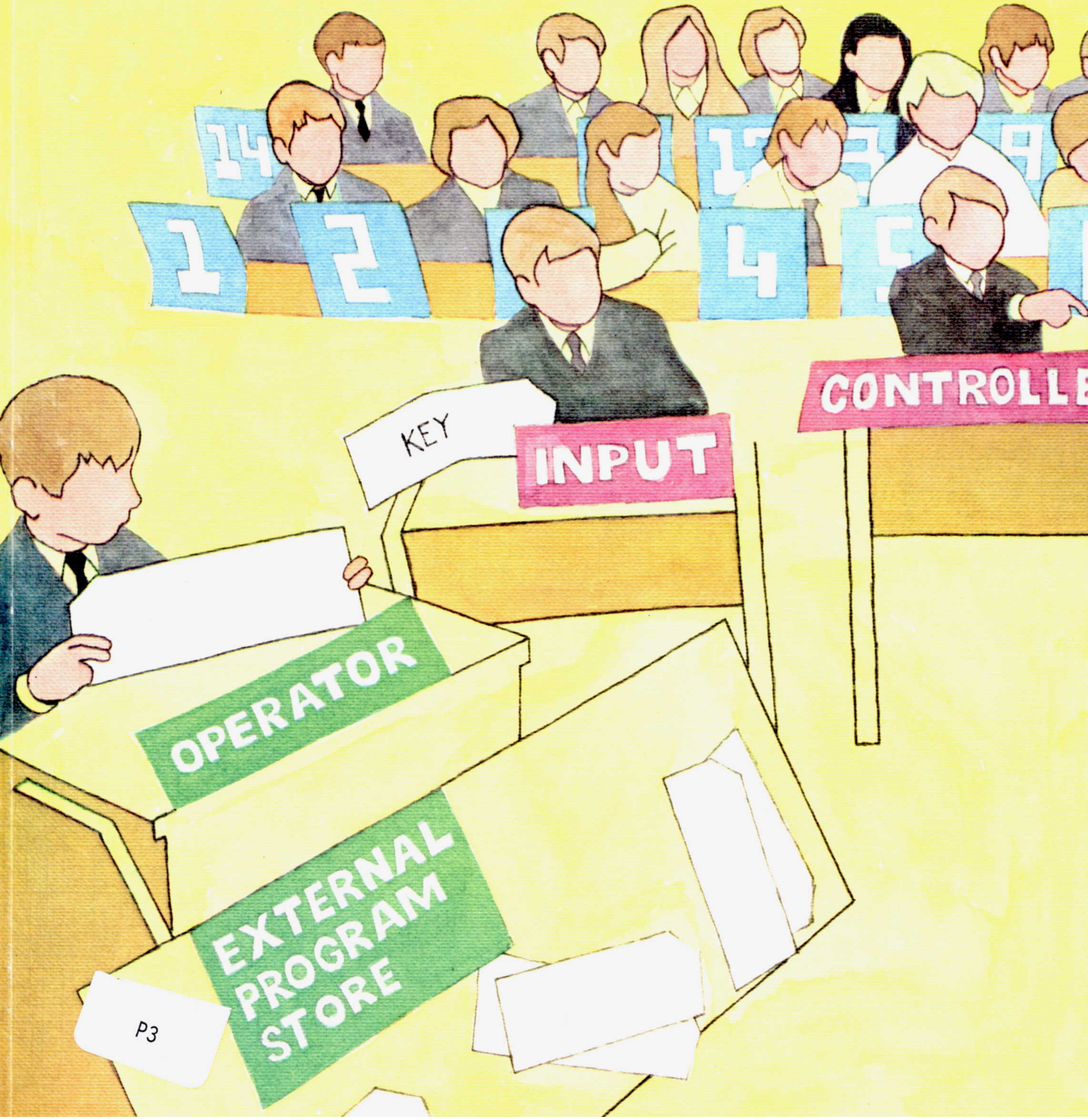
Computers and young children



STORES
21
13
7
8
9
11
12
WIRES
12
PRINT OUT
9
14
ARITH. UNIT
OUTPUT

Computers and young children

W & R CHAMBERS Edinburgh
JOHN MURRAY London
JOHN WILEY New York

Published for the
Nuffield Foundation by
W & R Chambers, Edinburgh;
John Murray, London; and
John Wiley and Sons Inc., New York

ISBN 0 550 77023 2 (Chambers)
ISBN 0 7195 2653 1 (Murray)
ISBN 0 471 65179-6 (Wiley)

Library of Congress
Catalog Card No: 75-39285

Printed in Great Britain by
Newgate Press Limited
London EC1

ieneral introduction

'he aim of the Nuffield Mathematics Project is to devise a :ontemporary approach for children from 5 to 13'. The uides do not comprise an entirely new syllabus. The stress s on *how to learn,* not on what to teach. Running through ll the work is the central notion that the children must be et free to make their own discoveries and think for hemselves, and so achieve understanding, instead of earning off mysterious drills. In this way the whole attitude o the subject can be changed and 'Ugh, no, I didn't like naths' will be heard no more.

o achieve understanding young children cannot go straight o abstractions – they need to handle things ('apparatus' is oo grand a word for at least some of the equipment oncerned – conkers, beads, scales, globes, and so on).

ut 'setting the children free' does not mean starting a riot vith a roomful of junk for ammunition. The changeover to he new approach brings its own problems. The guide *I do, nd I understand* (which is of a different character from the thers) faces these problems and attempts to show how they an be overcome.

he other books fall into three categories: Teachers' Guides, Veaving Guides and Check-up Guides. The Teachers' iuides cover three main topics: ● Computation and tructure, ▼ Shape and Size, ■ Graphs Leading to Algebra. n the course of these guides the development of nathematics is seen as a spiral. The same concept is met ver and over again and illustrated in a different way at very stage. The books do not cover years, or indeed any pecific time; they simply develop themes and therefore how the teacher how to allow one child to progress at a ifferent pace from another. They contain direct teaching uggestions, examples of apparently un-mathematical ubjects and situations which can be used to develop a nathematical sense, examples of children's work, and uggestions for class discussions and out-of-school ctivities. The Weaving Guides are single-concept books vhich give detailed instructions or information about a articular subject.

he third category of books, as the name implies, provides :heck-ups' on the children's progress. The traditional tests re difficult to administer in the new atmosphere of ndividual discovery and so our intention has been to replace these by individual check-ups for individual children. These have been prepared by a team from the Institut des Sciences de l'Education in Geneva under the general supervision of Piaget. These check-ups, together with more general commentary, are published in the same format as the other guides and they form an integral part of the scheme.

While the books are a vital part of the Nuffield Mathematics Project, they should not be looked on as guides to the only 'right' way to teach mathematics. We feel very strongly that development from the work in the guides is more important than the guides themselves. They were written against the background of teachers' centres where ideas put forward in the books could be discussed, elaborated and modified. We hope very much that they will continue to be used in this way. A teacher by himself may find it difficult to use them without the reassurance and encouragement which come from discussion with others. Centres for discussion do already exist and we hope that many more will be set up.

The children's work that has been reproduced in these books, like the books themselves, is not supposed to be taken as a model of perfection. Some of it indeed contains errors. It should be looked upon as an example of work that children *might* produce rather than a model of work that they *should* produce.

Foreword to the Nuffield Mathematics Project

The last few years have been exciting ones for teachers of mathematics; and for those of us who are amateurs in the subject but have a taste for it which was not wholly dulled by the old methods that are so often stigmatised, there has been abundant interest in seeing the new mathematical approach develop into one of the finest elements in the movement towards new curricula.

This is a crucial subject; and, since a child's first years of work at it may powerfully affect his attitude to more advanced mathematics, the age range 5 to 13 is one which needs special attention. The Trustees of the Nuffield Foundation were glad in 1964 to build on the forward-looking ideas of many people and to set up the Nuffield Mathematics Project; they were also fortunate to secure Professor Geoffrey Matthews and other talented and imaginative teachers for the development team. The ideas of this team have helped in the growth of much lively activity, throughout the country, in new mathematical teaching for children: the Schools Council, the Local Education Authority pilot areas, and many individual teachers and administrators have made a vital contribution to this work, and the Trustees are very grateful for so much readiness to co-operate with the Foundation. The fruits of co-operation are in the books that follow; and many a teacher will enter the classroom with a lively enthusiasm for trying out what is proposed in these pages.

Brian Young
Director of the Nuffield Foundation, 1964–70

Contents

Why computers ? 9

Flow charts 11

A classroom human 'computer' 28

Presenting the program to the computer 42

Working with a computer 48

Conclusion 61

Introduction

There is no attempt in this book to give the age for which each particular part of the work is appropriate. Chapter 2 begins with work that is suitable for children at the lower end of the primary school : the end of the book deals with work that is appropriate for pupils at the top end of the primary school and the first years of the secondary school. It must be left to individual teachers to decide just when the time is right to introduce the different stages of the work.

Spreading the activities over a long period and integrating some of the work with other subjects is probably the best way of introducing the new ideas. This is likely to be more satisfactory than a compressed course in which new ideas are introduced in quick succession.

Why computers?

ne computer is probably the most important result of cientific advancement in recent times. Compared with space avel and nuclear energy, the computer is less spectacular ut its effects may well be much greater.

any people have acquired a limited idea of the capabilities a computer in certain fields but few are aware of the likely nsequences of its full effect on our civilisation.

part from its effect on the organisation of clerical and dustrial work, the computer is a very powerful tool which, in sociation with the general adaptability of the human brain, as greatly accelerated research, discovery and invention. The mputer, even in its present state of development, has helped an to make progress in his work; part of the resulting owledge has led to the creation of a more advanced achine which, in turn, will help man to reach yet further, and create a yet more useful tool. Man and machine together rm a loop, and from this loop will come the offshoots that ill lead to developments and inventions at an increasing te: and no longer will these often remain unused for long riods.

ience and technology have led to an ever-increasing degree specialisation and the accumulation of masses of technical ta. The computer's fast capability for storing, handling and ocessing specialised and complex knowledge will free many eople from the drudgery of this work and enable them to velop their full capabilities in a wider field of research, scovery and creativity.

ne effects of computers on our civilisation will be apparent all levels. The high level policy-makers of industry and mmerce must rely more and more on information produced a computer.

mputer education should not be looked upon as merely ing vocational training. It is true that many people must be lly trained to operate and use these machines; but the plosively expanding effect of computers on so many pects of our lives makes it important that all members of r interdependent society should have some understanding their capabilities. **An early introduction to the inciples on which a computer works would prevent e misconceptions which many people now have. As rt of his general education, every child should ecome aware of the capabilities and limitations of mputers; and of the effects they will have on our ciety.**

There are two different kinds of computer: the analogue and the digital.

The analogue computer uses analogies, in the form of measurements, to carry out computations but the digital computer works directly with digits. The ordinary clock is an example of an analogue device: a measurement of the rotation of the hands being used to indicate the time. The modern type of clock on which the time appears in figures is an example of a digital device. Similarly, 236 ÷ 17 worked out on a slide rule is an example of analogue computation; and an answer achieved by long division, or on an ordinary desk calculator, is the result of a digital procedure.

The digital type is used in most computer installations because, for most applications, it can be more accurate; and most information can be more satisfactorily processed when it is in digital form.

This book is concerned only with the digital type. Much of the work done on computers includes calculations of one form or another, some of these being of such length that they would take the cleverest of men years to complete, or of such complexity that they could never be sorted out by anybody. It is important, however, that the computer should not be thought of as being just a large version of an ordinary calculating machine. It is basically different from an ordinary calculator in that it can store, or 'remember', data and sets of instructions. With its great speed, and this capacity for 'remembering' sets of instructions (called programs), the digital computer can carry out a vast quantity of dull, repetitive work in a very short time.

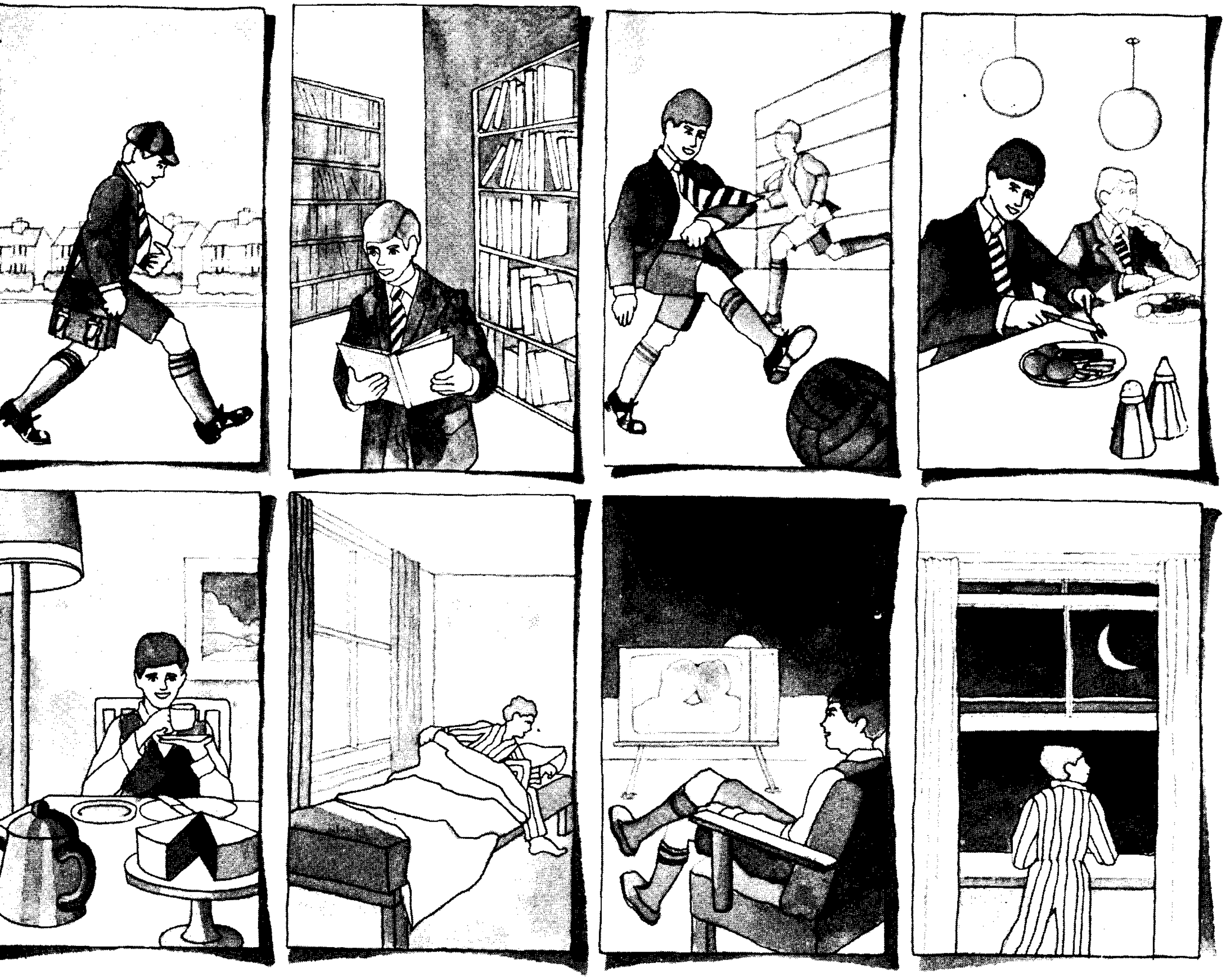
Flow charts

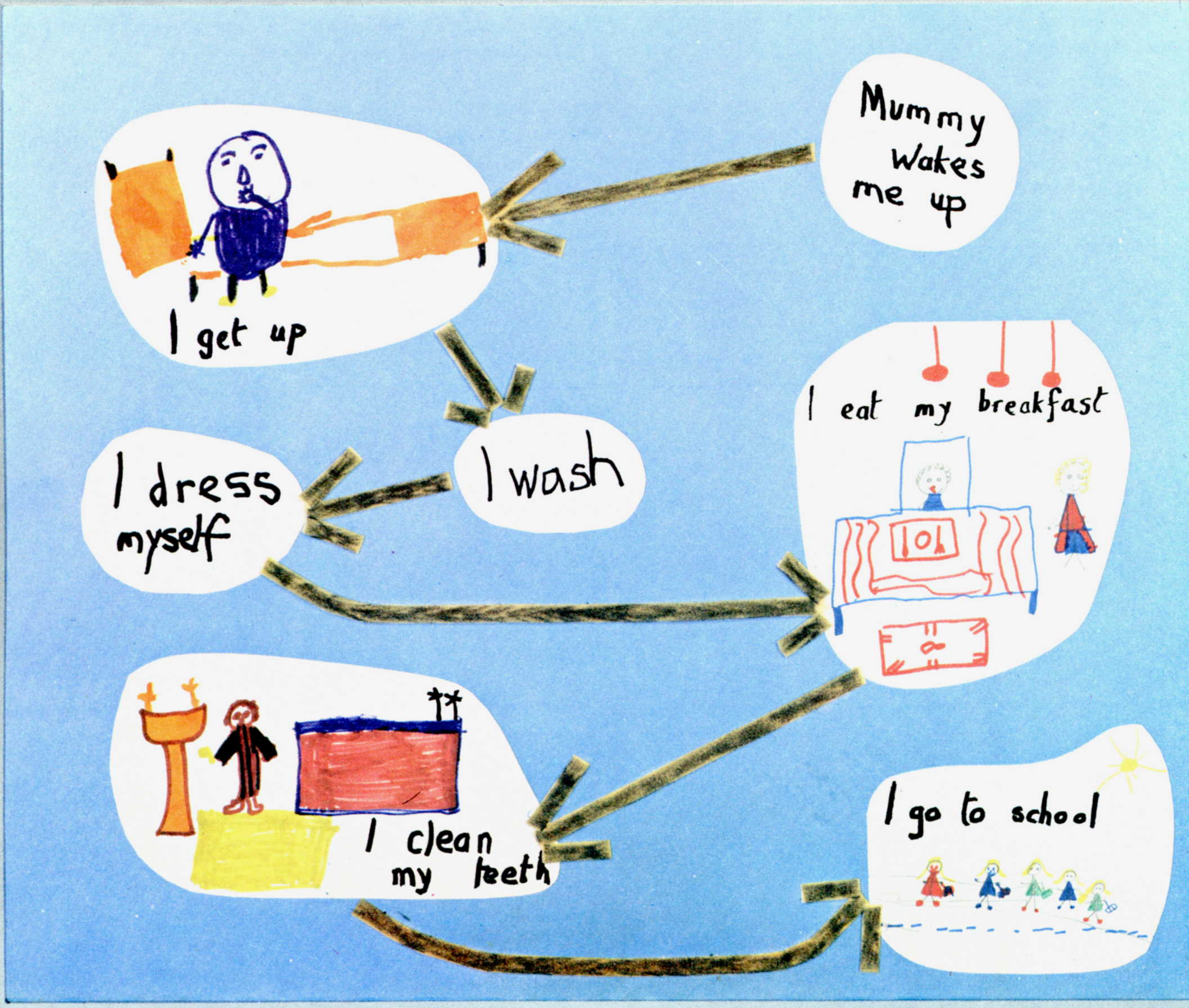
/hen it has been decided that a computer can help in the olution of a particular problem, the first thing to do is to work ut a method of solution and then break this method down to a series of very simple steps. Drawing flow charts is a mple way of showing these steps. The completed program ill be a series of instructions telling the computer how to arry out these steps.

ne preparation of flow charts is a valuable activity in many ıbjects, not just mathematics and computer programming. In e early stages of such work, as suggested in this book, there probably no value in mentioning to the children the ssociation with computer programming.

The flow chart for a computer program must have the steps in the correct order. Early practice with the idea of an ordered procedure can be given by activities in which the children examine the order in which the steps of everyday situations are carried out. The first work of this kind could be prepared in the form of sets of cards which must be arranged in order to give the correct sequence of events.

