FERRANTI
PEGASUS COMPUTER

The Pegasus Autocode

COMPUTER DEPARTMENT

Office and Works: WEST GORTON, MANCHESTER 12
London Computer Centre: 21 PORTLAND PLACE, LONDON W.1

LIST CS 217A

APRIL 1959

6/-

FERRANTI LTD

FERRANTI PEGASUS COMPUTER

THE PEGASUS AUTOCODE

CONTENTS

FERRANTI LTD

FERRANTI PEGASUS COMPUTER

THE PEGASUS AUTOCODE

0. INTRODUCTION

An Autocode is a simple method of preparing work for an electronic digital computer. The Autocode for the Pegasus computer can be learnt in a couple of days by someone previously knowing nothing about computers, and in a few hours by anyone with previous knowledge of computers. This makes it possible for anyone to put his own problems on a computer, without spending too much time learning how to use it.

When the work has been prepared, it may be run on any Pegasus computer, at computing centres, universities or industrial establishments. It is only necessary to have access to the machine for periods of minutes, or a few hours at most.

The ease with which problems can be prepared using the Autocode can only be gained by sacrificing some of the speed and flexibility of the computer. Often however, this is a very minor disadvantage and there is a wide range of problems for which the Autocode provides a great saving of time and money.

It has already been established that the Autocode will be useful in the following ways, many of which are of value to experienced users of Pegasus as well as to beginners:-

1. Technicians may easily run on the computer their work which is at present done by hand or with desk calculating machines. Not only are errors reduced and accuracy increased, but it becomes easier to see the interrelation between successive stages in design work.

2. Ad hoc problems which have to be solved only a few times, even though of considerable magnitude, may be prepared for the computer with much less effort.

3. It is a good way for an individual to begin to use a computer. The basic concepts of the Autocode and the experience gained in using it will be invaluable if he should go on to learn the normal technique of using a computer. And nothing in the Autocode has to be "unlearnt" when going on to Pegasus itself.

4. The Autocode is of particular value when an organisation is beginning to make use of a computer. Experience shows that the applications initially envisaged for a computer are often found to be inappropriate, whilst those which ultimately pay off most handsomely were not foreseen to begin with, but are only recognised as the staff of the organisation become accustomed to what a computer can do. The Autocode is invaluable for tackling all the early attempts.

5. When a new type of work is being put on a computer many successive attempts may be required. The early attempts will probably be over-simplified, or fail to meet some of the practical requirements. All this exploratory work can well be done in Autocode, and the final version rewritten to exploit the full facilities and performance of Pegasus itself.

6. In the majority of applications of a computer, the major part of the time is taken up in carrying out repetitively - perhaps tens of thousands of times - a basic group of operations (the instructions in the 'inner loop' of the programme). Since in the middle of a calculation it is possible to come out of the Autocode, use the computer in the normal manner, and then jump back into the Autocode, it is practical

for a fully trained user to write this basic group of operations in the machine code, with all the rest in Autocode. He thereby gains the advantages of simplicity in preparing the bulk of the work, together with the full speed of the computer where it counts most.

7. The Autocode has been found to be very valuable for testing the 'flow-diagram' of complicated problems, particularly those arising in clerical, commercial and data-processing applications. It is necessary to set down all the main steps in such a problem, showing the flow of the process from each step to others; often the inter-relation is very complex. Having set down the major steps, each of these has to be broken down into minor steps, to ensure that the problem has been adequately and precisely analysed. It is found that these minor steps can then be easily and quickly expressed in Autocode form, allowing the problem to be tried out on the computer. It is valuable thus to be able to test out the validity of one's plan, to check that it copes with every eventuality, before embarking on the major task of preparing it in detail for the computer or data-processing system.

8. It has been found that the Autocode is of great help for 'communication' between individuals tackling work for a computer. This is of especial value in data-processing - the applications are more difficult, they cannot be expressed with the precision of mathematics, and many of those responsible for office procedures are (at present) not over-well acquainted with computer methods. The person analysing such work in Autocode is able to explain what he is doing in a form which can be readily understood by others, thereby building up confidence and ensuring that he has not misunderstood or overlooked any important points.

This document provides a complete description of the Pegasus Autocode. It is written in three parts. The first part gives the basic concepts of using a computer and describes the main features of the Autocode; the second part gives examples of problems solved with the Autocode; and the third comprises a complete specification of the Autocode.

In the first and second parts the items printed in colour may be omitted at a first reading. They refer to points to be explained later, or to points of lesser importance, and include references to the third part of this document. Sections 1.0 and 1.1 may be omitted by those with previous experience of computers.

PART 1

1.0 THE PROGRAMME*

A digital computer is a general-purpose machine. It is prepared for any particular job by feeding into it a 'programme', or list of instructions. This programme is stored within the 'memory' of the computer. It is important to realise that it is only when a computer has got a programme within it, that it becomes a usable machine.

Because it is the programme which makes a computer to a particular job, it follows that it is only necessary to feed in a different programme to prepare the computer for a different type of work; there is no need to make any changes to the computer itself. This is one reason why it is possible for so many people to do such a variety of work on one machine. No matter what the machine was doing previously, it is immediately set up to do your particular job when you feed in your programme.

When a programme is fed into a computer, the **whole** programme goes in and is stored before the computer begins to obey the instructions of the programme. The instructions are **not**, repeat **not**, obeyed as they are read in. This is a very important idea, even though it is quite simple. The reader is therefore urged to read this paragraph again.

---

* This Section may be omitted by those with previous experience of computers.

Each programme is designed to do a particular **type** of work. When the programme has been stored in the computer, the instructions are then obeyed one by one, and one of the first instructions will be to read and store the **data for the particular job being done.** Thus the same programme may be used several times with different sets of data, without reading in the programme each time.

The computer, having stored the data it requires, then continues to work through the instructions of the programme, taking items of stored information, producing and storing intermediate results and finally producing the answers. These are then punched on to paper tape for subsequent printing on a teleprinter page, attention being paid to the layout in columns and the insertion of headings where desirable.

1.1 TACKLING THE PROBLEM*

It is important to realise that a computer is a pure automaton. It has no intelligence, no initiative, and no capacity for dealing with unforeseen circumstances. This means that the programme must be written in such a way that every possibility is catered for, and the programmer must foresee any exceptional cases that could arise.

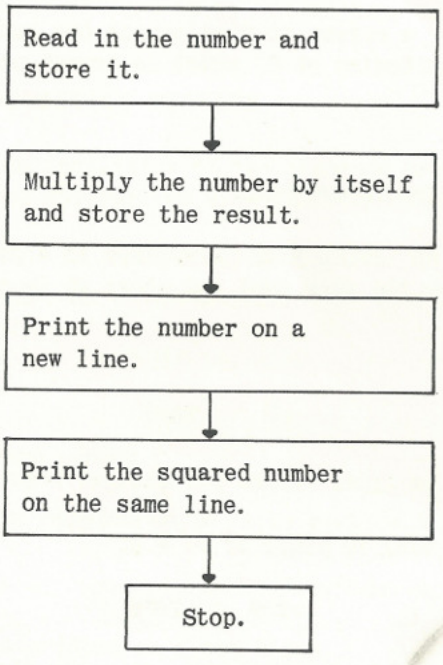
When preparing a problem for a computer the procedure is as follows:-

(1) The problem must first be precisely defined. In mathematical problems this is probably already done in the mathematical statement of the problem, but in other problems, particularly those involving human judgement, this first stage may in fact be a considerable part of the work.

(2) The steps of the process are written down in the form of a **flow diagram**:-

Example: Read in a number and print out the number followed by its square.

The flow diagram is written:



(3) The flow-diagram has then to be converted to a form that the computer can understand. This means splitting it up into simple instructions, each of which is written in a coded form intelligible to the computer. This coded form is described in the later parts of this document.

---

* This Section may be omitted by those with previous experience of computers.

(4) When the instructions of the programme have been written, they have to be put in a form which can be 'read' by the computer. This is done with a teleprinter, a machine similar to a typewriter, which also produces a punched paper tape. The typed sheet can be proof-read to detect errors, and there are facilities for correcting any errors.

Each time a number, letter, or symbol is printed a corresponding combination of up to 5 holes is punched across the tape, and these holes can be sensed by the photoelectric tape-readers on the computer. The information on this programme tape is read in by the computer, automatically converted to a form suitable to it, and stored.

The numbers involved in the calculation are similarly punched on a paper tape, usually referred to as the **data tape.** This may be a separate tape, or it may be attached to the end of the programme tape.

1.2 STORAGE OF NUMBERS

It is helpful to form a mental picture of a sequence of numbers being stored within the computer; arithmetic will be done on these, or they will be manipulated in various ways.

The stored numbers are identified **symbolically.** While the programme is being written it is inappropriate to consider the values of the numbers, they will in fact be different each time the programme is used. A symbolic form is therefore required and it is, of course, similar to simple algebraic notation.

All numbers on which arithmetic is done are called **'variables'** and are referred to as $v0$, $v1$, $v2$, ...up to $v1379$.

A second set of numbers, called **indices**, are used mainly for counting and organisation of the programme. They are whole numbers, stored separate from the variables, and are referred to as $n0$, $n1$, $n2$,...up to $n27$.

The number after v or n serves to distinguish one number from another. It will be noted that the numbering begins at 0, which is common practice in connection with a computer.

1.3 INSTRUCTIONS FOR INPUT OF DATA

We now describe how an instruction is written in a programme to make the computer read a number from the data tape and store it in the computer.

We write

$v1$ = TAPE

(and this is exactly what is typed on the teleprinter when punching the programme tape).

Any variable may be used in place of $v1$ e.g:

$v159$ = TAPE

This is a typical Autocode instruction; it means that the variable specified (in the orders above, $v1$, or $v159$) **after** the instruction has been obeyed takes the value of the number that was on the data tape. This number is read into the computer by the photoelectric tape reader.

We may read in a whole sequence of numbers and store them as consecutively numbered variables; we write

$v1$ = TAPE *

This instruction causes the computer to read in numbers from the tape and store them as $v1$, $v2$, $v3$, $v4$, etc., until a letter L is read from the tape. The computer then goes on to the next instruction.

We may also write

$v1$ = TAPE $n1$

This causes the computer to refer to $n1$ and whatever the value of $n1$, it reads in that number of numbers as $v1$, $v2$, $v3$, etc.

Any variable can be used instead of $v1$, and any index instead of $n1$. A number can also be put instead of the index. e.g. $v10$ = TAPE 3 will read 3 numbers from the tape and store them as $v10$, $v11$ and $v12$.

Similar instructions are used to read in integers to be used as indices,

e.g. $n5$ = TAPE
$n1$ = TAPE *
$n3$ = TAPE $n2$
$n2$ = TAPE 5

The computer has two tape readers. The second of these is controlled by rather similar instructions:

$v1$ = TAPEB	$n1$ = TAPEB
$v1$ = TAPEB *	$n1$ = TAPEB *
$v1$ = TAPEB $n1$	$n2$ = TAPEB $n1$

In all input instructions an 'L' on the data tape causes the computer to stop reading numbers and go on to the next instruction.

Whenever an input instruction is obeyed, $n0$ is set equal to the number of numbers read in. This is often very useful but the programmer must remember not to use $n0$ for other purposes when numbers are being read.

1.4 INSTRUCTIONS FOR OUTPUT OF RESULTS

The output and printing facilities of the Autocode are fully described in Section 3.6. For the present however, we will describe only the most frequently used facilities.

A typical output instruction is written:-

PRINT $v1$, 3123

This means that the number $v1$ (in this example) is to be printed in a 'style' determined by the 'style number' 3123.

The style number can be considered in three parts **a**, **b**, and **c**.

a is the first digit.

If **a** = 3 the number is printed on a new line.

If **a** = 4 the number is printed on the same line as the previous printing.

The last three digits have the value 20**b** + **c** where **b** is the maximum number of decimal places required before the decimal point,

and **c** is the number of decimal places required after the decimal point.

Thus 3123 prints the number on a new line with 6 digits before the point and 3 after, and 4005 prints the number on the same line with no places before the point and 5 after.

The following effects always occur:-

(1) Every number is immediately preceded by its sign, + or − ;

(2) If there are less digits before the point than the number specified, the extra places are filled with spaces instead of zeros to preserve the alignment of columns;

(3) Numbers having no integral part are printed with one zero before the decimal point;

(4) Numbers required as integers (**c** = 0) are printed without a decimal point;

(5) All numbers are correctly rounded to the prescribed number of decimal places.

