366	366.16	366.23	366.37
22	345.16	372	372	372	372
23	336.3	363.4	363.5	363.57	363.62
24	337.76	364	364	364	364
25	343.54	373	373	373	373
26	349.72	373.8	373.33	373.35	373.37
27	343.04	363	363	363.14	363.12
28	361.06	333	333	333	333
29	354.53	374.8	374.33	374.71	374.75

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 3			
		SAMPLE MEAN FOR GIVEN NO. OF SCANS.			
		9	10	11	12
12	4033.32	4033.33	4033.9	4033.31	4033.75
13	375.74	376	376.1	376.13	376.25
14	9999	9999	9999	9999	9999
15	1132.18	5014.33	4474	4033.72	3692.66
16	2797.34	2796.88	2796.8	2796.72	2796.66
17	3953.7	3955.11	3955	3954.9	3954.83
18	1276.44	1277.22	1277.2	1277.13	1277.03
19	3538.93	3541.44	3541.4	3541.36	3541.33
20	337.32	363.11	363.1	363.09	363.03
21	338.34	366.44	366.6	366.72	366.33
22	345.16	372	372	372	372
23	336.3	363.66	363.7	363.72	363.75
24	337.76	364	364	364	364
25	343.54	373	373	373	373
26	349.72	373.83	373.9	373.9	373.91
27	340.34	363.11	363.1	363.09	363.03
28	361.06	333	333	333.09	333.16
29	354.58	374.77	374.3	374.9	375

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 4			
		SAMPLE MEAN 13	FOR GIVEN 14	NO. OF 15	SCAJS. 16
12	4033.32	4033.69	4033.64	4033.66	4033.62
13	375.74	376.3	376.35	376.4	376.43
14	9999	9999	9999	9999	9999
15	1132.13	3422.15	3191.23	2937.2	2301.5
16	2797.34	2796.61	2796.57	2796.53	2796.5
17	3953.7	3954.76	3954.64	3954.53	3954.43
18	1276.44	1277	1276.92	1276.36	1276.31
19	3533.93	3541.3	3541.28	3541.26	3541.25
20	337.82	363.07	363.07	363.06	363.36
21	333.34	366.92	367	367.06	367.12
22	345.16	372	372.07	372.13	372.13
23	336.3	363.76	363.73	363.3	363.31
24	337.76	364	364	364	364.06
25	343.54	373	373	373.06	373.12
26	349.72	373.92	373.92	373.93	373.93
27	340.04	363.07	363.07	363.06	363.12
28	361.06	333.23	333.23	333.33	333.37
29	354.53	375.07	375.21	375.33	375.43

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 5 FOR GIVEN NO. OF SCANS.			
		17	13	19	20
12	4033.32	4033.53	4033.61	4033.63	4033.65
13	375.74	376.47	376.5	376.52	376.55
14	9999	9999	9999	9999	9999
15	1132.13	2633.53	2490.22	2360.42	2243.54
16	2797.34	2796.47	2796.44	2796.42	2796.4
17	3953.7	3954.29	3954.11	3953.94	3953.3
18	1276.44	1276.76	1276.72	1276.63	1276.64
19	3533.93	3541.23	3541.22	3541.15	3541.1
20	337.32	363.05	363.11	363.1	363.15
21	333.34	367.17	367.22	367.26	367.3
22	345.16	372.23	372.27	372.31	372.35
23	336.8	363.32	363.33	363.34	363.35
24	337.76	364.11	364.16	364.21	364.25
25	343.54	373.17	373.22	373.21	373.25
26	349.72	373.94	373.94	373.94	373.95
27	340.04	363.17	363.22	363.26	363.3
28	361.06	333.41	333.44	333.52	333.6
29	354.53	375.47	375.55	375.63	375.7

ACCURACY CODES.

RESULTS PAGE

6

INDICATING DEVIATION OF SAMPLE MEAN
FROM TRUE MEAN.

BLANK	>	10	%	
A	<	10	BUT >	8 %
B	<	8	BUT >	7 %
C	<	7	BUT >	6 %
D	<	6	BUT >	5 %
E	<	5	BUT >	4 %
F	<	4	BUT >	3 %
G	<	3	BUT >	2 %
H	<	2	BUT >	1 %
*	<=	1	%	

SCANNING RATE 7 CHANNELS/SEC.

A. CHARD.
SCANNING RATE

ENSEMBLE TEST.
9 CHANNELS/SECOND.

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 1			
		SAMPLE MEAN 1	FOR GIVEN 2	NO. OF 3	SCANS. 4
12	4036.34	4042	4039.5	4033.33	4037.75
13	336.9	333	334.5	335.66	336.25
14	9999	9999	9999	9999	9999
15	1735.36	2999	6499	7665.66	3249
16	2304.32	2600	2449	2393.66	2373.5
17	6493.32	6439	6491.5	6492.33	6492.75
18	1297.6	1307	1303	1301.33	1300.25
19	3512.6	3511	3512	3512	3512.25
20	902.06	907	905	904.33	904
21	902.64	903	903	903.33	903.5
22	905.8	907	906.5	906.33	906.25
23	834.92	837	836	836	836
24	867.12	867	867.5	863	863
25	893.88	893	893.5	893.66	893.75
26	897.92	893	893	893	893
27	869.28	870	870	870.33	870.25
28	992.53	995	995	994.66	994.75
29	967.24	957	963.5	965.66	966.75

CHANNEL NUMBER	TRUE MEAN.	SAMPLE MEAN	RESULTS PAGE 2		
			FOR GIVEN	NO. OF	SCANS.
		5	6	7	8
12	4036.34	4037.4	4037.16	4037	4036.37
13	336.9	336.6	336.33	337	337.12
14	9999	9999	9999	9999	9999
15	1735.06	3599	7722.5	6673.42	5846.37
16	2334.32	2353.4	2343.33	2341.14	2335.75
17	6493.32	6493	6493.16	6493.23	6493.37
13	1297.6	1299.6	1299.33	1299.14	1299
19	3512.6	3512.4	3512.33	3512.23	3512.37
20	902.06	903.3	903.66	903.57	903.5
21	902.64	903.6	903.5	903.42	903.37
22	905.8	906.2	906.16	906.14	906.12
23	834.92	835.3	835.66	835.71	835.75
24	867.12	868.2	863.16	863.14	863.12
25	393.33	393.3	393.33	393.35	393.37
26	397.92	393	393	393	393.25
27	369.23	370.2	370.16	370.23	370.25
28	992.53	994.6	994.5	994.57	994.5
29	967.24	967.4	967.33	963.14	963.37

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 3			
		SAMPLE MEAN 9	FOR GIVEN 10	NO. OF 11	SCANS. 12
12	4036.34	4036.77	4036.7	4036.63	4036.53
13	386.9	387.22	387.3	337.36	337.41
14	9999	9999	9999	9999	9999
15	1735.06	5174.22	4611.5	4151.45	3773.91
16	2304.32	2331.55	2323.4	2325.31	2323.66
17	6493.32	6493.44	6493.5	6493.54	6493.53
13	1297.6	1293.33	1293.3	1293.72	1293.66
19	3512.6	3512.33	3512.4	3512.36	3512.33
20	902.06	903.44	903.4	903.36	903.33
21	902.64	903.44	903.4	903.36	903.33
22	905.3	906.11	906.1	906.09	906.03
23	884.92	835.66	835.7	835.72	835.75
24	867.12	863.11	863.1	863.13	863.25
25	893.33	393.33	393.9	393.9	393.91
26	897.92	393.33	393.3	393.27	393.25
27	869.23	370.33	370.3	370.36	370.41
28	992.53	994.55	994.5	994.54	994.5
29	967.24	963.55	963.7	963.31	963.91

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 4			
		SAMPLE MEAN FOR GIVEN NO. OF SCANS.			
		13	14	15	16
12	4036.34	4036.53	4036.5	4036.46	4036.43
13	336.9	337.46	337.5	337.53	337.56
14	9999	9999	9999	9999	9999
15	1735.36	3483.69	3250.35	3048.93	2369.43
16	2304.32	2321.69	2320.14	2318.79	2317.62
17	6493.32	6493.61	6493.64	6493.66	6493.63
18	1297.6	1298.61	1298.57	1298.53	1298.5
19	3512.6	3512.38	3512.35	3512.33	3512.31
20	902.06	903.3	903.23	903.2	903.12
21	902.64	903.3	903.35	903.33	903.31
22	905.3	906.07	906.07	906.06	906.06
23	884.92	885.76	885.71	885.73	885.63
24	867.12	868.3	868.35	868.4	868.37
25	893.88	893.92	893.92	893.93	893.93
26	897.92	898.33	898.42	898.46	898.43
27	869.23	870.33	870.35	870.4	870.43
28	992.53	994.53	994.5	994.46	994.43
29	967.24	969	969.07	969.13	969.13

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 5			
		SAMPLE MEAN 17	FOR GIVEN 18	NO. OF 19	SCANS. 20
12	4036.34	4036.41	4036.33	4036.36	4036.35
13	336.9	337.53	337.61	337.63	337.65
14	9999	9999	9999	9999	9999
15	1735.06	2704.82	2550.27	2414.05	2296.35
16	2304.32	2316.53	2315.55	2314.63	2313.9
17	6493.82	6493.7	6493.66	6493.63	6493.65
18	1297.6	1293.47	1293.44	1293.42	1293.35
19	3512.6	3512.29	3512.33	3512.31	3512.3
20	902.06	903.05	903	902.94	902.9
21	902.64	903.29	903.27	903.26	903.25
22	905.8	906.05	906.05	906.05	906.05
23	834.92	835.64	835.61	835.63	835.6
24	867.12	868.41	868.33	868.42	868.4
25	893.33	893.94	894	894.05	894.1
26	897.92	898.41	893.33	898.42	893.4
27	869.23	870.47	870.5	870.52	870.55
28	992.53	994.47	994.44	994.42	994.4
29	967.24	969.23	969.27	969.36	969.4

ACCURACY CODES. RESULTS PAGE
INDICATING DEVIATION OF SAMPLE MEAN
FROM TRUE MEAN.

BLANK	>	10	%		
A	<	10	BUT	>	3 %
B	<	8	BUT	>	7 %
C	<	7	BUT	>	6 %
D	<	6	BUT	>	5 %
E	<	5	BUT	>	4 %
F	<	4	BUT	>	3 %
G	<	3	BUT	>	2 %
H	<	2	BUT	>	1 %
*	<=	1	%		

SCANNING RATE 9 CHANNELS / SECOND.

A. CHARD. ENSEMBLE TEST.
 SCANNING RATE 12 CHANNELS / SEC OND.

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 1			
		SAMPLE 1	MEAN FOR 2	NO. OF 3	SCANS. 4
12	4034.82	4051	4042.5	4040	4038.5
13	383.52	378	379	380	380.75
14	9999	9999	9999	9999	9999
15	1749.76	2729	6364	7575.66	8181.5
16	2343.44	3100	2713.5	2584.66	2520.25
17	6513.62	6499	6507	6510	6511.5
18	1296.96	1312	1304.5	1301.66	1300.5
19	3493.4	3487	3490	3491	3491.5
20	908.24	915	911.5	910.33	909.75
21	909.92	910	910	910	910
22	912.5	913	913	912.66	912.5
23	894	896	895	894.66	894.5
24	804.06	813	813	812.33	812
25	903.2	895	900	901.66	902.5
26	907.8	907	907.5	907.66	907.75
27	876.3	878	877.5	877.33	877.25
28	1000.94	1001	1002	1002.33	1002.5
29	977.24	994	986.5	984	982.75

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 2			
		SAMPLE MEAN FOR GIVEN 5	6	NO. OF SCANS. 7	8
12	4034.82	4037.6	4037.16	4036.85	4036.62
13	383.52	381.4	381.83	382.14	382.5
14	9999	9999	9999	9999	9999
15	1749.76	8545	7804.16	6752.14	5918.12
16	2343.44	2481.79	2456.16	2437.85	2424.12
17	6513.62	6512.4	6513	6513.42	6513.75
18	1296.96	1299.8	1299.16	1298.85	1298.5
19	3493.4	3491.8	3492	3492.14	3492.25
20	908.24	909.4	909.16	909	908.87
21	909.92	910.2	910.33	910.42	910.5
22	912.5	912.6	912.5	912.57	912.62
23	894	894.4	894.33	894.28	894.25
24	804.06	811.8	811.66	811.57	811.37
25	903.2	902.8	903	903.14	903.25
26	907.8	907.8	907.83	907.85	907.87
27	876.3	877.2	877.16	877.14	877.12
28	1000.94	1002.6	1002.66	1002.71	1002.75
29	977.24	982	981.5	981.28	981