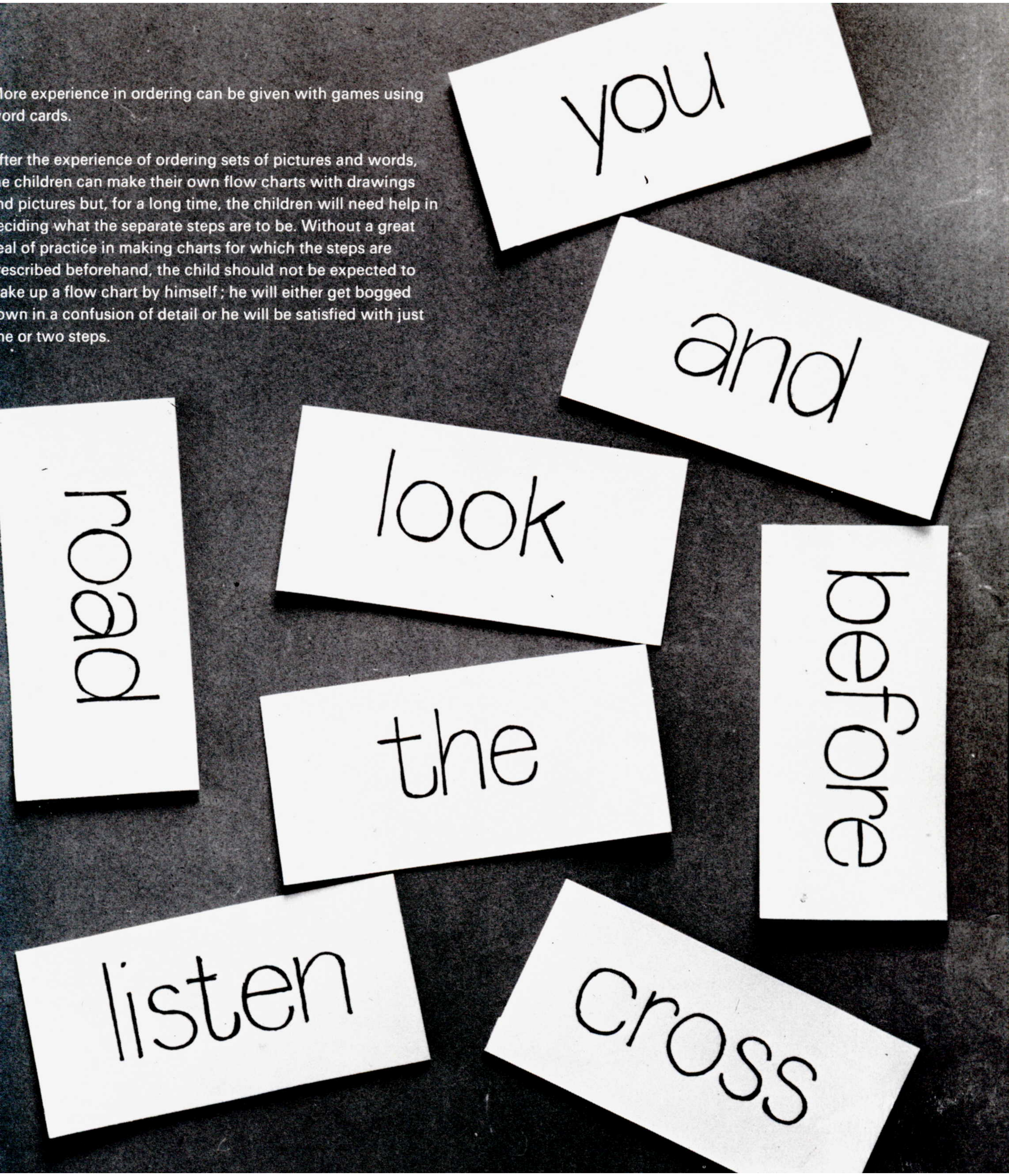
Further practice in preparation for drawing flow charts can be provided by having arrangements of pictures on which the children can pin arrows between the separate pictures to show the correct sequence of events.

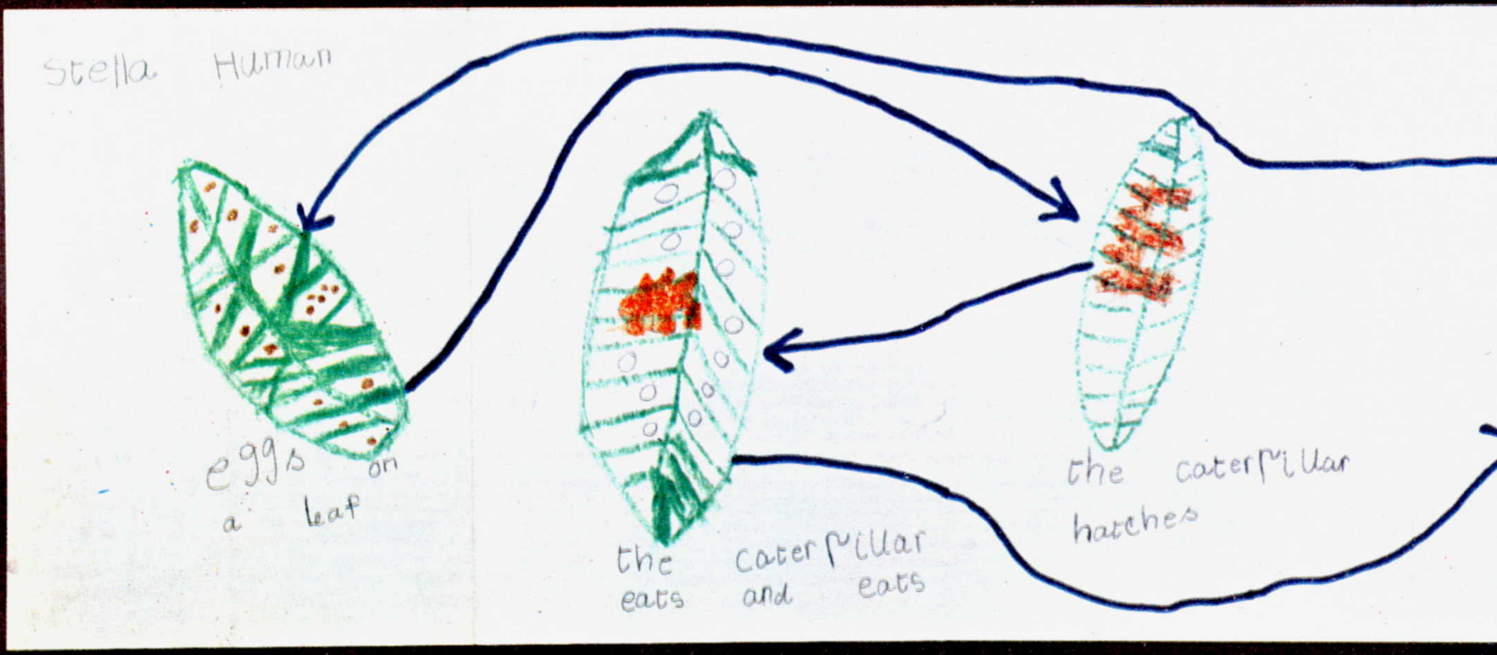


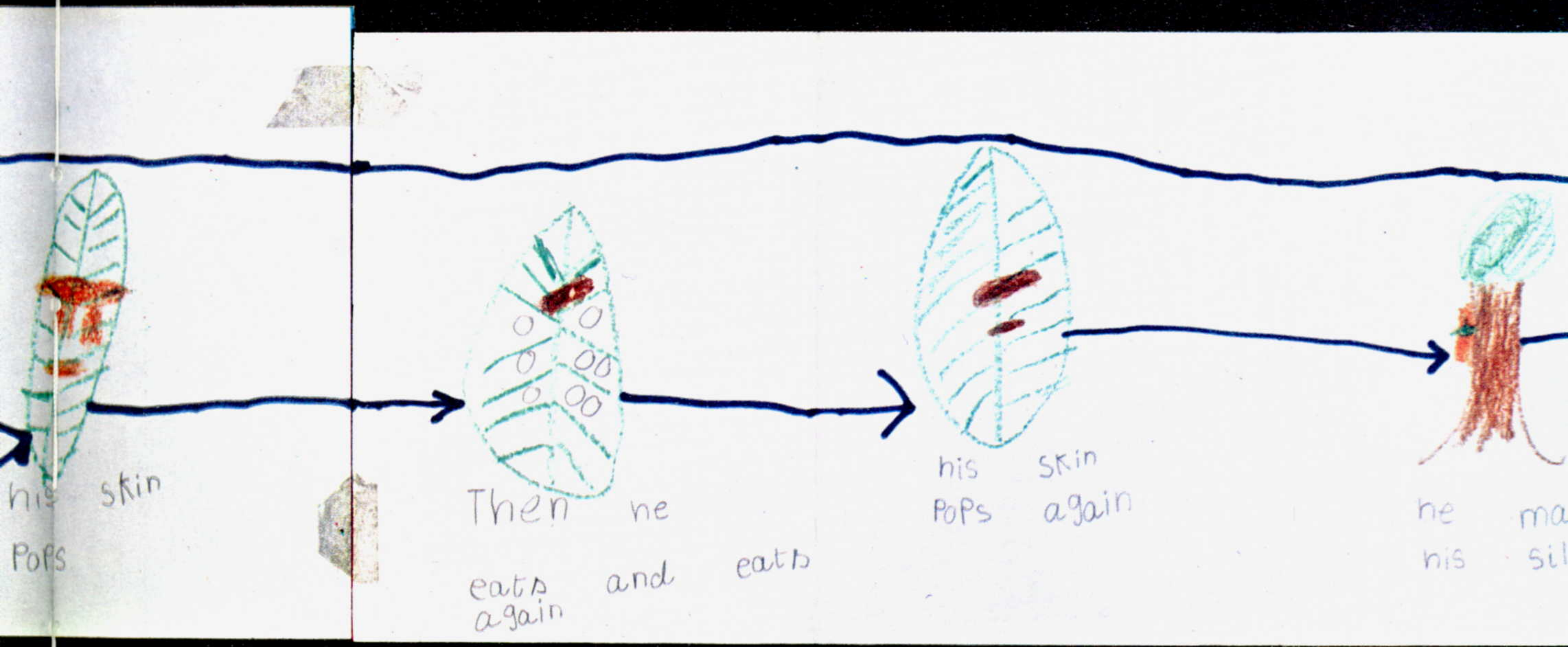

Mummy wakes me up
I get up
I wash
I dress myself
I eat my breakfast
I clean my teeth
I go to school

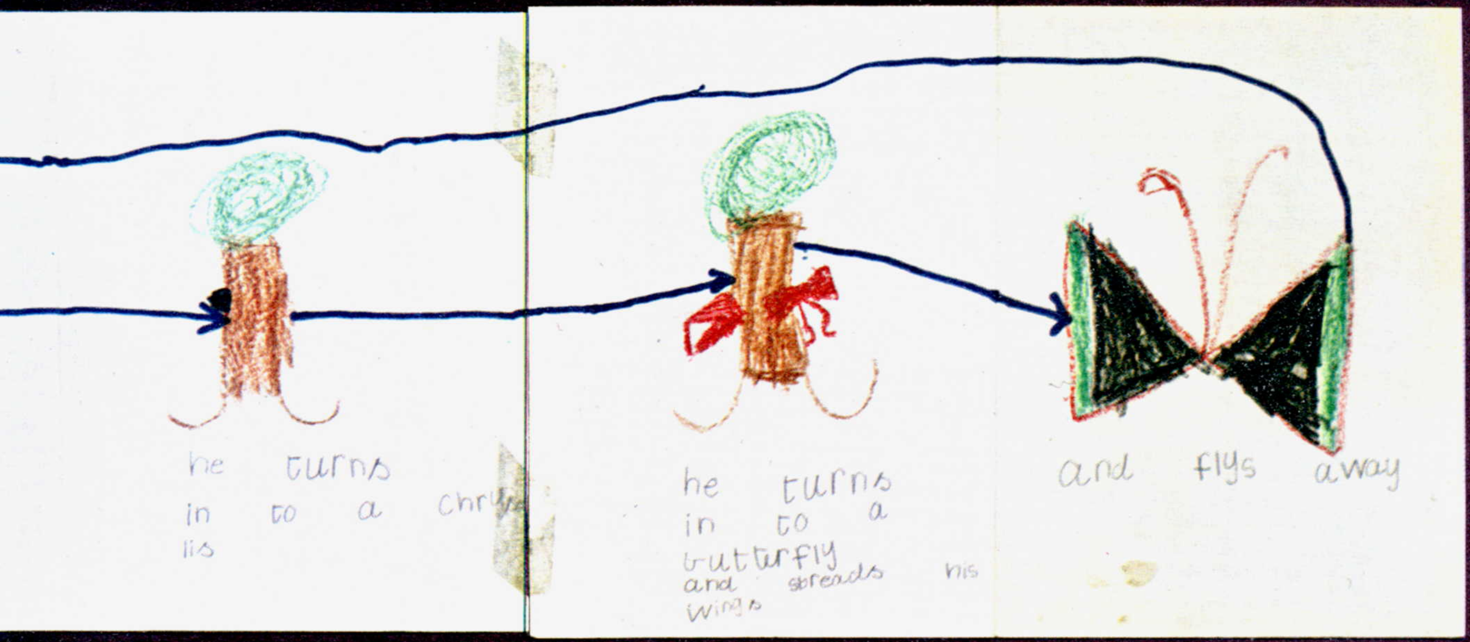
lore experience in ordering can be given with games using ord cards.

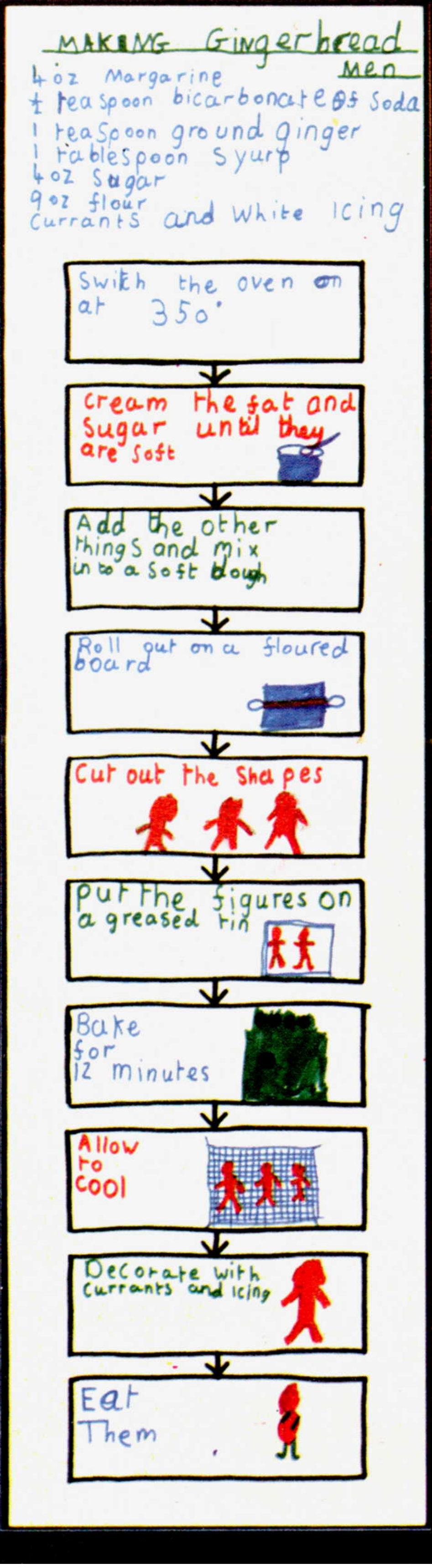
fter the experience of ordering sets of pictures and words, e children can make their own flow charts with drawings nd pictures but, for a long time, the children will need help in eciding what the separate steps are to be. Without a great eal of practice in making charts for which the steps are escribed beforehand, the child should not be expected to ake up a flow chart by himself ; he will either get bogged own in a confusion of detail or he will be satisfied with just ne or two steps.




Stella Human
eggs on a leaf
the caterpillar eats and eats
the caterpillar hatches


his skin
Pops
Then he
eats and eats
again
his skin
Pops again
he ma
his sil


he turns
in to a chry
lis
he turns
in to a
butturfly
and sbreads his
wings
and flys away



MAKING Gingerbread Men

4 oz Margarine
½ teaspoon bicarbonate of Soda
1 teaspoon ground ginger
1 tablespoon Syurp
4 oz Sugar
9 oz flour
Currants and White icing