The corresponding instruction for printing an index is written

PRINT *n*1, 3000 (for example)

In the case of indices, only the first digit of the style number has any effect, and all indices are printed as 4-digit integers (with spaces instead of left-hand zeros).

1.5 THE STOP INSTRUCTION

This is written:-

STOP

After this instruction the computer can be made to go on with the next instruction by operating the STOP/RUN key on the control desk.

1.6 INSTRUCTIONS FOR ARITHMETICAL AND OTHER FUNCTIONS

The majority of instructions in a programme take the form of equations in which the new value of a variable or index is computed in terms of the values of one or two previously calculated variables or indices. For example:-

$$v1 = v2 + v3$$

means that the new value of $v1$ is to be the sum of the previous values of $v2$ and $v3$. All the instructions for arithmetical and other functions are arranged in pairs of opposite sign.

Many of these instructions are self-explanatory but notes have been appended in some cases.

1.6.1 Instructions with Variables

$v1 = v2$	Copy	$v1 = -v2$
$v1 = v2 + v3$	Add	$v1 = -v2 + v3$
$v1 = v2 - v3$	Subtract	$v1 = -v2 - v3$
$v1 = v2 \times v3$	Multiply	$v1 = -v2 \times v3$
$v1 = v2 / v3$	Divide	$v1 = -v2 / v3$

$v1$ = MOD $v2$	(1) Modulus	$v1$ = −MOD $v2$
$v1$ = INT $v2$	(2) Integral Part	$v1$ = −INT $v2$
$v1$ = FRAC $v2$	(3) Fractional Part	$v1$ = −FRAC $v2$
$v1$ = SQRT $v2$	Square Root	$v1$ = −SQRT $v2$
$v1$ = SIN $v2$	(3) Sine	$v1$ = −SIN $v2$
$v1$ = COS $v2$	(3) Cosine	$v1$ = −COS $v2$
$v1$ = TAN $v2$	(3) Tangent	$v1$ = −TAN $v2$
$v1$ = CSC $v2$	(3) Cosecant	$v1$ = −CSC $v2$
$v1$ = SEC $v2$	(3) Secant	$v1$ = −SEC $v2$
$v1$ = COT $v2$	(3) Cotangent	$v1$ = −COT $v2$
$v1$ = ARCSIN $v2$	(3) Inverse Sine	$v1$ = −ARCSIN $v2$
$v1$ = ARCCOS $v2$	(3) Inverse Cosine	$v1$ = −ARCCOS $v2$
$v1$ = ARCTAN $v2$	(3) Inverse Tangent	$v1$ = −ARCTAN $v2$
$v1$ = LOG $v2$	(4) Natural Logarithm	$v1$ = −LOG $v2$
$v1$ = EXP $v2$	(5) Exponential	$v1$ = −EXP $v2$
$v1$ = EXPM $v2$	(5) EXP (−$v2$)	$v1$ = −EXPM $v2$

Notes. (1) The modulus of a number is its positive value, thus

the modulus of +2 is +2
the modulus of −2 is +2.

(2) The integral part of a number is defined as the largest integer less than or equal to the number; note that

the integral part of −3.4 is −4
the fractional part of −3.4 is +.6
the integral part of +3.4 is +3
the fractional part of +3.4 is +.4
INT $v2$ + FRAC $v2$ = $v2$ always.

(3) All angles are to be measured in radians;

(1 radian = $180/\pi$ = 57.2957795 degrees, and π = 3.14159265, to the full accuracy used in the Autocode.)

(4) Logarithms to the base e, Naperian logarithms.

(5) EXP $v2$ = e^{v2} This may be regarded as a natural antilogarithm.

1.6.2 Instructions with Indices

$n1$ = $n2$	Copy	$n1$ = −$n2$
$n1$ = $n2$ + $n3$	Add	$n1$ = −$n2$ + $n3$
$n1$ = $n2$ − $n3$	Subtract	$n1$ = −$n2$ − $n3$
$n1$ = $n2$ × $n3$	Multiply	$n1$ = −$n2$ × $n3$
$n1$ = $n2$ / $n3$	(1) Divide	$n1$ = −$n2$ / $n3$
$n1$ = $n2$ * $n3$	(2) $n2$ modulo $n3$	$n1$ = −$n2$ * $n3$
$n1$ = MOD $n2$	Modulus	$n1$ = −MOD $n2$

Notes. (1) $n1$ is taken as the integral part of $n2/n3$.
The negative form is treated as bracketed $-(n2/n3)$.
(See 1.6.1, note (2) above).

(2) $n2$ modulo $n3$ means the remainder when $n2$ is divided by $n3$.
$n1$ and $n3$ have the same sign. If $n3 = 0$, $n1 = n2$. The negative form is treated as bracketed, $-(n2 * n3)$.

1.6.3 Instructions with Variables and Indices

Variables and indices cannot normally be mixed in instructions. The only instructions in which this can be done are as follows:

$n1$	=	$v1$	(Nearest Integer)	$n1$	= $-v1$
$v1$	=	$n1$		$v1$	= $-n1$
$v1$	=	$n2$ / $n3$		$v1$	= $-n2$ / $n3$

1.6.4 N.B. **In all instructions with variables or indices or both, any variable or any index can be used instead of those mentioned in the instructions above.**

N.B. **In all instructions a positive constant may be written in place of any variable or index on the right hand side of the equation.** (In the case of indices integers only).

Examples. All the following instructions are permissible.

$v10$ = 2.932
$v10$ = -3.14159
$v5$ = $n11$ / 15
$n1$ = $n6$ x 4
$v196$ = SQRT 0.723

Note, however, that while

$$v1 = -3.14159$$

is permissible because a positive constant has been substituted for $v2$ in the instruction

$$v1 = -v2$$

nevertheless $v1$ = sin -0.721

or $v1$ = 0.56/-1.7329

are **not** permissible since the list of permissible instructions does **not** include

$v1$ = sin $-v2$ or $v1$ = $v2/-v3$.

1.6.5 Instructions of the following types are also permissible.

$v6$	=	$v6$	x	$v6$	(Take $v6$, square it, and leave the answer as the new value of $v6$.)
$n2$	=	$n2$	+	1	(Increase $n2$ by unity.)
$n6$	=	$-n6$	-	$n3$	

In general: **Only the value of the index or variable on the left hand side of the equation is changed. Any others mentioned in the equation are unaltered in value.**

Thus:- $v6$ = LOG $v22$ gives a new value to $v6$ but leaves $v22$ unchanged.
$n1$ = $n1$ + $n2$ leaves $n2$ unchanged but increases the old value of $n1$ by the value of $n2$.

1.7 A COMPLETE EXAMPLE

We can now give an example of a complete programme.

PROBLEM: Evaluate r and a where

$$r = \sqrt{x^2 + y^2}$$

and $\tan a = x/y$

for given values of x and y.

The values of x and y are assumed to be punched on the data tape in that sequence.

The programme is:-

Notes:-

```
D
N
EXAMPLE 1

J1.0
```

This is explained in Sections 1.11, 1.13, 1.14

```
STOP
v1 = TAPE 2          Pause to insert data tape.
v3 = v1 X v1         Set v1 = x, v2 = y from tape.
v4 = v2 X v2         v3 = x^2
v4 = v3 + v4         v4 = y^2
v4 = SQRT v4         v4 = x^2 + y^2
v3 = v1 / v2         v4 = r
v3 = ARCTAN v3       v3 = x/y
PRINT v4, 3123       v3 = a
PRINT v3, 4045       Print r on a new line.
STOP                 Print a on the same line.

(=>0)
```

This is explained in Section 1.11.

If the data tape consists of the numbers

```
+27.2
-351
```

then the output will look like this:-

```
1/1/1959----101      (Date and serial number)
EXAMPLE 1            (Name)

   +352.052  -0.07734 (r and a)
```

1.8 LABELS AND JUMP INSTRUCTIONS

In a programme, the instructions are usually obeyed in the sequence that they are written. The jump instructions, however, enable this sequence to be broken and can specify which instruction is to be obeyed next at a particular point in the programme.

To do this it is necessary to be able to identify the instruction to which the jump has to go. This is done by **labelling.** A label is a positive integer written with a bracket, before the instruction, e.g.

$$3)\quad v1 = v2 + v3$$

These labels can appear in any sequence in the final programme. If required an instruction could have more than one label {e.g. 2) 10) $n1 = n2 * n3$}, but no label can be used more than once. *

The simplest jump instruction is written

$$\rightarrow r$$

This means that the next instruction to be obeyed is that with label number r. r will usually be a number e.g.:

$$\rightarrow 3$$

will ensure that the next instruction to be obeyed is that labelled 3).

* The Swiss Autocode details (CS2704A) suggest on p 6 §7 that the same label can be used twice in Pegasus Autocode, but that "the last label written is the one used".

An index may be used to determine which label is to be used, e.g.:

$$\rightarrow n5$$

jumps to the instruction with the label whose number is that given by $n5$.

Jump instructions may be made **conditional** on some equality or inequality. e.g.: we can write

$$\rightarrow 3,\ v1 > v5$$

This means "Obey the instruction labelled 3) if $v1$ exceeds $v5$, otherwise go on to the next instruction as usual".

We have a great variety of jump instructions. Some variations are:-

$$\begin{aligned} &\rightarrow r,\ \ v2 > v3\\ &\rightarrow r,\ \ v2 > -v3\\ &\rightarrow r,\ -v2 > v3\\ &\rightarrow r,\ -v2 > -v3 \end{aligned}$$

These 4 instructions can be summarised as

$$\rightarrow r,\ \pm v2 > \pm v3$$

Thus we have the complete range of jump instructions:

$\rightarrow r$	(Unconditional Jump)	
$\rightarrow r,\ \pm v2 > \pm v3$	$\rightarrow r,\ \pm n2 > \pm n3$	(Greater than)
$\rightarrow r,\ \pm v2 \geqslant \pm v3$	$\rightarrow r,\ \pm n2 \geqslant \pm n3$	(Greater than or equal to)
$\rightarrow r,\ \pm v2 = \pm v3$	$\rightarrow r,\ \pm n2 = \pm n3$	(Equal to)
$\rightarrow r,\ \pm v2 \neq \pm v3$	$\rightarrow r,\ \pm n2 \neq \pm n3$	(Not equal to)
$\rightarrow r,\ \pm v2 =^* \pm v3$		(Approximately equal)
$\rightarrow r,\ \pm v2 \neq^* \pm v3$		(Not approximately equal)

"Approximately equal" means agreement to $n0$ significant binary digits, or approximately $(0.3 \times n0)$ significant decimal digits. In these instructions, $n0$ is always consulted to see what tolerance is to be allowed.

In all jump instructions any variable, or a positive integer, may be used.

For example:-

$$\begin{aligned} &\rightarrow 2,\ \ n5 > 6\\ &\rightarrow 10,\ v1 \neq^* -0.0723\\ &\rightarrow 1,\ \ \ 7 \geqslant n2 \end{aligned}$$

are all permissible

For the second of these, $n0$ would be set to a value of 8 or 9, thus giving rather less than three significant decimal figures.

1.9 LOOPS OF INSTRUCTIONS, AND COUNTING

An essential feature of using a digital computer is that repetitive processes are used. In this way a group of instructions which are only written once may be used repeatedly during the calculation, probably operating on different numbers each time. The speed of the computer makes it preferable to use a repetitive method - even if this obeys an unnecessarily large number of instructions - rather than one which needs more written instructions in the programme.

Often, jump instructions will jump back to instructions earlier in the programme, thus forming a repetitive loop of instructions. It is often required that the instructions of this loop are to be obeyed a fixed number of times. This is done by counting, and an index is used as a **counter**, unity being added to it (or subtracted) each time the loop is repeated.

Example:

$v1$ has the value x. Calculate $v2 = 5x^{10}$

The part of the programme to do this could be as follows:

```
      v2 = 5                    Set v2
      n1 = 0                    Set starting value of n1
1)    v2 = v2 X v1              Multiply by x
      n1 = n1 + 1               Count 10
      →1, n1≠10
```

Each time the loop is obeyed, $n1$ is increased by 1, and on the tenth time $n1 = 10$ and the computer leaves the loop and goes on to the next instruction.

