The Pegasus Autocode

by B. Clarke and G. E. Felton

Summary: A simplified method of preparing certain programs for the Ferranti Pegasus Computer is described from the point of view of the user, who need have no knowledge of ordinary programming. The programming techniques used to make the computer accept this kind of program are also discussed, together with the reasons for choosing them.

INTRODUCTION

Experience with the Ferranti Mark I Computer at Manchester University showed that it was possible to reduce considerably the time needed to prepare certain kinds of program by using a conversion program known as the Mark I Autocode (Brooker, 1956, 1958). It was consequently decided to prepare a similar program for the Ferranti Pegasus Computer. As far as the differences between the two machines allowed, the main features of the Mark I Autocode have been retained and, while there are some omissions, a number of useful additional facilities have been provided. The scheme is particularly well suited to occasional users with relatively small scientific or technical problems to be solved and who do not wish to go to the trouble of learning the normal programming methods; often only a day or less is needed to learn the simplified technique used with the Autocode, and programs can then be easily and quickly written down. Use of the scheme entails a reduction in the effective speed of computation, but this is frequently acceptable because of the very great reduction in programming time.

We give first a description of the scheme from the point of view of the user, and then describe some of the techniques used to make it realizable.

A BRIEF DESCRIPTION OF THE AUTOCODE

The calculation which is to be performed must first be broken down into a sequence of simple steps, each of which is written down as an Autocode *instruction* (or *order*); the sequence of instructions forms the *program* of the calculation. When this program has been prepared it is then typed out, more or less as written, on a teleprinter or a keyboard perforator. This produces a length of punched paper tape, called the *program tape*, which can be read into the computer by the Autocode routine. At this stage the instructions making up the program are simply converted into a suitable form and stored inside the computer; they are not obeyed. At the end of the tape are some special symbols which cause the machine to start obeying the program it has just read in, i.e. to start the calculation.

Numbers of two kinds are handled by Autocode instructions:

(*a*) *Variables* denoted by $v0, v1, v2, \ldots, v1379$. These are chiefly the numbers to be computed, intermediate quantities or data. They may be of virtually any size (up to about 10^{76} in magnitude) and are dealt with to a precision of about 8 or 9 significant figures.

(*b*) *Indices* denoted by $n0, n1, n2, \ldots, n27$. These are signed integers (up to 8191 in magnitude) which are intended mainly for counting and for use as suffixes for the variables (see below).

CALCULATING INSTRUCTIONS

Most of the 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

$$v1 = v2 + v3$$

is an instruction to replace $v1$ by the sum of $v2$ and $v3$. Numbers can be written instead of variables on the right-hand side, thus the instruction

$$v9 = v9 - 1{\cdot}46$$

causes the value of $v9$ to be reduced by 1·46. As an example of a sequence of instructions, suppose we have to evaluate

$$(1 + 0{\cdot}32v2)/(1{\cdot}68 + v3)$$

from previously calculated values of $v2$ and $v3$ and put the result in $v0$. The following instructions can be used ($v1$ is used as working space):

$$v0 = 0{\cdot}32 \times v2$$
$$v0 = 1 + v0$$
$$v1 = 1{\cdot}68 + v3$$
$$v0 = v0/v1$$

This sequence of instructions could form part of a complete program.

Notice how the process has been broken down into steps involving only two variables or numbers on the right; some instructions involve only one, for example

$$v8 = v4 \text{ or } v6 = -v2 \text{ or } v6 = 43{\cdot}3$$

There are similar instructions for handling indices, for example

$$n8 = n3 \quad n2 \text{ or } n7 = 95 \quad n0$$

but indices can take only integral values. As a rule indices and variables cannot be mixed in the same instruction, but a few simple instructions of this type have been provided. Certain elementary *functions* can be evaluated by a single instruction; for example

$$v28 = \text{SQRT } v8$$

finds the square root of $v8$ and places it in $v28$.

The permissible instructions of this type are summarized in Table 1, in which $v1$, $v2$, $v3$ and $n1$, $n2$, $n3$ are simply representative variables and indices. In any of these instructions a minus sign may be written after the equals sign, thus $v1 = -v2 + v3$ and $v4 = -\mathrm{LOG}\ v8$ are permissible. Non-negative numbers (or integers) may be put instead of any variable (or index) on the right-hand side.

The results of the calculation can be printed by using output instructions; these usually consist of the word PRINT followed by the variable or index to be printed and an integer (or index) specifying the style desired. To aid in diagnosing errors we may also write XP (or SP) before any calculating instruction to cause printing of the result of the instruction on a new line (or the same line); this kind of output can be suppressed by the use of a key on the control panel of the computer. The available output instructions are summarized in Table 2; the style number may be given as an index or may be written explicitly as an integer in the instruction.