Swith the oven on at 350°

Cream the fat and sugar until they are soft

Add the other things and mix in to a soft dough

Roll out on a floured board

Cut out the Shapes

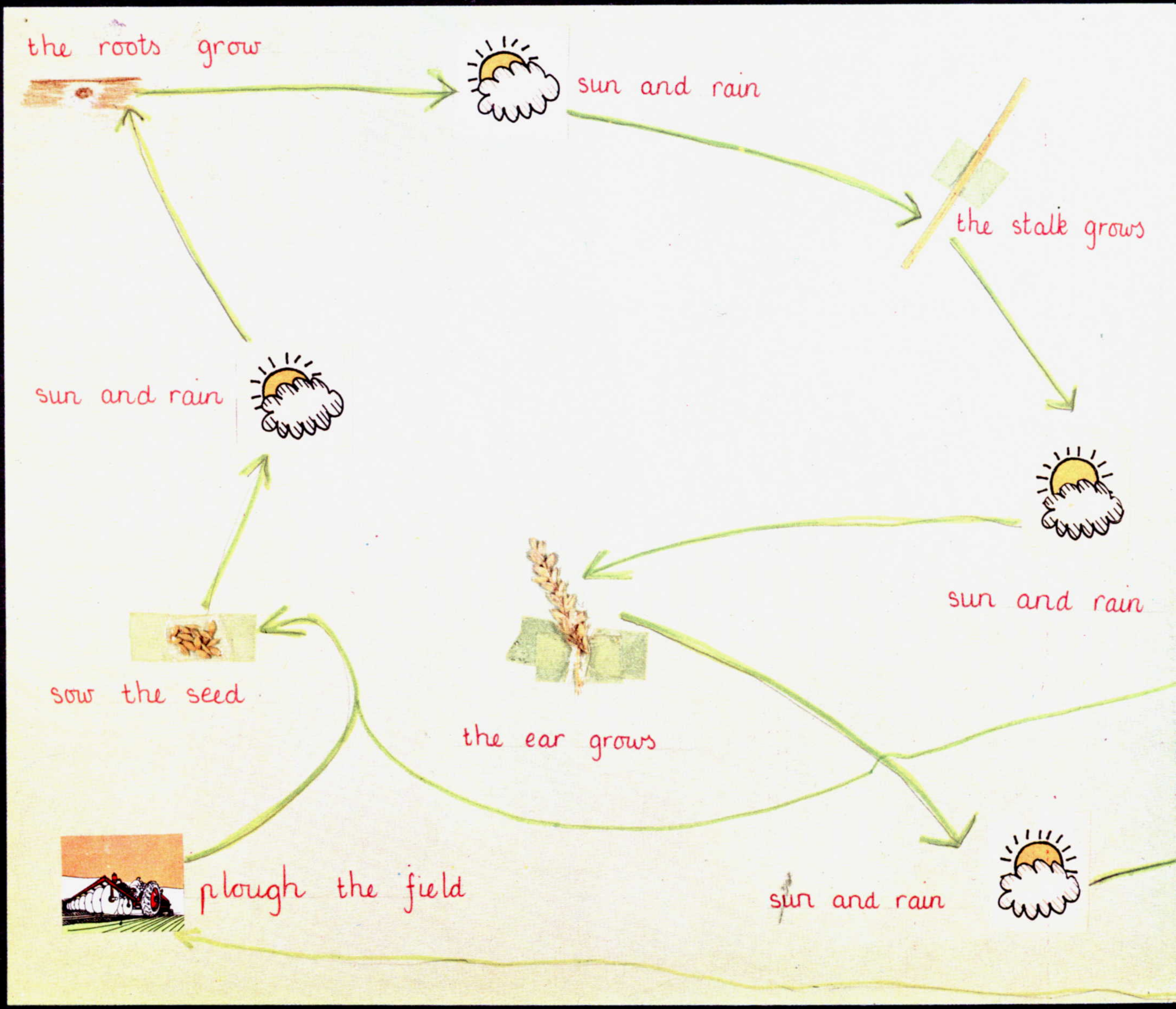
put the figures on a greased tin

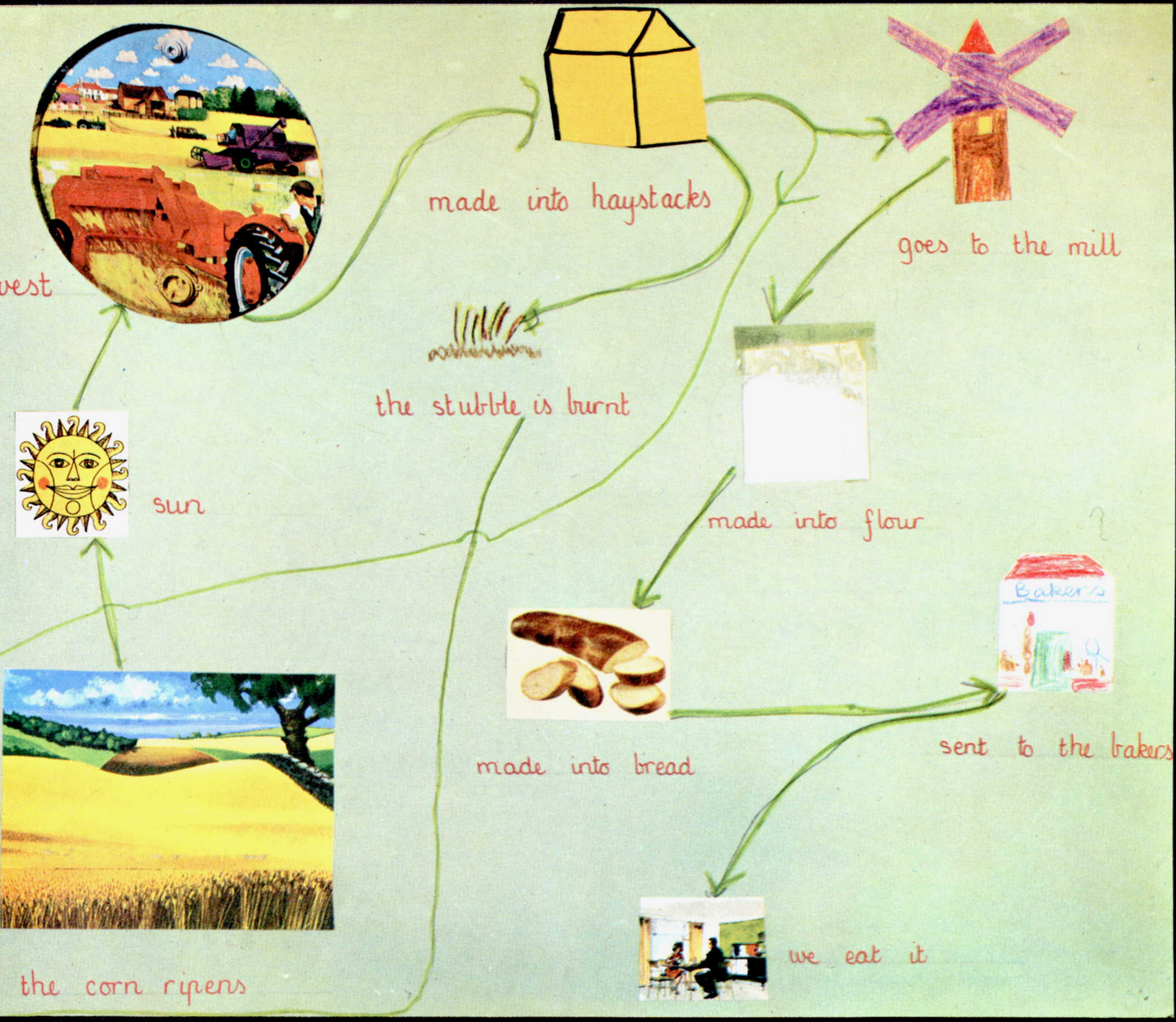
Bake for 12 minutes

Allow to cool

Decorate with currants and icing

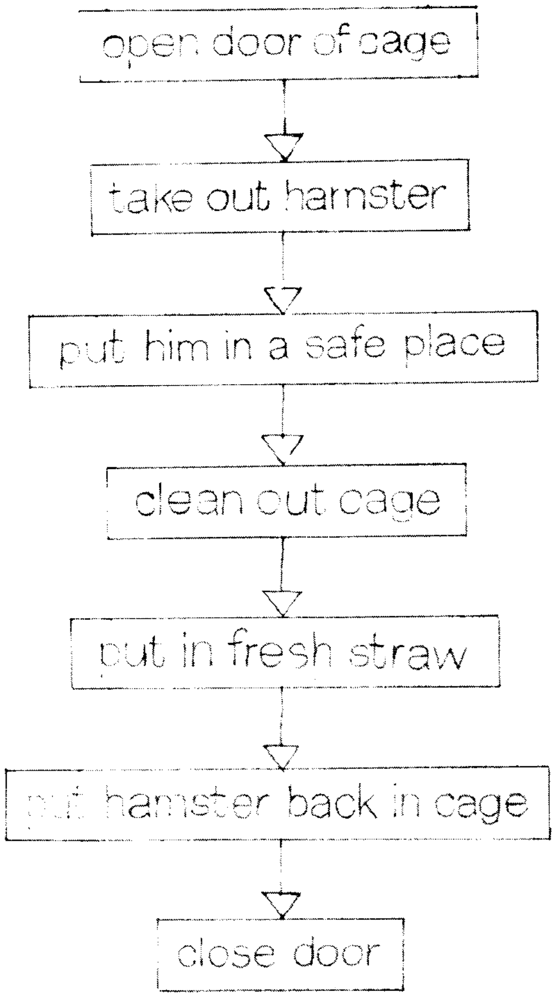
Eat Them


the roots grow
sun and rain
the stalk grows
sun and rain
the ear grows
sun and rain
sow the seed
plough the field
sun and rain

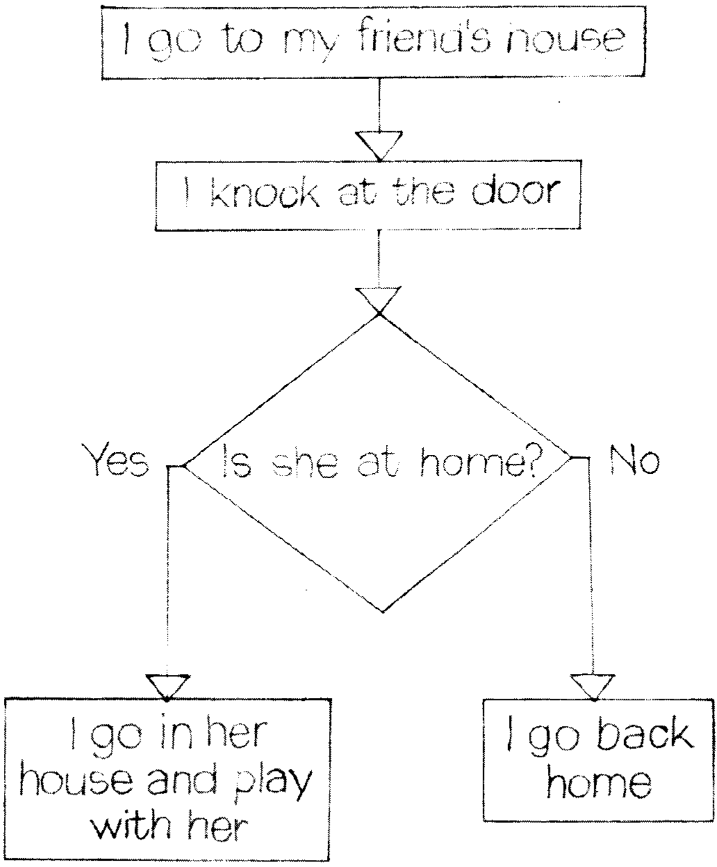

vest
made into haystacks
goes to the mill
the stubble is burnt
sun
made into flour
Bakers
made into bread
sent to the bakers
the corn ripens
we eat it

At a later stage the children can make flow charts in which the steps are described in writing.

Cleaning hamster cage



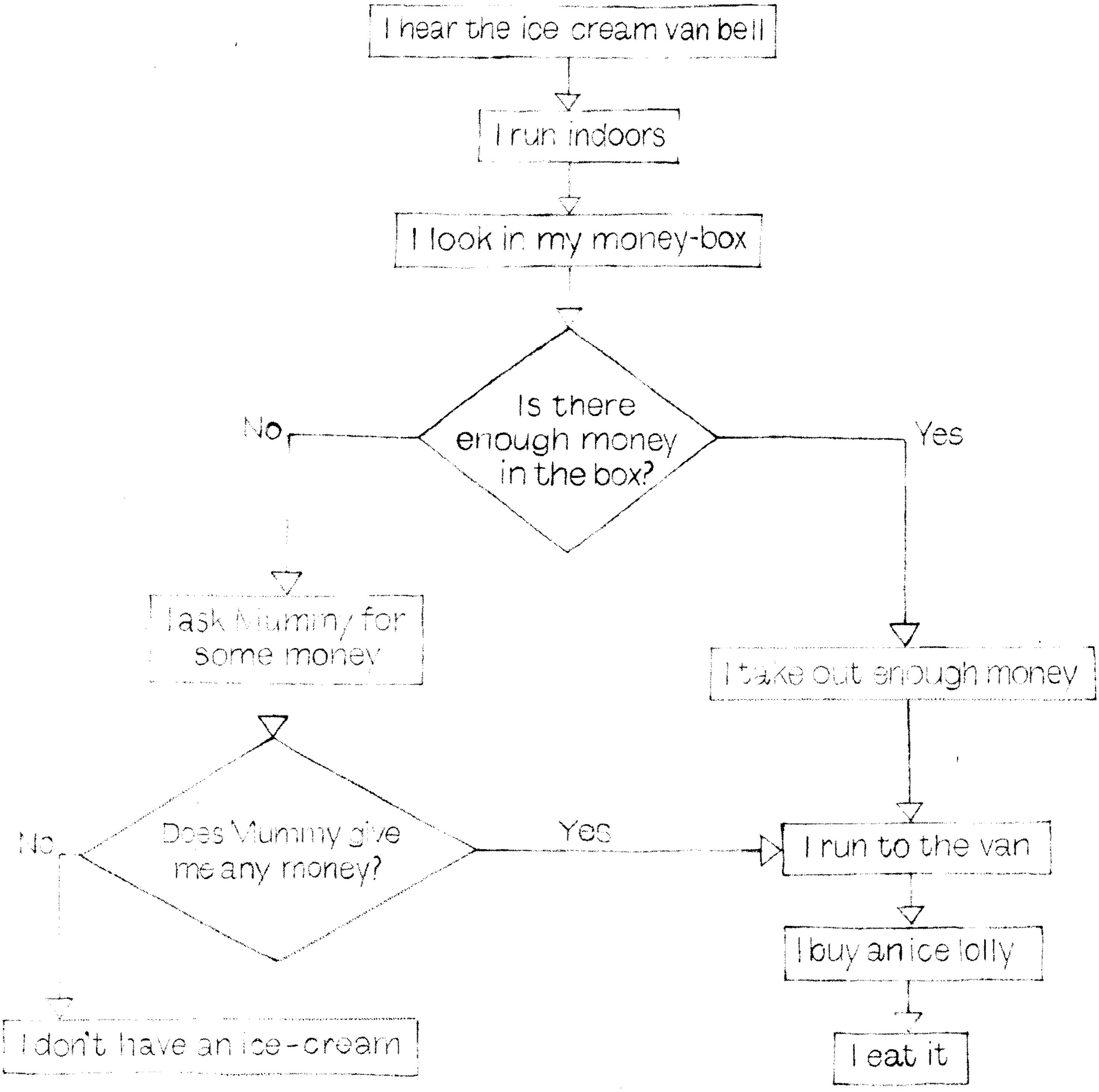
Going to play with my friend


I go to my friend's house
I knock at the door
Is she at home?
Yes
No
I go in her house and play with her
I go back home

Many of the subjects which the children may suggest for flow charts will have more than one possible arrangement for some of the stages: it is most important to discuss this to see which stages can be re-arranged, and which ones must be in a certain order. In later computational programs this will be an important part of the preparation.

Decision stages After the children have had a considerable amount of practice with straightforward flow charts, it should be possible to introduce some examples in which, at some particular stage, the steps branch off in separate directions.

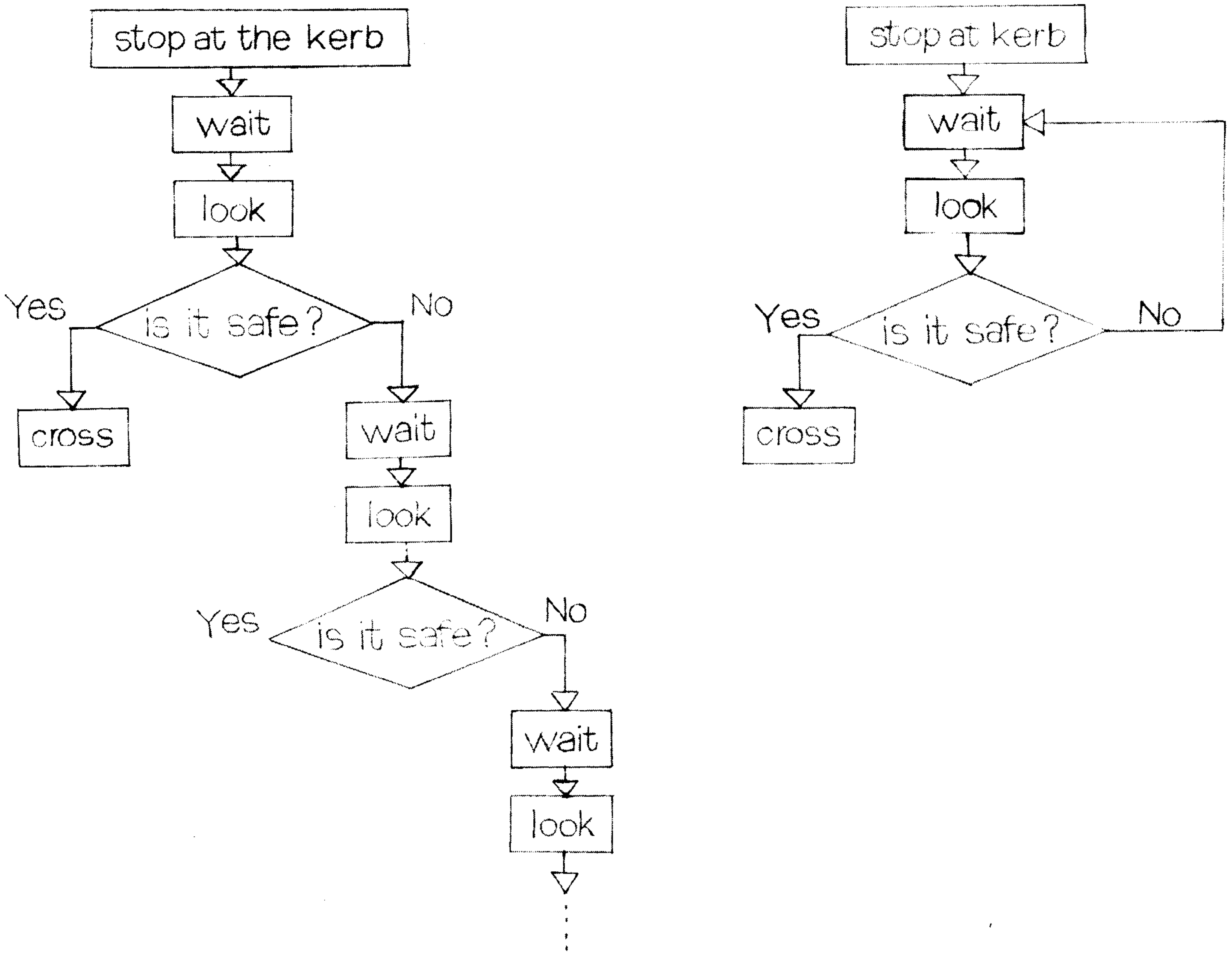
It is usual practice for this branching step to be enclosed in a 'box' of a different shape – a rhombus.


I hear the ice cream van bell
I run indoors
I look in my money-box
Is there enough money in the box?
No
Yes
I ask Mummy for some money
I take out enough money
Does Mummy give me any money?
No
Yes
I run to the van
I buy an ice lolly
I don't have an ice-cream
I eat it

Even if a group of children all start off with the same agreed set of steps, the resulting flow charts for some situations may be very different because some of the charts will have repeat stages incorporated in them. For example: making a chart to describe one particular crossing of a busy road could result in a very long piece of work in which many of the steps would be repetitions of earlier stages.

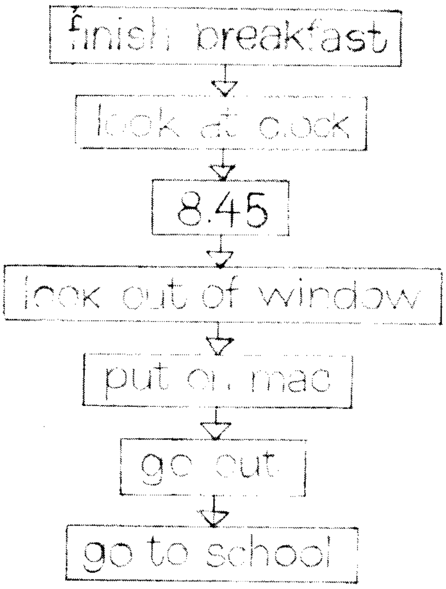
To simplify such examples, and to prepare a chart which will show the **general procedure** for crossing a road, it will be necessary to introduce the idea of a loop. The set of steps included in the loop is repeated as often as necessary.

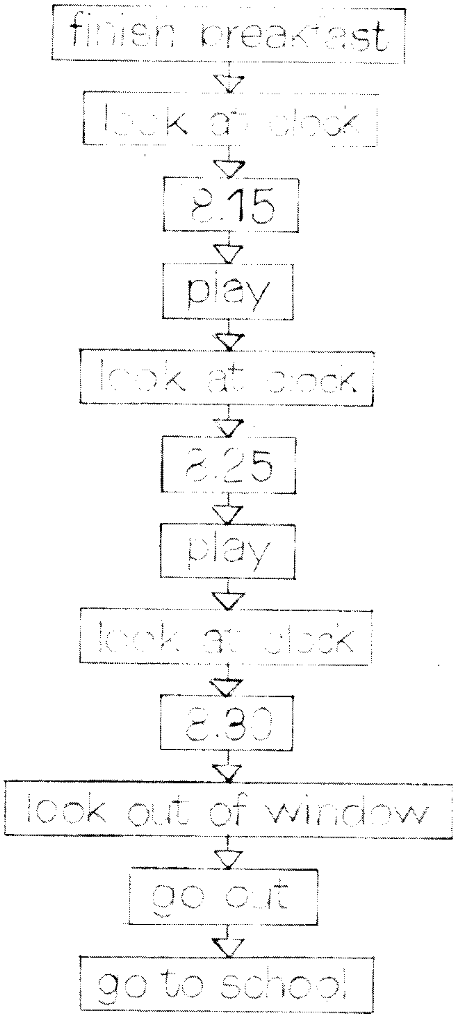
Crossing the road

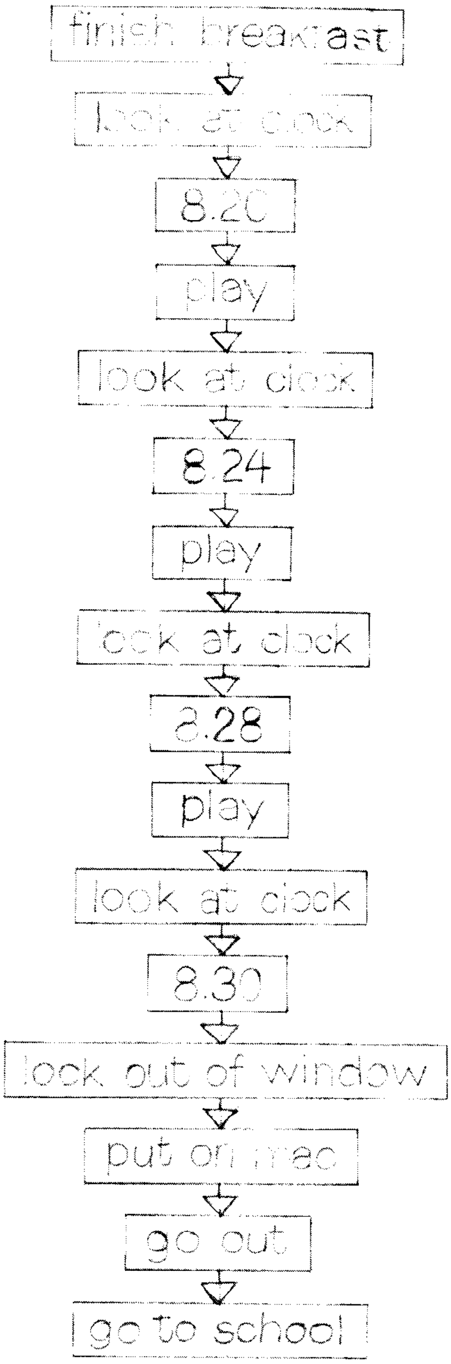


important at this stage to discuss the difference between a v chart describing one particular occurrence and one that ws the **general procedure** for **all** situations. The owing examples show the sequence of events for a child ng to school on three different mornings.