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 3			
		SAMPLE MEAN FOR GIVEN 9	10	NO. OF SCANS. 11	12
12	4034.82	4036.33	4036.2	4036.09	4036
13	383.52	382.77	383	383.18	383.33
14	9999	9999	9999	9999	9999
15	1749.76	5237.55	4666.8	4198	3817.16
16	2343.44	2413.44	2404.9	2397.9	2392.08
17	6513.62	6514	6514.2	6514.27	6514.33
18	1296.96	1298.33	1298.2	1298.09	1298
19	3493.4	3492.33	3492.4	3492.54	3492.66
20	908.24	909	908.9	908.9	908.91
21	909.92	910.55	910.7	910.72	910.75
22	912.5	912.66	912.7	912.72	912.75
23	894	894.22	894.3	894.27	894.33
24	804.06	811.33	811.3	811.18	811.08
25	903.2	903.33	903.3	903.36	903.41
26	907.8	907.88	907.9	907.9	907.91
27	876.3	877.11	877.1	877.09	877.08
28	1000.94	1002.77	1002.8	1002.81	1002.83
29	977.24	980.88	980.7	980.63	980.58

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 4			
		SAMPLE MEAN FOR GIVEN NO. OF SCANS.			
		13	14	15	16
12	4034.82	4035.92	4035.92	4035.86	4035.87
13	383.52	383.46	383.57	383.66	383.62
14	9999	9999	9999	9999	9999
15	1749.76	3514.3	3277.78	3074.8	2894.93
16	2343.44	2387.15	2382.92	2379.26	2376.06
17	6513.62	6514.38	6514.42	6514.4	6514.37
18	1296.96	1297.92	1297.85	1297.8	1297.75
19	3493.4	3492.76	3492.85	3492.93	3493
20	908.24	908.92	908.92	908.93	908.87
21	909.92	910.76	910.78	910.86	910.87
22	912.5	912.76	912.78	912.8	912.81
23	894	894.38	894.42	894.4	894.37
24	804.06	811.07	811	810.93	810.93
25	903.2	903.46	903.5	903.53	903.56
26	907.8	907.92	907.92	907.93	907.93
27	876.3	877.07	877.07	877.06	877.06
28	1000.94	1002.84	1002.85	1002.86	1002.87
29	977.24	980.46	980.42	980.4	980.37

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 5			
		SAMPLE MEAN 17	FOR GIVEN 18	NO. OF 19	SCANS. 20
12	4034.82	4035.76	4035.72	4035.68	4035.65
13	383.52	383.58	383.61	383.63	383.65
14	9999	9999	9999	9999	9999
15	1749.76	2730.47	2576.38	2435.94	2316.79
16	2343.44	2373.23	2370.72	2368.57	2366.54
17	6513.62	6514.35	6514.33	6514.31	6514.3
18	1296.96	1297.7	1297.66	1297.63	1297.6
19	3493.4	3493.05	3493.11	3493.15	3493.2
20	908.24	908.88	908.83	908.78	908.75
21	909.92	910.88	910.88	910.89	910.9
22	912.5	912.82	912.83	912.84	912.85
23	894	894.41	894.44	894.47	894.5
24	804.06	810.94	811	811.05	811.05
25	903.2	903.58	903.61	903.63	903.65
26	907.8	907.94	907.94	907.94	907.95
27	876.3	877	877	877	877
28	1000.94	1002.88	1002.88	1002.89	1002.9
29	977.24	980.29	980.22	980.21	980.2

ACCURACY CODES. RESULTS PAGE 6
INDICATING DEVIATION OF SAMPLE MEAN
FROM TRUE MEAN.

BLANK	>	10	%				
A	<	10	BJT	>	8	%	
B	<	8	BJT	>	7	%	
C	<	7	BJT	>	6	%	
D	<	6	BJT	>	5	%	
E	<	5	BJT	>	4	%	
F	<	4	BJT	>	3	%	
G	<	3	BJT	>	2	%	
H	<	2	BJT	>	1	%	
*	<=	1	%				

SCANNING RATE 12 CHANNELS / SECOND.

A. CHARD. ENSEMBLE TEST.
 SCANNING RATE 14 CHANNELS/SECOND.

CHANNEL NUMBER	TRUE MEAN.	SAMPLE MEAN	RESULTS PAGE 1 FOR GIVEN NO. OF SCANS.			
			1	2	3	4
12	4029.02	4104	4065.5	4053	4046.5	
13	363.2	367	363	367.66	363	
14	9999	9999	9999	9999	9999	
15	1747.63	2549	6269.5	7512.66	3134.25	
16	2371.02	5400	3343.5	3324.66	3065.25	
17	6470	6399	6436	6443.33	6454.5	
18	1295.13	1360	1327.5	1316.66	1311.25	
19	3503.42	3469	3437	3493	3496	
20	896.7	920	909.5	905.33	903.25	
21	899.36	895	897	897.66	893	
22	896	903	899.5	893	897.5	
23	877.62	879	879	879	879	
24	863.94	861	860	859.66	859.5	
25	887.3	910	899.5	896	894.25	
26	894.54	893	893.5	894	894.25	
27	863.32	863	863.5	863.33	863.25	
28	1016.13	1013	1015	1015.66	1016	
29	935.9	1006	999	996	994.25	

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 2			
		SAMPLE MEAN 5	FOR GIVEN 6	NO. OF 7	SCANS. 8
12	4029.02	4042.6	4040.16	4033.28	4037
13	363.2	363.2	363.33	363.42	363.5
14	9999	9999	9999	9999	9999
15	1747.63	8507.19	7322.66	6763	5929.5
16	2371.02	2909.6	2806.16	2732.28	2676.37
17	6470	6453.2	6460.66	6462.14	6463.25
18	1295.13	1308	1305.33	1304.14	1302.37
19	3503.42	3497.3	3499	3499.35	3500.5
20	396.7	902	901.16	900.57	900.12
21	399.36	393.2	393.33	393.42	393.5
22	396	397.2	397	397	397
23	377.62	379	379	379	379
24	363.94	359.4	359.33	359.28	359.25
25	387.3	393	392.16	391.71	391.25
26	394.54	394.2	394.33	394.42	394.5
27	363.32	363.2	363.16	363.14	363.25
28	1016.13	1016	1016	1016	1016
29	935.9	993.6	993.16	992.71	992.25

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 3			
		SAMPLE MEAN FOR GIVEN NO. OF SCANS.			
		9	10	11	12
12	4029.02	4035.33	4035.1	4034.36	4033.75
13	363.2	363.55	363.6	363.63	363.66
14	9999	9999	9999	9999	9999
15	1747.68	5249.33	4680	4213.63	3335
16	2371.02	2633.77	2599.29	2571.09	2547.53
17	6470	6464.33	6465.2	6465.54	6465.33
18	1295.13	1302	1301.3	1300.72	1300.25
19	3503.42	3501	3501.5	3501.81	3502.16
20	896.7	899.77	899.5	899.27	899.03
21	899.36	893.55	898.6	893.63	893.66
22	896	897	897	897	897
23	877.62	879	879	879	879
24	863.94	859.66	860	860.27	860.66
25	887.3	890.77	890.4	890.27	890.41
26	894.54	894.44	894.5	894.54	894.53
27	863.32	863.44	863.6	863.72	863.33
28	1016.13	1016	1016	1016	1016.03
29	935.9	992.11	992	991.9	991.75

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 4			
		SAMPLE MEAN FOR GIVEN NO. OF SCANS.			
		13	14	15	16
12	4029.02	4033.33	4033	4032.66	4032.37
13	363.2	363.69	363.71	363.73	363.75
14	9999	9999	9999	9999	9999
15	1747.63	3533.84	3295.21	3033.4	2903.93
16	2371.02	2527.69	2510.64	2496	2433.12
17	6470	6466.07	6466.23	6466.46	6466.62
18	1295.13	1299.34	1299.5	1299.2	1293.93
19	3503.42	3502.46	3502.64	3502.36	3503.06
20	396.7	393.92	393.73	393.66	393.56
21	399.36	393.69	393.71	393.73	393.75
22	396	397	397	397	397
23	377.62	379	379	379	379
24	363.94	361	361.35	361.66	362
25	337.3	390.33	390.23	390.2	390.25
26	394.54	394.61	394.64	394.66	394.63
27	363.32	363.92	364	364.06	364.13
28	1016.13	1016.07	1016.07	1016.06	1016.12
29	935.9	991.69	991.5	991.33	991.13

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 5			
		SAMPLE MEAN 17	FOR GIVEN 18	NO. OF 19	SCANS. 20
12	4029.32	4032.11	4031.33	4031.63	4031.45
13	363.2	363.76	363.77	363.73	363.7
14	9999	9999	9999	9999	9999
15	1747.63	2735.11	2577.33	2435.39	2316.34
16	2371.32	2471.7	2461.55	2452.47	2444.29
17	6470	6466.76	6466.33	6467	6467.1
18	1295.13	1293.64	1293.38	1293.21	1293
19	3503.42	3503.23	3503.33	3503.47	3503.6
20	396.7	393.47	393.33	393.31	393.25
21	399.36	399	399.16	399.31	399.45
22	396	397	397	397	397
23	377.62	373.33	373.33	373.73	373.75
24	363.94	362.29	362.55	362.73	362.95
25	337.3	393.29	393.33	393.36	393.4
26	394.54	394.7	394.72	394.73	394.75
27	363.32	364.23	364.27	364.31	364.35
28	1016.13	1016.17	1016.22	1016.26	1016.3
29	935.9	991.05	990.94	990.34	990.75

ACCURACY CODES. RESULTS PAGE 6
INDICATING DEVIATION OF SAMPLE MEAN
FROM TRUE MEAN.

BLANK	>	10	%		
A	<	10	BUT	>	3 %
B	<	8	BUT	>	7 %
C	<	7	BUT	>	6 %
D	<	6	BUT	>	5 %
E	<	5	BUT	>	4 %
F	<	4	BUT	>	3 %
G	<	3	BUT	>	2 %
H	<	2	BUT	>	1 %
*	<=	1	%		

SCANNING RATE 14 CHANNELS / SECOND.

ACCURACY CODES.

CHANNEL
NUMBER

ACCURACY CODE.
1 TO 10 SCANS.

12	H	*	*	*	*	*	*	*	*	*
13	*	*	*	*	*	*	*	*	*	*
14	*	*	*	*	*	*	*	*	*	*
15										
16										A
17	H	*	*	*	*	*	*	*	*	*
18	D	G	H	H	*	*	*	*	*	*
19	*	*	*	*	*	*	*	*	*	*
20	G	H	*	*	*	*	*	*	*	*
21	*	*	*	*	*	*	*	*	*	*
22	*	*	*	*	*	*	*	*	*	*
23	*	*	*	*	*	*	*	*	*	*
24	*	*	*	*	*	*	*	*	*	*
25	G	H	*	*	*	*	*	*	*	*
26	*	*	*	*	*	*	*	*	*	*
27	*	*	*	*	*	*	*	*	*	*
28	*	*	*	*	*	*	*	*	*	*
29	G	H	H	*	*	*	*	*	*	*

A. CIAPD

ENSEMBLE TEST

WARM UP PERIOD

PERIOD 1/2 SECOND

TABLE 7.6.

Ensemble Test and Accuracy Codes - an execution during the warm up period showing how this affects the results.

PUNCH
CONTINUEA. CHARD. ENSEMBLE TEST.
SCANNING RATE 13 CHANNELS/SECOND.