JUMP INSTRUCTIONS

Autocode instructions are usually obeyed in the order in which they are written down. Jump instructions have been provided to break this sequence and so to provide the possibility of selecting alternative courses of action. The instruction to which a jump is made is identified by giving it a *label*; this is a small unsigned integer written in front of the instruction and separated from it by a right bracket. Thus the instruction

$$6)\ v3 = v4 + v2$$

is labelled 6. Any instruction can be labelled. The first instruction in a program is automatically labelled 0: there is no need to write this label in. Typical jump instructions are

$\rightarrow 6$	jump (unconditionally) to the instruction labelled 6,
$\rightarrow 6,\ v1 > v7$	jump to the instruction labelled 6 if $v1 > v7$,
$\rightarrow 6,\ v1 \geq -v7$	jump to the instruction labelled 6 if $v1 \geq -v7$.

The available jump instructions are summarized in Table 3; any combination of the signs shown is permissible. If desired, numbers (or integers) may be written in place of any variable (or index), and expressions such as $-n1$ are allowed.

USE OF SUFFIXES

Indices may be used as suffixes to any variables appearing in any instruction. For example we might use an instruction like this:

$$vn1 = v(2 + n5) - v(-15 + n2)$$

Here the $n1$, $(2 + n5)$ and $(-15 + n2)$ are to be thought of as suffixes and may, if desired, be so written, even though they cannot be printed as such on a teleprinter. Suffixes can be attached only to variables and must take

TABLE 1

CALCULATING INSTRUCTIONS

$v1 = v2$ $v1 = v2 + v3$ $v1 = v2 - v3$ $v1 = v2 \times v3$ $v1 = v2/v3$

$n1 = n2$ $n1 = n2 + n3$ $n1 = n2 - n3$ $n1 = n2 \times n3$ $n1 = n2/n3$

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

$n1 = n2*n3$ (remainder when $n2$ is divided by $n3$)

$v1 = \mathrm{MOD}\ v2$	(modulus)	$v1 = \mathrm{SIN}\ v2$
$n1 = \mathrm{MOD}\ n2$	(modulus)	$v1 = \mathrm{COS}\ v2$
$v1 = \mathrm{SQRT}\ v2$	(square root)	$v1 = \mathrm{TAN}\ v2$
$v1 = \mathrm{INT}\ v2$	(integral part)	$v1 = \mathrm{CSC}\ v2$
$v1 = \mathrm{FRAC}\ v2$	(fractional part)	$v1 = \mathrm{SEC}\ v2$
$v1 = \mathrm{LOG}\ v2$	(natural logarithm)	$v1 = \mathrm{COT}\ v2$
$v1 = \mathrm{EXP}\ v2$		$v1 = \mathrm{ARCSIN}\ v2$
$v1 = \mathrm{EXPM}\ v2$	(negative exponential)	$v1 = \mathrm{ARCCOS}\ v2$
		$v1 = \mathrm{ARCTAN}\ v2$

Note: There are analogous instructions with a minus sign on the right-hand side.

TABLE 2

OUTPUT INSTRUCTIONS

PRINT $v1, n2$ — Print $v1$ in a style determined by $n2$. Thus if $n2 = 1{,}000a + 20b + c$, then print b figures before the decimal point and c after.
- If $a = 1$ print in floating decimal form on a new line.
- If $a = 2$ print in floating decimal form on same line.
- If $a = 3$ print in fixed-point form on a new line.
- If $a = 4$ print in fixed-point form on same line.

PRINT $n1, n2$ — If $a = 3$ print $n1$ on new line, or if $a = 4$ on same line.

XP before a calculating instruction. Print the result of the instruction on a new line, unless suppressed.

SP as XP but print on the same line.

X (or S) before a calculating instruction. Print carriage return and line feed (or space). Used to lay results out neatly.

TABLE 3

JUMP INSTRUCTIONS

$\rightarrow 1$ (unconditional jump)

$\rightarrow 1,\ \pm v2 \geq \pm v3$	$\rightarrow 1,\ \pm n2 \geq \pm n3$
$\rightarrow 1,\ \pm v2 > \pm v3$	$\rightarrow 1,\ \pm n2 > \pm n3$
$\rightarrow 1,\ \pm v2 = \pm v3$	$\rightarrow 1,\ \pm n2 = \pm n3$
$\rightarrow 1,\ \pm v2 \neq \pm v3$	$\rightarrow 1,\ \pm n2 \neq \pm n3$

$\rightarrow 1,\ \pm v2 =* \pm v3$ (jump if approximately equal, i.e. if the two variables agree to $n0$ significant binary digits, or say $0{\cdot}3 \times n0$ significant decimal figures).