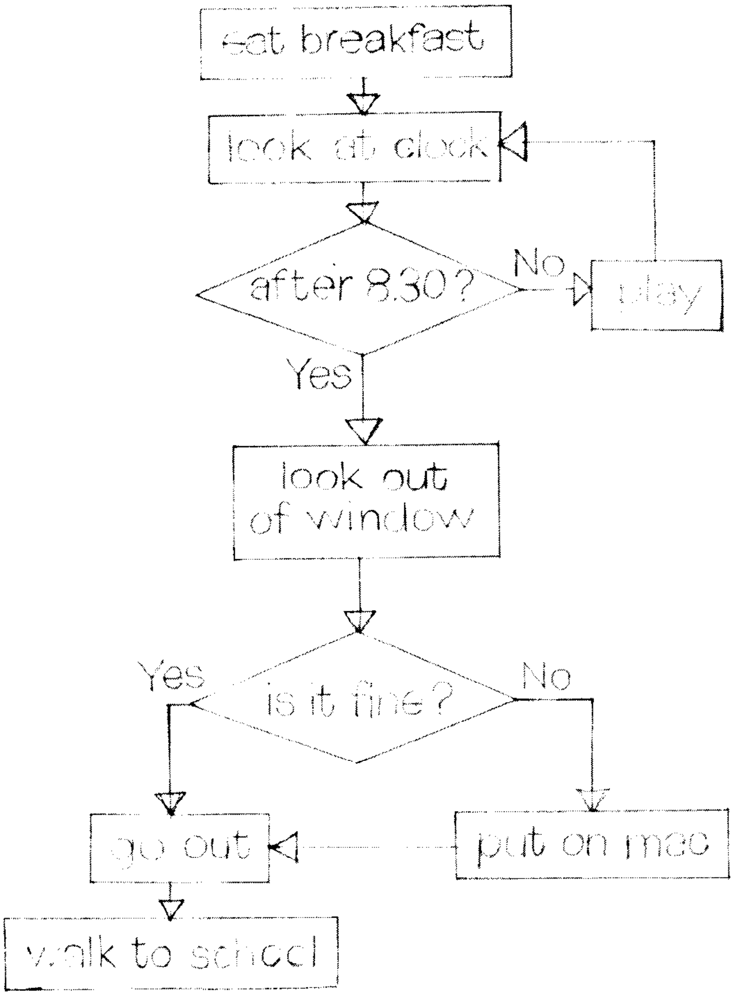
Even with this small set of steps, there could be a great many variations of the procedure for different mornings. By introducing decision stages and loops it is possible to produce a chart that shows the general procedure followed every morning.





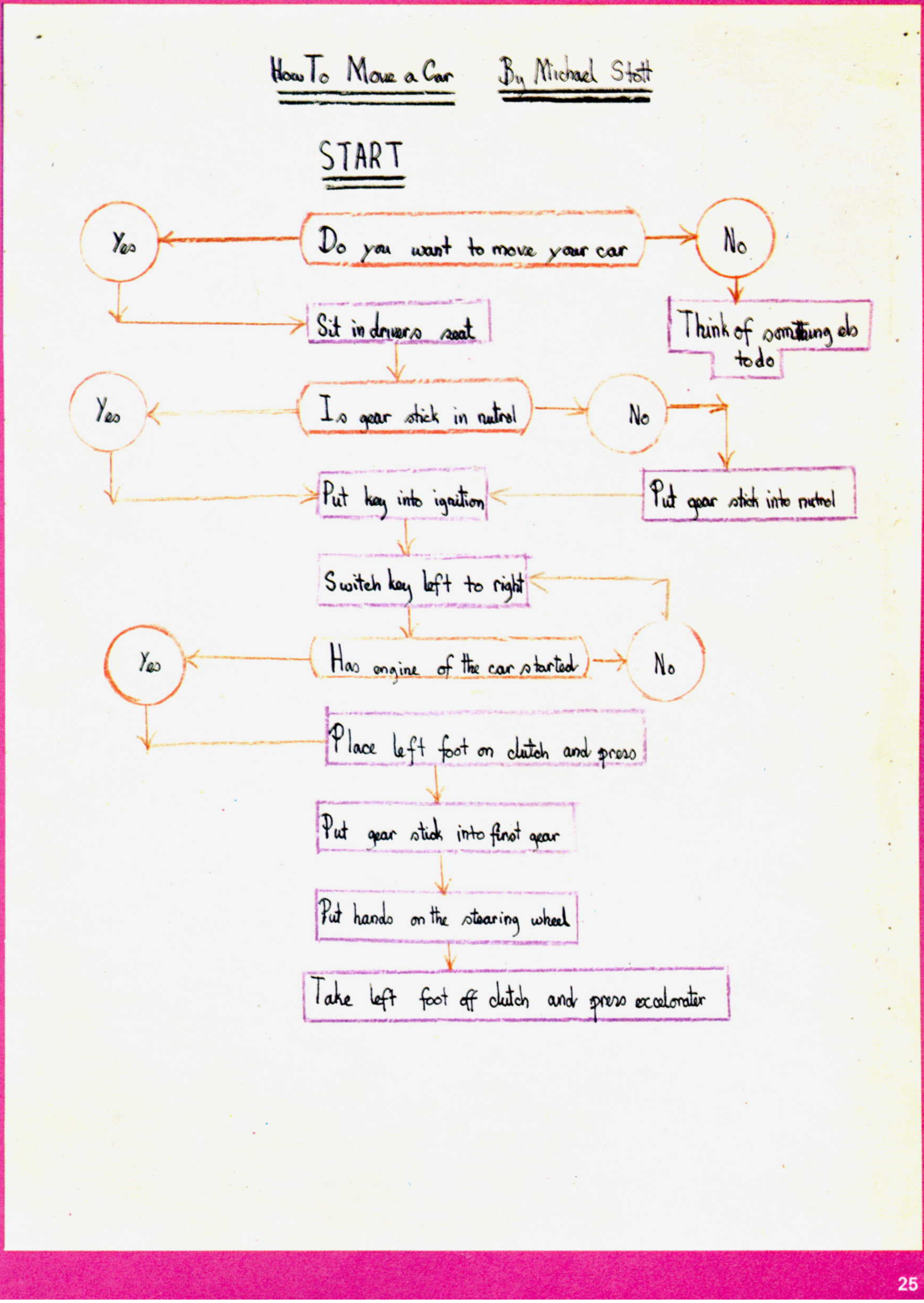


Going to school

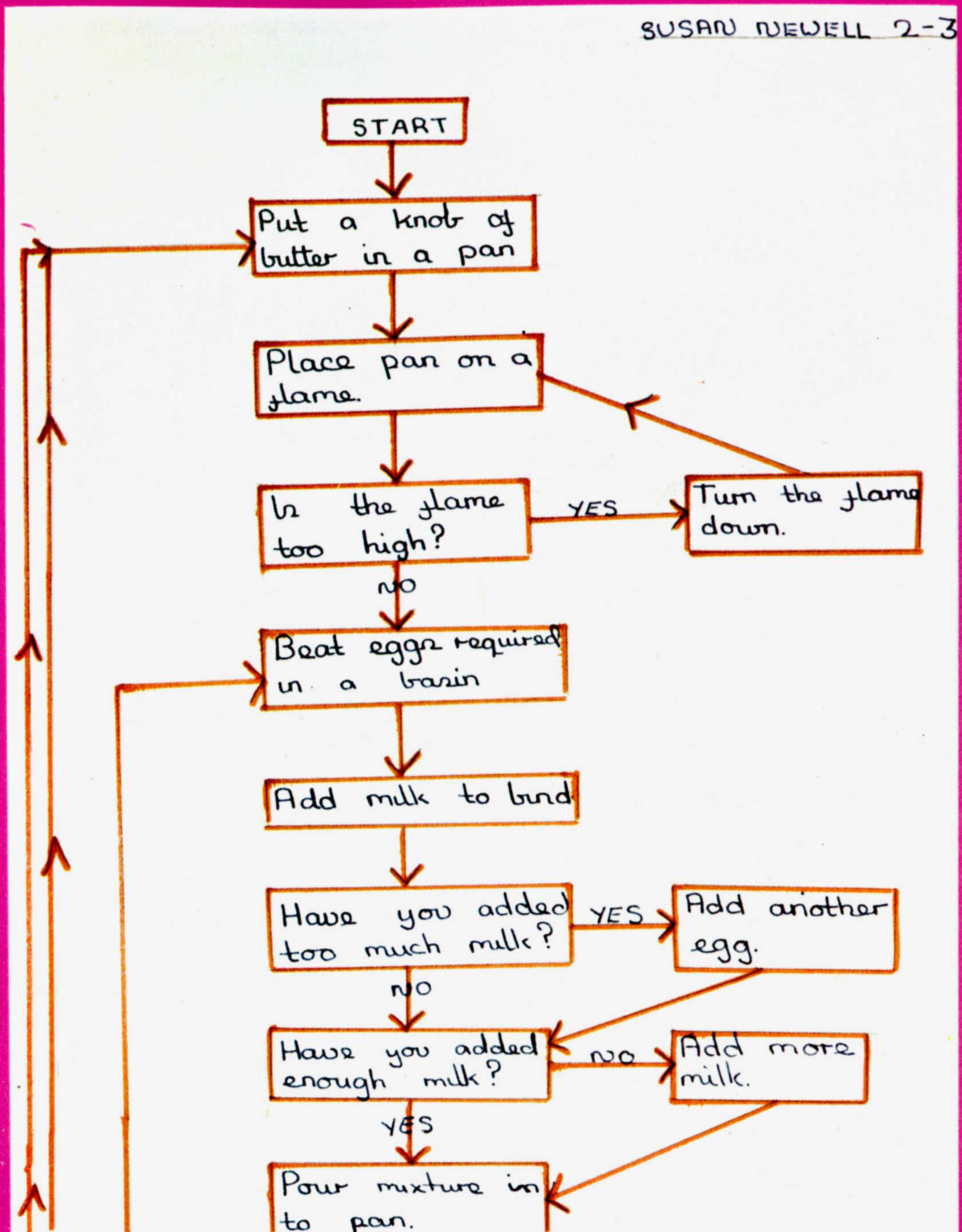


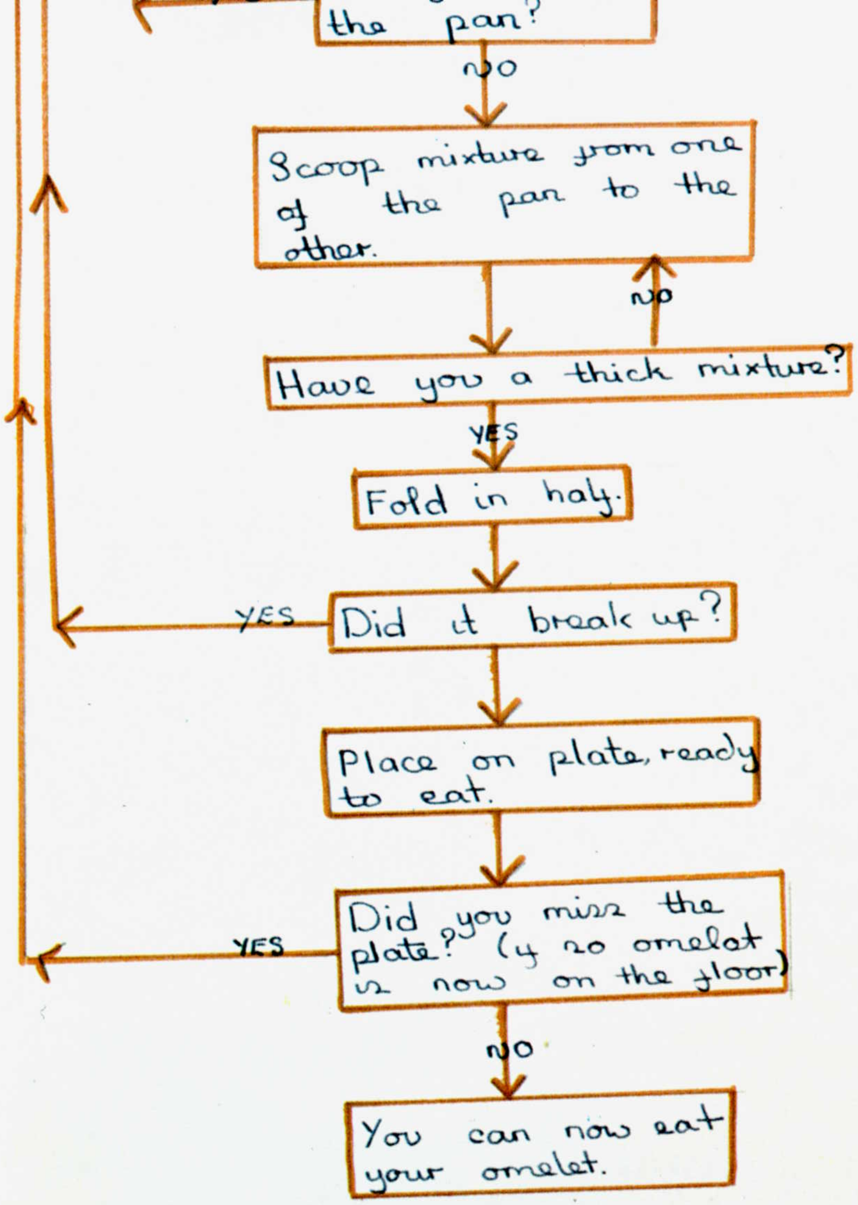
Later chapters will show flow diagrams being used as a preliminary stage in the writing of programs. A full program for a big problem could need thousands of separate steps. Such a program could take weeks or months to prepare but, once prepared, it could always be available for instant use with a new set of data. Results for very complicated and lengthy computations can then be produced almost instantly.

If the program contains a number of loops, the computer may go through millions of steps in the course of carrying out the computation but this will present little difficulty as far as time is concerned since a modern computer can perform many millions of calculations in one second. A convenient unit of time used in modern computer research and development is the nanosecond : one thousand million nanoseconds make one second.


How To Move a Car By Michael Stott

START
Do you want to move your car
Yes
No
Think of somthing els to do
Sit in drivers seat
Is gear stick in nutrol
Yes
No
Put gear stick into nutrol
Put key into ignition
Switch key left to right
Has engine of the car started
Yes
No
Place left foot on clutch and press
Put gear stick into first gear
Put hands on the stearing wheel
Take left foot off clutch and press excelorater


SUSAN NEWELL 2-3
START
Put a knob of butter in a pan
Place pan on a flame.
Is the flame too high?
YES
Turn the flame down.
NO
Beat eggs required in a basin
Add milk to bind
Have you added too much milk?
YES
Add another egg.
NO
Have you added enough milk?
NO
Add more milk.
YES
Pour mixture in to pan.


the pan?
NO
Scoop mixture from one of the pan to the other.
NO
Have you a thick mixture?
YES
Fold in half.
Did it break up?
YES
Place on plate, ready to eat.
Did you miss the plate? (if so omelet is now on the floor)
YES
NO
You can now eat your omelet.

A classroom human 'computer'

The principles involved in the programming of a computer can be introduced by a class activity in which the children act as the different units of the computer.

The accompanying diagram shows the main units of a computer, and the links between them.

Functions of the computer units

The **Control** unit tells the other units what to do, and when to do it. In large computers it can control the working of each unit in such a way that a number of tasks can be performed simultaneously. Its control of operations ensures that the computer is kept as productively occupied as possible.

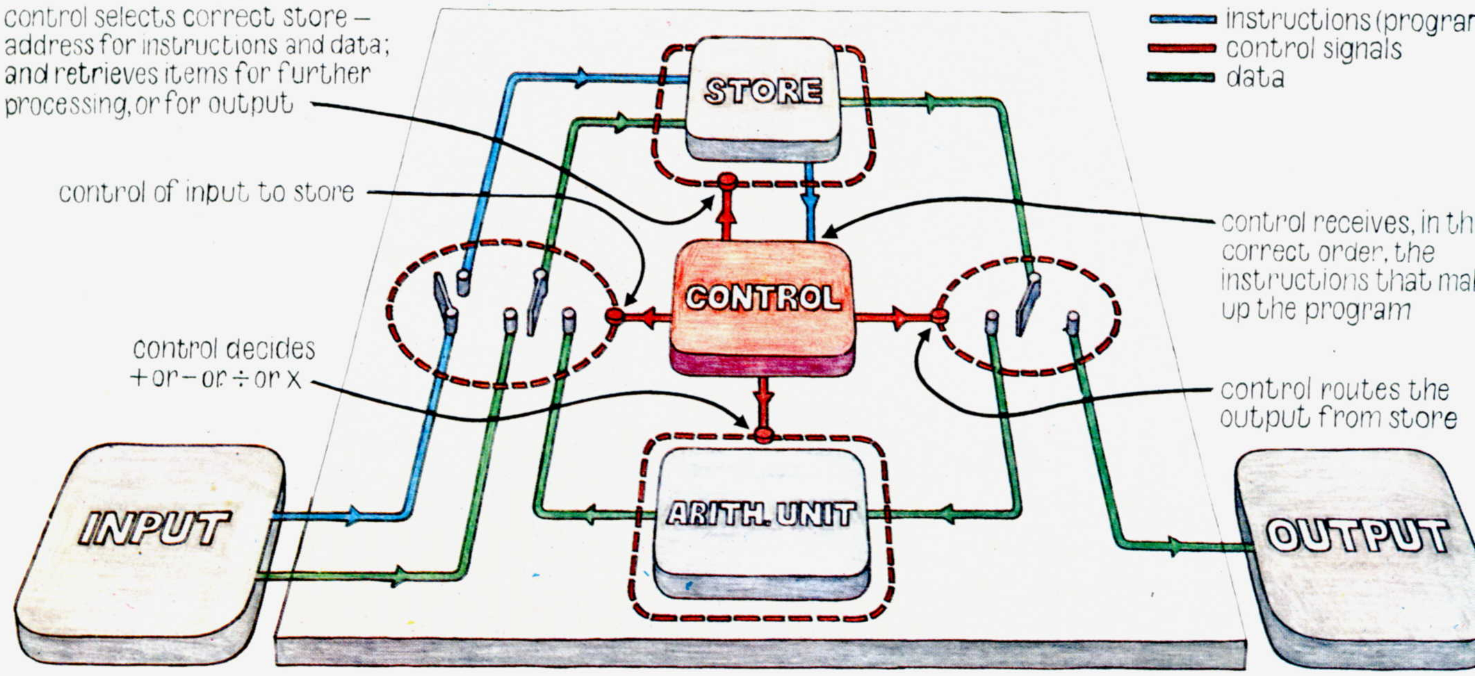
The **Input** unit accepts the instructions and information from the operator and converts the separate items into a form suitable for the control unit (a pattern of electrical impulses in a real computer). The programs and data will usually be presented to the input unit in the form of punched cards or tapes.

The **Output** unit presents the required results of computati to the operator. This is usually done by printing, punching cards or tape, or by an illuminated display on a screen.

The **Arithmetic Unit** performs all the calculations incorpora in the program. It can add, subtract, multiply and divide. It also compare numbers and decide whether a number is positive, negative or zero.

The **Store** is the computer's 'memory'. It can 'remember' bo instructions and numerical data. It is made up of a large number of individual cells in which the separate items of th program and data input can be held. Each cell has a numbe or store-address. Instructions and data are conveyed to the store, and held there, as patterns of electrical charges.