CHANNEL NUMBER	TRUE MEAN.	SAMPLE MEAN 1	RESULTS FOR GIVEN NO. OF SCANS.		PAGE 1	
			2	3	4	5
12	3972.42	939	2438	3033.66	3261.5	
13	293.32	4035	1326.5	1033.66	719.75	
14	9806.73	333	5193.5	6795.33	7596.25	
15	1987.26	9999	9999	9999	9999	
16	2230.22	22	1151.5	1543.33	1739.5	
17	6434.32	2323	4425	5123.33	5472.75	
18	1401.2	6513	3933	3037.66	2602.5	
19	3450.52	1296	2396.5	2763	2946.25	
20	943.98	3495	2203	1772.33	1556.75	
21	895.53	910	909.5	909	903.75	
22	905.56	910	913.5	915	915.75	
23	837.6	918	903.5	906	904.75	
24	809.4	899	863	857.33	852	
25	392.36	835	871	833	889	
26	900.24	907	910	910.66	911	
27	873.36	912	897	892	839.5	
28	955.62	882	927	941.66	949	
29	933.92	971	961.5	953.33	956.75	

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 2			
		SAMPLE MEAN 5	FOR GIVEN 6	NO. OF 7	SCANS. 8
12	3972.42	3416.2	3519.33	3593	3643.12
13	298.32	493.2	350.5	245	165.87
14	9306.73	3076.3	3397.16	3326	3797.62
15	1937.26	9999	9999	3713.23	7649
16	2230.22	1857.2	1935.66	1991.71	2033.75
17	6434.32	5632.2	5321.66	5921.14	5995.37
18	1401.2	2341.4	2167.16	2042.71	1949.37
19	3450.52	3056.2	3129.5	3131.35	3221
20	943.93	1427.4	1341.16	1279.57	1233.37
21	895.53	903.6	903.5	903.42	903.37
22	905.56	916.2	916.5	916.71	916.37
23	837.6	904	903.5	903	902.5
24	809.4	348.3	346.33	345.23	344.12
25	892.36	392.6	395	396.71	393
26	900.24	911.2	911.33	911.42	911.5
27	870.36	833	337	336.23	335.75
28	955.62	953.4	956.33	953.42	963
29	933.92	955.3	955.16	954.71	954.37

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 3 FOR GIVEN NO. OF SCANS.			
		SAMPLE MEAN 9	10	11	12
12	3972.42	3691.11	3725.5	3753.63	3777
13	293.82	104.33	55.1	14.81	13.75
14	9806.78	8931.11	9037.9	9125.27	9193.83
15	1987.26	6792.66	6031.5	5437.31	4993.91
16	2230.22	2066.44	2392.6	2114	2131.83
17	6434.32	6054	6100.6	6133.63	6170.33
18	1401.2	1376.77	1313.7	1771.13	1731.53
19	3450.52	3251.44	3275.9	3295.9	3312.53
20	943.93	1197.44	1163.7	1145.13	1125.53
21	395.53	903.33	903.3	903.45	903.53
22	905.56	917	917.1	917.13	917.25
23	387.6	902.22	902	901.31	901.66
24	309.4	343.22	342.5	341.9	341.41
25	392.36	899	399.3	900.36	900.33
26	900.24	911.55	911.6	911.63	911.66
27	370.36	335.33	335	334.72	334.5
28	955.62	961.22	962.2	963	963.66
29	938.92	954.11	953.9	953.31	953.75

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 4 FOR GIVEN NO. OF SCANS.			
		13	14	15	16
12	3972.42	3796.84	3313.35	3323.6	3541.5
13	293.32	47.15	71.5	92.6	111.06
14	9336.73	9259.69	9312.5	9353.26	9393.31
15	1937.26	4599.34	4231.21	4011.4	3773.37
16	2230.22	2146.92	2159.35	2171.06	2130.37
17	6434.32	6197.15	6220.14	6240.06	6257.5
18	1401.2	1693.07	1669.35	1644.46	1622.63
19	3450.52	3326.69	3333.73	3349.26	3353.43
20	943.93	1109	1094.73	1032.46	1071.63
21	895.53	903.69	903.73	903.36	933.93
22	905.56	917.3	917.35	917.4	917.43
23	337.6	901.53	901.42	901.33	901.25
24	309.4	341	340.64	340.33	340
25	392.36	901.23	901.57	901.36	902.12
26	900.24	911.69	911.71	911.73	911.75
27	370.36	334.3	334.14	334	333.37
28	955.62	964.23	964.71	965.13	965.5
29	933.92	953.61	953.5	953.4	953.31

CHANNEL NUMBER	TRUE MEAN.	RESULTS PAGE 5			
		SAMPLE MEAN 17	FOR GIVEN 18	NO. OF SCANS. 19	20
12	3972.42	3352.83	3363	3372.35	3333.2
13	293.32	127.35	141.33	154.73	166.45
14	9306.73	9433.64	9465.05	9493.15	9513.44
15	1937.26	3559.29	3361.33	3133.63	3024.4
16	2230.22	2139.52	2197.22	2234.1	2210.29
17	6434.32	6272.32	6236.44	6293.63	6309.6
18	1401.2	1603.47	1536.33	1571.1	1557.35
19	3450.52	3366.47	3373.61	3330	3335.75
20	948.93	1062.17	1053.72	1046.21	1039.45
21	895.53	909	909.05	909.1	909.15
22	905.56	917.47	917.5	917.52	917.55
23	887.6	901.17	901.11	901.05	900.95
24	809.4	839.76	839.55	839.36	839.2
25	892.36	902.41	902.66	902.39	903.1
26	900.24	911.76	911.77	911.73	911.35
27	870.86	883.76	883.66	883.57	883.5
28	955.62	965.82	966.11	966.36	966.6
29	933.92	953.23	953.22	953.21	953.2

ACCURACY CODES. RESULTS PAGE
INDICATING DEVIATION OF SAMPLE MEAN
FROM TRUE MEAN.

6

BLANK	>	10	%		
A	<	10	BUT	>	3 %
B	<	8	BUT	>	7 %
C	<	7	BUT	>	6 %
D	<	6	BUT	>	5 %
E	<	5	BUT	>	4 %
F	<	4	BUT	>	3 %
G	<	3	BUT	>	2 %
H	<	2	BUT	>	1 %
*	<=	1	%		

SCANNING RATE 13 CHANNELS / SECOND.

ACCURACY CODES.

RESULTS PAGE 7

CHANNEL
NUMBER

ACCURACY CODE.
1 TO 10 SCANS.

12						A	A	B	C
13									
14								A	B
15									
16								A	A
17						A	B	C	D
18									
19						A	B	C	D
20									
21	H	H	H	H	H	H	H	H	H
22	*	*	H	H	H	H	H	H	H
23	F	G	G	H	H	H	H	H	H
24		B	D	D	E	E	E	E	E
25	C	G	H	*	*	*	*	*	*
26	*	H	H	H	H	H	H	H	H
27	E	F	G	G	H	H	H	H	H
28	B	G	H	*	*	*	*	*	*
29	F	G	G	H	H	H	H	H	H

8. CONCLUSION

The system has been developed and implemented, in use it has enabled programs to be developed quickly and easily because of the inter-active nature of BASIC. It could be argued that the running speed is slow compared with that which might be obtained using FORTRAN as the written language. However, the ease and speed of program development far outweighs this consideration.

The data acquisition subroutines are themselves in machine code and hence, operate at the maximum possible speed which compensates to a large extent for the relatively slow BASIC processing.

APPENDIX 1

BASOON -- ON-LINE FROM BASIC

Basoon is an inter-active on-line package.

On Line capability is provided via the MDP-200 data logging system and the inter-active facility by the computer programming language BASIC. The user does not need any previous knowledge of data logging or of the MDP-200 system. All the conversions and programs which refer to the data logging are done by sub-routines which are called from BASIC.

There are five sub-routines in this system.- these are:-

1. Timer and delay
2. Channel Reader
3. MDP Clock and Scan Data Switches.
4. Sense Switch Tester.
5. Audible Warning.

SUBROUTINE CALLS

Timer and Delay

Calling Sequence -CALL (1, ARG1, ARG2, ARG3)

Use This subroutine is used to time the running of a program and to cause time delays for a given time interval.

e.g. To examine a data value every minute and print its value:-

1. First start the timer.
2. Scan the channel and print the result.
3. Stop timer.
4. Subtract elapsed time from one minute to give required delay.
5. Enter delay routine for this length of delay.
6. Go to beginning to repeat.

ARGUMENTS

ARG1 - TYPE NUMBER

1. For delay. In this case the delay will be ARG2 seconds and ARG3 will be a dummy argument.
2. Start timing. This starts the real time clock in the H 316; the arguments ARG2 and ARG3 are both dummies for this call.
3. Stop timing. This stops the real time clock and sets ARG3 equal to the time elapsed in seconds. For this call ARG2 is a dummy.

After calling for a time start (a "2" call) a time delay may be called for; this will have no effect on the elapsed time except that the delay will be included in it.

During a time delay the user may wish to take action. This he may do in two ways:-

- i) If an R is typed on the ASR the delay will be terminated as if it had run its full course. (Note: if this is done during timing the elapsed time will be the actual time and not the time which would have elapsed after the full delay had run its course.)
- ii) If a C is typed an immediate return will be made to the command mode of BASIC.

Other characters will have no effect in any way.

ERROR MESSAGES

Subroutine 1 incorporates the following error detections:-

- TW - TYPE WRONG. Argument 1 is not a
1,2 or 3.
- TU - TIME UNDERFLOW. Argument 2 is too small.
- RU - RUNNING. Clock is already running when a
start clock call is given.
- NR - NOT RUNNING. Clock was not running when
a stop clock call was given.

Argument 2 may be integer or non-integer. e.g. 1,1.0, 9.02 are all acceptable.

Numbers may be used in place of identifiers in the arguments and I recommend using a number for ARG1 only. Numbers in place of variables for ARG2 and ARG3 may cause undetectable errors.

2. Channel Reader

Calling Sequence CALL (2,ARG1,ARG2).

Use To input the value of a channel from the data logger.

Arguments:-

- ARG1 is the channel number which is to be interrogated.
- ARG2 is what the resulting value is to be called. This will be a 4 digit integer which must be scaled by the user. (see example iii).

(-9999 to 9999 in the actual range
-9.999 to 9.999 volts).

Argument 1 may be a number in the range 1 to 39 because only these channels are built into the system.

However, a wrong call is detected and leads to the following error messages:-

CO Channel Overflow i.e. channel >39

CU Channel Underflow i.e. channel <1

With all error messages the line number is given.

Argument 1 may be a variable or a number.

Program Examples:-

i) To fetch and store

5 DIM K (39)

10 FOR J=1,39

20 CALL (2,J,K(J))

30 NEXT (J)

40 REM THIS WILL SCAN CHANNELS 1 TO 39

50 REM AND PUT THE VALUES IN ARRAY K

ii) Calling by value

10 CALL (2,22,V)

20 PRINT V

will have the same effect as:-

10 C=22

20 CALL (2,C,V)

30 PRINT V

Program Examples

i) To time a calculation:-

```
5      A=0

10     CALL  (1,2,A,A)

20     REM A IS A DUMMY ARGUMENT

30     FOR H=1, 1000

40     G=H+1

50     NEXT H

60     CALL (1,3,A,T)

70     REM A IS A DUMMY

80     REM T IS THE ELAPSED TIME

90     PRINT "THAT TOOK"; T; "SECONDS".

100    STOP
```

In this example T 6:06 seconds.

ii) To cause a delay:-

```
5     D=10

10    CALL (1,1,D,A)

20    REM D IS THE DELAY

30    REM A IS A DUMMY ARGUMENT.
```

Delays up to 655 seconds are allowed. Larger than this overflows the machine word length.

Further use of the delay will be explained under subroutine 2 - Channel reader.

Program Examples (cont'd)

iii) To scan a channel every N seconds:-

```
10 INPUT N,C
20 REM INPUT TIME BETWEEN SCANS N, AND CHANNEL C
30 CALL (1,2,A,A)
40 REM A IS A DUMMY
50 CALL (2,C,V)
60 REM CHANNEL C
70 PRINT V
80 CALL (1,3,A,T)
90 D=N-T
100 REM TIME TO NEXT SCAN=N-TIME ELAPSED
110 CALL (1,1,D,A)
120 REM DELAY FOR D
130 GO TO 30
140 REM GO AND DO IT AGAIN
```

V will be the form 1234, knowing the range is 0-10 volts multiplication by 10^{-4} will give 0.1234 which is the actual voltage of the analogue input.

3. MDP Clock and Scan Data Switches

Calling Sequence CALL (3,ARG1,ARG2,ARG3)

Use i) To read the MDP clock

ii) To input the current value which is set on the scan data switches situated on the MDP mobile cabinet.

Arguments

These must all be variables. If numbers are used the input values will be lost.

ARG1 - will be set equal to the current value of the HOURS and MINUTES from the MDP 200 clock. This is in the form of a four digit integer. e.g. 1249 is 12 hours, 49 minutes.

ARG2 - will be the current value of the seconds and tenths of seconds in a similar form. e.g. 3046 is 30.46 seconds.

ARG3 - is a 4 digit number taken from the thumb switches on the MDP mobile cabinet; again a four digit integer range 0000 to 9999.