1.10 AUTOMATIC SELECTION OF NUMBERS, MODIFICATION

Often during a computation we require to work systematically through sets of numbers stored as variables in the computer. To do this we require the methods of counting mentioned above, but we also need to be able to select in turn a succession of variables, referring to a different variable each time we go round the loop.

This technique is made possible by the following facility:-

Any variable may be specified by an index, or by a constant and an index.

Thus we can write

$$vn1$$

Here the computer uses the index $n1$ to determine the variable. If $n1 = 8$ then

$vn1$ would be interpreted as $v8$.

We can also write

$$v(h + n1)$$

where h is any integer in the range $-2048 \leqslant h < 2048$. For example, if we write $v(7 + n5)$, the computer will consult $n5$ at the time the order is obeyed and if $n5 = 15$, the computer would refer to $v22$. It is important to write $v(7 + n5)$ and **not** $v(n5 + 7)$ as the latter is **not permitted.**

An index employed in this way is called a **modifier**, because the selection of the variable is determined by what the programmer has written, altered - or modified - by the **value of the index.**

When using this facility, the variables in a loop of instructions are defined in terms of one (or more) modifiers, and during each repetition of the loop the modifier will be altered, thus referring to **successive** items of data, and not the same items each time.

This technique of modification is one of the most valuable features available in a computer. It is however one of the more difficult ideas to grasp, mainly because there is nothing else quite like it in any other field of technology or of common life. Examples using modification will be found in Sections 2.1 and 2.2.

1.11 STARTING A PROGRAMME

Immediately before the first instruction of an Autocode programme we must punch: J 1.0

This tells the computer that Autocode instructions follow. As the Autocode programme is read into the computer the instructions are stored away sequentially,

and the first Autocode instruction is automatically labelled 0). These instructions are **not** obeyed as they are read, they are only read in and stored.

An instruction, or group of instructions, can be enclosed in brackets. In this case this instruction or group of instructions, is obeyed **as soon as the right-hand bracket is read.**

Thus we can start to obey a programme at the first instruction by punching

(→ 0)

at the end of the programme tape. When this has been read the instruction → 0 is obeyed and this causes the first instruction of the programme to be obeyed.

1.12 READING MORE PROGRAMME

The computer can be made to read in more programme by the instruction

TAPE

In this case the first new instruction will replace the first of the bracketed instructions at the end of the previous piece of programme.

The instruction

TAPE *n*1

writes the first new instruction on top of the instruction labelled *n*1, and

TAPE *n*1, *n*2

writes the first new instruction in place of the one *n*2 after that labelled *n*1.

As usual any index, or an integer, may replace *n*1 and *n*2.

There is no instruction

TAPE B

to use the second tape reader for this purpose.

1.13 DATE HEADINGS

It is usual to make the computer print out the date, and serial number in the day, at the head of each set of results. This is done by punching the letter D (λD ϕ) at the head of the programme tape.

1.14 TITLES AND HEADINGS

The practice is encouraged of making the computer insert titles, headings, and explanatory notes into the typed output sheet.

This is done by punching at the head of the programme or data tape the letter N (λNϕ) followed by the required title including all layout characters required, i.e. Carriage Returns (CR), Line Feeds (LF), and spaces. When the tape is read, everything after the N is exactly copied on to the output tape until a piece of blank tape is read. (Actually at least 2 blank tape characters). The computer then stops copying and starts to read the rest of the tape.

Thus the programme tape will usually start with

```
D
N
NAME OF PROGRAMME
Blank Tape

J 1.0
```

and end with

```
(→ 0)
```

1.15 THE DIRECTIVE Z

The letter Z on a data tape causes the computer to stop.

1.16 FORM OF NUMBERS IN THE COMPUTER

All **variables** are held in the computer to an accuracy of 8 - 9 significant decimal digits. The value of a variable must not be greater than about 10^{76} or less than -10^{76}. Numbers between $-1/10^{77}$ and $+1/10^{77}$ are treated as zero.

In practice this means that all numbers likely to be encountered can be used, and an overall accuracy in the answers of one part in ten million can be expected except in some very exceptional processes.

All **indices** are held with complete accuracy as integers. No index must take a value above 8191 or below -8191.

1.17 SPEED

It takes about a minute to read the master programme into the computer to make it work in Autocode. Any number of Autocode programmes can then be run.

Autocode instructions are read in at about 2 instructions per second.

Times for reading data are usually relatively short. The tape reader works at 200 tape characters per second and the computer can usually assimilate data at this speed.

Autocode instructions are obeyed at 15 - 20 instructions per second. Output instructions, however, take rather longer, and times for output are often significant.* The maximum speed of the output punch is 33* tape characters per second. On the average the time in seconds for output can be calculated by working out the total number of characters (including spaces, decimal points, etc.) to be punched, and dividing by about 20 - 25.* The output tape is then printed on a teleprinter at a speed of 7* characters per second, but this can be done away from the computer while the computer is used for other problems.

The times mentioned above are considerably slower than the usual operation of the computer. This is the price that has to be paid for the ease of programming which is given by the Autocode. In many problems the reduction in programming time is of much greater value than the increase in running time, and very many programmes, even in Autocode, take only four or five minutes of computer time.

---

* A new fast punch is expected to become available soon for Pegasus. This will have a speed of 240 - 300 tape characters per second. A new type of teleprinter, available soon, gives printing at 10 characters per second.

* Tape Reader 60 p/sec.
* " - 10 "

1.18 PUNCHING THE TAPES

When the programme has been written it must be punched on to paper tape. The data must also be similarly punched. This is done on teleprinter equipment available at any Pegasus (or Mercury) computer installation and some computer users also have their own equipment.

A teleprinter is like a typewriter except that:

(1) It has a key for "letter shift" and a key for "figure shift". On striking the letter shift key, a character is punched, and the figures keys are locked, thus permitting only the letters keys to be depressed. Similarly, pressing the figure shift key permits only figures, symbols, etc. to be punched.

(2) Keys are provided to control the carriage of the teleprinter. "Carriage-return" starts printing at the left-hand end of the line, and "line-feed" moves up the paper by one line. Thus "carriage-return" followed by "line-feed" will start a new line.

The following abbreviations are used.

λ	=	Letter Shift
ϕ	=	Figure Shift
CR	=	Carriage Return
LF	=	Line Feed
Sp	=	Space
Er	=	Erase (all 5 holes)

None of these characters, except Er, itself causes any printing on the paper.

The following recommendations are made for punching all tapes:-

(1) Start every tape by running out at least 6 inches of blank tape.

(2) 1 or 2 inches of blank tape should be inserted between distinct sections of programme and at the end of the tape.

(3) Punch 6 or 8 Er at the finish of the tape. This distinguishes one end of the tape from the other when inserting the tape in the tape reader.

The following rules for punching programme tapes **must** be observed.

(1) In each Directive (e.g. D, N, J 1.0, L) the letter must be immediately preceded by λ and immediately followed by ϕ.

(2) All titles must be terminated by blank tape.

(3) Each Autocode instruction (and J 1.0) must be terminated by CR LF.

(4) Spaces may occur anywhere in an Autocode instruction except among the digits of any number.

(5) Erases may occur anywhere except between CR and LF.

The following rules for punching data tapes must be observed.

(1) Every number must start with + or − and be terminated by Sp or CR LF.

(2) Er may occur anywhere except between CR and LF.

Facilities are provided on the equipment for correcting errors and editing tapes. Tapes may be joined using adhesive tape.

/// CRLF must be punched after blank tape.

1.19 TRYING THE PROGRAMME ON THE COMPUTER

Most programmes, even Autocode programmes, will not work the first time they are tried. This may be due to one of three causes:-

(1) The computer may break down. This is a very rare cause of trouble and **if it occurs the computer will stop.** If the computer produces answers it may be assumed that it is working correctly.

(2) The tapes may be wrongly punched. This is generally immediately apparent, as most punching errors will be detected as the tape is read in, and the computer will stop just after the incorrect piece of tape has been read.

(3) The programme may be wrong. This is by far the commonest cause of trouble. It may cause wrong answers to be produced, or it may cause the computer to stop (e.g. when dividing by zero, or taking the square root of a negative number). In either case the programmer must then find out where he went wrong. This is not nearly as difficult as it might at first appear and after a little experience on the computer most errors can be traced quite easily. Various aids to tracing faults in programmes are described in Section 3.6 (the XP and SP facilities) and 3.16 (Error Tracing).

In general, all programmes should be very carefully checked before going on the computer, and if possible, some sample calculation to which an answer is known should be carried out as a check.

1.20 OPERATING THE COMPUTER

The two main control keys are in the centre of the control panel. They are labelled

START		RUN
NORMAL	and	STOP
MANUAL		SINGLE SHOT

These are referred to as the "Start Key" and the "Run Key". They have to be pulled out before being moved.

The sequence of operations required to run an Autocode programme is as follows:-

(1) Put the blank leader of the master Autocode tape under the reading position of the main tape reader. Clamp it down and pull the tape along very slightly until the light labelled "INPUT BUSY" goes out.

(2) Press down the key labelled "0" of the row across the top of the control panel (the third from the left). This suppresses some "Optional Printing" which is not useful with Autocode programmes.

(3) Push the Start key up to START. It will spring back to NORMAL.

(4) Push the Run key up to RUN and leave it there. The computer will now start to read the tape.

(5) At the end of the master Autocode tape, the computer will stop and the light labelled "STOP ORDER 77" will go on. The computer will hoot when it stops if the key labelled "HOOT ON STOP" is down.

(6) Return the RUN key to STOP.

(7) Replace the master Autocode tape by your programme tape.

(8) Push the Run key to RUN. Your programme tape will now go in. At the end of this tape the computer starts to obey the programme.

(9) The data tape should be put into the tape reader specified by the programme. A "STOP" instruction is usually put in the programme to allow time for this to be done.

(10) Whenever the computer stops, (for example when obeying the instruction "STOP" or reading a "Z" on a data tape) it can be made to continue by putting the Run key to STOP and back to RUN.

(11) The Start key is only used at the beginning of a new piece of programme. It enters the "Initial Orders" of the computer which reads the tape up to "J 1.0". After this point the Autocode takes over.

1.21 ADDITIONAL ITEMS NOT MENTIONED IN THIS PART

Some less frequently used facilities are mentioned only in Part 3 and not in this part:-

	Section
The Directive Q	3.5
Full description of Style Numbers	3.6
The XP and SP, X and S, facilities	3.6
Use of Machine Orders with Autocode	3.12
The ALTER facility	3.13
Storage Space and alteration of maximum numbers of Instructions, Variables, Indices and Labels	3.15
Error Tracing	3.16
Complete Table of Instructions	3.19

P A R T 2

2.0 INTRODUCTION

This Part gives some typical, short, Autocode programmes designed to illustrate various programming techniques and show how to use various features of the Autocode. The notes after each programme are intended to indicate how and why these facilities are used.

2.1 EXAMPLE 1

Problem

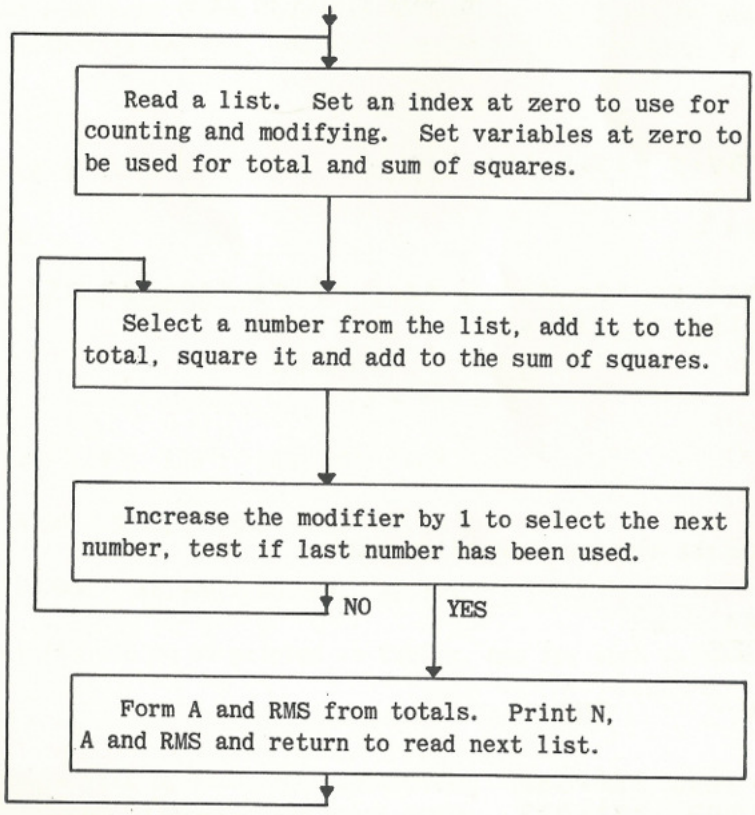
Several lists of numbers are given. The numbers may be expected to have up to 3 decimal digits before, and 3 after the decimal point. Print out for each list:-

N – The number of numbers on the list
A – The average of the numbers
RMS – The root mean square of the numbers [This means the square root of the average of the squares of the numbers].