Programs and data can also be stored in an **External Store,** usually in the form of reels of tape, or cards.



control selects correct store – address for instructions and data; and retrieves items for further processing, or for output
control of input to store
control decides + or – or ÷ or x
instructions (prograr
control signals
data
STORE
CONTROL
ARITH. UNIT
INPUT
OUTPUT
control receives, in th correct order, the instructions that ma up the program
control routes the output from store

e photograph shows a class of children working as a uman computer'. The **Operator** has a store of programs in e form of punched cards ; and **Input** has a 'key' for erpreting the punched-hole instructions and data. This thod of input is not used in the first 'runs' ; it is explained in e chapter 'Working with real computers'.

e messenger, who is carrying a copy of the contents of one the store-cells, is being directed to the **Arithmetic Unit** by **Controller**.

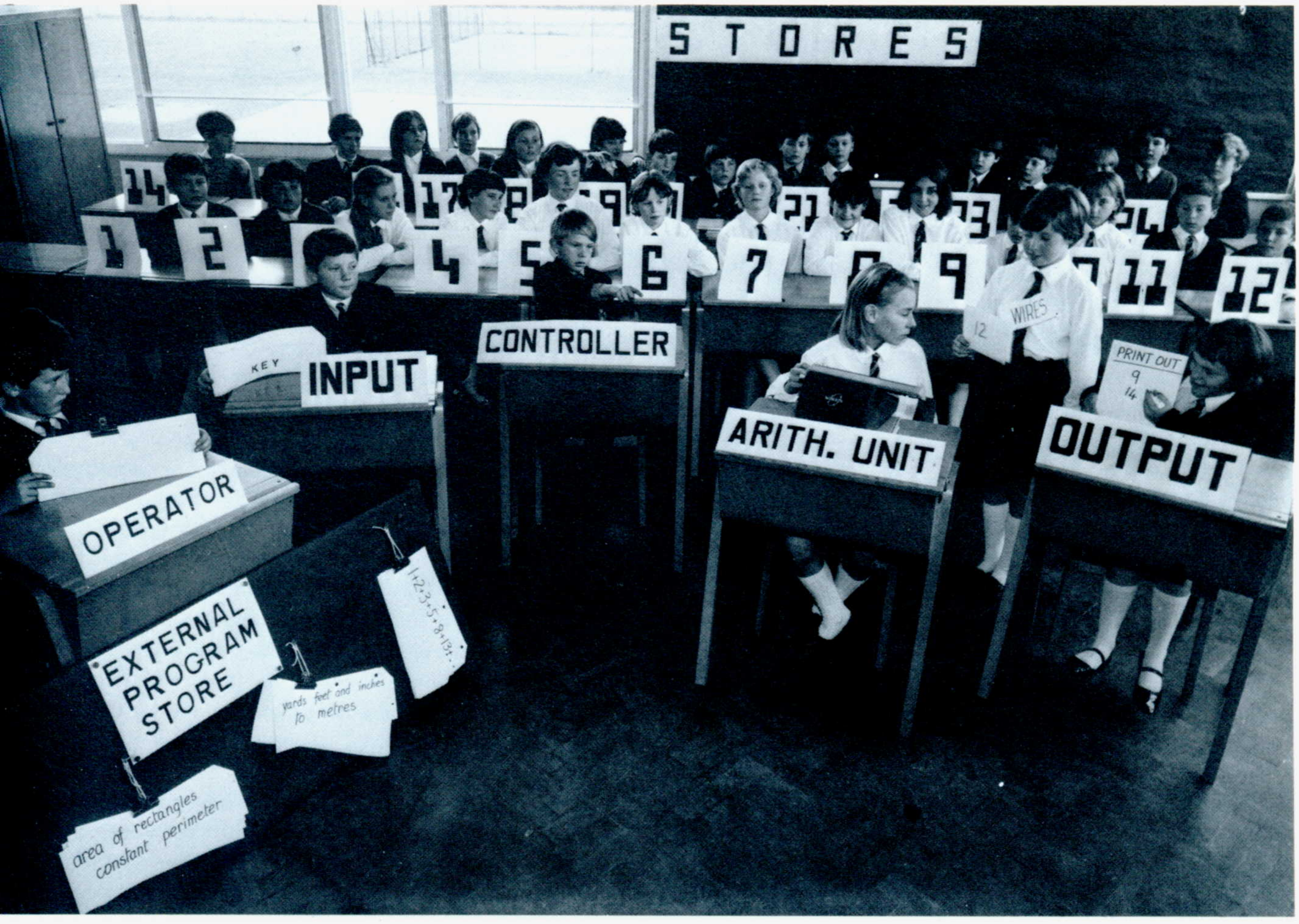
e **Arithmetic Unit** has a calculating machine for carrying out arithmetic operations needed in the program, but this is an essential piece of equipment for the activity.

e store-addresses were written in 'computer' figures to give ded interest. The children will possibly have seen examples of these numerals on cheques. The form of the figures can be seen as one more example of the detail in which input must be prepared before it can be processed by a computer. Even with wide variations of the shape and size of our numerals we can still recognise them but the computer can recognise only a special standardised set of shapes.

Output has a simple display system for printing out results when they are brought by the messenger.

Any computer, however large and expensive it may be, can obey only a limited number of instructions. Before it is possible to start working with the class 'computer', it is necessary to decide what these instructions are to be.

The diagram on page 28 shows the 'highways' along which information must flow. For work in the early stages of learning



computer principles, the following set of instructions will be adequate to direct this flow.

Set of instructions

Copy the information from the card waiting at **Input**, take it to **Store**, and put it in cell . . . .

Take a copy of the contents of **Store** cell . . . . and put it in the **Arithmetic Unit**.

Take a copy of the contents of the **Arithmetic Unit** and put it in **Store** cell . . . . .

Take a copy of the contents of **Store** cell . . . . . Add this to the contents of the **Arithmetic Unit**, leaving only the answer in the **Arithmetic Unit**.

Take a copy of the contents of **Store** cell . . . . . Subtract this from the contents of the **Arithmetic Unit**, leaving only the answer in the **A.U.**

Take a copy of the contents of **Store** cell . . . . . Multiply the contents of the **Arithmetic Unit** by this number, leaving only the answer in the **A.U.**

Take a copy of the contents of **Store** cell . . . . . Divide the contents of the **Arithmetic Unit** by this number, leaving only the answer in the **A.U.**

Do not continue with the instructions in the next cell, but instead go straight to **Store** cell . . . . . and obey the instruction there. After that, continue obeying the instructions in order from that point.

Look at the contents of the **Arithmetic Unit**. If it is greater than zero, go straight to **Store** cell . . . . . and obey the instruction there. After that, continue obeying the instructions in order from that point.

Copy the contents of **Store** cell . . . . . and take it to **Output** who displays it.

Stop all activity and wait for further instructions.

After the flow chart stage, all programs will be written as an arrangement of these instructions.

Before any calculations are performed by the computer, each instruction must be stored in a separate **Store** cell. For this to be done the computer needs a directive at the beginning of the program telling it to store the instructions and data that follow the directive.

During this first stage of storing the program, no calculations are performed. The process is simply one of copying the program from **Input** and putting the items in separate store cells.

When the program has been stored the computer will be reac for the directive to start running the program. Not until this directive is received will the computer start on the actual computing process. When the directive is received the computer will jump to the beginning of the program and ther obey the instructions in the correct sequence until it reaches the instruction to stop, or an improper instruction that it does not understand or cannot perform.

For all computers the instructions of the program must be written in accordance with a strictly observed set of rules, otherwise the computer would not be able to 'understand' what it was expected to do. Even a very small mistake can upset a whole program.

At first the children should use the longhand version of the s of instructions as used in this book. After some practice with this lengthy procedure they could be led to suggest that a shorter form of writing would be gladly accepted. The abbreviated form will be their first introduction to the idea of computer programming language.

The abbreviations in this book are only a suggested means for understanding basic principles. They are n suitable for input to any existing computer.

There are many different computer languages in current use : none of these is suitable for all computer installations. These languages have been designed for the convenience of peopl who use computers in their work ; they are not designed to a understanding of computer principles, and are unsuitable for work with young pupils.

Those pupils who progress to more advanced work and are able to make use of a large computer will, of course, need to learn the strict rules and rigid procedure for one or more of these languages. They should find the task much easier if the ave had previous experience in making up programs with a mple set of instructions based on first principles such as that sed in this book.

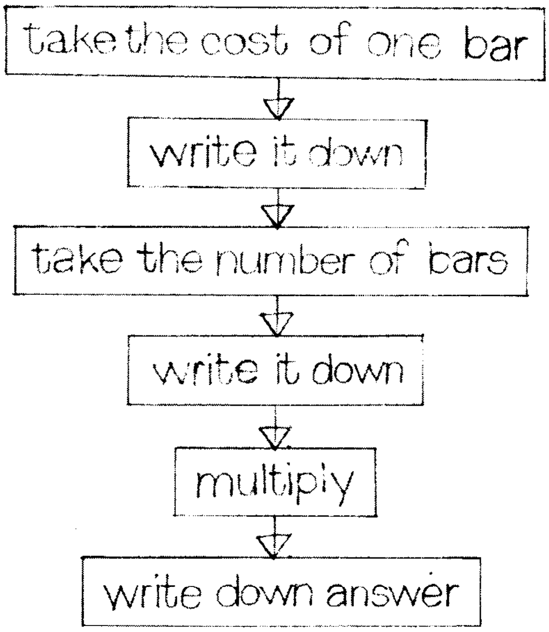
hortened form for 'Set of Instructions'	Mnemonic Abbreviation
ead next card, copy, and STore in cell	RST( )
NTer copy of contents of cell . . . . . in ithmetic unit	ENT( )
Tore copy of contents of Arithmetic unit in ell	STA( )
DD copy of contents of cell . . . . . to ontents of arithmetic unit	ADD( )
UBtract copy of contents of cell . . . . . from ontents of arithmetic unit	SUB( )
luLTiply contents of arithmetic unit by copy contents of cell	MLT( )
IVide contents of arithmetic unit by copy of ontents of cell	DIV( )
ımp UNconditionally to cell	JUN( )
ımp to cell . . . . . if contents of arithmetic nit equal to or GReater than 1. If not, go on next instruction	JGR( )
ake copy of contents of cell . . . . . to OUTput	OUT( )
TOP all activity	STOP
to first instruction of program in cell . . . . . nd RUN through programme	RUN( )
TORE copies of following items sequentially, eginning at cell	STORE( )

ome children may have wanted to use their own abbreviated rms of the long instructions. The necessity for standardising e abbreviations should then be discussed. Since every struction must have a precise meaning, and can in no ossible way be ambiguous, this is most important. For ample: the instruction **Sub(20)** means that the contents of **Store(20)** must be taken **from** the contents of the arithmetic unit, and at no time can this instruction be used for the reverse operation.

Program for finding the total cost of any given number of bars of chocolate at any given price

This first example of a program for the class 'computer' is a very simple one in which there are no branching instructions and no loops. The procedure for finding the answer to the problem may well appear to be in the 'using a sledge-hammer to crack a nut' class but, aside from its value as a simple introduction to programming, it will serve to show that, for a computer, even the simplest problem must be programmed properly.

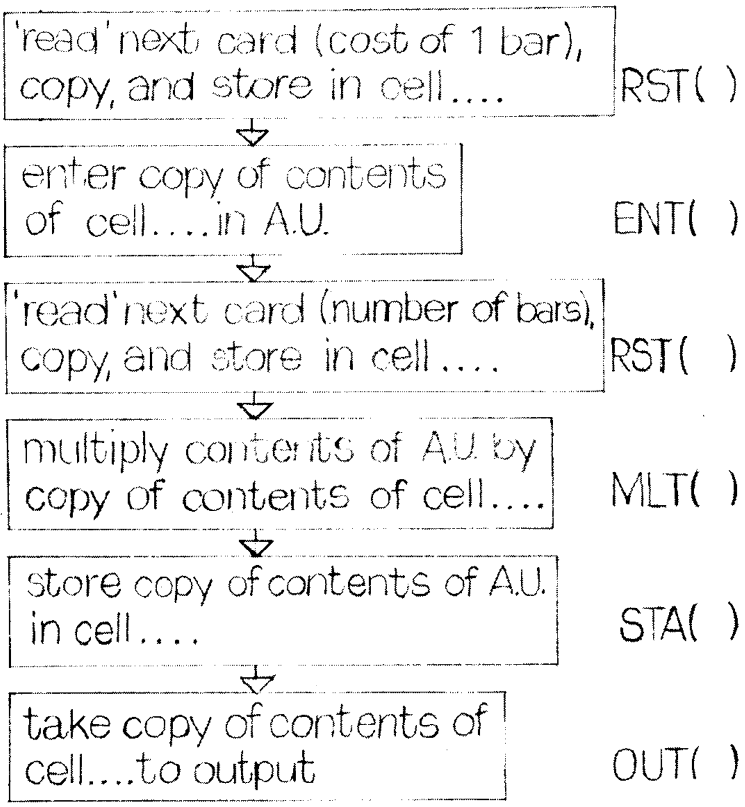
A procedure for ordinary arithmetic could be shown as follows:



This, however, is not sufficient for the class 'computer'. The computer cannot think what 'it' means; it cannot go looking for 'the cost of one bar'. The instructions must tell the computer precisely where the information can be found.

Knowing the capabilities and limits of the 'computer', as shown in the diagram and as described in the 'Set of Instructions', it is possible to prepare a suitable set of instructions.

Mnemonic Abbreviations



'read' next card (cost of 1 bar), copy, and store in cell.... RST( )

enter copy of contents of cell....in A.U. ENT( )

'read' next card (number of bars), copy, and store in cell.... RST( )

multiply contents of A.U. by copy of contents of cell.... MLT( )

store copy of contents of A.U. in cell.... STA( )

take copy of contents of cell....to output OUT( )

This is the general program. To use it for a particular purchase, the data must be entered. For this example it can be 24 bars costing 5p. each. These two numbers are variables i.e. they can be changed for different purchases. The rest of the program will be unchanged for any purchase.

Each of the instructions must be written on a separate card.

The two data cards (containing 5 and 24) will follow at the end of the program. They will be written on cards of a different colour to show that they are not part of the general program. After the general program has been stored the data cards will be waiting at **Input** for the **Controller** to 'read' them.

Each instruction and item of data can now be allocated a storage-address.

STORE( ) this is a directive to the **Controller**; it is not part of the stored program

RST( ) can go in **Store(1)**

ENT( ) can go in **Store(2)**

RST( ) can go in **Store(3)**

MLT( ) can go in **Store(4)**

STA( ) can go in **Store(5)**

OUT( ) can go in **Store(6)**

STOP can go in **Store(7)**

RUN( ) this is a directive; it is not part of the stored program

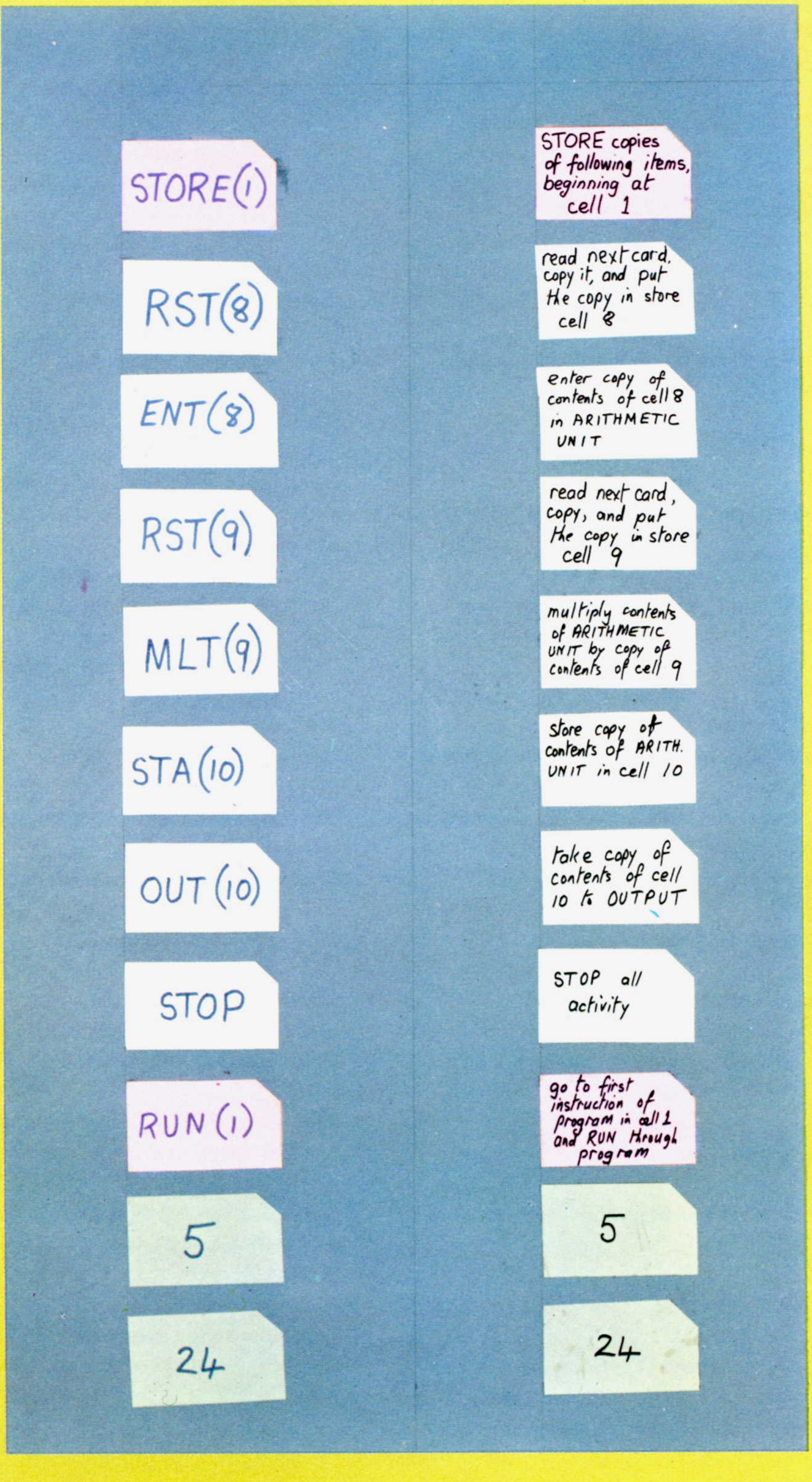
cost of 1 bar (5 in this example) can go in **Store(8)**

number of bars (24 in this example) can go in **Store(9)**

Now that the addresses of the instructions and data are known, it is possible to fill in the brackets in the instructions, and write the complete program.

```
STORE(1)
RST(8)
ENT(8)
RST(9)
MLT(9)
STA(10)   the next available storage cell
OUT(10)
STOP
RUN(1)
  5
24
```

In later work it will be more convenient if a generous allocation of storage cells is reserved for program instruction The next address can then be specified as the point at which the storage of data begins.



STORE(1)	STORE copies of following items, beginning at cell 1
RST(8)	read next card, copy it, and put the copy in store cell 8
ENT(8)	enter copy of contents of cell 8 in ARITHMETIC UNIT
RST(9)	read next card, copy, and put the copy in store cell 9
MLT(9)	multiply contents of ARITHMETIC UNIT by copy of contents of cell 9
STA(10)	store copy of contents of ARITH. UNIT in cell 10
OUT(10)	take copy of contents of cell 10 to OUTPUT
STOP	STOP all activity
RUN(1)	go to first instruction of program in cell 1 and RUN through program
5	5
24	24

Notes for operation of class 'computer'
The number of a storage cell (its store-address) must not be confused with the number in a cell (its contents).

Putting anything into a store automatically destroys the previous contents of the store.

Copying something from a store does not destroy its contents.

The arithmetic unit retains only the answer after each calculation.

The whole of the general program must be stored before any computation is done.

Directives (on red cards) are not stored in the computer.

Data cards (green) wait at **Input** until they are needed in the program.

After a program has been run once, it is only necessary to enter new data and a **Run(   )** directive for a repeat computation.

The computer does not think (it merely follows a series of instructions). The thinking is done by the program writer and the computer designer. A computer will obey mistakes in a program just as carefully as it will follow a correct program. As an example of this unthinking obedience, a computer can go on and on repeating the steps of an incorrectly programmed loop, producing meaningless results, and never stopping until the operator intervenes.

Procedure for operation of class 'computer'
Operator collects program (complete set of cards) from **External Store** and presents it to **Input.** He then switches on 'computer' i.e. he alerts the **Controller.**

Controller sends messenger to **Input** for copy of first card.