Program Examples:-

```
10 CALL (3,A,B,C)
20 PRINT A: "HOURS, MINS"; B/100; "SECS".
30 PRINT "TEST NUMBER"; C
40 REM A USEFUL WAY OF INCLUDING A TEST NO.
```

4. Sense Switches

Calling Sequence CALL (4,ARG1,ARG2)

Use To test sense switches on the computer control panel or the MDP 200 mobile cabinet.

NB S.S. one is used to cause a "program break" in BASIC; this returns the user to the command mode. So only switches 2,3, and 4 are available for user options.

Arguments

ARG 1 is the sense switch which is to be tested, and may be a number or a variable name.

ARG 2 is set to 1 if the switch is set, and to 2 if the switch is reset.

Program Examples:-

```
5 . R = 0
10 FOR S = 2,4
20 CALL (4,S,R)
30 IF R = 1 THEN PRINT "SET"
40 IF R = 1 THEN GO TO 60
50 PRINT S; "RESET"
60 NEXT S
70 REM THIS WILL PRINT THE STATUS OF ALL SWITCHES
```

5. Audible Warning

Calling Sequence CALL (5)

Use To ring the bell on the ASR

Arguments

There are none.

Program Examples:-

```
10 FOR H = 1,39
20 CALL (2,H,Z)
30 IF Z 0 THEN CALL (5)
40 NEXT H
50 REM)THIS WILL RING THE BELL WHEN
   )
60 REM)A NEGATIVE VALUE IS FOUND
```

Complete Test and Demonstration Program

The following program has been written to show how each of the subroutines can be used. A printout of its execution is also included for reference.

LIST

```
1  A=0:T=0: PRINT "DEMONSTRATION PROGRAM"
2  DIM Z(39)
5  X=20
6  PRINT : PRINT : PRINT
10 PRINT "ONE SCAN OF CHANNELS 1-39"
20 PRINT "BELL INDICATES START & END OF SCAN"
25 CALL (5)
26 CALL (1,2,1,1)
30 FOR G=1,39
40 CALL (2,G,Z(G))
50 NEXT G
55 CALL (5)
60 CALL (1,3,A,T)
61 PRINT "THAT TOOK";T;"SECONDS"
62 PRINT : PRINT
63 PRINT "THE VALUES ARE....."
66 FOR Q=1,39
67 PRINT Q,Z(Q)
68 NEXT Q
80 PRINT "NOW A DELAY OF 20 SECONDS"
90 PRINT "RETURN TO COMMAND MODE DURING DELAY BY TYPING C"
100 PRINT "RETURN AND CONTINUE PROGRAM BY TYPING R"
101 PRINT
105 CALL (1,1,X,B)
106 PRINT : PRINT
110 PRINT "MDP CLOCK TIME"
120 CALL (3,A,B,C)
130 PRINT A,B
135 PRINT
140 PRINT "SCAN DATA SWW";C
145 PRINT "SENSE SW STATES ARE..."
150 FOR F=2,4
155 M=0
160 CALL (4,F,M)
170 IF M=1 THEN PRINT F;"SET": GOTO 190
180 PRINT F;"RESET"
190 NEXT F
195 PRINT
196 PRINT
200 PRINT "THAT'S YOUR LOT"
210 PRINT : PRINT : PRINT
220 STOP
```

ONE SCAN OF CHANNELS 1-39
FLL INDICATES START & END OF SCAN
THAT TOOK 2.16 SECONDS

THE VALUES ARE.....

1	-9999
2	9999
3	-9999
4	-9999
5	-9999
6	-9999
7	-9999
8	-9999
9	-401
10	-9999
11	9999
12	-9999
13	-9999
14	-9999
15	-9999
16	-9999
17	-9999
18	-9999
19	-4401
20	-9999
21	9999
22	-9999
23	9999
24	-9999
25	-9999
26	-9999
27	-401
28	-9999
29	-7101
30	-9999
31	9999
32	-9999
33	-9999
34	-9999
35	-7999
36	-9999
37	-9999
38	-3799
39	-9999

NOW A DELAY OF 20 SECONDS
RETURN TO COMMAND MODE DURING DELAY BY TYPING C
RETURN AND CONTINUE PROGRAM BY TYPING R
R

MDP CLOCK TIME
1340 3130

SCAN DATA SWW 5676
SENSE SW STATES ARE...
2 RESET
3 SET
4 RESET

THAT'S YOUR LOT

220 EXIT

APPENDIX 2

HONEYWELL 316 INSTRUCTION COMPLEMENT

Doc. No. 424003434058

Programmers
Reference Card

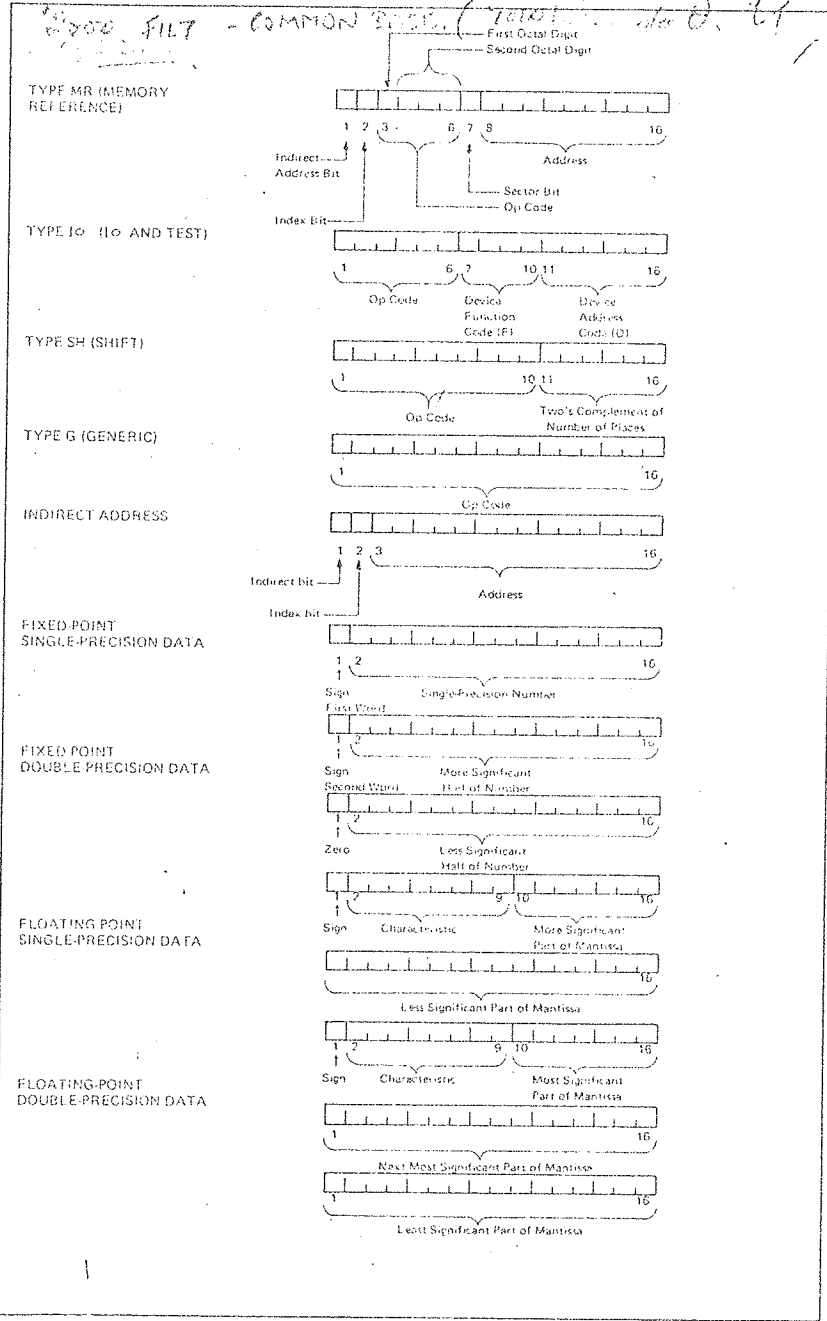
316

General Purpose Digital Computers

516

Honeywell

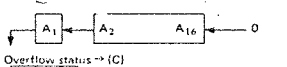
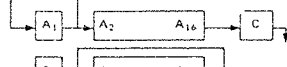
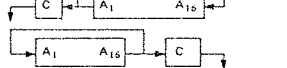

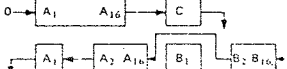
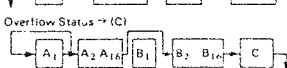
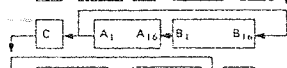
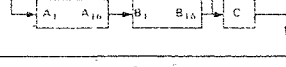


WORD FORMATS *202-8P 575 → 27600 where 001shap goes.*



KEY-IN LOADER -- (X = 1 for paper-tape reader or 4 for ASR-33/35)

OCTAL ADDRESS	OCTAL INSTRUCTION	MEANING	OCTAL ADDRESS	OCTAL INSTRUCTION	MEANING
00001	010767	STA '57	00011	002010	JMP -- 1
00002	00000X	OCF '000X	00012	141470	LGL R
00003	13100X	INA '100X	00013	10000X	INA '000X
00004	000000	JMP -- 1	00014	002013	JMP -- 1
00005	101146	SNZ	00015	110000	STA 0
00006	002004	JMP -- 3	00016	024000	IRS 0
00007	010000	STA 0	00017	100010	SZC
00010	13100X	INA 100X			

INSTRUCTION REPERTOIRE

MNEMONIC	TYPE	OP CODE	DEFINITION	DESCRIPTION	*NO. OF CYCLES
Load and Store Instructions					
CRA	G	140040	Clear A	0 → [A]	1
LDA	MR	02	Load A	[EA] → [A]	2
DLDA ^{1,4}	MR	02	Double Precision Load	[EA] → [A], [EA + 1] → [B]	3
STA	MR	04	Store A	[A] → [EA]	2
DSTA ^{1,4}	MR	04	Double Precision Store	[A] → [EA], [B] → [EA + 1]	3
LDX ²	MR	15	Load X	[EA] → [X]	3
STX ²	MR	15	Store X	[X] → [EA]	2
IAB	G	000201	Interchange A and B	[A] ↔ [B]	1
SCA ¹	G	000041	Shift Count to A	[SC] → [A _{12:16}]	1
IMA	MR	13	Interchange Memory and A	0 → [A _{1:11}] [A] → [EA]	3
INK	G	000043	Input Keys	[C] → [A ₁] [DP Model] → [A ₂] [PMI] → [A ₃] 0 → [A _{4:10}] [SC] → [A _{11:16}]	1
OTK	G	171020	Output Keys	[A ₁] → [C] [A ₂] → [DP Model] [A ₃] → [EXT Model] [A _{1:16}] → Shift Count	2
Arithmetic Instructions					
ADD	MR	06	Add	[A] + [EA] → [A] Overflow status → [C]	2
DAD ^{1,4}	MR	06	Double-Precision Add	[A, B] + [EA, EA + 1] → [A, B] Overflow status → [C]	3
SUB	MR	07	Subtract	If [(EA + 1)] ≠ [B ₁] sum invalid [A] - [EA] → [A] Overflow Status → [C]	2
DSB ^{1,4}	MR	07	Double-Precision Subtract	[A, B] - [EA, EA + 1] → [A, B] Overflow Status → [C]	3
MPY ¹	MR	16	Multiply	If [(EA + 1)] ≠ [B ₁] sum invalid [A] × [EA] → [A, B]	5%
DIV ¹	MR	17	Divide	[A, B] ÷ [EA] → [A] Remainder → [B] Improper Divide Status → [C]	11 (max)
ACA	G	141216	Add C to A	[A] + [C] → [A] Overflow status → [C]	1
AOA	G	141206	Add One to A	[A] + 1 → [A] Overflow status → [C]	1
TCA	G	140407	Two's Complement A	[A] + 1 → [A]	1%
Logical Instructions					
ANA	MR	03	AND to A	[A] ∧ [EA] → [A]	2
ERA	MR	05	Exclusive OR to A	[A] ∨ [EA] → [A]	2
CMA	G	140401	Complement A	[A] → [A]	1
CSA	G	140320	Copy Sign and Set Sign Plus	[A ₁] → [C], 0 → [A ₁]	1
SSM	G	140500	Set A Sign Minus	1 → [A ₁]	1
SSP	G	140100	Set A Sign Plus	0 → [A ₁]	1
CHS	G	140024	Complement A Sign	[A ₁] → [A ₁]	1
Shift Instructions					
ALS	SH	0415	Arithmetic Left Shift		1 + XN
ARS	SH	0405	Arithmetic Right Shift		1 + XN
ALR	SH	0416	Logical Left Rotate		1 + XN
ARR	SH	0406	Logical Right Rotate		1 + XN
LGL	SH	0414	Logical Left Shift		1 + XN
LGR	SH	0404	Logical Right Shift		1 + XN
LLS	SH	0411	Long Arithmetic Left Shift		1 + XN
LRS	SH	0401	Long Arithmetic Right Shift		1 + XN
LLR	SH	0412	Long Left Rotate		1 + XN
LRR	SH	0402	Long Right Rotate		1 + XN