Method

The lists will be punched on a tape which will be read from the second Tape Reader. Each list of numbers will finish with an "L", and there will be a "Z" at the end of the tape. The lists will be dealt with one at a time.

Flow Diagram



(If there is not another list the computer finds Z instead and stops).

Programme

```
D                                  To print Date and Serial Number
N
LIST AVERAGES                      Name and headings

   N       A          RMS

J1.0                               To enter Autocode

STOP                               Pause to put data tape into second Tape Reader.
1)v10=TAPEB*                       Read a list: n0 = number of numbers on the list.
n1=0                               Start counter/modifier at zero.
v1=0                               To be used for total.
v2=0                               To be used for sum of squares.
2)v1=v1+v(10+n1)                   Select a number (with n1). Add into total.
v3=v(10+n1)xv(10+n1)               Square
v2=v2+v3                           Add to sum of squares.
n1=n1+1                            Advance n1 to select next number.
→2,n1≠n0                           Go back until the end of the list is reached.
PRINTn0,3000                       Print  N  (New line)
v4=n0                              Calculate A
v5=v1/v4
PRINTv5,4063                       Print  A  (Same line)
v6=v2/v4                           Calculate  RMS
v6=SQRTv6
PRINTv6,4063                       Print  RMS  (Same line)
→1                                 Go back to read a new list.
(→0)                               To start programme.

*********                          To mark finish of tape.
```

Data Tape

```
+10.62 -5 +99.5 +15.2 -75.623 L                                  First List

+100 +55 -50 L                                                   Second List

-52 -16.3 -56.52 +70 +30.3 +1.5 -6.23 +17.562 -55 -77            Third List
+10 -10.4 -155.42 -77.4 +.4 L

Z                                                                Stop
**********
```

Results

The output from the computer looks like this:-

```
23/9/58---44
LIST AVERAGES

   N       A          RMS

  +5     +8.939     +56.547
  +3    +35.000     +71.937
 +15    -25.101     +58.674
```

Notes:

(i) Note how the heading is inserted by reading it as a title from the programme tape.

(ii) The instruction $v10$ = TAPEB* reads from the second tape reader and sets the variables $v10$, $v11$, $v12$ etc. equal to the numbers read, until the "L" is read from the tape. $n0$ is set equal to the number of numbers read and is used later to test when the last number is used.

(iii) $n1$ is used as a modifier to select a number from the list. $v(10 + n1)$ selects a new number each time round the loop of instructions.

(iv) The loop is repeated the correct number of times, by testing $n1$ against $n0$

$$\rightarrow 2,\ n1 \neq n0$$

(v) The two instructions

$$v4 = n0$$
$$v5 = v1 / v4$$

must be used since an instruction like

$$v5 = v1 / n0 \quad \text{is } \textbf{not permitted.}$$

(see Section 1.6.3).

(vi) Note the style numbers, and the printing they produce.

(vii) $\rightarrow 1$ at the end of the programme goes back to the beginning to read a new list, or Z if there is no other list.

Time

The Autocode tape takes less than 1 minute to go into the computer; the programme tape takes another 10 seconds, and the computation and punching of the results, a further 15 seconds.

2.2 EXAMPLE 2

Problem

Evaluate the formula:

$$\frac{2Fr(\sin\phi - \phi\cos\phi)}{1 - \cos\phi} \quad \text{for}$$

ϕ = $133^{\circ}\ 30'$; 123°; $105^{\circ}\ 30'$; 90°

r = 4.33; 5.75; 7.25; 8.5; 10.25

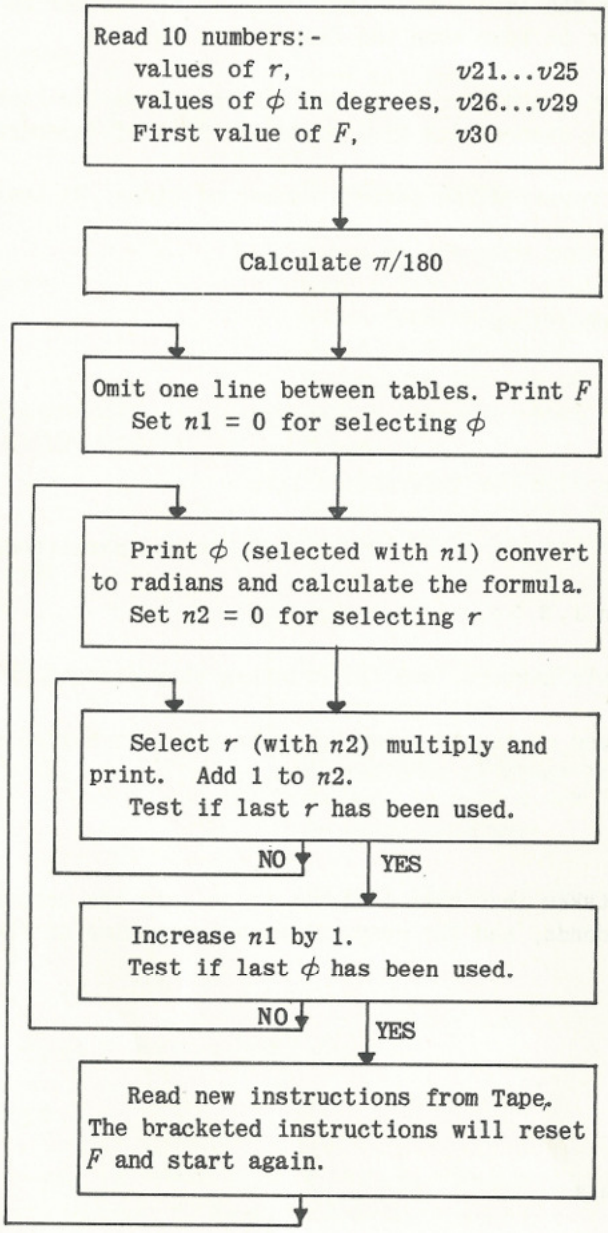
F = 16,000; 18,000; 20,000; 22,000;

The results are to be printed as tables, one for each value of F.

Method

Each table will be calculated separately. After each table, a set of bracketed orders will be read to reset F and start again.

Flow Diagram



Programme

```
D
N
TABULATION                                                    Name and Heading

          4.33    5.75    7.25    8.5    10.25
J 1.0                                   To Enter Autocode.

First Table → STOP                      Pause before reading numbers.
              v21 = TAPE 10             Read 10 numbers.
              v5 = 3.14159 / 180        Calculate π/180
Other Tables → 3) X n1 = 0              Omit one line (X) and set n1 for selecting φ.
              PRINT v30, 3140           Print F
              1) PRINT v(26+n1), 3061   Print φ
              v6 = v(26+n1) X v5        Convert to radians
              v3 = COS v6
              v2 = 1 - v3
              v0 = SIN v6               Calculate
              v1 = v6 X v3              v0 = 2F(sin φ − φ cos φ) / (1 − cos φ)
              v0 = v0 - v1
              v0 = v0 / v2
              v0 = v0 X 2
              S v0 = v0 X v30           'S' prints space
              S n2 = 0                  Set n2 for selecting r
              2) v10 = v0 X v(21+n2)    Multiply by r
              PRINT v10, 4120           Print entry in table
              n2 = n2 + 1               Advance to next r
              →2, n2 ≠ 5                Go back unless end of row
              n1 = n1 + 1               Advance to next φ
              →1, n1 ≠ 4                Go back unless last row has been printed
              TAPE                      Read instructions
              (→0)                      To start programme initially.
```

Data Tape

(This could well be attached to the programme tape in this example)

```
+4.33 +5.75 +7.25 +8.5 +10.25   Values of r
+133.5 +123 +105.5 +90          Values of φ
+16000                          F

(v30 = 18000                    Reset F and print next table
→3)

(v30 = 20000                    Third table
→3)

(v30 = 22000                    Fourth table
→3)

( STOP )                        Stop

**********                      To mark finish of tape.
```

Output

```
23/9/58----45
TABULATION

                4•33      5•75      7•25      8•5       10•25

  +16000
+133•5    +191157 +253846 +320067 +375250 +452508
+123•0    +180114 +239182 +301577 +353573 +426367
+105•5    +159167 +211364 +266503 +312452 +376780
 +90•0    +138560 +184000 +232000 +272000 +328000

  +18000
+133•5    +215052 +285577 +360075 +422157 +509071
+123•0    +202628 +269079 +339274 +397769 +479663
+105•5    +179062 +237785 +299816 +351508 +423877
 +90•0    +155880 +207000 +261000 +306000 +369000

  +20000
+133•5    +238946 +317307 +400083 +469063 +565635
+123•0    +225143 +298977 +376971 +441966 +532959
+105•5    +198958 +264205 +333129 +390565 +470975
 +90•0    +173200 +230000 +290000 +340000 +410000

  +22000
+133•5    +262841 +349038 +440091 +515969 +622198
+123•0    +247657 +328875 +414668 +486162 +586255
+105•5    +218854 +290626 +366442 +429621 +518072
 +90•0    +190520 +253000 +319000 +374000 +451000
```

Notes:

(i) When typing this programme, spaces have been inserted in many places. These make the programme slightly clearer to read and check, but they are entirely optional and their insertion or omission is a matter of taste.

(ii) This time two counters are used and we have two loops, one inside the other. $n1$ counts the number of items in the row and $n2$ the number of rows. At the end of each row, $n2$ is advanced by 1 and $n1$ set back to zero ready to start the next row.

(iii) 'X' is used to give a spare line between successive tables, and 'S' to give two extra spaces between the value of ϕ and the main table.

(iv) The constant $\pi/180$ could be put in directly if known, but it is very easy to calculate it and the time wasted is approximately 0.05 second.

(v) The first table is calculated for $v30\ (F) = 16000$. At the end of the calculation the instruction 'TAPE' reads more programme. This turns out to be a 'bracketed interlude' which resets F and re-enters the programme for the next table.

(vi) After the last table, more programme is read, but this is the single instruction 'STOP' which, since it is bracketed, is obeyed immediately. The computer stops in a condition ready to read more programme if required.

Time

Input of programme tape	10 seconds
Calculation and output (4 tables)	70 seconds

2.3 EXAMPLE 3

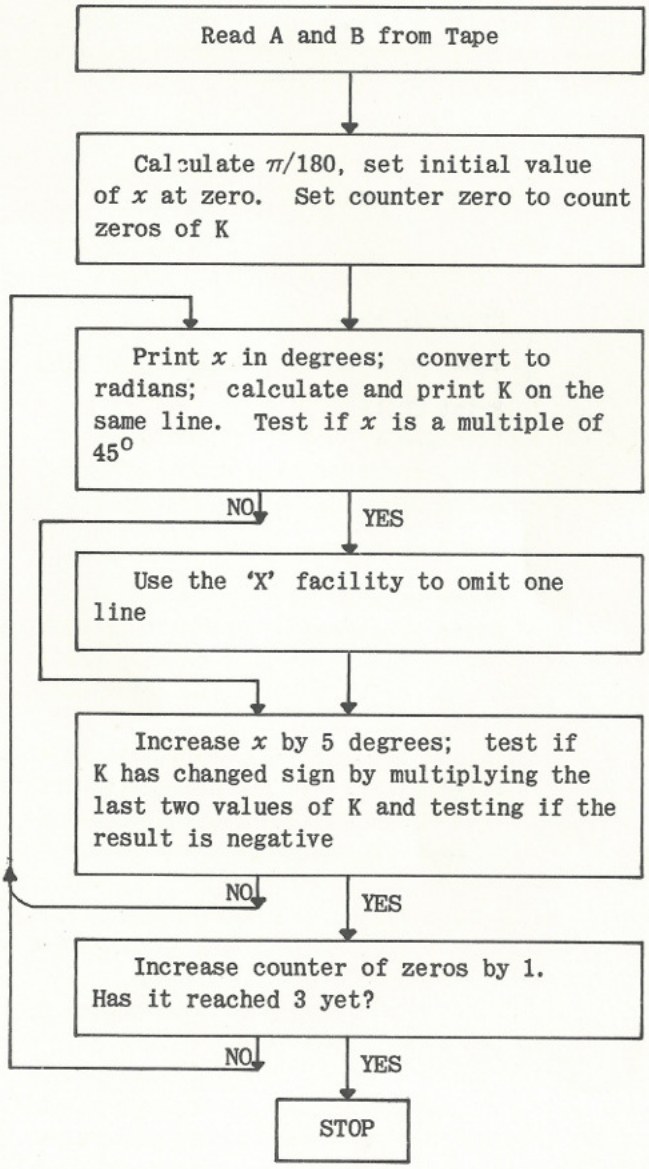
Problem

Calculate

$$K = A \sin x + B \cos 2x$$

for any given pair of constants A and B. Print the values of x and K at intervals of 5 degrees of x, starting at zero and going on until K has passed its third zero value. The results are to be printed in two columns with one line omitted after each multiple of 45^{o}.