Input copies card and gives copy to messenger who takes it to **Controller.**

Input puts the copied card to one side.

Controller keeps the directive – **Store (1).** He must now store whatever follows in a sequence of store cells, beginning at **Store (1).** He will do this until he receives another directive.

Controller sends messenger to **Input** for copy of next card an directs him to put it in **Store(1).**

Controller......messenger......**Input**......copy......next card......**Store(**
....**Store(**
....**Store(**
....**Store(**
....**Store(**

The next instruction – **Stop** – is not obeyed but is put into the next store cell in the same way as the previous instructions. The **Controller** is still 'remembering' the first directive telling him to store whatever follows.

When **Controller** receives the copy of the next card – the directive **Run(1)** – he destroys his previous directive. He must now obey the stored instructions, in sequence, starting with the one in **Store(1).** He sends his messenger to **Store(1)** for a copy of the contents.

Controller reads the copy of the instruction – **RST(8)** – and directs his messenger to take a copy of the next card from **Input** and put it in **Store(8).** He destroys the copy of the instruction.

Controller sends messenger for copy of next instruction – **ENT(8).** He tells messenger to copy the contents of **Store(8)** and take it to the **Arithmetic Unit,** who keeps it.

Controller sends messenger for copy of contents of next stor cell. He tells messenger to go to **Input** for a copy of the next card and put it in **Store(9).**

Controller sends messenger for copy of next instruction – **MLT(9).** Copy of contents of **Store(9)** is taken to **Arithmetic Unit,** together with a multiplication sign.

Arithmetic Unit multiplies his previous number by this new number, keeping only the answer.

Controller sends messenger for copy of next instruction – **STA(10).** Copy of contents of **Arithmetic Unit** is put in **Store(10),** ready to be taken to **Output.**

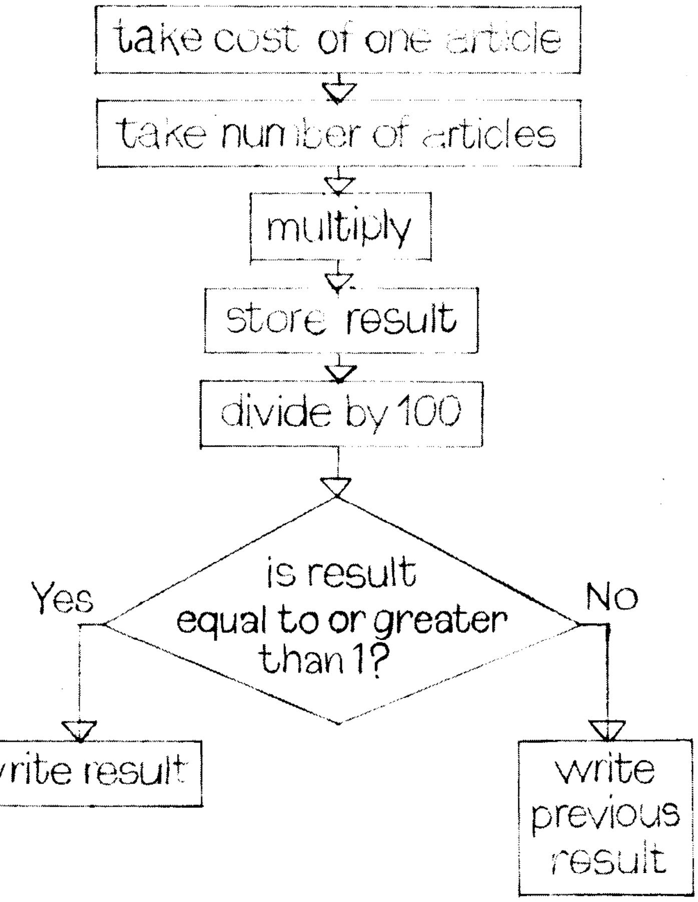
Controller sends messenger for copy of next instruction – **Out(10).** Copy of contents of **Store(10)** is taken to **Output,** who displays it.

ontroller sends messenger for copy of next instruction – **op.** On receiving it **Controller** destroys his previous ective and waits for further orders.

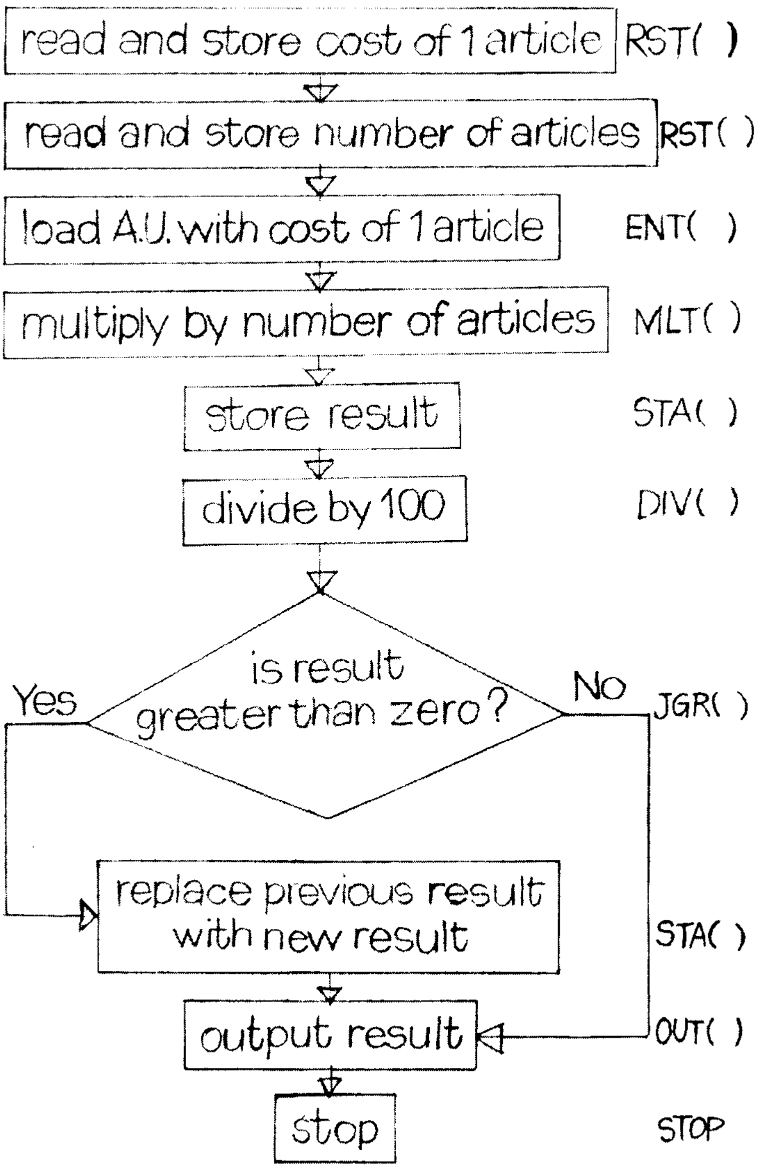
he program is not going to be repeated with another set data, the **Operator** collects the set of program cards from **put** and returns it to **External Store.** He then directs **ntroller** to 'switch off' all units. On this instruction all units, cept **Output,** destroy their contents.

he 'computer' is not 'switched off', it can be used for a eat run. For this, it is only necessary to enter different data rds (preceded by a **Run(1)** directive) at the **Input.**

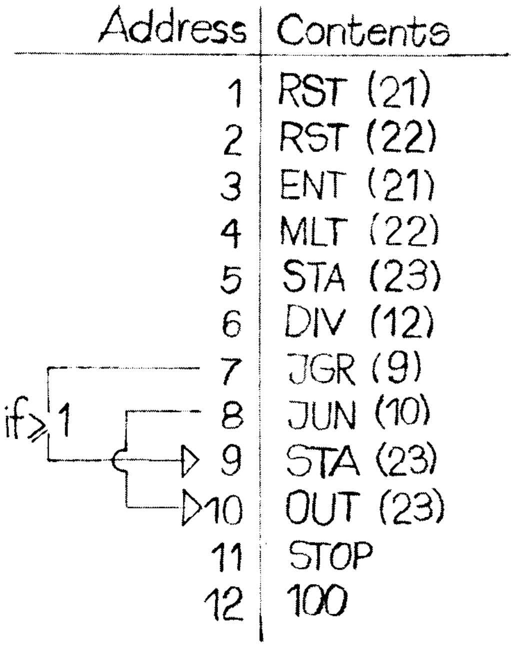
e results from the previous program will have been splayed at **Output** in pence. For some purchases the total st might well be more than £1, in which case the results are st displayed in £ and p. To do this, it is only necessary to ide by 100, but this must not be done if the total cost is less an 100p. To extend the previous program in such a way at this decision is built-in, it is necessary to include a anching instruction.



Written in more detail for the 'computer' the flow chart would be:


read and store cost of 1 article
RST( )
read and store number of articles
RST( )
load A.U. with cost of 1 article
ENT( )
multiply by number of articles
MLT( )
store result
STA( )
divide by 100
DIV( )
is result greater than zero?
Yes
No
JGR( )
replace previous result with new result
STA( )
output result
OUT( )
stop
STOP

If the first 20 storage addresses are reserved for program storage, then cells 21 onwards can be used for working stores. The constant 100 is needed in the general program so it can be stored with the instructions.



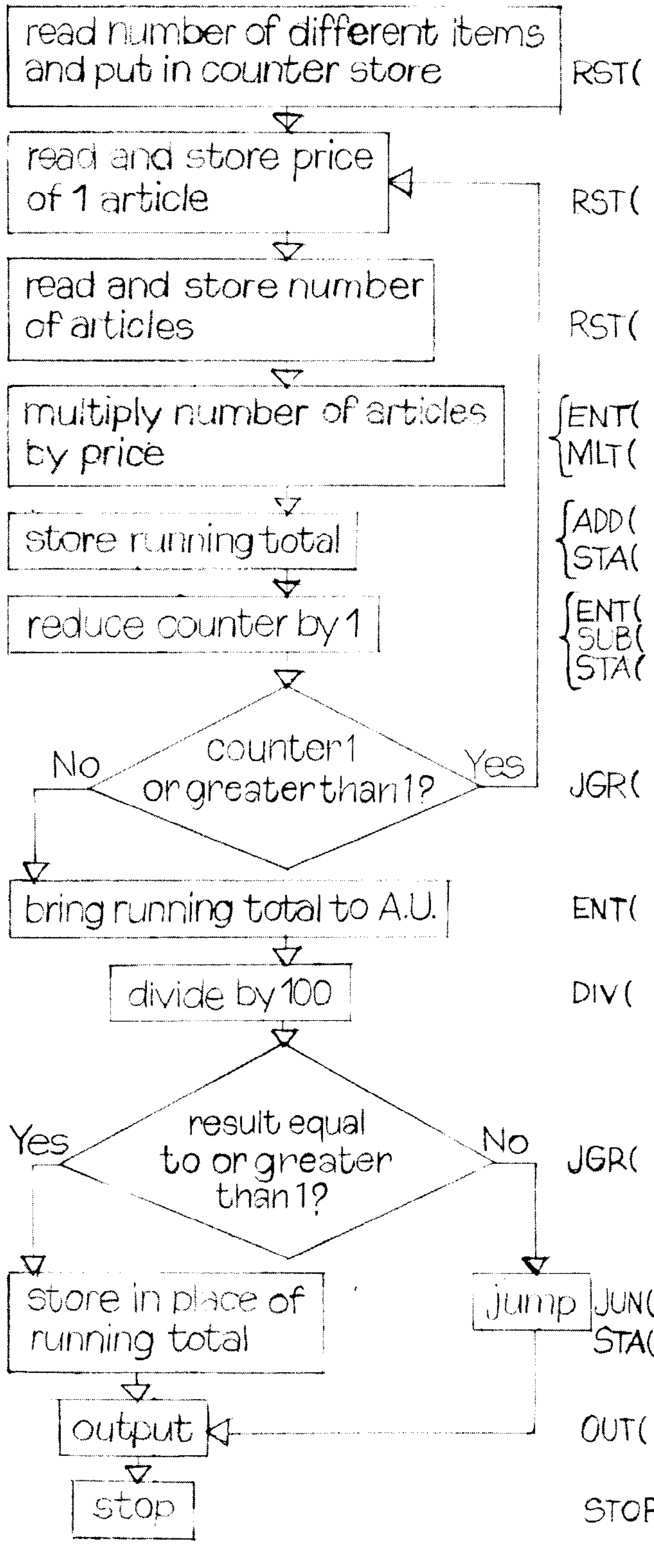
Address	Contents
1	RST (21)
2	RST (22)
3	ENT (21)
4	MLT (22)
5	STA (23)
6	DIV (12)
7	JGR (9)
8	JUN (10)
9	STA (23)
10	OUT (23)
11	STOP
12	100

This set of instructions will be preceded, as before, by the directive **Store(1)** and followed by the directive **Run(1)**, and the data.

A further extension of the program could be the inclusion of instructions telling the computer to find, from a list of purchases, the total cost of the first . . . . . items, each consisting of . . . . . articles at . . . . . p. each.

For example : to find the total cost of
14 bottles of orangeade at 8p. each
15 boxes of chocolates at 7p. each
6 bars of chocolate at 2p. each

For this form of the problem it will be necessary to include a loop in the program because some of the instructions must be repeated a number of times, according to how many different items there are. In the above example, for instance, there are three different items so the computer must be told to keep a running total until it has computed the total cost of all three items. To do this, it starts with the number three in store as a counter, reducing it by one each time it computes the cost of an item. No results are sent to the output until this counter reaches zero.


read number of different items and put in counter store
RST(
read and store price of 1 article
RST(
read and store number of articles
RST(
multiply number of articles by price
ENT(
MLT(
store running total
ADD(
STA(
reduce counter by 1
ENT(
SUB(
STA(
counter 1 or greater than 1?
No
Yes
JGR(
bring running total to A.U.
ENT(
divide by 100
DIV(
result equal to or greater than 1?
Yes
No
JGR(
store in place of running total
jump
JUN(
STA(
output
OUT(
stop
STOP

e constants needed in the program are 100 (for bringing p. £) and 1 (for reducing the counter store as each stage of e computation is carried out).

he first 25 cells are reserved for program storage, then 26 wards can be used for working stores.

s not usually necessary to check that the store cells are pty before a program is stored and run because the entry of item into a store cell automatically destroys the previous ntents. However, in this particular program it is necessary to rt off with a nil total cost. For a first run it is likely that the res will be empty so there will be no trouble but if the gram is run again without 'switching off' (which would stroy all contents) the previous final total will still be in the re and will be added to the next total. To avoid this it will necessary to include, in the program, instructions which ll ensure that the contents of the running total store will be o at the beginning of a program run. Real computers have nple ways in which this can be done but this 'computer' will ed a set of instructions. Using cell (26) as the running total re, the set of instructions **ENT(26), SUB(26)** and **STA(26)** ll effectively clear it.

e full program will then be:

```
store address       program
                    STORE (1)
           1        ENT (26)
           2        SUB (26) to clear cell (26)
           3        STA (26)
           4        RST (27)
           5◁       RST (28)
           6        RST (29)
           7        ENT (28)
           8        MLT (29)
           9        ADD (26)
          10        STA (26)
          11        ENT (27)
general   12        SUB (23)
program   13        STA (27)
          14        JGR (5)
          15        ENT (26)
          16        DIV (22)
          17        JGR (19)
          18        JUN (20)
          19◁       STA (26)
          20◁       OUT (26)
          21        STOP
          22        100
          23        1
                    RUN (1)
```

Each item of the program will be written on a separate card. The set of program cards will be followed by the data cards.

```
variables   .......... (number of different items)
            .......... (cost of 1 article in 1st. item)
            .......... (number of articles in 1st. item)
            .......... (cost of 1 article in 2nd. item)
            .......... (number of articles in 2nd. item)
            .......... (cost of 1 article in 3rd. item)
            .......... (number of articles in 3rd. item)
```

In this example the program from page 37 is being run through the 'computer'. The instruction in (13) has just been carried out.

From this stage the controller will move on to (14). The instruction in this cell tells the controller to go back to (5) if the number in the **A.U.** is greater than zero. At this point in the run, the number in the **A.U.** is 2 so the controller will jump back to (5) for his next instruction. Having obeyed this instruction he will then continue with the instructions from (6) onwards.

Having achieved some basic understanding of computer principles by taking part in the class computer activity it is likely that there will be a need for an individual approach in further work. At first, this can be met by encouraging the more capable pupils to write their own programs for testing with the class 'computer'.

After this stage pupils will be ready to write and test their own programs by themselves. For the testing part it will still be advisable to have some practical means – a simple storage system – by which the programs can be run through the computer processes.

There are many ways in which a battery of storage cells can be provided for this purpose but, with simplicity and cheapness in mind, an arrangement of egg boxes can be quite satisfactory. Alternatives could be plastic ice-cube trays, matchboxes, or small storage drawers.

The photograph shows an arrangement of boxes that will provide 30 storage addresses, an input section, an arithmetic unit and an output section. The pupil will be programmer, operator and controller.

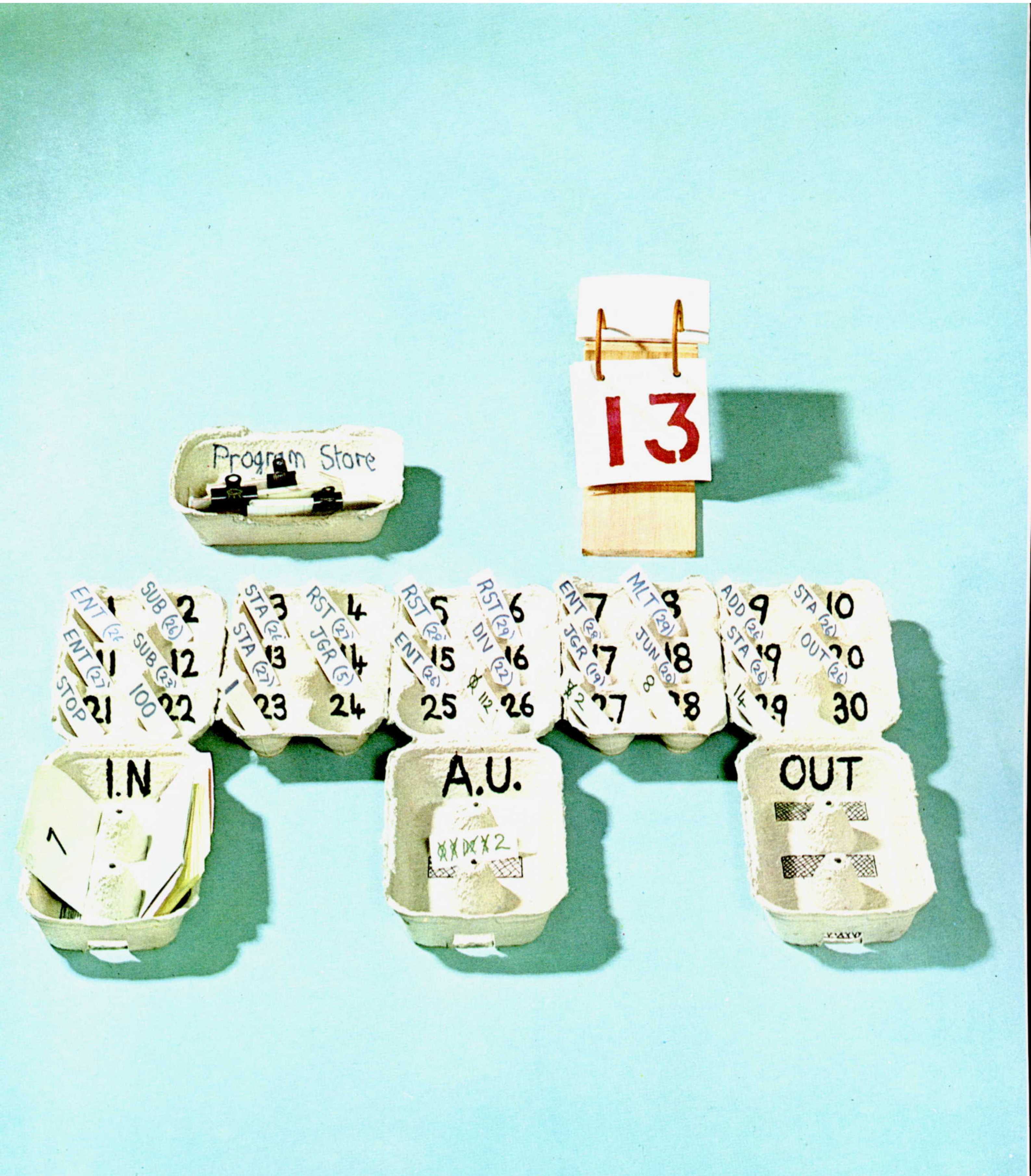
Notes on operation of desk top, egg box 'computer'
The program pack is entered on one side of the **Input** tray. After each card has been read, copied and stored, or obeyed it is a directive, it is placed on the other side of the tray. This simple procedure will ensure that the green data cards are each exposed for reading, in their correct order, at the proper time.