MNEMONIC	TYPE	OP CODE	DEFINITION	DESCRIPTION	*No. OF CYCLES
LLL	SH	0410	Long Left Logical Shift		1 + %N
LRL	SH	0400	Long Right Logical Shift		1 + %N
NRM ¹	G	000101	Normalise	 Shift until (A ₁) ≠ (A ₁) Number of shifts → (SC)	1 + %N
Half-Word Instructions					
CAL	G	141050	Clear A, Left Half	0 → (A ₁₋₈) (A ₉₋₁₆) are unchanged	1
CAR	G	141044	Clear A, Right Half	0 → (A ₉₋₁₆) (A ₁₋₈) are unchanged	1
ICA	G	141340	Interchange Characters in A	(A ₁₋₈) ↔ (A ₉₋₁₆) A ₁ is interchanged with A ₉ , A ₂ with A ₁₀ , etc.	1
ICL	G	141140	Interchange and Clear Left Half of A	(A ₁₋₈) ↔ (A ₉₋₁₆) 0 → (A ₁₋₈)	1
ICR	G	141240	Interchange and Clear Right Half of A	(A ₉₋₁₆) ↔ (A ₁₋₈) 0 → (A ₉₋₁₆)	1
Control Instructions					
CAS	MR	11	Compare and Skip	If (A) > (EA), execute next instruction If (A) = (EA), skip next instruction If (A) < (EA), skip two instructions EA → (P)	3
JMP	MR	01	Unconditional Jump	Next inst. to be executed is at EA	1
JST	MR	10	Jump and Store Location	(P ₃₋₁₆) ↔ (EA ₃₋₁₆), (EA ₁₂) not changed (EA ₃₋₁₆) + 1 → (P ₃₋₁₆) Next inst. to be executed is at EA + 1	3
IRS	MR	12	Increment, Replace and Skip	(EA) + 1 → (EA), if original (EA) + 1 = 0, skip next inst.	3
SKP	G	100030	Unconditional Skip	Skip next instruction	1
SPL	G	100400	Skip if A Sign Plus	Skip next inst. if (A ₁) = 0	1
SMI	G	101400	Skip if A Sign Minus	Skip next inst. if (A ₁) = 1	1
SZE	G	100040	Skip if A Zero	Skip next inst. if (A) = 0	1
SNZ	G	101040	Skip if A Nonzero	Skip next inst. if (A) ≠ 0	1
SLZ	G	100100	Skip if (A ₁₆) Zero	Skip next inst. if (A ₁₆) = 0	1
SLN	G	101100	Skip if (A ₁₆) Nonzero	Skip next inst. if (A ₁₆) = 1	1
SRC	G	100001	Skip if C Reset	Skip next inst. if (C) = 0	1
SSC	G	101001	Skip if C Set	Skip next inst. if (C) = 1	1
SR1	G	100020	Skip if Sense Switch 1 is Reset	Skip next inst. if SS1 is off	1
SR2	G	100010	Skip if Sense Switch 2 is Reset	Skip next inst. if SS2 is off	1
SR3	G	100004	Skip if Sense Switch 3 is Reset	Skip next inst. if SS3 is off	1
SR4	G	100002	Skip if Sense Switch 4 is Reset	Skip next inst. if SS4 is off	1
SSR	G	100036	Skip if No Sense Switch Set	Skip next instruction if all sense switches are off	1
SS1	G	101020	Skip if Sense Switch 1 is Set	Skip next inst. if SS1 is on	1
SS2	G	101010	Skip if Sense Switch 2 is Set	Skip next inst. if SS2 is on	1
SS3	G	101004	Skip if Sense Switch 3 is Set	Skip next inst. if SS3 is on	1
SS4	G	101002	Skip if Sense Switch 4 is Set	Skip next inst. if SS4 is on	1
SSS	G	101056	Skip if Any Sense Switch Set	Skip next instruction if any sense switch is on	1
SPN ^{1,5}	G	100200	Skip on No Memory Parity Error	Skip next instruction if parity error flip-flop is reset	1
SPS ^{1,5}	G	101200	Skip on Memory Parity Error	Skip next instruction if parity error flip-flop is set	1
RMP ^{1,5}	G	000021	Reset Memory Parity Error	Reset parity error flip-flop	1
HLT	G	000000	Halt	Stop computer operation	1½
NDP	G	101000	No Operation	Next inst. to be executed is at EA + 1 Performs no operation	1
RCB	G	140200	Reset C Bit	0 → (C)	1
SCB	G	140000	Set C Bit	1 → (C)	1
ENB	G	000401	Enable Program Interrupt	Enable interrupt (PI indicator lights)	1
INH	G	001001	Inhibit Program Interrupt	Inhibit interrupt (PI indicator is extinguished)	1
SGL ¹	G	000005	Enter Single-Precision mode	Places computer in single precision mode	1
DBL ¹	G	000007	Enter Double-Precision mode	Places computer in double-precision mode	1
EXA ^{1,5}	G	000013	Enable Extended Addressing	Places computer in extend mode	1
DXA ^{1,5}	G	000011	Disable Extended Addressing	Restores computer to normal addressing	1
ERN ^{1,5}	G	001401	Enter Restricted Mode	Places computer in restricted mode	1
Input/Output Instructions					
OCP	IO	14	Output Control Pulse	(F _{D₃₋₁₆}) ↔ (A _{D₃₋₁₆}) Direct Control Pulse (F) to IO Device D	2
INA	IO	54	Input to A	If not ready, no input, execute next instruction If ready, and (F ₃) = 1, (INB) ↔ (A) and skip next inst.	2
OTA ¹	IO	74	Output from A	If ready and (F ₃) = 0, (INB) ↔ (A) and skip next inst. If not ready, no output, execute next instruction If ready, (A) ↔ (OTB), skip next instruction	2

MNEMONIC	TYPE	OP CODE	DEFINITION	DESCRIPTION	No. OF CYCLES
SMK ³	IO	74	Set Mask	(A)~(OTB)	2
SKS	IO	34	Skip if Ready Line set	Skip or execute next instruction depending on sense condition	2
					4

NOTES

- * The nominal cycle time for the H316 is 1.6µs; the nominal cycle time for the DDP-516 is 0.96µs.
- 1 Instruction used with H316 or DDP-516 options
- 2 Instructions STX and LDX have the same operation code (15). STX has an index bit of 0; LDX has an index bit of 1.
- 3 Instructions OTA and SMK have same operation code (74). SMK has device address D = '20 or '24; OTA has D = neither '20 nor '24.
- 4 CPU must be in double precision mode
- 5 This instruction is used with options available only on the DDP-516.

ABBREVIATIONS

A	A register (16 bits)	X	Index register (16 bits)
ADB	Address bus	()	Contents of a hardware register
B	B register (16 bits)	[]	Contents of memory location
C	Carry bit (C-bit)	A	Logical AND
D	Input-output device code	V	Logical OR
DP	Double-precision mode bit	v	Exclusive OR
EA	Effective address	-	Logical NOT
F	Input output function code	+	Algebraic addition
G	Generic instruction	→	Replaces
INB	Input bus	↔	Is exchanged with
IO	Input-output and test instruction	↓	Discarded
MR	Memory reference instruction (index bit, indirect address bit, and sector bit applicable)		
OTB	Output bus		
P	Program counter register (14 or 15 bits)		
PI	Program interrupt enable indicator		
PMI	Previous mode indicator (extended addressing option)		
SC	Shift count		
SH	Shift instruction		
N	Specific number of shifts to be performed		

TAP E

TRACKS

8 7 6 5 4 3 2 1

A REG BITS

9 10 11 12 13 14 15 16

SUMMARY OF DAP-16 PSEUDO-OPERATIONS

OPERATION MNEMONIC	MEANING	EFFECT
...	Op code set by program	Zeros put into op-code
ABS	Absolute mode	Subsequent instructions assembled in absolute mode
BCI	Binary coded information	ISO code characters converted into binary
BES	Block ending with symbol	Increases value of location counter by value of expression in the v.f.
BSS	Block starting with symbol	Same as BES
BSZ	Block storage of zeros	Same as BES (used for defining storage blocks that are initially cleared)
CALL	Call subroutine	Generates a JST to call referenced external subroutine
CF1	Configuration of DDP-116	
CF3	Configuration of H316	
CF4	Configuration of DDP-416	
CF5	Configuration of DDP-516	Specifies which 16-bit computer will execute the object program
COMN	Put in common storage	Assigns a location in a common data pool for symbol in location field
DAC	Define address or constant	Causes DAP to assemble a 16-bit address word
DBP	Double-precision conversion	Decimal characters converted into binary with double-precision option
DEC	Decimal-to-binary conversion	Decimal characters converted into binary
EJCT	Eject	Causes DAP to begin or resume listing on a new page
END	End of source program	Terminates assembly pass
EQU	Equals	Assigns value and mode of expression in the variable field to symbol in location field
EXD	Enter extended desectoring	Subsequent instructions assembled in extended addressing mode
FIN	Finish	Output literals
LIST	Generate listing	Causes output of source and object programs, side-by-side
LOAD	Load mode	Subsequent instructions assembled in load mode
LXD	Leave extended desectoring	Subsequent instructions assembled in normal addressing mode
MOR	More	Halt during assembly and reset line count
NLST	No listing	Inhibits output of listing
OCT	Octal-to-binary	Octal characters converted into binary
ORG	Origin	Value and mode of expression in variable field are evaluated and location counter is set accordingly
FZE	Plus Zero	Zeros put into op-code
REL	Relocatable mode	Subsequent instructions assembled in relocatable mode
SETB	Set base sector	Specify a sector other than zero as the base sector
SUBR	Entry point	Outputs external name for identification by loader
XAC	External address constant	Causes DAP to assemble 16-bit address word defining location outside the program