Flow Diagram



Programme

```
D
N
ZEROS OF    K = A SINx + B COS2x        } Name and Heading
   X          K
J1.0                                    To Enter Autocode

STOP                                    Pause to insert Data Tape
v9=TAPE2                                Read A and B
v8=3.14159/180                          π/180
v6=0                                    Starting value for checking change of sign
n10=0                                   Counter of zeros
n1=0                                    x in degrees
1)PRINTn1,3000                          Print x, new line
v0=n1                                   } convert x to radians
v0=v0xv8                                }
v1=SINv0                                }
v0=v0x2                                 }
v2=COSv0                                } Calculate K
v3=v1xv9                                }
v4=v2xv10                               }
v5=v3+v4                                }
PRINTv5,4086                            Print K, same line
n2=n1*45                                } Test for x being a multiple of 45°
→2,n2≠0                                 }
Xn0=n0                                  Print CR.LF
2)n1=n1+5                               Increase x
v7=v6xv5                                Test change of sign
v6=v5                                   Store previous value
→1,v7≥0                                 } Add 1 to counter after each zero
n10=n10+1                               }
→1,n10≠3                                Count zero's
(→0)                                    Start Programme
```

Data Tape

```
+1    +2.5          A and B

**********          To mark finish of tape.
```

Output

23/9/58----46

ZEROS OF K = A SINx + B COS2x

X	K
+0	+2.500000
+5	+2.549175
+10	+2.522880
+15	+2.423883
+20	+2.257132
+25	+2.029588
+30	+1.750002
+35	+1.428629
+40	+1.076911
+45	+0.707110
+50	+0.331927
+55	-0.035895
+60	-0.383971
+65	-0.700658
+70	-0.975416
+75	-1.199135
+80	-1.364422
+85	-1.465824
+90	-1.500000
+95	-1.465826
+100	-1.364426
+105	-1.199141
+110	-0.975423
+115	-0.700667
+120	-0.383981
+125	-0.035906
+130	+0.331916
+135	+0.707098
+140	+1.076899
+145	+1.428618
+150	+1.749992
+155	+2.029581
+160	+2.257126
+165	+2.423879
+170	+2.522878
+175	+2.549175
+180	+2.500003
+185	+2.374869
+190	+2.175591
+195	+1.906254
+200	+1.573103
+205	+1.184365
+210	+0.750016
+215	+0.281491
+220	-0.208649

Notes:

(i) 'X' and 'S' must be associated with an instruction. Thus the dummy instruction '$n0 = n0$' is introduced so that 'X' may be put in front of it.

(ii) The instruction $n2 = n1 * 45$ gives the remainder when $n1$ is divided by 45. The instruction $\rightarrow 2$, $n2 \neq 0$ thus **jumps round** the instruction to omit a line, **unless** x is a multiple of 45^{0}.

(iii) The instruction PRINT $v5$, 4086 asks for 4 places before the decimal point. This allows plenty of spaces between columns as K will usually have only one or two digits before the decimal point.

(iv) In this example a sign is tested to decide when to leave the central loop of instructions, instead of a counter. The computer thus goes on until the given condition is satisfied, rather than for a predetermined number of steps.

(v) This programme is not quite perfect. If some value of K were exactly zero it would find 4 zeros and not 3. This is highly unlikely due to very small rounding errors (for example in the value of π given), but the reader may like to deduce what would happen if a value of K were exactly zero.

(vi) In this case, the data tape would probably be attached to the end of the programme tape.

2.4* TESTING FOR SPECIAL CASES

If the computer tries to divide by zero it will stop. This may not always be wanted and it is sometimes desirable to test before a division to see if the denominator is zero.

For example:

$\frac{1}{x} \sin x$ cannot be directly calculated when $x = 0$, but

$$\lim_{x \to 0} \frac{\sin x}{x} = 1$$

Thus to print $\frac{1}{x} \sin x$ for **any** value of x, including zero, (where $v1 = x$) we could write:

```
    →2, v1 = 0
    v2 = SIN v1
    v2 = v2 / v1
    →3
2)  v2 = 1
3)  PRINT v2, 3027
```

Similar precautions may be needed for the square root of a negative number, and for the functions TAN, COT, SEC, CSC.

2.5 ROUNDING ERRORS

The computer works in the binary scale and not the decimal scale. This means that decimal fractions may have a rounding error in their 9th or 10th significant figure. This is insignificant in almost every case but it should be watched when using conditional

---

* The remainder of this Part will not give complete programmes but will illustrate points of interest.

jump instructions. For example, an instruction such as '→ 2, v10 = 0.15' should not be used, since it is unlikely that exact equality will occur even if v10 has been calculated simply by multiplying 0.05 by 3.

Instead, the conditions '>' or '=*' should be used.

2.6 SUBROUTINES

It may be that a particular piece of Autocode programme may be required several times during a calculation, returning to a different position each time it is used. This piece of programme is then called a **subroutine**. It will start with a labelled instruction and finish with an instruction such as '→ n5'. To use the subroutine it is then necessary to set n5 to the number of the label to which it is required to return, and jump to the beginning of the subroutine.

For example: A subroutine performs an interpolation in a table; it replaces v10 by its interpolated value. The first instruction is labelled 5) and the last instruction is '→ n2'.

To perform an interpolation the instructions are:-

```
v10 = --------  (required value to be looked up)

n2  = 3

→ 5

3)  ------------  (continue with interpolated value)
```

On a later occasion we may wish to use the subroutine again and return to label 15). To do this we simply set n2 = 15 and the subroutine returns to label 15) instead of 5).

2.7 PRINTING OF SUBTITLES AND NOTES

Subsidiary titles may be read and copied from a tape as follows. Usually the second tape reader is used but this is not essential.

We write an instruction such as

```
v100 = TAPEB
```

This should set v100 to a number read from the second tape-reader. Suppose, however that we punch on the tape:

```
CR LF λ N ϕ CR LF
λ P A G E ϕ Sp 2 CR LF  Blank Tape CR LF
λ L ϕ
```

On reaching the instruction

```
v100 = TAPEB,
```

the computer reads the tape and finds 'N'. It therefore copies the following name, ending with blank tape. It then finds 'L' and this causes the computer to go back and obey the next instruction. Thus the name is copied but v100 is unchanged. Any variable or index can be used as it is unchanged by the instruction.

N.B. n0 is set to 0 before the tape is read; and therefore any number in n0 before this instruction is lost unless steps taken to preserve it.

P A R T 3

3.0 INTRODUCTION

This Part gives a complete Specification of the Pegasus Autocode. It is intended to be used for reference while using the Scheme.

Certain Sections are printed in colour. These refer to the use of ordinary machine orders with the Autocode, and may be intelligible only to those who have learnt the conventional method of programming Pegasus.

3.1 AUTOCODE PROGRAMMING

To use the Autocode on a particular problem the calculation must first be broken down into a sequence of steps, each of which is the calculation of a number from one or two previously calculated numbers. Each step is written as an **instruction** and the whole sequence forms the **programme** of the calculation. The programme is punched, as written, on a teleprinter or a keyboard perforator to give an Autocode programme tape. After the master Autocode tape has been read into Pegasus, the computer can read in the Autocode programme tape (using the instruction input section); the programme is converted into a suitable internal form and stored. At the end of the tape are some special symbols which cause the programme to be obeyed. At various stages while it is being obeyed the programme can read in numerical data from tapes in either of the tape readers, or it can cause printing of results (they are actually punched and printed later).

3.2 NUMBERS

Two types of numbers are handled by Autocode instructions:

(a) **Variables** denoted by $v0$, $v1$, $v2$, $v3$,...

The number after a v serves to distinguish one variable from another, and is to be thought of as a suffix.

Variables may be signed integers, (decimal) fractions, or mixed numbers. Variables as large as 10^{76} or as small as 10^{-77} are held with a precision of between 8 and 9 significant decimal digits. Variables smaller than 10^{-77} (approximately) are treated as zero.

(Variables are stored as standard, packed, floating-point numbers with 9 bits for the exponent and may therefore lie in the range

$$-2^{254} \leqslant v \leqslant 2^{254}(1 - 2^{-28}).$$

Numbers in the range

$$-2^{-257} \leqslant v \leqslant 2^{-257}(1 - 2^{-28})$$

are stored as zero.)

(b) **Indices** denoted by $n0$, $n1$, $n2$, $n3$, ...

These are signed integers in the range

$$-8191 \leqslant n \leqslant 8191.$$

(They are stored in the modifier position of words and are handled in fixed-point form.)

Most of the numbers entering into a calculation are variables; the indices are introduced as auxiliary quantities, mostly to facilitate counting and modifying. The v and n symbols are available in figure-shift on the teleprinter.

The total available number of variables and indices is adjustable (see Section 3.15), but they are normally fixed so that there are:

(a) 1380 variables $v0$, $v1$, ..., $v1379$
(b) 28 indices $n0$, $n1$ ...., $n27$.

3.3 ADDRESSES OF STORAGE LOCATIONS

Conventional methods of programming a digital computer employ the concept of storage locations, or registers, in which numbers are placed; a distinction is made between the locations and the numbers. The storage locations are defined by a system of addresses.

Experience in training shows that this concept can be puzzling to a beginner, and it may take say half a day to assimilate it, with the nomenclature of addresses.

This concept is however unnecessary for the Autocode, and has consequently been avoided. Instead, the simpler concept is used of the computer holding a series of numbers, expressed symbolically (Section 1.2). The experienced programmer will, however, recognise these as being equivalent to storage locations, addressed with vs or ns.

This point has been explained so that it may be borne in mind by an experienced programmer when assisting a colleague who has knowledge only of the Autocode.

3.4 AUTOCODE INSTRUCTIONS

Most Autocode instructions take the form of an equation giving the new value of a variable (or index) in terms of one or two numbers or previously calculated variables (or indices). For example

$$v1 = v2 + v3$$

means the new value of $v1$ is to be the sum of $v2$ and $v3$.

Any v or n, within the total number permitted, may be used in these instructions.

On the right-hand side of the equation, any variable may be replaced by a number, and any index by an integer. (Positive numbers, only, in the case of the function instructions.)

To provide the facility of **modification**, any instruction involving variables may have any variable specified by means of any index, e.g.

$$v8 = v998 + vn6$$

or by means of a constant (h) and any index, e.g.

$$v(8 + n2) = v(53 + n4) + v(-2 + n11)$$

The integer h must be written before the n, and can have any value in the range

$$-2048 \leqslant h \leqslant 2047$$

but the result after the addition of the n must not be negative. The index must be **added**, never subtracted.

3.5 INPUT OF DATA

A set of instructions is provided for the input of numerical data; each reads one (or more) complete numbers punched on the data tape. A simple one is the following

$v6$ = TAPE

This causes $v6$ to be set equal to a number read from the tape in the main tape reader. The instruction

$v5$ = TAPE 13

causes 13 numbers to be read and placed in $v5$, $v6$, ..., $v17$.

The instruction

$v28$ = TAPE *

causes numbers to be read and placed in $v28$, $v29$, $v30$, ... until an L-directive is read. The input instructions are given in full in Table 1. It should be noted that *any* input instruction has the following properties

(a) input ceases and the next instruction is obeyed when L is read.