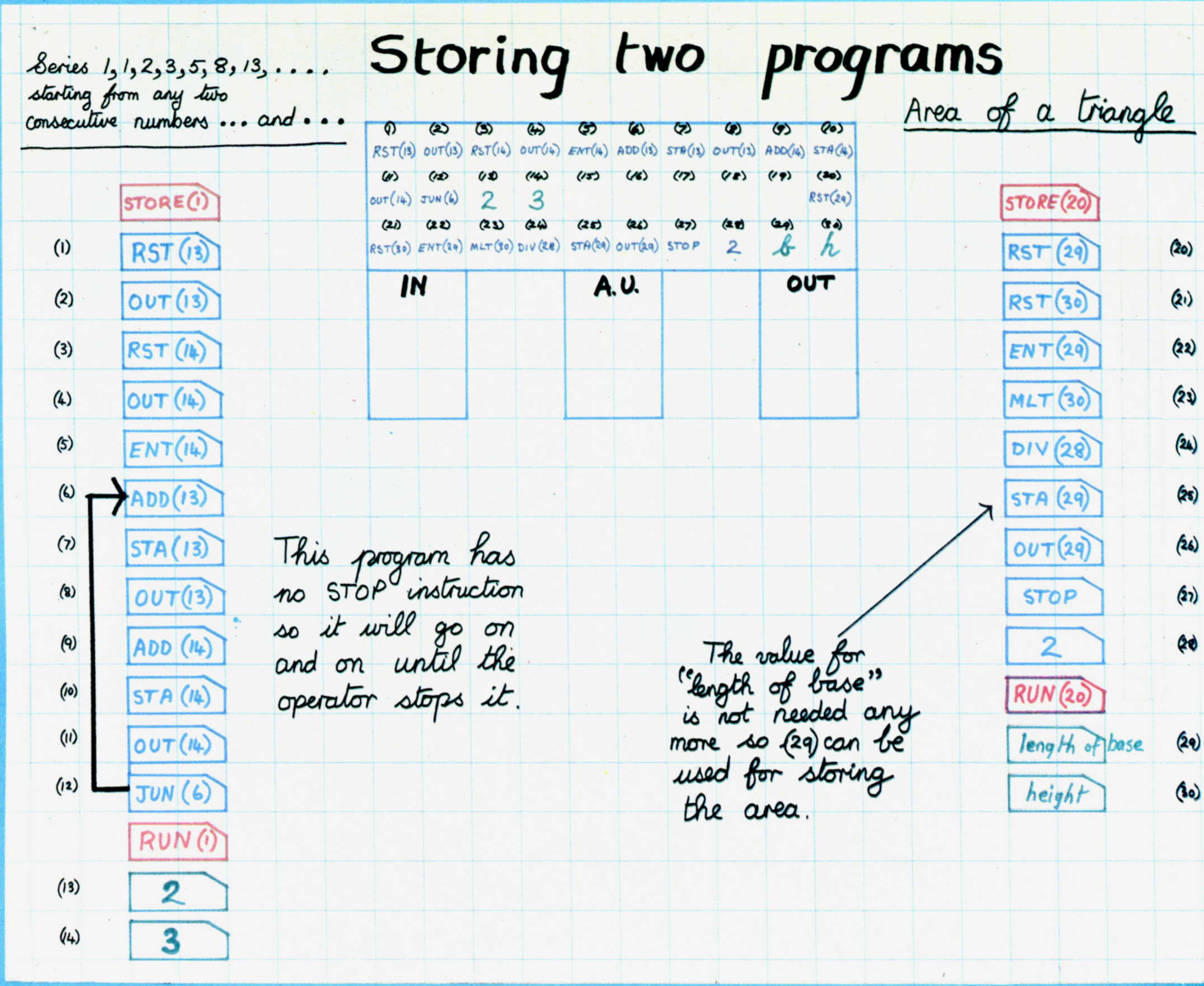
The **Arithmetic Unit** must never contain more than one number after each operation.

The previous content of a store is destroyed when it is replaced by another item. This means that with new data and a **Run( )** directive a stored program can be used more than once without taking the previous data slips out of the stores before each run.

After the program has been stored, and the computation part of the process has been reached, it is helpful to have a flip-over counter (see photograph) to show which storage-address has been reached.

If there is enough storage space, it is possible for the 'computer' to have more than one program stored at any given time. The **Store( )** and **Run( )** directives will tell th computer the address of the first instruction in the particular program being used.


Program Store
13
ENT (26) 1
SUB (26) 2
STA (26) 3
RST (27) 4
RST (29) 5
RST (29) 6
ENT (28) 7
MLT (29) 8
ADD (26) 9
STA (26) 10
ENT (27) 11
SUB (23) 12
STA (27) 13
JGR (5) 14
ENT (26) 15
DIV (22) 16
JGR (19) 17
JUN (20) 18
STA (26) 19
OUT (26) 20
STOP 21
100 22
23
24
25
112 26
2 27
8 28
14 29
30
I.N
1
A.U.
2
OUT



Storing two programs

Series 1, 1, 2, 3, 5, 8, 13, ... starting from any two consecutive numbers . . . and . . .

Area of a triangle

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
RST(13)	OUT(13)	RST(14)	OUT(14)	ENT(14)	ADD(13)	STA(13)	OUT(13)	ADD(14)	STA(14)
(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)
OUT(14)	JUN(6)	2	3						RST(29)
(21)	(22)	(23)	(24)	(25)	(26)	(27)	(28)	(29)	(30)
RST(30)	ENT(29)	MLT(30)	DIV(28)	STA(29)	OUT(29)	STOP	2	b	h

IN		A.U.		OUT

```
     STORE(1)
(1)  RST(13)
(2)  OUT(13)
(3)  RST(14)
(4)  OUT(14)
(5)  ENT(14)
(6)  ADD(13)
(7)  STA(13)
(8)  OUT(13)
(9)  ADD(14)
(10) STA(14)
(11) OUT(14)
(12) JUN(6)
     RUN(1)
(13) 2
(14) 3
```

This program has no STOP instruction so it will go on and on until the operator stops it.

```
STORE(20)
RST (29)        (20)
RST (30)        (21)
ENT (29)        (22)
MLT (30)        (23)
DIV (28)        (24)
STA (29)        (25)
OUT (29)        (26)
STOP            (27)
2               (28)
RUN (20)
length of base  (29)
height          (30)
```

The value for "length of base" is not needed any more so (29) can be used for storing the area.

ving prepared and run a number of successful programs th apparatus, the pupils can progress to a stage of testing eir programs by recording the steps on a chart.

ample : find the total cost of the first two items on page 36.

			constants		working store				
tore ddress	program	A.U.	22	23	26	27	28	29	output
1	ENT (26)	?	100	1	?	?	?	?	
2	SUB (26)	0	"	"	"	"	"	"	
3	STA (26)	"	"	"	0	"	"	"	
4	RST (27)	"	"	"	"	2	"	"	
5	RST (28)	"	"	"	"	"	8	"	
6	RST (29)	"	"	"	"	"	"	14	
7	ENT (28)	8	"	"	"	"	"	"	
8	MLT (29)	112	"	"	"	"	"	"	
9	ADD (26)	112	"	"	"	"	"	"	
10	STA (26)	"	"	"	112	"	"	"	
11	ENT (27)	2	"	"	"	"	"	"	
12	SUB (23)	1	"	"	"	"	"	"	
13	STA (27)	"	"	"	"	1	"	"	
14	JGR (5)	"	"	"	"	"	"	"	
5	RST (28)	"	"	"	"	"	7	"	
6	RST (29)	"	"	"	"	"	"	15	
7	ENT (28)	7	"	"	"	"	"	"	
8	MLT (29)	105	"	"	"	"	"	"	
9	ADD (26)	217	"	"	"	"	"	"	
10	STA (26)	"	"	"	217	"	"	"	
11	ENT (27)	1	"	"	"	"	"	"	
12	SUB (23)	0	"	"	"	"	"	"	
13	STA (27)	"	"	"	"	0	"	"	
14	JGR (5)	"	"	"	"	"	"	"	
15	ENT (26)	217	"	"	"	"	"	"	
16	DIV (22)	2·17	"	"	"	"	"	"	
17	JGR (19)	"	"	"	"	"	"	"	
19	STA (26)	"	"	"	2·17	"	"	"	
20	OUT (26)	"	"	"	"	"	"	"	2·17
21	STOP	"	"	"	"	"	"	"	

e operator interprets the output.   £2 17p.

Presenting the program to the computer

Punched cards are a very convenient way of storing information in a form suitable for mechanical processing.

A set of such cards can contain a vast amount of information, all represented by punched holes, but much simpler cards should be used for introducing the idea of storing information in this way.

A set of seven large cards (say 7 inches by 5 inches), each having three finger-sized holes, can be used to show how a simple mechanical process can sort a number of cards into their correct numerical order.

After the cards have been shuffled, hold them upright on a flat surface and push one finger through the right hand holes of the pack. Those cards which can be lifted from the others are put at the back of the pack. When this process has been repeated with the second and third holes, the cards will be in their correct numerical order.

The children's attention should be drawn to the importance of having one corner cut off so that it will be obvious if a card in the pack has been reversed. For a similar reason, commercial cards must not be square.

By taking note of the black marks over the uncut holes, the children can prepare a table in which the figure 1 represents uncut holes and a 0 represents those that have been cut av

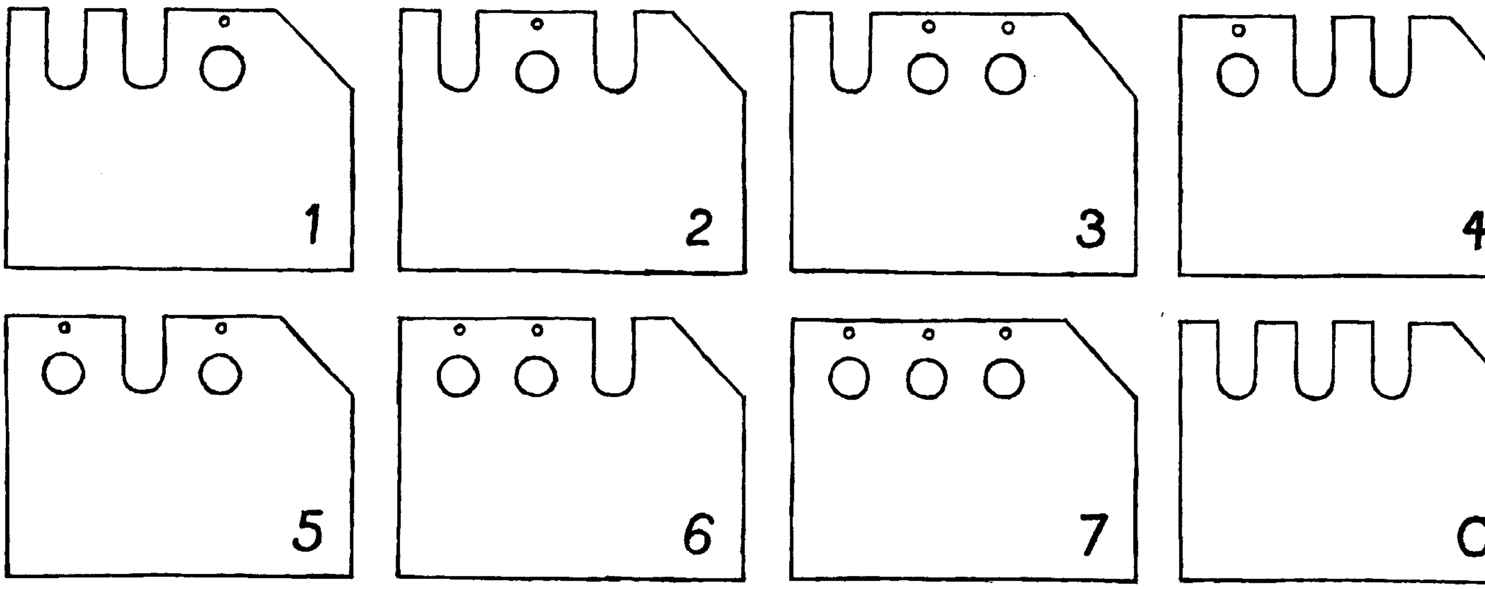
1	001
2	010
3	011
4	100
5	101
6	110
7	111

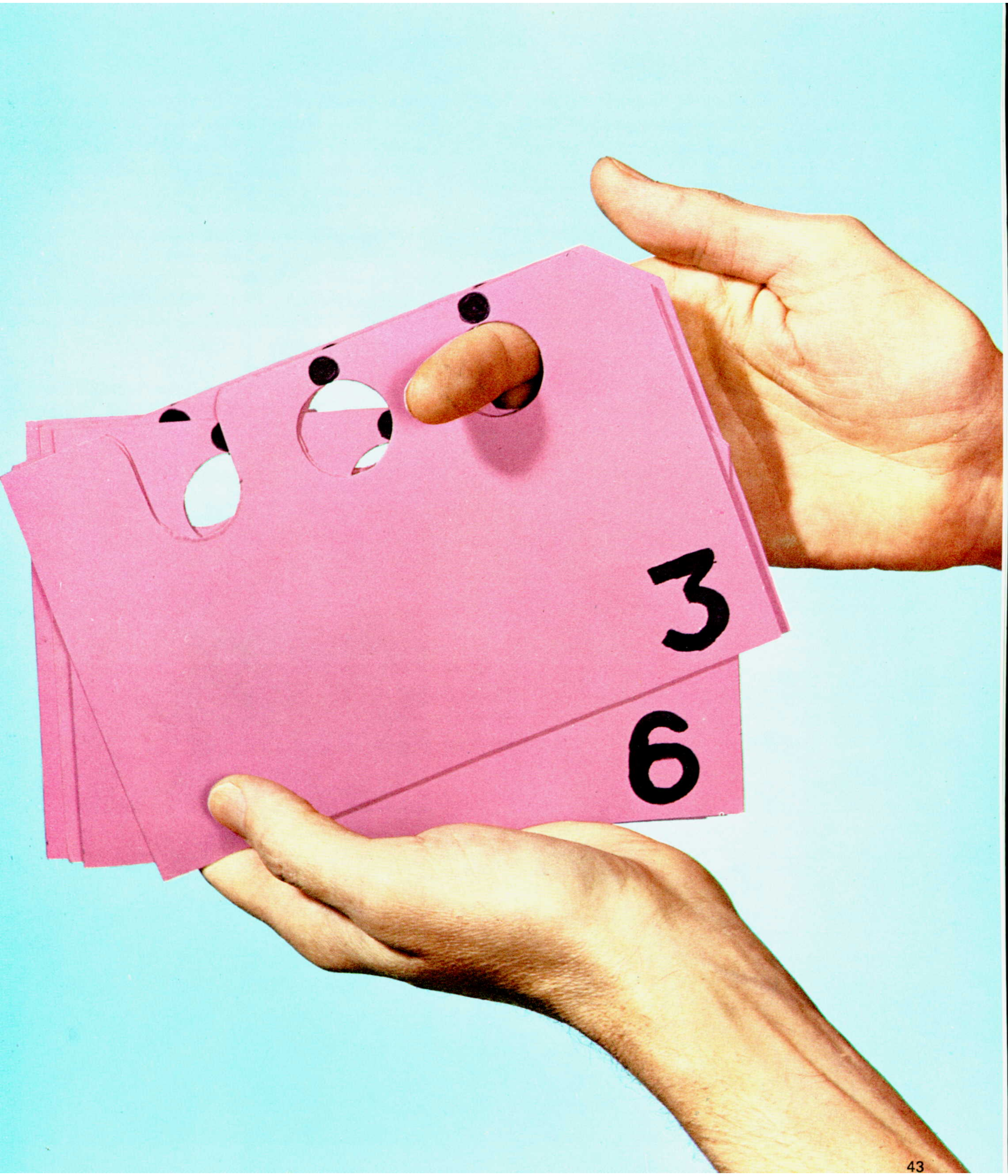
This will provide a possible introduction to the binary numb system.

When they have had adequate experience with this small se they can experiment with a four-hole set of cards to see hov an extra hole can increase the range of numbers which can represented by this method.

Cards with a greater number of holes can be used for storin information based on the children's own interests.

For success in the use of these cards it is essential that the holes are accurately positioned. Because of the possible practical difficulties of doing this, it is often preferable to bu pre-punched cards. Six-hole cards are obtainable from: Allen-Glenold Ltd., East Farndon, Market Harborough, Leicestershire, at 75p. per 1,000 plus postage.




3
6

All the information to be stored on the cards must be in binary form, i.e. the question used for preparing the information must be worded in such a way that YES or NO will be sufficient for the answer.

The set of cards which have the answer YES to any particular question can be extracted by pushing a long needle through the holes corresponding to the question, and lifting them from the rest.

Children sometimes get confused over whether they should cut away the hole for YES or for NO.

There is no definite rule about this – indeed you will find that in *Shape and Size* ▼ it is suggested that a cut hole means 'Yes' and an uncut one means 'No'. However, we feel that the more logical convention is to be able to lift the ones with the word 'Yes' and this is the practice we follow in the Guide.

For punched cards, punched tape and magnetic tape storage, all information is converted into a form in which it can be recorded in a two-state system (punched or not punched; on or off; positive or negative). This system of storage and presentation is very convenient for computers because it can simplify design problems. A machine designed to store and manipulate decimal digits requires devices capable of dealing with ten different figures in each column: the typical mechanical calculating machine is an example of this. Such a system would be too large and too slow for a computer. The speed needed in a modern computer requires the use of electronic systems. In such a system it is very convenient to represent each digit by a device which is ON or OFF. Much of the circuitry of an electronic computer consists of devices that can 'hold' or 'lose' a charge of electricity. These charges can be 'read' by the computer and easily transferred from one location to another in the course of running a program.

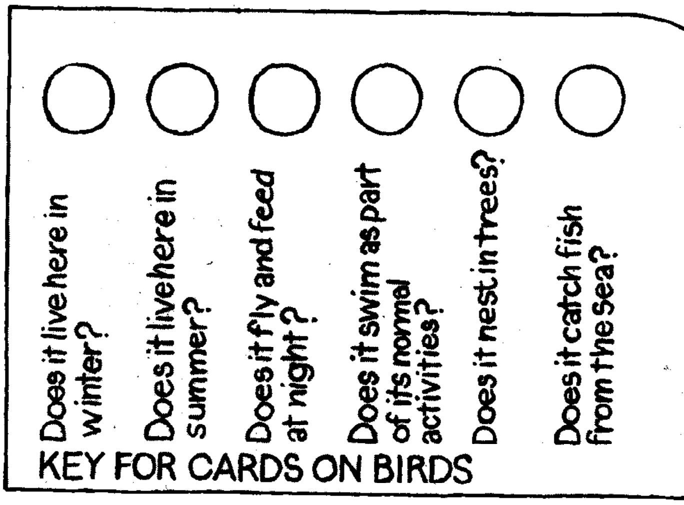
Limited activities in working with binary numbers can help children to understand this part of the working of a computer. This work (combined with experience in using other number bases) will also help them to understand the principles on which our denary system is based, but an excessive amount of conversion is unhelpful.