R. A. CHARD.

PERIPHERAL CONTROL AND SENSING CODES

CARD READER				
OCF	'0005	Read one card in Hollerith mode	SKS '0305	Skip if card reader operational
OCF	'0105	Read one card in binary mode	SKS '0405	Skip if not interrupting
OCF	'0205	Offset stack the card currently in transport	SKS '0505	Skip if no cycle check detected
SKS	'0005	Skip if ready	SKS '0605	Skip if no illegal punch detected
SKS	'0105	Skip if not busy	INA '0006	Input if ready
SKS	'0205	Skip if ETX code not detected	INA '1005	Clear A and then input if ready
			SMK '0020	Set interrupt mask (A ₁)
ASR-33/35				
OCF	'0004	Enable ASR-33/35 in input mode	INA '0204	Input binary code if ready
OCF	'0104	Enable ASR-33/35 in output mode	INA '1004	Clear A and input ISO code if ready
SKS	'0004	Skip if ready	INA '1204	Clear A and input binary code if ready
SKS	'0104	Skip if not busy	OTA '0004	Output ISO code if ready
SKS	'0204	Skip if input not X-OFF (DC ₃)	OTA '0204	Output binary code if ready
INA	'0004	Input ISO code if ready	SKS '0404	Skip if not interrupting
			SMK '0020	Set interrupt mask (A ₁)
LINE PRINTER				
OCF	'0100	Inhibit PCU. No paper advance	SKS '0300	Skip if not advancing paper
OCF	'0200	Set PCU in DMCI/DMA mode	SKS '0400	Skip if not interrupting
OCF	'0700	Reset DMCI/DMA logic. Return PCU to I/O mode	SKS '1100	Skip if not in printing mode
OCF	'1000	Space to head of form	SKS '1500	Skip if line printed and paper not advanced
OCF	'1700	Space one line vertically	SMK '0020	Set PCU interrupt mask (A ₁₄)
SKS	'0000	Skip if ready	OTA '0000	Transfer data from A-register to PCU
SKS	'0100	Skip if not busy		
SKS	'0200	Skip if no cycle error detected		
HIGH-SPEED PAPER-TAPE READER				
OCF	'0001	Start paper-tape reader	INA '1001	Clear A and input if reader ready
OCF	'0101	Stop paper-tape reader	SKS '0401	Skip if paper-tape reader not interrupting
INA	'0001	Input if paper-tape reader ready	SKS '0001	Skip if paper-tape reader is ready
			SMK '0020	Set interrupt mask (A ₉)
HIGH-SPEED PAPER-TAPE PUNCH				
OCF	'0002	Enable paper-tape punch (BRPE Punch Only)	SKS '0602	Skip if paper-tape punch is ready
OCF	'0102	Paper-tape punch power off (BRPE Punch Only)	SKS '0102	Skip if paper-tape punch is enabled
OTA	'0002	Output to paper-tape if ready	SKS '0402	Skip if paper-tape punch not interrupting
			SMK '0020	Set interrupt mask (A ₁₀)
MAGNETIC TAPE				
OCF	'001X	Read BCD 2 char/word	SKS '001X	Skip if ready
OCF	'011X	Read binary 2 char/word	SKS '011X	Skip if not busy
OCF	'021X	Read binary 3 char/word	SKS '021X	Skip if error not detected
OCF	'031X	Set up normal DMCI/DMA mode	SKS '031X	Skip if not BOV
OCF	'041X	Write BCD 2 char/word	SKS '041X	Skip if not interrupting
OCF	'051X	Write binary 2 char/word	SKS '051X	Skip if EOV not detected
OCF	'061X	Write end of file	SKS '061X	Skip if EOP not detected
OCF	'071X	Reset DMCI/DMA mode	SKS '071X	Skip if output permitted
OCF	'101X	Write binary 3 char/word	SKS '111X	Skip if M1 is operational
OCF	'111X	Space forward 1 block	SKS '121X	Skip if DMCI/DMA sub-channel is not using chan 2
OCF	'121X	Space forward 1 file	SKS '131X	Skip if DMCI/DMA sub-channel is not in auto switch mode
OCF	'131X	Set up DMCI/DMA in auto switch mode	SKS '141X	Skip if not reformatting
OCF	'141X	Reformat	INA '001X	Input from TCU if ready
OCF	'151X	Backspace 1 block	INA '101X	Clear A and input from TCU if ready
OCF	'161X	Backspace 1 file	OTA '001X	Output data to TCU
OCF	'171X	Stop write	SMK '0070	Set interrupt mask (A ₁ for TCU, A ₂ for TCU ₂)

SERIES 16 PERIPHERAL DEVICE CODES

CHAR	ISO CODE	EVEN PARITY	SIX-BIT CODE	CARD PUNCHING	CHAR	ISO CODE	EVEN PARITY	SIX-BIT CODE	CARD PUNCHING	CHAR	ISO CODE	EVEN PARITY	SIX-BIT CODE	CARD PUNCHING
space	240	240	20	blank	G	216	066	06	6	L	314	314	43	11-3
!	241	011	15	R-6	7	267	267	37	7	M	315	115	41	11-4
"	242	042	37	0-8-7	8	270	270	10	8	N	316	116	45	11-5
#	243	713	33	3-9-2	9	271	071	11	9	O	317	317	46	11-6
\$	244	044	52	11-2-3	0	272	072	15	0-5	P	320	120	47	11-7
%	245	245	75	12-5-5	1	273	273	52	11-8-2	Q	321	321	50	11-8
&	246	246	77	12-8-7	2	274	074	57	11-5-7	R	322	322	51	11-9
'	247	047	14	8-4	3	275	275	13	8-3	S	323	123	22	0-2
(250	050	34	0-9-4	4	276	276	17	6-7	T	324	324	23	0-3
)	251	251	74	12-2-4	5	277	077	26	0-8-5	U	325	125	24	0-4
*	252	252	54	11-8-4	@	300	300	76	12-8-6	V	326	126	25	0-5
+	253	053	60	12	A	301	101	61	12	W	327	327	26	0-6
,	254	254	33	0-6-3	B	302	102	62	12-2	X	330	330	27	0-7
-	255	055	40	11	C	303	303	65	12-3	Y	331	131	30	0-8
.	256	256	73	12-8-3	D	304	104	64	12-4	Z	332	132	31	0-9
/	257	257	21	0-1	E	305	305	05	12-5	[333	333	55	11-8-5
0	260	060	00	0	F	306	306	66	12-6	\	334	134	2	
1	261	261	01	1	G	307	107	67	12-7]	335	335	36	0-8-6
2	262	262	02	2	H	310	110	70	12-8	^	336	336	72	12-8-2
3	263	063	03	3	I	311	311	71	12-9	~	337	137		
4	264	264	04	4	J	312	312	41	11-1					
5	265	065	05	5	K	313	113	42	11-2					

ASR 33/35 FUNCTION CONTROLS

FUNCTION	ISO CODE	EVEN PARITY
NUL	'200	'000
SOH	'201	'201
ETX	'203	'003
ENQ	'205	'005
BEL	'207	'207
LF	'212	'012
CR	'215	'215
X ON (DC ₁)	'221	'321
TAPE (DC ₂)	'222	'022
X-OFF (DC ₃)	'223	'223
DEL	'377	'377

NOTES

- On many Teletypes, E is printed as 3.
- On Honeywell line printers with an EDP type roll, & is printed as 8, 0 as 0, 1 as 1, 1 as 1, 1 as 2, 1 as 2 and 1 as 7.
- On some printers, left arrow may be replaced by underline, and upwards arrow by bar over it.
- On paper tape, left arrow signifies "delete" and backslash means "tab"; these characters are not available on necessary on tapes or magnetic tape.
- The six-bit codes are used for both cards (Hollerith mode) and punched magnetic tape (even parity or BCD mode). On writing magnetic tapes, '00 is written as '12 in reverse; when reading in even parity, '12 is read '00. In punched cards, code '06 corresponds to card punching 11-8-6, which is interpreted as ETX (end of text).

APPENDIX 3

PREPARATION OF A DAP-16 PROGRAM

The stages involved in using the assembler to produce a working program are very similar to those described for FORTRAN. (Section 3). i.e.

- i) Prepare program - by writing it
- ii) Punch a source tape
- iii) Load assembler into computer
- iv) Use assembler to convert source to object tape.
- v) Load "loader" into computer
- vi) Use loader to load object tape
- vii) Run program

(i) Program preparation consists of the writer detailing exactly what he wants the machine to do. Each program line represents one machine code instruction, which can make very long programs which run extremely quickly.

(ii) Preparation of Source Tape. Each DAP instruction must be put in the appropriate columns of a coding sheet, similar to FORTRAN.

There are four "fields":-

- Location field - used for defining variable names or identifiers
- Operation field - defines the operation performed by this instruction
- Variable - the variable which is to be used for the operation
- Comments - to enable remarks to be put in; this can make the program much easier to understand.

Instead of spacing the fields out in full, a back slash character / may be typed which causes the field to be terminated. This enables programs to be typed very quickly without having to worry about spaces.

iii) Assembler loading. This is a self-loading tape, as the FORTRAN compiler.

iv) Assembly. The assembler may be used in two modes; one pass and two pass.

One pass assembly In this mode the source tape is read only once by the assembler and the object code tape produced immediately. The disadvantage is that references to variables which have not yet been defined have to be filled in at the end. This is done automatically, the only difference in the final tape being the physical length, because of the extra information.

Two Pass Assembly The source tape is read by the assembler twice; during the first pass all the variables are defined, and given symbolic locations. No object tape is produced until the second pass, when reference is made to the variable table. In this way all the variable's addresses are known and can be filled in.

Although two passes of the source tape are necessary, this mode of assembly is much better than single pass because a better object code tape is produced.

v) Load "loader" into computer. Once the assembler has been used and the object tape is satisfactory the loader is put into the machine ready to load the object tape.

vi) The loader is then used, and puts the object program into the store.

vii) Finally the program can be run by putting its starting address into the P register and pressing the start button.

If there are any user written subroutines (also in DAP-16) these would be loaded after the main program, but before execution.

APPENDIX 4

COMMUNICATION BETWEEN DAP-16 AND FORTRAN ON THE HONEYWELL 316

Introduction

Communication between DAP and FORTRAN can be achieved by loading together subprograms in DAP object and FORTRAN object format.

In this context a subprogram may be a DAP or FORTRAN main program, DAP subroutine, FORTRAN subroutine or FORTRAN function.

Transfer of information may be achieved in either of two ways.

These are:-

- a) by reference to items in COMMON
- b) by argument transfer.

The first of these requires a knowledge of the way DAP and FORTRAN allocate COMMON storage and the second requires a knowledge of how FORTRAN compiles CALL statements. These will be dealt with separately.

Common Storage Allocation

Common storage declared in FORTRAN programmes is allocated in the following way:-

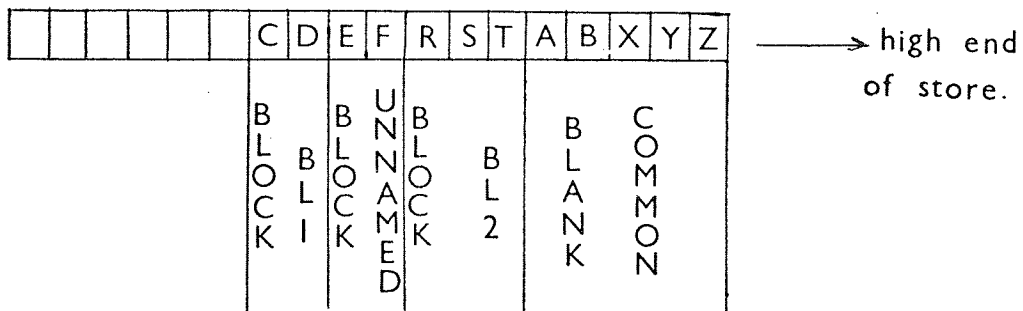
- a) Blank common storage is allocated first in such a way that the variables in blank common are assigned storage, in the reverse order to which they are declared, starting at the high end of store, immediately below the initial common base, and working downwards in the store.

b) Variables in both named block and unnamed blocks of common storage are then assigned storage in the reverse order to which they are declared starting immediately below any variables in blank common (or, if there is no blank common, below the initial common base) and, again, working downwards in the store. Thus the following statements:-

```
COMMON A,B/BL1/C,D/ / E,F
```

```
COMMON X,Y,Z/BL2/R,S,T
```

will result in the following allocations:-



DAP Common

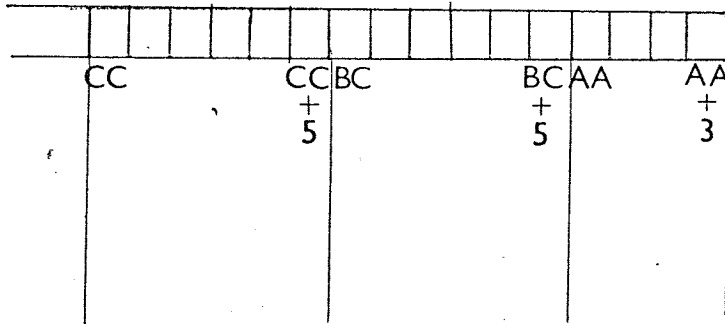
Common storage declared in DAP programmes is illustrated by the following example:-

```

AA  COMN  4
BC  COMN  6
CC  COMN  6
DD  COMN  8

```

The first statement allocates identifier 'AA' 4 words below the initial common base, and the second allocates 'BC' 6 words below 'AA' (i.e. 10 words below the initial common base).



Thus if the DAP program with the COMN statements above is loaded with the FORTRAN example, and the common base is the same, there will be the following association of identifiers:-

AA	-----	Y
AA+2	-----	Z
BC	-----	A
BC+2	-----	B
BC+4	-----	X
CC	-----	R
CC+2	-----	S
CC+4	-----	T
DD	-----	C
DD+2	-----	D
DD+4	-----	E
DD+6	-----	F

Common Base In the case of FORTRAN compiled code the initial COMMON base is a user option which can be set by the loader, in default this is set to '100 below the top of the store (in the case of the 12K machine '27700). To change the COMMON base to any other value, the location FIL7 ('2000 above the bottom of the loader) is changed to the desired value.

In the case of DAP programmes using the DAP MOD-2 assembler the COMMON base is set by the pseudo operation SETC (set common).

Thus SETC '27700.

This allows COMMON to be located anywhere in the store, thus saving the change of FIL7 referred to above.