(b) $n0$ is put equal to the number of numbers read in.

(c) Directive N causes the characters following it to be copied on the output tape as a name. As with the Initial Orders the name must be terminated by blank tape.

(d) Directive Z causes the computer to wait on a 77 stop in 0.2. Input may be resumed by operating the STOP/RUN key.

(e) Directive Q sets a decimal block exponent. Q must be followed by an integer $\pm q$ representing a power of 10 by which succeeding variables are to be multiplied. All succeeding variables in that input instruction, but not in later instructions, will be multiplied by 10^q unless a second Q is read. Q can not be used with indices.

$v1$ = TAPE $v1$ = TAPEB	read one number and set in $v1$.
$n1$ = TAPE $n1$ = TAPEB	read one integer and set in $n1$.
$v1$ = TAPE $n2$ $v1$ = TAPEB $n2$	read $n2$ numbers and set in $v1$, $v2$ .... ($n2 > 0$)
$n2$ = TAPE $n1$ $n2$ = TAPEB $n1$	read $n1$ integers and set in $n2$, $n3$, ... ($n1 > 0$)
$v1$ = TAPE * $v1$ = TAPEB *	read numbers up to L on tape, set in $v1$, $v2$, ...
$n1$ = TAPE * $n1$ = TAPEB *	read integers up to L on tape, set in $n1$, $n2$, ...

TAPE instructions refer to the main tape reader, TAPEB to the second tape reader.

Table 1 Autocode instructions - input

Suppose, as an example, that the instruction

$v1$ = TAPE *

is obeyed and that the following tape is in the main tape reader:

```
N
DATA 3
(blank tape)
+1.5
Q + 10     -.657     +0.9876
Q - 15
+0.55332
-13
L
```

When the instruction has been obeyed the following will be stored:

$$v1 = +1.5, \qquad v2 = -0.657 \times 10^{10}, \qquad v3 = +0.9876 \times 10^{10},$$

$$v4 = +0.55332 \times 10^{-15}, \qquad v5 = -13 \times 10^{-15}, \qquad n0 = 5.$$

The name DATA 3 will be printed when the tape is read.

Each number on the tape is punched as written, immediately preceded by its sign and must be terminated by Sp or CR LF. (If indices are to be read in they must be punched without a decimal point).

The TAPE input instructions actually read in numbers from whichever tape reader was last selected. Since all input instructions leave the main reader selected trouble can only occur when the Autocode programme is entered from machine orders or from Initial Orders. In both cases care should be taken to leave the main tape reader selected before entry.

3.6 OUTPUT OF RESULTS

There are two main methods of output. The most used is a special instruction of which the following is an example:

PRINT $v6$, 3045

This instruction causes the value of $v6$ to be printed in a way determined by the style-number 3045. To find the style-number to write in a print instruction we must first decide whether the number is to be printed on a new line or on the same line as the previous number and whether it is to appear in floating-point form (i.e. as a number and a decimal exponent) or in fixed-point form (with the decimal point in its proper position). This determines the first digit (a) of the style-number which can be found from the following table:

	Floating-point	Fixed-point
Print on a new line (CR LF ϕ before number)	$a = 1$	$a = 3$
Print on same line (Sp before number)	$a = 2$	$a = 4$

The rest of the style-number is determined by the number of digits (b) to be printed before the decimal point and after it (c); the style-number is

$$1000a + 20b + c$$

so that style 3045 causes the number to be printed in fixed-point form on a new line with two digits before the decimal point and five after it. The number is preceded by its sign and is rounded.

If desired an instruction of the form

PRINT $v3$, $n9$

may be written, when the current value of $n9$ is taken as style-number.

A similar instruction may be used for printing indices; here only the first digit of the style number matters and the index is always printed as a 4- digit signed integer. Thus the instruction

PRINT $n3$, 4000

causes the value of $n3$ to be printed, preceded by a single space. In all printing non-significant zeros in the integral part are replaced by spaces, except that the digit before the decimal point is always printed. If $b = 0$ then zero is printed before the decimal point. If $c = 0$ the decimal point is not printed.

An alternative method of output is mainly useful to provide extra printing in the development stage of an Autocode programme. We can write XP (for printing on a new line) or SP (for the same line) **before** an instruction; this causes the result of the instruction to be printed if handswitch 0 on the computer is up when the instruction is obeyed. No printing occurs if this handswitch is down. Thus the instruction

$$\text{XP } v9 \;=\; v3 + v5$$

causes the new value of $v9$ to be printed on a new line provided handswitch 0 is in the up position.

To help in laying out the printing an X before an instruction causes printing of CR LF and an S causes printing of Sp, after the instruction has been obeyed. This printing can not be suppressed by handswitch 0.

Only one of XP, SP, X, S may be written before any one instruction and only arithmetic and function instructions can be preceded by XP, SP, X or S.

The output instructions are given in detail in Table 2.

PRINT $v1$, $n2$. Print $v1$ in style $n2 = 1000a + 20b + c$.
If $a = 1$ print CR LF ϕ then number in floating-point form with b digits before point and c digits after point, then Sp and two digit signed exponent, then Sp Sp. Width of printing $8 + b + c$ unless $b = 0$ (width $9 + c$) or $c = 0$(7 + b).
If $a = 2$ print Sp then number as for $a = 1$. Width $9 + b + c$ unless $b = 0$(width $10 + c$) or $c = 0$(8 + b)
If $a = 3$ print CR LF ϕ then number in fixed-point form with b digits before the point and c digits after. Width $2 + b + c$ unless $b = 0$(3 + c) or $c = 0$(1 + b). If b is too small print in floating-point form as if $a = 1$.
If $a = 4$ print Sp then number as for $a = 3$. Width $3 + b + c$ unless $b = 0$(4 + c) or $c = 0$(2 + b). If b is too small print as if $a = 2$.

PRINT $n1$, $n2$. Print $n1$ in style-number $n2 = 1000a$
If $a = 3$ print CR LF ϕ then 4-digit index. Width 5.
If $a = 4$ print Sp then 4-digit index. Width 6.

XP before an instruction (arithmetical or functional) the result of which is a **variable.** Obey instruction, then print CR LF ϕ and result in floating-point form as for PRINT with $a = 1$, $b = 0$, $c = 9$. Width 18.

XP before an instruction the result of which is an **index.** Obey instruction, then print CR LF ϕ and result as four digit integer. Width 5.

SP is similar to XP but Sp is printed instead of CR LF ϕ. Width 19 for variables, 6 for indices.

Note: XP and SP only cause printing if handswitch 0 is up when the instruction is obeyed.

X, S } before any arithmetical or functional instruction. Print CR LF or Sp respectively, after obeying the instruction. This printing is not affected by the setting of handswitch 0.

Table 2 Autocode instructions - output

The following are typical results of some output instructions

PRINT *vn*10, 1064	CR LF ϕ + 345.6789 Sp Sp + 1 Sp Sp
PRINT *v*3, 2064	Sp − 100.0000 Sp Sp + 6 Sp Sp
PRINT *v*4, 2044	Sp + 12.3456 Sp + 12 Sp Sp
PRINT *v*9, 3060	CR LF ϕ − 123
PRINT *vn*4, 3062	CR LF ϕ Sp Sp + 2.36
PRINT *v*(1 + *n*2), 4026	Sp + 9.876543
PRINT *vn*6, 4063	Sp Sp Sp + 0.016
PRINT *n*2, 3000	CR LF ϕ Sp Sp + 12
PRINT *n*5, 4000	Sp − 7732
XP *v*4 = *v*3 + *v*2	CR LF ϕ + 0.123456789 Sp Sp + 2 Sp Sp
SP *v*7 = *v*9 + 10	Sp − 0.987654321 Sp + 11 Sp Sp
XP *n*3 = 1 + *n*3	CR LF ϕ Sp + 103
X *n*2 = 0	CR LF
S *n*4 = *n*7 − *n*1	Sp

3.7 ARITHMETICAL INSTRUCTIONS

A wide range of arithmetical instructions is provided which give the new value of a variable (or index) in terms of one or two numbers or previously calculated variables (or indices). Note that each instruction involves not more than two variables or numbers on the right-hand side.

The purely arithmetical instructions are summarised in Table 3. In this table *v*1, *v*2 and *v*3 are given simply as representative variables, and *n*1, *n*2 and *n*3 represent any three indices.

Variables		Indices	
*v*1 = *v*2	*v*1 = −*v*2	*n*1 = *n*2	*n*1 = −*n*2
*v*1 = *v*2 + *v*3	*v*1 = −*v*2 + *v*3	*n*1 = *n*2 + *n*3	*n*1 = −*n*2 + *n*3
*v*1 = *v*2 − *v*3	*v*1 = −*v*2 − *v*3	*n*1 = *n*2 − *n*3	*n*1 = −*n*2 − *n*3
*v*1 = *v*2 × *v*3	*v*1 = −*v*2 × *v*3	*n*1 = *n*2 × *n*3	*n*1 = −*n*2 × *n*3
*v*1 = *v*2 / *v*3	*v*1 = −*v*2 / *v*3	*n*1 = *n*2 / *n*3	*n*1 = −*n*2 / *n*3
		*n*1 = *n*2 * *n*3 (rem)	*n*1 = −*n*2 * *n*3
Mixed:	*n*1 = *v*2	*n*1 = −*v*2 (nearest integer)	
	*v*1 = *n*2	*v*1 = −*n*2	
	*v*1 = *n*2/*n*3	*v*1 = −*n*2/*n*3	

Table 3 Autocode instructions - arithmetic

The instruction *n*1 = *n*2/*n*3 gives the integral quotient when *n*2 is divided by *n*3 and the instruction *n*1 = *n*2 * *n*3 gives the corresponding remainder (zero or with the same sign as *n*3). The analogous instructions with a minus sign give the same numbers with their signs changed. In any of the instructions any variable (or index) may be replaced by a positive number (or integer).

3.8 INSTRUCTIONS FOR FUNCTIONS

Certain elementary **functions** may be evaluated by a single instruction; for example we can write

$$v6 = \text{SQRT } v9$$

to evaluate a square root, or

$$v3 = \text{SIN } v99$$

to evaluate a sine.

The functions available are listed in Table 4; they all apply to variables only, except for MOD which can be used to form the modulus of an index or a variable.

Again, any variable on the right may be replaced by a positive number; for example

$$v37 = \text{EXP } 5.1058809$$

$v1 = \text{MOD } v2$	$v1 = -\text{MOD } v2$	modulus
$v1 = \text{INT } v2$	$v1 = -\text{INT } v2$	integral part
$v1 = \text{FRAC } v2$	$v1 = -\text{FRAC } v2$	fractional part
$v1 = \text{SQRT } v2$	$v1 = -\text{SQRT } v2$	square root
$v1 = \text{SIN } v2$	$v1 = -\text{SIN } v2$	circular functions
$v1 = \text{COS } v2$	$v1 = -\text{COS } v2$	
$v1 = \text{TAN } v2$	$v1 = -\text{TAN } v2$	
$v1 = \text{CSC } v2$	$v1 = -\text{CSC } v2$	
$v1 = \text{SEC } v2$	$v1 = -\text{SEC } v2$	
$v1 = \text{COT } v2$	$v1 = -\text{COT } v2$	
$v1 = \text{ARCSIN } v2$	$v1 = -\text{ARCSIN } v2$	inverse circular functions
$v1 = \text{ARCCOS } v2$	$v1 = -\text{ARCCOS } v2$	
$v1 = \text{ARCTAN } v2$	$v1 = -\text{ARCTAN } v2$	
$v1 = \text{LOG } v2$	$v1 = -\text{LOG } v2$	natural log
$v1 = \text{EXP } v2$	$v1 = -\text{EXP } v2$	exponential
$v1 = \text{EXPM } v2$	$v1 = -\text{EXPM } v2$	exp $(-v2)$
$n1 = \text{MOD } n2$	$n1 = -\text{MOD } n2$	modulus

Table 4 Autocode instructions - functions

3.9 JUMP INSTRUCTIONS

As usual, Autocode instructions are obeyed in the sequence in which they are written, until a **jump instruction** is encountered. If a jump occurs the next instruction to be obeyed is identified by its **label** which is simply a small positive integer written in front of the instruction and separated from it by a right bracket. For example

$$7)\ v4 = -9.44/v5$$

is labelled 7. Any instruction can be labelled but the same label should not be used twice. If required we can attach two or more labels to the same instruction, thus

$$9)\ 2)\ v3 = \text{LOG } v0$$

The first instruction in an Autocode programme is always automatically labelled 0, there is no need to write this label in.