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

one of the three forms indicated in the instruction above.

By using suffixes like these together with jump instructions we can often take advantage of repetitive features of a calculation. For example to evaluate and print

$$(v5 \times v55) + (v6 \times v56) + \ldots + (v24 \times v74)$$

we can use the following instructions:

```
    n1 = 5
    v0 = 0
3)  v1 = vn1 × v(50 − n1)
    v0 = v0 + v1
    n1 = n1 + 1
    → 3, n1 ≠ 25
    PRINT v0, 3043
    STOP
```

The last instruction causes the computer to wait until a certain key is operated on the control panel.

INPUT ORGANIZATION

Normally the complete program is punched on paper tape and is fed into the store of the computer before any of its instructions are obeyed. The punching of the program tape is quite straightforward; each instruction is typed out on a teleprinter (or a keyboard perforator) and is terminated by the special symbols carriage return and line feed. The teleprinter is normally on figure shift and can then be used to type out most of the instructions (the symbols *v* and *n* are available in this shift); occasional changes to letter shift are necessary, for example, to type out words such as LOG. The printing produced when the teleprinter keyboard is operated can be checked against the original program.

At the end of the program tape we must punch some special instructions to cause the Autocode scheme to stop reading in the tape and start obeying the program; these instructions are written in brackets. Any group of instructions included within brackets is obeyed, starting with the first, as soon as the whole group has been read in; if, therefore, we punch some instructions such as the following

```
(v2 = 6·4284
 → 0)
```

at the end of the program tape, the value of *v*2 is set and a jump occurs to the start of the program (which is always automatically labelled 0). Such a group of instructions is called a bracketed interlude. If the last instruction of a bracketed interlude is obeyed, and does not cause a jump, the Autocode input is re-entered, and more instructions are read and stored in the space which was occupied by the instructions of the interlude. Thus the instructions of the bracketed interlude do not form part of the stored program.

When the program is being obeyed it can cause further numbers to be read in by using an input instruction; for example,

$$v6 = \text{TAPE}$$

causes a number to be read in from the punched paper tape and put in *v*6. Several numbers can be read in at a time, if desired; the permissible input instructions are indicated in Table 4.

TABLE 4

INPUT INSTRUCTIONS

*v*1 = TAPE	Read in one number and set in *v*1.
*v*1 = TAPE *n*2	Read in *n*2 numbers and set in *v*1, *v*2, *v*3, . . .
*v*1 = TAPE*	Read in numbers up to L on tape, set in *v*1, *v*2, . . .

Notes:

1. These read from the main tape reader; to use the second tape reader write TAPEB instead of TAPE.
2. Integers can be read in and placed in indices by instructions such as *n*1 = TAPE, etc.
3. Input ceases and the next instruction is obeyed if L is read by *any* input instruction; also *n*0 is *always* set equal to the number of numbers read in.
4. The instruction *v*1 = TAPE 3 causes 3 numbers to be read into *v*1, *v*2, *v*3.

There are two tape readers, either of which can be used. We can alternatively read in further instructions by writing

TAPE

The new instructions are then added at the end of the program. There are facilities for leaving the Autocode program and causing the computer to start obeying ordinary machine orders, and also for returning to or calling in the Autocode from machine orders.

The time to read in an Autocode program tape is about $\frac{1}{2}$ second per instruction. The instructions are obeyed at the rate of about 15 to 20 per second.

PEGASUS AND THE AUTOCODE

Pegasus is a medium-sized binary computer (Elliott, Owen, Devonald and Maudsley, 1956). It has a two-level store, consisting of:

(*a*) The *computing store* (or working store), containing principally 7 accumulators and 48 ordinary registers grouped into 6 blocks of 8 registers each. To all of these there is immediate access.

(*b*) The *main store* (or backing store), of 7,168 storage locations (4,096 in earlier machines) together with isolated storage for the Initial Orders and the engineers' test programs. This store is a magnetic drum with a revolution time of 16 milliseconds.