ARGUMENT TRANSFER

In this case the way in which FORTRAN compiles the CALL statement must be known. The code generated is:-

```
JST Routine ) when there is only
DAC ARG1   ) one argument.
```

```
JST Routine )
DAC ARG1   ) where there are
----- ) N arguments (N > 1)
DAC ARGN   )
OCT 0
```

Knowing this it is possible to write a DAP subprogram which will make a CALL on a FORTRAN subprogram by following this format. The return will be made to the location following the OCT 0.

When writing a DAP subprogram that is CALLED by a FORTRAN program it is possible to use the standard Honeywell FORTRAN argument transfer library subroutine F\$AT (doc.no.180071000) to effect the transfer. This has the advantage that all levels of indirect addressing are removed from the argument on transfer, which can increase the operating speed.

An example of such a call would be:-

```
DAC **      Link word of subprogram
             (used to store return address)

CALL F$AT

OCT 3      Number of arguments

OCT 0      ) space for the addresses
OCT 0      ) of the arguments to be
OCT 0      ) stored in
```

The call to JSAT must be immediately following the link word of the called subprogram.

JSAT does not transfer the argument itself, but the storage location at which it is located.

For example

if the following piece of program was in sector '05

		<u>Storage</u>
'5100	CALL DAP	
'5101	DAC A	'5 200 A
'5102	DAC B	'5 201 B
'5103	DAC C	'5 410 C
'5104	DAC D	'5274 D
'5105	LDA A	
'5106	ADD B	
'5107	STA SUM	
'5110	LDA C	
'5111	SUB D	
'5112	STA DIFF	
'5113	IAB	

Upon execution of the JSAT, control would be transferred to '26600, where the following might be stored:-

```

'26 600      DAC **
'26 601      CALL - F$AT
'26 602      OCT 4
'26 603      OCT 0
'26 604      OCT 0
'26 605      OCT 0
'26 606      OCT 0
'26 607      LDA * *-4
'26 600      STA AL
'26 611      LDA * *-3
'26 612      STA BL
'26 613      JMP * '26 600

```

upon execution of F\$AT, the locations '26 602 to '26 606 would be given the values '5200, '5201, '5410, '5274, i.e. the places where the arguments are stored. Because there are arguments stored after the CALL, the return cannot be made to the usual place i.e. immediately after the CALL ('5101). F\$AT increments the return address by the number of arguments, in this case 4, so return will be made to the correct place '5 105.

The subroutine may use the arguments by indirect addressing or can transfer them into itself. In the example at 126-607 the first argument is fetched by indirectly loading it and then storing in AL. Similarly B is fetched and stored locally. New values of arguments can be passed back by indirectly storing the new values through the addresses transferred.

APPENDIX 5

BASOON PROGRAMS

1. DISTRIBUTION ANALYSIS
2. SAMPLING TEST
3. ENSEMBLE TEST

DISTRIBUTION ANALYSIS

Page 1 of 5

```
10 PRINT "CHANNEL?": INPUT C
15 PRINT "TIME BETWEEN SAMPLES?": INPUT T
20 PRINT "SCANS/SAMPLE?": INPUT S
22 IF S>25 GOTO 20
25 PRINT "SAMPLES?": INPUT S5
35 IF S5>200 GOTO 25
36 GOTO 61
40 PRINT : PRINT
45 FOR I=1,10: PRINT : NEXT I
50 PRINT "ANALYSIS OF DATA FROM CHANNEL":C
55 PRINT S:"SCANS TO EACH SAMPLE."
60 PRINT "SAMPLES TAKEN EVERY ";T;"SECONDS"
61 DIM K(25),A(2,200),J(2,25),T(200)
65 X=0:X1=0:X2=0:X3=0
70 FOR S6=1,S5
71 FOR I=1,25:J(1,I)=0:J(2,I)=0: NEXT I
75 REM TIME START GOES HERE.
76 CALL (1,2,X,X1)
80 FOR S7=1,S
85 CALL (2,C,K(S7))
90 IF K(S7)=0 GOTO 85
95 NEXT S7
100 T1=0:M2=0
105 FOR N1=1,S
110 T1=T1+K(N1)
115 NEXT N1
120 M1=T1/S
121 N=S
125 FOR N1=1,S
130 M2=M2+(K(N1)-M1)^2
135 NEXT N1
140 M3=SQR((M2/(N-1)))
145 REM M1 MEAN, M3 ST. DEV.
```

DISTRIBUTION ANALYSIS

Page 2 of 5

```
146 A(1,S6)=INT(M1):A(2,S6)=M3
150 CALL (4,4,X): IF X=1 GOTO 282
155 CALL (1,3,X,X1): REM STOP TIMING
160 REM ORDER INPUT ARRAY.
165 PRINT : PRINT "SAMPLE NO.":S6
170 FOR I=1,S
175 E=K(I)
180 JI=0
185 FOR J=1,S
190 IF K(J)<E GOTO 226
195 NEXT J
200 IF JI=0 GOTO 220
205 J2=K(JI)
210 K(JI)=K(I)
215 K(I)=J2
220 NEXT I
222 REM FINISHES HERE.
224 GOTO 232
226 E=K(J)
228 JI=J
230 GOTO 195
232 I1=0
234 FOR J=1,S
236 U(1,J)=K(I1+1)
238 FOR I=1,S
240 IF U(1,J)=K(I) GOTO 244
242 GOTO 248
244 U(2,J)=U(2,J)+1
246 I2=I
248 O=0
250 NEXT I
252 I1=I2
254 IF I1=S GOTO 258
```

DISTRIBUTION ANALYSIS

Page 3 of 5

```
256 NEXT J
258 PRINT "RANGE";K(1);"T";K(S)
260 PRINT "MEAN";M1;"ST. DEV.";M3: PRINT
262 PRINT "DISTRIBUTION": PRINT "VALUE NO. OF POINTS"
264 PRINT
266 FOR I=1,S
268 IF U(1,I)=0 GOTO 273
270 PRINT U(1,I),U(2,I)
273 NEXT I
275 GOTO 283
280 FOR I=1,8: PRINT : NEXT I
281 GOTO 283
282 CALL (1,3,X,X1)
283 D1=T-X1: IF D1<=0 GOTO 287
284 IF S6=(S5-1) GOTO 287
285 CALL (1,1,D1,X)
287 NEXT S6
290 FOR I=1,10: PRINT : NEXT I
300 PRINT "ANALYSIS OF SAMPLE MEANS."
305 PRINT
310 R3=0
315 FOR J=1,S5
320 R3=R3+A(1,J)
325 NEXT J
330 M1=R3/S5: REM MEAN
335 M2=0
340 FOR J=1,S5
345 M2=M2+(A(1,J)-M1)^2
350 NEXT J
355 M3=SQR((M2/(N-1))): REM ST. DEV.
360 B4=A(1,1):D4=B4:B6=A(2,1):D6=B6
365 FOR J=1,(S5-1)
370 IF A(1,J+1)>B4 THEN B4=A(1,J+1)
```

DISTRIBUTION ANALYSIS

Page 4 of 5

```
375 IF A(1,J)<D4 THEN D4=A(1,J)
380 IF A(2,J+1)>B6 THEN B6=A(2,J+1)
385 IF A(2,J)<D6 THEN D6=A(2,J)
390 NEXT J
400 PRINT "LARGEST SAMPLE MEAN";B4
405 PRINT "SMALLEST";D4
410 PRINT "LARGEST ST. DEV.";B6
415 PRINT "SMALLEST";D6
420 PRINT
425 PRINT "THE MEAN OF THE SAMPLES IS";M1
430 PRINT " AND THEIR ST.DEV. ";M3
435 FOR J=1,10: PRINT : NEXT J
440 PRINT "DISTRIBUTION OF SAMPLES"
445 FOR J=1,100:T(J)=0: NEXT J
450 B1=B4-D4+1
452 S1=D4
455 FOR I=1,B1
460 FOR J=1,S5
465 IF A(1,J)=S1 THEN T(I)=T(I)+1
470 NEXT J
475 S1=S1+1
480 NEXT I
485 PRINT "VALUE NO. OF POINTS"
490 PRINT
500 S1=D4
505 FOR I=1,B1
510 IF T(I)=0 THEN GOTO 520
515 PRINT S1,T(I)
520 S1=S1+1
525 NEXT I
530 FOR I=1,10: PRINT : NEXT I
535 S1=D4
540 FOR I=1,B1
```

DISTRIBUTION ANALYSIS

Page 5 of 5

```
545 PRINT S1;
550 IF T(I)=0 GOTO 570
551 IF T(I)>50 THEN PRINT "TOO MANY FOR ONE LINE!"
552 IF T(I)>50 GOTO 570
555 FOR J=1,T(I)
560 PRINT "*";
565 NEXT J
570 PRINT
575 S1=S1+1
580 NEXT I
585 FOR I=1,10: PRINT : NEXT I
590 STOP
```

SAMPLING TEST

Page 1 of 4

```
1 DIM D(50),E(39),Q(2,4)
5 S=12
6 S1=12
7 S2=12
8 S3=12
9 S4=35
15 PRINT TAB(S);"A. CHARD SAMPLING TEST."
20 PRINT
25 PRINT TAB(S);"PURPOSE: TO TEST THE ACCURACY OF DATA BEING"
30 PRINT TAB(S+12);"SCANNED FROM A CHANNEL CONSECUTIVELY"
35 PRINT TAB(S+12);"AND IN THE MIDST OF OTHER CHANNELS."
40 PRINT : PRINT : PRINT
45 A=0:B=0:C=0
50 CALL (3,A,B,C)
55 PRINT TAB(S);"TEST BEGINS. STARTING TIME";A,B
60 PRINT TAB(S);". . . . . ."
65 PRINT : PRINT : PRINT
68 PRINT
70 REM SAMPLING PROPER BEGINS HERE.
75 FOR NI=12,29
76 PRINT TAB(S);"A. CHARD SAMPLING TEST. PAGE ";(NI-11)
77 PRINT : PRINT : PRINT : PRINT
80 PRINT TAB(S1);"CHANNEL";NI;" 50 CONSECUTIVE SCANS."
85 PRINT
86 A=0:B=0
87 CALL (1,2,A,B)
90 FOR C=1,50
95 CALL (2,NI,D(C))
100 NEXT C
101 Z9=0
102 CALL (1,3,Z9,Z9)
103 Z8=INT(50/Z9)
105 R=1
```


SAMPLING TEST

Page 2 of 4

```

110 GOSUB 500
115 PRINT TAB(S1);"CHANNEL";N1;" 50 INTERRUPTED SCANS."
116 PRINT
120 FOR B=1,50
121 C5=N1-1:C6=N1+1
125 FOR C=C5,C6
130 CALL (2,C,E(C))
135 NEXT C
140 D(B)=E(N1)
145 NEXT B
150 R=2
155 GOSUB 500
160 PRINT : PRINT
161 PRINT : PRINT
165 PRINT TAB(S3);" DATA ANALYSIS . CHANNEL ";N1: PRINT
170 PRINT TAB(30);"CONSECUTIVE INTERRUPTED."
171 PRINT TAB(30);" SCANS"
175 PRINT TAB(S);"SAMPLE RANGE ";Q(1,1);"TO ";Q(1,2);" ";Q(2,1);"T
";Q(2,2)
180 PRINT
181 S7=30:S8=54
185 PRINT TAB(S);"MEAN";TAB(S7);Q(1,3);TAB(S8);Q(2,3)
190 PRINT
195 PRINT TAB(S);"ST. DEVN.";TAB(S7);Q(1,4);TAB(S8);Q(2,4)
196 PRINT
200 PRINT
205 PRINT TAB(S1);"DIFFERENCE IN MEAN VALJES ";
210 H=(100*ABS(Q(2,3)-Q(1,3)))/(Q(1,3))
211 H=(INT(100*H))/100:PRINT H;"%":PRINT
212 PRINT TAB(S);"SCANNING RATE ";Z8;"CHANNELS / SECOND."
220 FOR A=1,30:PRINT : NEXT A
225 NEXT N1
230 STOP