A jump instruction always includes an arrow; the simplest is the unconditional jump, for example

$\rightarrow 7$

means jump to the instruction labelled 7.

Consider the following instruction:

$\rightarrow 8,\ v1 \geqslant v6$

This means jump to the instruction labelled 8 if $v1 \geqslant v6$, otherwise carry on with the next instruction as usual. The following three instructions resemble this one closely:

$\rightarrow 8,\ v1 \geqslant -v6$
$\rightarrow 8,\ -v1 \geqslant v6$
$\rightarrow 8,\ -v1 \geqslant -v6$

These four instructions can be summarised as

$\rightarrow 8,\ \pm v1 \geqslant \pm v6$

All the available jump instructions are summarised in this way in Table 5; as before 1,2,3 represent any three numbers.

$\rightarrow 1$ (unconditional jump)
$\rightarrow 1,\ \pm v2 \geqslant \pm v3$ $\quad \rightarrow 1,\ \pm v2 > \pm v3$
$\rightarrow 1,\ \pm v2 = \pm v3$ $\quad \rightarrow 1,\ \pm v2 \neq \pm v3$
$\rightarrow 1,\ \pm n2 \geqslant \pm n3$ $\quad \rightarrow 1,\ \pm n2 > \pm n3$
$\rightarrow 1,\ \pm n2 = \pm n3$ $\quad \rightarrow 1,\ \pm n2 \neq \pm n3$
$\rightarrow 1,\ \pm v2 = * \pm v3$ (jump if approximately equal; more exactly, jump if the two variables agree to $n0$ significant binary digits – i.e. to about $0.3 \times n0$ significant decimal figures) $(1 \leqslant n0 \leqslant 28)$
$\rightarrow 1,\ \pm v2 \neq * \pm v3$ (jump if not approximately equal)

Table 5 Autocode instructions - jumps

In a jump instruction any variable may be replaced by a number and any index by an integer. For example

$\rightarrow 4,\ 0 > v62$
$\rightarrow 8,\ n6 \neq -20$

Jump instructions can also be modified. The following are examples:

$\rightarrow n7,\ v0 > v67$
$\rightarrow n9,\ -n8 \geqslant 21$
$\rightarrow (-4 + n8),\ n0 \neq 8$

Care should be exercised in using the conditional jumps involving variables because decimal fractions are stored in binary form and therefore cannot in general be held exactly. Integers in the range $-2^{28} \leqslant v \leqslant 2^{28}$ may be stored exactly but inaccuracies may arise in calculating them.

The jumps testing approximate equality are intended to allow for the effects of the rounding errors inevitable in most floating-point work; the value of $n0$ must be set before using them (e.g. by an instruction such as $n0 = 20$).

3.10 STOP INSTRUCTION, BRACKETED INTERLUDES, COMPLETE PROGRAMME

The instruction STOP

causes a 77-stop in U 5.0. If the RUN key is operated the next Autocode instruction will be obeyed.

The programme tape for an Autocode Programme is headed as follows:

```
D
N
(name of programme)
blank tape
J 1.0
```

The tape up to and including the J-directive is read by the Initial Orders; the name must be terminated by blank tape. The remainder of the tape is read by the Autocode instruction input (the Autocode itself having been read previously). The Autocode instructions making up the programme are punched (as written) in their correct sequence after the J 1.0. Each instruction is converted into a block of machine orders and numbers and stored. About half a second is required to read each instruction. At the end of the programme certain special bracketed symbols must be punched to cause the programme to be entered.

An instruction or a group of instructions written in brackets is obeyed as soon as it has been read. Since the first instruction in the programme is automatically labelled 0 we can enter a programme (i.e. start obeying it) by punching

```
(→0)
```

at the end of the tape. This is an unconditional jump to the beginning of the programme and since it is written in brackets it is obeyed the moment it has been read.

A sequence such as the following may be punched:

```
(v1 = 1.76
 n2 = 10
 →3)
```

This sequence is all read and stored but a note is made of the left bracket; when the right bracket is read the instruction after the left bracket is obeyed, then the next, and so on. The last instruction in the example is an unconditional jump to the instruction labelled 3 and the main programme will be entered at this point. Such a sequence of instructions is called a **bracketed interlude.** If the last instruction of the interlude does not cause a jump then further instructions, punched after the interlude on the tape, will be read and will obliterate the interlude. If we wish, the bracketed interlude can be quite a complicated programme including loops and print instructions, for example. In fact, the whole programme could be put in brackets, so that it would be entered when the right bracket is read.

The Autocode uses one block beyond those where the programme has been stored, to record the right bracket and to cause more programme to be read if the last instruction of the interlude does not cause a jump.

3.11 FURTHER FACILITIES

The instruction TAPE

causes more instructions to be read in from the input tape (in the tape reader last selected i.e. the main tape reader unless the programmer has taken measures to select the second reader) and to be added at the end of the programme (in fact they overwrite the last bracketed interlude). This instruction can be used in a programme designed

to do a rather complicated calculation on a few numbers or parameters. At the end of this calculation a TAPE instruction can be used to read a bracketed interlude such as

```
(n4 = 3
 v1 = 41.509
 →6)
```

This interlude can set new values of the numbers or parameters and cause the calculation to be repeated.

A variant of the TAPE instruction is written, for example

```
TAPE 6
```

This reads in more instructions and puts the first one in place of the instruction labelled 6. Similarly the instruction

```
TAPE 6,3
```

will place the first of the new instructions over the third instruction after that labelled 6. Instructions such as

```
TAPE n1
TAPE n1, n2
```

can also be used.

The instructions of Sections 3.10 and 3.11 may be summarized thus:-

STOP	Stop(77), proceed with next instruction when RUN key is operated.
TAPE	Read in more instructions and add them at the end of the programme.
TAPE *n*1	Read in more instructions, the first to replace that labelled *n*1.
TAPE *n*1, *n*2	Read in more instructions, the first to replace the instruction *n*2 after that labelled *n*1.

Table 6 Miscellaneous Autocode Instructions

3.12 USING MACHINE ORDERS WITH THE AUTOCODE

It is sometimes desirable to stop obeying an Autocode programme and to start obeying ordinary Pegasus machine orders; after some special calculation has been done the programmer may wish to return to the Autocode programme, either at the next instruction or at some other specified instruction.

The instruction →M 939

means: start obeying machine orders at decimal address 939 (i.e. at B117.3). The effect is similar to a J-directive with decimal address 939 so that B117 will be transferred to U0, the next three blocks to U1,2 and 3 and a jump will occur to 0.3 (a-order). A link is set in X1 and a special word in X2; if the link is obeyed with C(2) undisturbed the computer will return to the next Autocode instruction (i.e. the one after the →M 939). Similary the instruction

```
→M 939, v2
```

will cause exit to machine orders at decimal address 939 but in addition the variable

$v2$ will be unpacked; X7 will contain its argument and X5 will contain 256 + its exponent. Instructions such as

→M $n3$, $v5$
→M 939, $n1$

can also be used. The last will leave $n1$ in 7_m. With any →M instruction the settings of the accumulators are reproduced in B0.

Entry from machine orders to an Autocode programme at the instruction labelled 15 (for example) can be done by putting 15 into X2 as an integer and obeying cue 01 to R 600:

```
35+ [4] 72
4.4  0 60
```

i.e. normally

```
37  [4] 72
4.4  0 60
```

Before obeying this cue a link may be set in X1; this is obeyed by the Autocode instruction

→L

Machine orders should normally be punched after the name of the programme but before the J 1.0 entering the Autocode. The space available for storing machine orders may be obtained by reference to section 3.15. It is usually convenient to use for machine orders the space reserved for the higher numbered variables.

The Initial Orders may be called in as a subroutine by the instruction

→I0

A link is set which causes return to the next Autocode instruction when an L-directive is read, provided the contents of B0.1 and 0.2 are undisturbed. Similarly the instruction

→I0, 1200

calls in the Initial Orders after setting the Transfer Address to decimal address 1200 (i.e. to B150.0), and the instruction

→I0, 1200, 153

will in addition set the Relativiser to 153. Instructions such as the following may also be used

→I0, $n1$
→I0, $n1$, $n2$

The cue to enter the Autocode programme from machine orders is:-

01

```
35+ [4] 72

4.4  0 60
```

Cues 02 and 03 are partial cues containing the address of the first block of the master Autocode programme (R 600), other than B1, in the address parts of the a-order and b-order respectively.

These various Autocode instructions are summarized thus:-

→ M *n*1	Exit to machine orders at decimal address *n*1
→ M *n*1,*v*2	The same, and leave *v*2 unpacked in X7 and X5
→ M *n*1,*n*2	The same but leave *n*2 in 7_m.
→ L	Obey link set in X1 on entry from machine orders
→ IO	Call in Initial Orders as a subroutine
→ IO,*n*1	The same, but set T.A. = *n*1
→ IO,*n*1,*n*2	The same and set Relativiser = *n*2

Table 7 The Autocode instructions to permit use of machine orders

3.13 RE-ENTRY AND ALTERATIONS

At the beginning of the Autocode programme tape the directive J 1.0 is punched; this causes the input programme to treat the following instructions as a new Autocode programme and to assign the label 0 to its first instruction. Sometimes we may wish to **restart** an Autocode programme that has already been read and stored, perhaps after changing one or two parameters or after making some alterations (this is described below). This can be done by reading a bracketed interlude ending with a jump, for example

```
(n3 = -48
→1)
```

J 1.2 must be punched at the head of this tape; this has the same effect as the instruction TAPE so that any instructions read will overwrite the last bracketed interlude. J 1.0 must not be punched at the beginning of this tape for the input programme would then treat it as the start of a new programme and store it over the beginning of the original programme.

Occasionally it may be necessary to alter an instruction in an Autocode programme. This can be done by using the tape-editing equipment to alter the tape but this may be inconvenient if the tape is long. Instead of doing this a correction may be inserted near the end of the tape, before the bracketed interlude causing entry to the programme. This correction is written, for example

```
ALTER 6,3
```

followed by one Autocode instruction (on a new line); this instruction will then replace the one 3 after that labelled 6. Similarly

```
ALTER 6
```

causes the instruction labelled 6 to be altered. This sequence forms a kind of **Autocode directive** and is not stored; it does not form a part of the programme. When the alteration has been made the input section of the Autocode scheme returns to read more instructions or alterations. If the alteration is to be made by a separate tape then it should be headed J 1.2 as described above.

3.14 TAPE PREPARATION

Instructions are punched exactly as written. During input CR LF, LF, Sp, ϕ and Er are ignored between instructions. Er is ignored everywhere except between CR and LF, Sp is ignored in instructions except among the digits of a number. Every instruction must be terminated by CR LF.

Punching errors or tape reader errors detected during input of instructions will cause a 77-stop in 0.2. The instruction in which the stop occurs can be read in again by pulling the tape back to the beginning of the instruction and operating the STOP/RUN key.

Numbers to be taken in by input instructions must be preceded by their sign and terminated by CR LF or Sp. CR LF, LF, Sp, ϕ and Er are ignored between numbers and Er is ignored in numbers.

Punching errors or tape reader errors during input of numbers will cause a 77 stop in 0.4. The number in which the stop occurs can be read in again by pulling back the tape, clearing the handswitches and operating the STOP/RUN key. The input instruction can be repeated from the beginning by first ensuring that the handswitches are not clear. Most punching errors in numbers cause a stop immediately they are read. It should be noted however that if the sign of a number is not punched the stop does not occur until the terminating character is read.

3.15 STORAGE SPACE AND OPERATION*

Ordinarily the master Autocode tape is read in with handswitch 1 up, and the Autocode programme is read after it by operating the RUN key. Space in the Main Store is then allocated as follows:-

B1 – 111.1	Autocode Scheme
B111.2 onwards	Labels (see below)
B124.0 – 127.3	Indices ($n0$ to $n27$, one location each)
B127.4 – 299.7	Variables ($v0$ to $v1379$, one location each)
B300 onwards	Instructions (one block each)

There is room for 210 instructions if the date and serial number stored in B511.6 and 511.7 are not to be disturbed. Each instruction occupies one block except for the last instruction of a bracketed interlude which takes two blocks.