Each word consists of 39 bits and represents a number or a pair of orders (or instructions): the latter can best be thought of as single-address orders, although any accumulator can normally be specified. There are facilities for modification of the address in any order by a modifier in a selected accumulator. Orders are obeyed from the ordinary registers in the computing store and chiefly affect numbers in the accumulators; simple orders take about 0·3 msec, multiplication takes 2 msec and division $5\frac{1}{2}$ msec. The main store is divided into 896 blocks (512 on earlier machines), each of 8 words; information can be transferred into the computing store, or vice versa, either by blocks or by single words.

In order to reduce scaling difficulties the Autocode scheme has its variables in floating-point form $A \times 2^a$, with A represented by a 30-bit fraction (including a repeated sign-bit) and $a + 256$ by a 9-bit integer; these two numbers are packed into a single 39-bit word. The arithmetical operations are therefore carried out to a precision of rather more than 8 significant decimal figures, and can deal with numbers up to about 10^{76} in magnitude. The floating-point operations are all programmed, since the computer has fixed-point orders only: this was greatly facilitated in fact by the nature of the order-code. A programmed floating-point operation takes about 8 to 15 msec, including unpacking the operands, packing up the result and testing for overflow.

For ease of use the variables of the Autocode scheme are presented as a continuous series so that, in effect, there is a single-level store; these variables are kept in the main store and are transferred when required to the computing store by single-word transfer orders. When read in, each Autocode instruction is converted into an 8-word block of information in the main store; the first half of this block is made up of orders, and the second half of addresses and parameters. When the program is obeyed five of the blocks of ordinary registers in the computing store are more or less permanently occupied by the "inner loop" of the Autocode: this transfers the "instruction-blocks" into the remaining block of the computing store, selects, transfers and unpacks the operands, carries out the appropriate arithmetical operation and then normalizes, packs and stores the result. Much of this inner loop is made up of subroutines for the floating-point arithmetic; processes not included in the inner loop are effected by subroutines read in from the main store. In obeying most Autocode instructions access is required four times to the main store, involving a total mean waiting time of about 32 msec; this is roughly equal to the time spent in obeying the machine orders, giving an overall speed of about 15 to 20 Autocode instructions per second.

Normally there is room for 28 indices, 1,380 variables and 594 instructions (or 210 in earlier machines), but these numbers can readily be changed by the user should he require a different allocation of the available storage space.

APPLICATIONS

The Autocode is intended mainly for relatively small scientific or technical calculations, especially those in which the results are required quickly. Many of these calculations can be put on to the computer in a very short time, often a few hours only, once the problem has been precisely described. This should be contrasted with the weeks, or even months, which might be required to program the problem in the ordinary way. The fact that the running of the program may take much longer if the Autocode is used may not matter, since it is often quite brief in any event. There are some problems in which the cost of the extra machine time is saved by the reduction in programming costs; many "one-off" problems are of this kind, the program being used only once after it has been written. Even if the program is to be used several times there are advantages in having the originator prepare the program himself, which may be impracticable if he has to learn ordinary programming methods first. In this way many people throughout an organization can be using the computer directly on their own calculations, and a general computer consciousness can spread. In some Autocode programs it may be advantageous to re-write the "inner loops" in ordinary machine language, and hence to achieve nearly the same speed as if the Autocode had not been used.

An important application, in business as well as technical calculations, is that in which an Autocode program is written as a preliminary to the preparation of a machine-language program. The Autocode version can be used to test proposed methods and flow-diagrams, and can very readily be changed in the light of results: it can also be used to provide a few sets of accurate results and intermediate values, which can be very useful in later testing of the machine program; it may also provide indications of how scaling should be done if the ultimate program is to work with fixed-point numbers. This application is specially useful in exploratory work where it is difficult to see what sort of processes should be used in the final program.

ACKNOWLEDGEMENTS

The authors wish to acknowledge their indebtedness to Mr. R. A. Brooker of Manchester University and to their colleagues, with whom they have had many stimulating discussions; they wish also to thank the Directors of Ferranti Ltd. for permission to publish this paper and to thank Mr. B. B. Swann for his continued support and encouragement.

REFERENCES

Elliott, W. S., Owen, C. E., Devonald, C. H., and Maudsley, B. G. (1956). "The Design Philosophy of Pegasus, a Quantity Production Computer," *Proc. I.E.E.*, Vol. 103, Part B, Supplement No. 2, p. 188.

Brooker, R. A. (1956). "The Programming Strategy used with the Manchester University Mark 1 Computer," *Proc. I.E.E.*, Vol. 103, Part B, Supplement No. 1, p. 151.

Brooker, R. A. (1958). "The Autocode Programs developed for the Manchester University Computers," *The Computer Journal*, Vol. 1, p. 15 (1958).