```

SAMPLING TEST

Page 3 of 4

```
500 PRINT
505 REM THIS SUBR. DOES THE MEAN & ST. DEVN. CALCNS.
510 FOR A=0,45 STEP 5
515 FOR B=1,4
520 PRINT TAB(S2);D(A+B);
525 NEXT B
526 PRINT D(A+5)
530 NEXT A
535 PRINT : PRINT
540 REM RANGE DETERMINATION.
545 REM DESIGNATE M SMALLEST, Z LARGEST
550 M=D(1):Z=M
555 FOR J=1,49
560 IF D(J+1)>Z THEN Z=D(J+1)
565 IF D(J)<M THEN M=D(J)
570 NEXT J
575 FOR N=1,50
580 T=0:M1=0
585 FOR N=1,50
590 T=T+D(N)
595 NEXT N
600 M1=T/50
605 REM M1=MEAN
610 REM NOW ST. DEVN.
615 M2=0
620 FOR N=1,50
625 M2=M2+(D(N)-M1)*2
630 NEXT N
635 M2=SQR(M2/49)
640 REM M1 MEXN, M2 ST. DEVN.
645 REM DATA STORAGE.
650 Q(R,1)=M
655 Q(R,2)=Z
```

SAMPLING TEST

Page 4 of 4

```
660 Q(R,3)=M1
665 Q(R,4)=M2
670 REM NOW GO BACK TO MAIN PROGRAM.
675 RETURN
5000 Q(1,1)=1234
5001 Q(1,2)=4321
5002 Q(2,1)=9876
5003 Q(2,2)=6543
5005 S=12
5010 RETURN
```

ENSEMBLE TEST

Page 1 of 5

```
2  U9=1
10 REM
15 S=12
20 DIM T(39,20),R(39,21)
30 A=0:B=0
60 CALL (1,2,A,B)
65 FOR C=12,29
70 CALL (2,C,F)
75 NEXT C
80 CALL (1,3,A,A)
85 H=ABS(INT(18/A+.5))
90 REM      H CHANNELS PER SECOND.
95 FOR N=12,29
100 REM      START OF MAIN LOOP.
105 M=0:P=0
110 FOR C=1,50
115 CALL (2,N,P)
120 M=M+P
125 NEXT C
130 M1=M/50
135 REM      M1=TRUE MEAN.
140 R(N,1)=ABS(M1)
141 NEXT N
145 REM
146 FOR N=12,29
147 N8=0
150 FOR N2=1,20
160 CALL (2,N,N9)
165 N8=N8+N9
200 R(N,N2+1)=ABS(N8/N2)
205 R(N,N2+1)=(INT(R(N,N2+1)*100))/100
310 NEXT N2
315 REM      N2 ENSEMBLE.
```

ENSEMBLE TEST

Page 2 of 5

```
320 NEXT N
325 REM      N CHANNEL
330 REM      T ARRAY HOLDS DATA BEFORE CALCN.
335 REM      R ARRAY = MEANS AFTER CALCN.
340 PRINT TAB(5);"A. CHARD.      ENSEMBLE TEST."
345 PRINT TAB(5);"SCANNING RATE";H;"CHANNELS/SEC OND."
350 PRINT : PRINT : PRINT
390 J=10
395 U1=10
400 FOR K=2,18 STEP 4
402 PRINT TAB(40);"RESULTS      PAGE";U9:U9=J9+1
405 GO SUB 1000
410 FOR I=12,29
415 U1=10
420 PRINT TAB(U);I;TAB(U+U1);R(I,1);
425 I2=K+3
430 FOR J=K,I2
435 U1=J1+10
440 PRINT TAB(U+U1);R(I,J);
445 NEXT J
450 PRINT : PRINT
455 NEXT I
456 FOR T=1,30: PRINT : NEXT T
460 NEXT K
480 X1=10/100
481 X2=8/100
482 X3=7/100
483 X4=6/100
484 X5=5/100
485 X6=4/100
486 X7=3/100
487 X8=2/100
488 X9=1/100
```

ENSEMBLE TEST

Page 3 of 5

```
498 GOSUB 3000
499 FOR V=1,30: PRINT : NEXT V
500 FOR I5=2,12 STEP 10
505 I9=I5-1:I8=I9+9
510 I2=I5:I3=I5+9
515 GOSUB 2000
520 FOR I=12,29
525 PRINT TAB(10);I;TAB(25);
530 FOR J=12,I3
550 Y=ABS(R(I,J))-R(I,1)
555 M6=ABS(R(I,1))
570 IF Y>X1*M5 THEN PRINT " ";
572 IF Y<=X1*M6 THEN IF Y>X2*M6 THEN PRINT " A ";
574 IF Y<=X2*M6 THEN IF Y>X3*M6 THEN PRINT " B ";
576 IF Y<=X3*M6 THEN IF Y>X4*M6 THEN PRINT " C ";
578 IF Y<=X4*M6 THEN IF Y>X5*M6 THEN PRINT " D ";
580 IF Y<=X5*M6 THEN IF Y>X6*M6 THEN PRINT " E ";
582 IF Y<=X6*M6 THEN IF Y>X7*M6 THEN PRINT " F ";
584 IF Y<=X7*M6 THEN IF Y>X8*M6 THEN PRINT " G ";
586 IF Y<=X8*M6 THEN IF Y>X9*M6 THEN PRINT " H ";
588 IF Y<=X9*M6 THEN PRINT " * ";
590 NEXT J
592 PRINT : PRINT
593 NEXT I
594 FOR V=1,30: PRINT : NEXT V
595 NEXT I5
600 STOP
1000 PRINT TAB(S);"CHANNEL TRUE SAMPLE MEAN FOR GIVEN NO. OF SCANS."
1005 PRINT TAB(S);"NUMBER MEAN.";
```

ENSEMBLE TEST

Page 4 of 5

```
1010 U4=5
1015 U3=25
1020 FOR L=0,3
1025 PRINT TAB(U3+U4);(L+K-1);
1030 U4=J4+10
1035 NEXT L
1040 PRINT
1041 PRINT
1045 RETURN
2000 PRINT TAB(10);"ACCURACY CODES."
                                     RESULTS PAGE";U9:U9=U9+1
2010 PRINT
2015 PRINT TAB(10);"CHANNEL
2020 PRINT TAB(10);"NUMBER
2025 PRINT : PRINT : PRINT
2100 RETURN
3000 PRINT TAB(15);"ACCURACY CODES.";TAB(40);"RESULTS PAGE";U9:U9=U9+1
3010 PRINT TAB(15);"INDICATING DEVIATION OF SAMPLE MEAN"
3020 PRINT TAB(15);"FROM TRUE MEAN."
3030 PRINT
3040 PRINT TAB(15);"BLANK >";TAB(22);X1*100;"%
3044 PRINT
3050 PRINT TAB(15);"A    <";TAB(22);X1*100;TAB(25);"BUT  >";TAB(34);X2*1
0;"%
3055 PRINT
3060 PRINT TAB(15);"B    <";TAB(22);X2*100;TAB(25);"BUT  >";TAB(34);X3*1
0;"%
3065 PRINT
3070 PRINT TAB(15);"C    <";TAB(22);X3*100;TAB(25);"BUT  >";TAB(34);X4*1
0;"%
3075 PRINT
3080 PRINT TAB(15);"D    <";TAB(22);X4*100;TAB(25);"BUT  >";TAB(34);X5*
00;"%
                                     1
```

ENSEMBLE TEST

Page 5 of 5

```
3085 PRINT
3090 PRINT TAB(15);"E" <" ;TAB(22);X5*100;TAB(25);"BUT >" ;TAB(34);X6*1
0;"%"
3095 PRINT
3100 PRINT TAB(15);"F" <" ;TAB(22);X6*100;TAB(25);"BUT >" ;TAB(34);X7*
00;"%"
3105 PRINT
3110 PRINT TAB(15);"G" <" ;TAB(22);X7*100;TAB(25);"BUT >" ;TAB(34);X8*
00;"%"
3113 PRINT
3115 PRINT TAB(15);"H" <" ;TAB(22);X8*100;TAB(25);"BUT >" ;TAB(34);X9*1
0;"%"
3117 PRINT
3120 PRINT TAB(15);"*" <=" ;TAB(22);X9*100;"%"
3150 PRINT : PRINT
3160 PRINT TAB(15);"SCANNING RATE";H;"CHANNELS / SECOND."
3190 FOR V=1,40: PRINT : NEXT V
3200 PRINT : PRINT : RETURN
7000 A=0
7010 CALL (1,2,A,A)
7020 FOR H=1,29
7030 CALL (2,H,I)
7040 NEXT H
7050 CALL (1,3,A,A)
7060 PRINT "TIME";A;"RATE";29/A;"INVERSE RATE";A/29
7070 STOP
```


* BASIC INIT. NOIS.

Page 1

0001

* BASIC INIT. NOIS.

0002

* BASIC INIT. NOIS. A NEW REFIN

0003

* PRINTOUT * SET THE HIGH ALIGNED

APPENDIX 6

ASSEMBLY LISTING OF MODIFICATIONS TO THE BASIC
COMPILER TO REVISE INITIALISATION.

0001			* BASIC INIT. MODS.
0002			* THESE MODS GIVE A NEW HEADING
0003			* PRINTOUT & SET THE HIGH ADDRESS
0004			* FOR THE USER.
0005			*
0006			ORG '166
0007	00166	141301	BCI 13, BASOON ON-LINE FROM BASIC
	00167	151717	
	00170	147716	
	00171	120317	
	00172	147255	
	00173	146311	
	00174	147305	
	00175	120306	
	00176	151317	
	00177	146640	
	00200	141301	
	00201	151711	
	00202	141640	
0008	00203	000000	OCT 0
0009			*
0010			ORG '305
0011	00305	152310	BCI -20, THE HIGH OCTAL ADDRESS IS SET
	00306	142640	
	00307	144311	
	00310	143710	
	00311	120317	
	00312	141724	
	00313	140714	

00314 120301

00315 142304

00316 151305

00317 151723

00320 120311

00321 151640

00322 151705

00323 152240

00324 152317

00325 120262

00326 132267

00327 133667

00330 120240

1 0012 00331 000000

OCT 0

1 0013

*

0014

* NOW FOR THE PROGRAM BITS.

0015

*

0016

ORG '7242

0017 07242 101000

NOF

0018 07243 101000

NOF

0019

*

0020

*

0021

ORG '7245

- 1. Char. I.C. IBM, Model. 315, 66 (1970)
- 2. Franklin H.C.A. Computer. B. 15, 440, (1970)
- 3. Macintosh Q.W. Computer 1, 45, (1970)

* BASIC INIT. MODS.

0022 07245 0 01 07301 JMP **'34

0023 *

0024 ORG '7367

0025 07367 024777 OCT 24777

0026 *

0027 *THAT'S YOUR LOT

0028 *

0029 *RAC 13.7.72

0030 *

0031 END

NO ERRORS IN ABOVE ASSEMBLY.

DAP-16 REV. E

9. BIBLIOGRAPHY

1. Chua L.O. IEEE, Educat. E15, 66 (1972)
2. Hankins H.C.A. Computer. B. 16, 440, (1972)
3. Rosentha C.W. Computer 5, 48, (1972)
4. Sternick E.S. Physics in medicine. 17, 679, (1972)
5. Feldmann R.J. J. Chem. Doc. 11, 234, (1972)
6. Meyer E.F. Nature 232, 255, (1971)
7. Heller S.R. J. Chem. Educat. 49, 291, (1972)
8. Biemann Et. Al. Anal Chem. 43, 681, (1971)
9. Heller S.R. Abs. Paps. ACS 20, (1972)
10. Idem J. Chem. Doc. 11, 248, (1971)
11. Neilsen W.B. J. Chem. Doc. 48, 414, (1971)
12. Peddie J.G. Research/development 22, 26, (1971)
13. Mazda F.F. Control & Instr 4, 27, (1972)
14. Ammosov V.V. Instruments & Exptal. Techniques 70, 1598, (1970)
15. Kompass E.J. Control Eng. 19, 52, (1972)
16. Burke M.F. Research/Development 23, 32, (1972)
17. Lord D. & Macleod G.R. J. Sci. Instr. 2, 1, (1969)
18. McCullough R.D. J. Gas Chrom. 5, 635, (1967)
19. Bowen H.C. J. Sci. Instr. 44, 343, (1967)
20. Bowen H.C. & Fish G.B. On-line Computer Methods Relevant to Chem. Eng. I Chem. E./ Brit. Comp. Soc. Sept 1971 Nottingham.
21. Landowne R.A. Analytical Chem. 44, 1961, (1972)
22. Desiderio D.M. Idib. 40, 2090, (1968)
23. Bliselius P.A. Behavioral Research & Meas 3, 37, (1971)
24. Reilly C.N. Abs. Paps. A.C.S. 28, (1971)
25. Payne J.P. Bio. Med. Eng. 6, 554, (1971)
26. Jewell B.R. J. Phys. Lon. 224, 117, (1972)
27. Olsson T. Physics in Medicine 17, 731, (1972)
28. Payne S.G. Private Communication 1972