The number of locations used for labels is one more than the highest number label used. For instance if the highest number label is 24 then the 25 locations B111.2 to 114.2 are used.

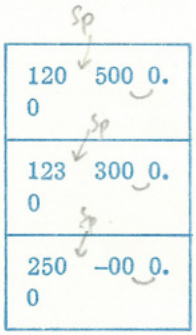
During input of the Autocode B126.0 to 128.6 are used temporarily for two interludes and a list of parameters.

If handswitch 1 is down a 77 stop in 2.6 occurs near the end of the tape which gives the programmer an opportunity to change the numbers of indices, variables and instructions available. The following tape puts the index $n0$ in B120.5, $v0$ in B123.3 and the first instruction in B250:

---

* If using a computer fitted with the larger magnetic drum, the available storage space is 896 blocks (7168 words). Standard allocation of space is as shown above and so there is space for 594 instructions if the date and serial number (now in B895.6 and 895.7) are not to be disturbed. There is no other difference.

T128.0



J126.2
A3
Z

There is then no need to read the remaining portion of the Autocode tape. The variables must begin in or before B127.7 and there must be room for at least one index (stored before $v0$).

It is possible to use the Assembly facilities with the Autocode. Master programme and subroutines may be read in by going to run after the 77 stop, taking care to set the transfer address to leave room for the labels used. A1 must not appear on this tape. The tape must end with

J126.2
A3
Z

Alternatively programme, including directive A1, may be read in before R 600. The Autocode tape must be read from after the A1 directive with which it begins. The Autocode will then occupy B0+ - B109+.1 and B1. An interlude ensures that the storage used for labels begins in B109+.2.

3.16 ERROR TRACING

Each instruction occupies one block in the Main Store. The number of this block is held in 2_B while the instruction is being obeyed, except sometimes during input and output. The rest of X2 is not used. Hence, if a stop occurs while programme is being obeyed it is quite easy to trace the instruction concerned.

The following loop stops may occur:

0.0 663	SQRT argument negative
0.0+ 760	Floating-point division by zero. Infinite result in SEC, CSC, COT, TAN.
0.1 760	Division by zeros, Indices.
0.2+ 463	ARCCOS, ARCSIN argument outside range $-1 \leqslant x \leqslant 1$
0.3+ 660	LOG argument zero.
0.4+ 663	LOG argument negative.
1.4 060	Floating-point overflow.

A write with overflow stop is caused if the result of any index instruction overflows.

The blocks of programme are transferred into U5 before being obeyed. Hence a punch on block transfers while the programme is being obeyed will show the whole course of the programme by giving the block numbers of the instructions obeyed (interspersed with some other lower block numbers).

It may sometimes be useful to put an optional stop in an instruction: to obtain, for example a punch on block transfers after part of the programme has been obeyed.

Depress key for P. on BT

Outputted in binary

The following is an example of how a programme tape might end in order to achieve this:

```
(→I0)
S351.0
L
(→0)
```

Such an optional stop, which should be in location 0 of the block concerned, will occur in U5.0 after the variables or indices involved have been read and unpacked but before the rest of the instruction has been obeyed. Note that instruction number n is in B300 + n.

If an optional stop is inserted in B44+.5 of R 600 a stop will occur between instructions as the programme is obeyed. The number of the next instruction to be obeyed will be found in 2_B.

3.17 ACCURACY OF FUNCTIONS

The function instructions give a *maximum* error of 1 in the eighth significant decimal place (the maximum error is actually 2^{-29} in an argument between ¼ and ½) with the following exceptions:

LOG with argument near 1. The answer which is near 0 is correct to eight decimal places (not 8 significant figures).

SIN, COS, TAN, CSC, SEC and COT are evaluated by dividing the argument by 2π and disregarding the integral part. The accuracy falls off as the argument increases. Argument 11000 gives six decimal figures correct. These functions give the full number of significant figures for small arguments.

3.18 ORGANISATION OF FUNCTIONAL SUBROUTINES

The subroutines performing the function instructions occupy storage as follows:

SIN, COS, TAN, CSC, SEC, COT	B77+ – 83+
LOG	B84+ – 86+
SQRT (also used by ARCCOS, ARCSIN)	B87+ – 88+.3
FRAC	B88+.4 – 89+
EXP, EXPM,	B90+ – 93+.3
ARCCOS, ARCSIN, ARCTAN	B93+.4 – 101+

A dictionary of functions starts in B104+.6, each entry occupying two words. The first word is an integer calculated as follows: let the function be $\lambda_r\lambda_{r-1}\ldots\lambda_0$. The integer is

$$32^r(31-\lambda_r) + 32^{r-1}(31-\lambda_{r-1}) + \ldots + (31-\lambda_0)$$

where λ denotes the numerical value of the letter. The second word in the entry is the cue to a subroutine to perform the calculation.

Consider, for example, the LOG function. L is the 12th letter of the alphabet, O the 15th, G the 7th

$$\therefore \text{LOG} = 32^2(31-12) + 32(31-15) + (31-7) = 19992$$

The entry in the dictionary for LOG would therefore be

+ 19992	function number
84+ 0 72	cue to LOG subroutine
0.3 0 60	

The layout of the Autocode tape is as follows:

```
A1
Programme in B1
Parameters and interludes
in B126.0 - 128.6
L
Programme headed by cue list and name
J127.0  enters an interlude which sets the
        beginning of the list of labels at
        the current Transfer Address
L
J126.0  examines handswitches
J126.2  calculates further parameters from
        those supplied on Autocode tape or
        by programmer.
A3
Z
```

The entries in the dictionary are in the order MOD, TAPE, TAPEB, EXP, EXPM, LOG, SQRT, INT, FRAC, COS, SEC, COT, SIN, CSC, TAN, ARCCOS, ARCSIN and ARCTAN. Alterations made to this list should be made on the Initial Orders tape before the directive J127.0. The word in B128.3 must be altered to contain the address of the beginning of the dictionary in the modifier position and the number of entries in the counter position. The sign digit must be present in this word. The dictionary must begin in an even address.

Any subroutine written to extend the list of functions must satisfy the following conditions: U5.0 - 5.5 and X2 must not be disturbed. Any Computing Store blocks used must be restored, U0 from B41+, U1 from B42+, ..., U4 from B45+. On entry the argument is unpacked with (256 + exponent) in X4 and numerical part in X6 (in the case of an index it is in 6_m on entry). The result should be left unpacked in the same form. Exit is by a jump to U5.1.

The Transfer Address at the J127.0 must be the address at which the list of labels is to begin. J127.0 enters an interlude which sets the beginning of the list of labels at the current Transfer Address.

3.19 Table of Instructions

Arithmetic Instructions

$v1 = v2$	$v1 = -v2$	$n1 = n2$	$n1 = -n2$
$v1 = v2 + v3$	$v1 = -v2 + v3$	$n1 = n2 + n3$	$n1 = -n2 + n3$
$v1 = v2 - v3$	$v1 = -v2 - v3$	$n1 = n2 - n3$	$n1 = -n2 - n3$
$v1 = v2 \times v3$	$v1 = -v2 \times v3$	$n1 = n2 \times n3$	$n1 = -n2 \times n3$
$v1 = v2 / v3$	$v1 = -v2 / v3$	$n1 = n2 / n3$	$n1 = -n2 / n3$

$n1 = n2 * n3$ (remainder of $n2/n3$)
$n1 = -n2 * n3$ (minus remainder of $n2/n3$)

$n1 = v2$	$n1 = -v2$	(nearest integer)
$v1 = n2$	$v1 = -n2$	
$v1 = n2 / n3$	$v1 = -n2 / n3$	

Functions

$v1 =$ MOD $v2$	$v1 =$ -MOD $v2$	$n1 =$ MOD $n2$	$n1 =$ -MOD $n2$
$v1 =$ INT $v2$	$v1 =$ -INT $v2$		
$v1 =$ FRAC $v2$	$v1 =$ -FRAC $v2$		
$v1 =$ SQRT $v2$	$v1 =$ -SQRT $v2$		
$v1 =$ SIN $v2$	$v1 =$ -SIN $v2$		
$v1 =$ COS $v2$	$v1 =$ -COS $v2$		
$v1 =$ TAN $v2$	$v1 =$ -TAN $v2$		
$v1 =$ CSC $v2$	$v1 =$ -CSC $v2$		
$v1 =$ SEC $v2$	$v1 =$ -SEC $v2$		
$v1 =$ COT $v2$	$v1 =$ -COT $v2$		
$v1 =$ ARCSIN $v2$	$v1 =$ -ARCSIN $v2$		
$v1 =$ ARCCOS $v2$	$v1 =$ -ARCCOS $v2$		
$v1 =$ ARCTAN $v2$	$v1 =$ -ARCTAN $v2$		
$v1 =$ LOG $v2$	$v1 =$ -LOG $v2$		
$v1 =$ EXP $v2$	$v1 =$ -EXP $v2$		
$v1 =$ EXPM $v2$	$v1 =$ -EXPM $v2$		

Notes: INT $v2 \leqslant v2$, FRAC $v2 \geqslant 0$

Input Instructions

$v1 =$ TAPE	read one number
$v1 =$ TAPEB	
$n1 =$ TAPE	read one integer
$n1 =$ TAPEB	
$v1 =$ TAPE $n2$	read $n2$ numbers
$v1 =$ TAPEB $n2$	
$n2 =$ TAPE $n1$	read $n1$ integers
$n2 =$ TAPEB $n1$	
$v1 =$ TAPE *	read numbers up to L
$v1 =$ TAPEB *	
$n1 =$ TAPE *	read integers up to L
$n1 =$ TAPEB *	

Notes:
TAPE implies main tape reader
TAPEB implies second tape reader
$n0$ = number of numbers read

Jump Instructions

→1 (unconditional jump)

→1, $\pm v2 \geqslant \pm v3$	→1, $\pm v2 > \pm v3$
→1, $\pm v2 = \pm v3$	→1, $\pm v2 \neq \pm v3$
→1, $\pm n2 \geqslant \pm n3$	→1, $\pm n2 > \pm n3$
→1, $\pm n2 = \pm n3$	→1, $\pm n2 \neq \pm n3$

→1, $\pm v2 = * \pm v3$ (jump if approximately equal)
→1, $\pm v2 \neq * \pm v3$ (jump if not approximately equal)

Output Instructions

PRINT $v1$, $n2$ (print $v1$ in style $n2$)
PRINT $n1$, $n2$ (print $n1$ in style $n2$)
XP before an instruction (obey instruction and print result on new line)
SP before an instruction (as XP but print on same line)
X before an instruction (print CR LF after obeying instruction)
S before an instruction (print Sp after obeying instruction)

Note: XP and SP printing is suppressed if H0 = 1

Printing Styles

Style number $= 1000a + 20b + c$
For variables b = number of digits before the decimal point
c = number of digits after the decimal point
Indices are always printed as 4 digit integers

	Floating-point (Variables only)	Fixed-point
Print on a new line (CR LF ϕ before number)	$a = 1$	$a = 3$
Print on same line (Sp before number)	$a = 2$	$a = 4$

Miscellaneous Instructions

STOP (77 stop)
TAPE (read more instructions to end of programme)
TAPE $n1$ (read more instructions to $n1$ onwards)
TAPE $n1$, $n2$ (read more instructions to replace instruction $n2$ after $n1$)
→M $n1$ (exit to machine orders at decimal address $n1$)
→M $n1$, $v2$ (as →M $n1$ and leave $v2$ unpacked in X7 and X5)
→M $n1$, $n2$ (as →M $n1$ and leave $n2$ in 7_m)
→L (obey link set in X1 on entry from machine orders)
→I0 (call in Initial Orders as a subroutine)
→I0, $n1$ (as →I0 and set T.A. = $n1$)
→I0, $n1$,$n2$ (as →I0, $n1$ and set Relativiser = $n2$)

Entry to Autocode J1.0
Re-entry J1.2
Entry from machine orders – cue 01 with number of label in X2

Table 8 Autocode Instructions

Ferranti Ltd.,
London Computer Centre,
21 Portland Place,
London, W.1.

List CS 217A
April 1959
G.E.F. B.C.
H.McG.R. W.F.M.P.

Printed in England by Specialised Printing Services Ltd., 30-34 Langham Street, LONDON W.1.