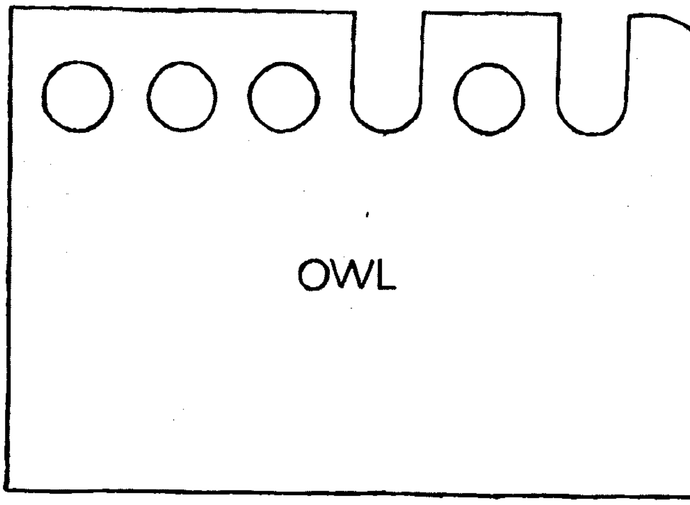
It is not necessary for a computer operator, or programmer, be competent in binary computation. The computer is designed to translate the input notation to its own before starting work on it; and it can perform the reverse operation before it presents the results at the output.

Putting a program on punched cards

Cards for real computers are not pre-punched with holes which are cut away to represent information like the simple examples for children's activities at the beginning of this chapter. It is normal practice for holes to be punched only where they are needed to represent information.

When the children have had sufficient experience with the first type of card, and have written programs for the class 'computer', they can progress to making cards that are base on similar principles to those used in actual computer pract


Does it live here in winter?
Does it live here in summer?
Does it fly and feed at night?
Does it swim as part of its normal activities?
Does it nest in trees?
Does it catch fish from the sea?
KEY FOR CARDS ON BIRDS



e illustration on page 46 shows a suggested design for a which can be used to decide the position of the holes in cards, and to interpret the cards when they are used as ut to the class 'computer'.

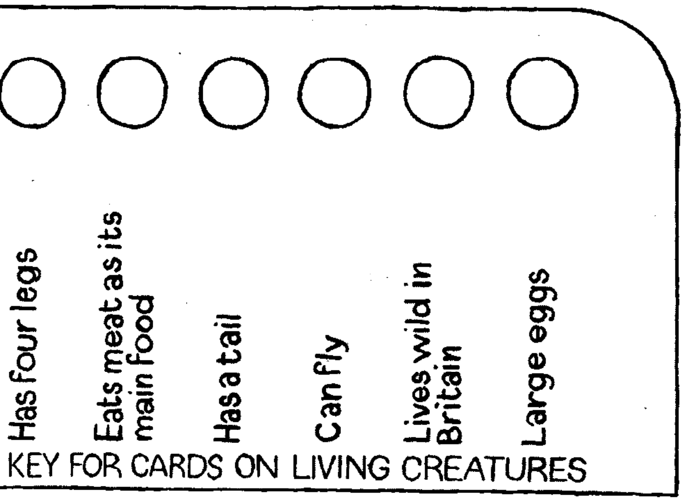
e accurate positioning of the holes is made much pler if $\frac{1}{2}$ inch squared paper is used as the material for the ds.

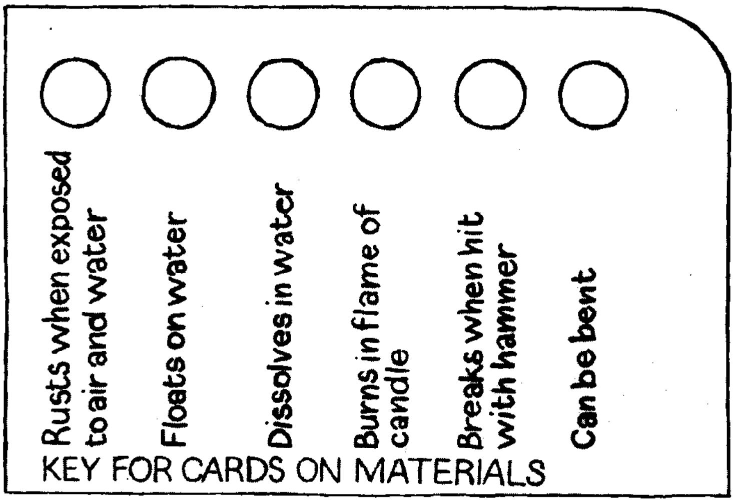
e holes are made with a $\frac{1}{2}$ inch punch and hammer. For t results the punching should be done on a firm base of **d** wood or soft metal (lead is the best material of all).

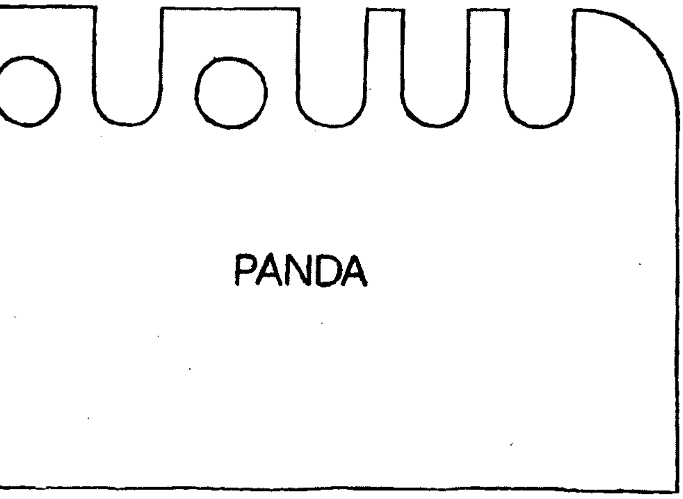
e row of abbreviations along the top of the key card ludes the full 'Set of Instructions' as detailed on page 31. **st'** and **'Data'** are added to make it clear whether the ormation on the card is an instruction or a number. The ures in blue (on the left) are for storage-addresses. The figures in green (on the right) are used to record numbers, either as constants or as the data for a program.

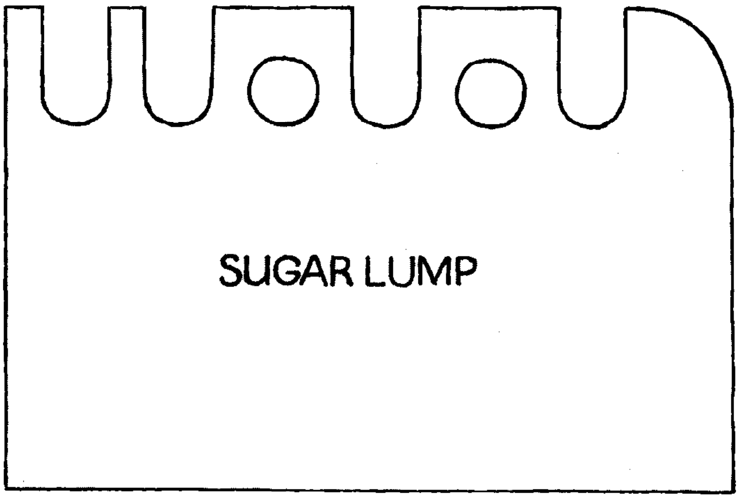
Using this method, a complete set of cards for a program can be made quite easily. When the program is run through the 'computer', **Input** has the task of interpreting the information before passing it to the **Controller.** The cards can be 'read' easily by simply placing each in turn over the key card and reading whatever can be seen through the holes. This interpretive procedure is similar to actual computer practice when a computer must have, as part of its equipment, the means whereby the particular language being used on the tape or cards can be changed into its own machine language.

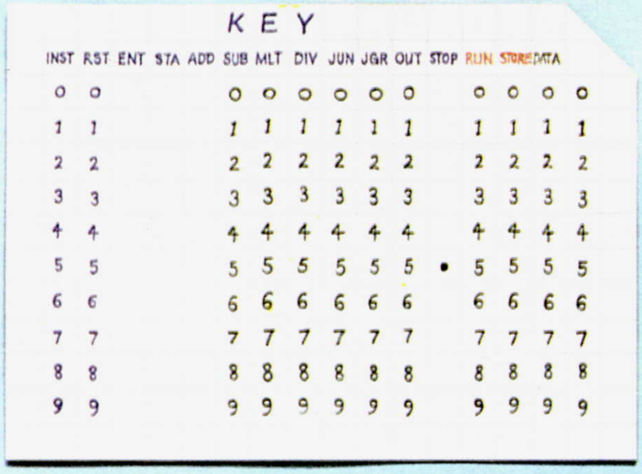
It was pointed out in chapter 3 that there is no single standardised form in which programs can be written for running on all computers. This applies also to punched cards and tape. A set of cards punched for one computer




Rusts when exposed to air and water
Floats on water
Dissolves in water
Burns in flame of candle
Breaks when hit with hammer
Can be bent
KEY FOR CARDS ON MATERIALS

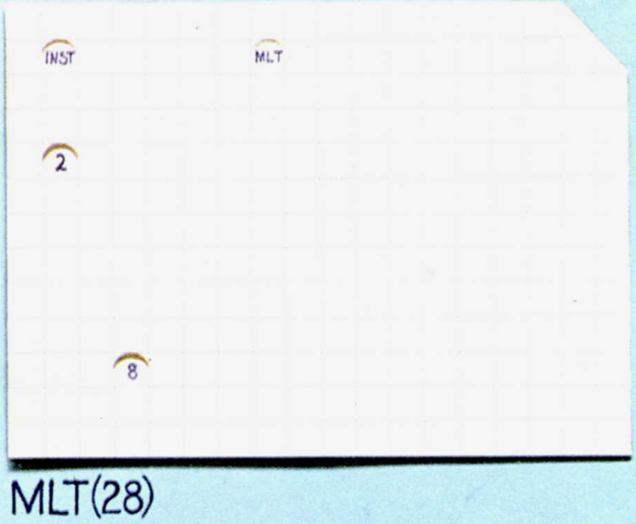




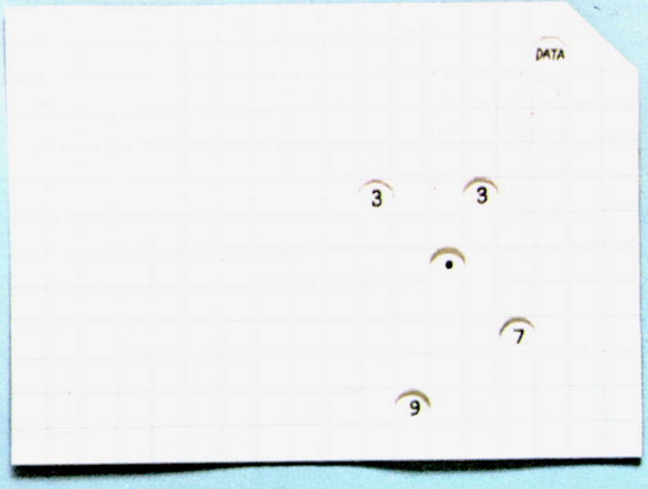


KEY

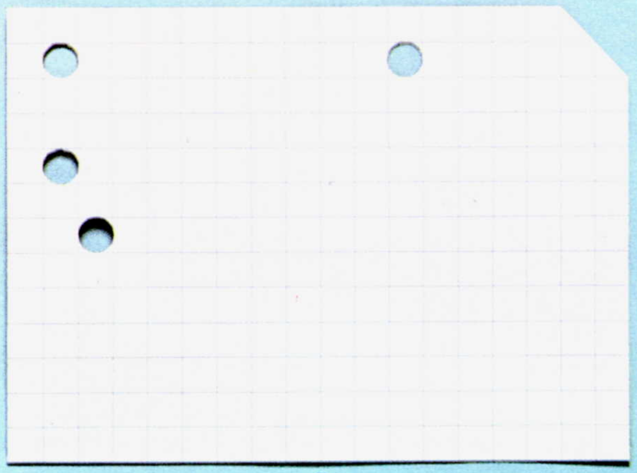
INST	RST	ENT	STA	ADD	SUB	MLT	DIV	JUN	JGR	OUT	STOP	RUN	STORE	DATA	
0	0				0	0	0	0	0	0		0	0	0	0
1	1				1	1	1	1	1	1		1	1	1	1
2	2				2	2	2	2	2	2		2	2	2	2
3	3				3	3	3	3	3	3		3	3	3	3
4	4				4	4	4	4	4	4		4	4	4	4
5	5				5	5	5	5	5	5	•	5	5	5	5
6	6				6	6	6	6	6	6		6	6	6	6
7	7				7	7	7	7	7	7		7	7	7	7
8	8				8	8	8	8	8	8		8	8	8	8
9	9				9	9	9	9	9	9		9	9	9	9



MLT(28)



39•37



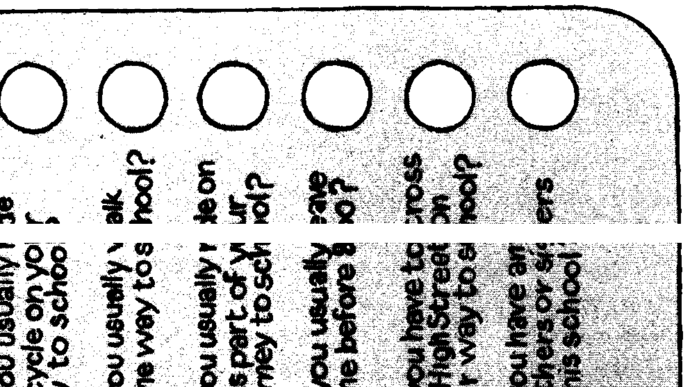
what's this one?

allation is not always suitable for running on a different nputer. The cards in this chapter are suggested as a simple ans by which the children can be introduced to the idea of unched card input system. The format of the cards does follow the pattern of any existing computer system.

ard input system such as is actually used in computing will well within the understanding of children who have rked with the simple cards shown in this chapter. It will be easy step for them to move on to using real cards and the ociated punching equipment when they have the chance un programs on a real computer.

nched tape input is used in some computer installations but is not a satisfactory system for most schools because the ipment is too expensive ; and it is a very inconvenient tem when there is a need to alter or correct a program.

en a program, or a collection of data, has been punched on ds it is easy to make changes by removing, replacing or ling individual cards, but with a punched tape system it is ally necessary to change the whole tape. This is a most portant point in computer education where the main phasis is on the preparation of programs rather than the lity with which they can be used again and again. Even in early stages, however, the children can be made aware of use of tape as a form of input. They may enjoy writing ssages in one of the punching codes (by dots on strips of ared paper). This activity will help to make them see that at appears to be nothing more than a confusing array of es can, in fact, be meaningful.



a bicycle on yo
way to schoo
Do you usually walk all the way to school?
Do you usually ride on a bus part of your journey to school?
Do you usually leave home before
Do you have to cross the High Street on your way to school?
Do you have any brothers or sisters in this school?

Working with a computer

Children who have worked on the ideas of the first sections of this book will have achieved some understanding of the nature of computers and they will be aware of some of the consequent effects that are likely to be caused by the widespread introduction of a computer system.

This work will have been valuable in itself, but as the children progress it is only to be expected that they will want to have direct experience with a real computer. As a first step the children may benefit from a visit to a computer installation. It should be seen, however, that such a visit is of little or no value as an introduction to computing for young children. The main interest in a planned visit to a computer installation will lie in the input and output facilities, and the speed with which the working is carried out. The rest of the computer will be seen as nothing more than one or more uninteresting metal boxes.

The opportunities for pupils to make use of a real computer in their work are depressingly limited at the present time but there are various ways in which persistent pioneers have managed to overcome these difficulties. This section will include a brief outline of some of these approaches.

Only a small proportion of the pupils who are fortunate enough to benefit from direct experience with a computer will progress to the stage where they are using the computer as a necessary tool, beyond the extent for which a calculating machine would be adequate. Such a stage will be reached only by some pupils in the top forms of the secondary school. For those children who do not reach this level, the value of running programs through a computer will lie in the increased understanding of computer principles and the benefits to be derived from practice in logical thinking. Most children will not need the full facilities of a large computer since their overall attainment will set limits on the kind of problem that they will be capable of tackling. Indeed, many of the younger children will deliberately restrict their first programs to a level where they can check, by ordinary arithmetic, 'to see if the computer is right'.

Long time lags between the writing and running of the programs tend to kill interest at all levels but such delays are particularly detrimental with young children. Postal/courier services to a distant computer are comparatively inexpensive, and useful to older pupils, but the interest of beginners is best maintained by a system which provides quick results. The ideal situation in these early years is therefore one in which the computer can be used in the classroom.

This classroom computing facility must be capable of being programmed in a very simple way. It is pointless if children have to devote excessive time to learning a complex language or if they become bogged down with technical difficulties.

Two ways of providing this facility have been found to be successful with children. The first of these uses a small desk-top computer (which may be no more than a programmable calculator) and the second a terminal connected by telephone line to a distant computer.

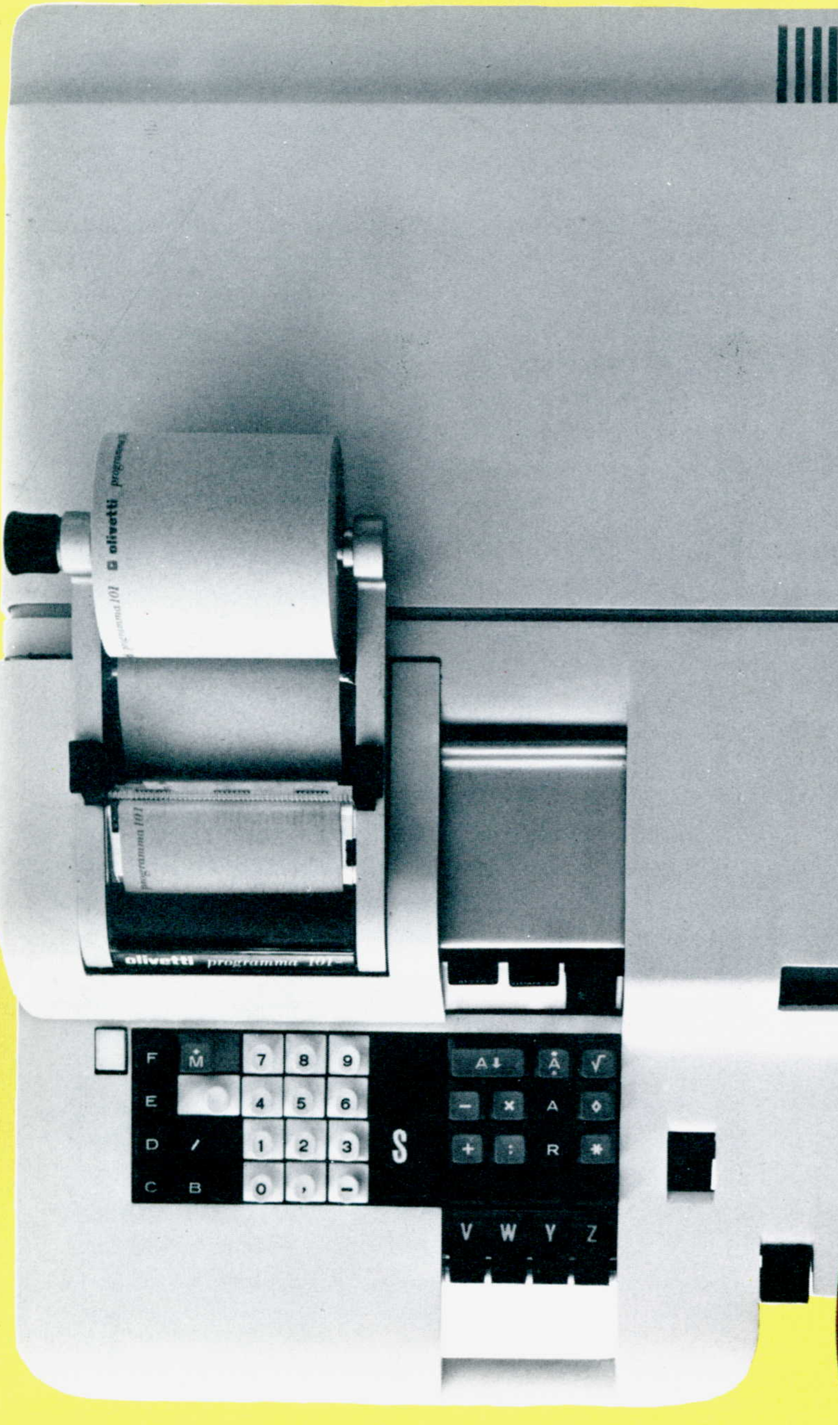
Much successful work has been done with desk-top computers or calculators but all need to be programmed in a low-level language and some of these do not fulfil the criteria set out above. These machines do not usually allow for the solution of non-numeric problems, which may be desirable for some children, and have very limited storage so that even the simplest data processing problems are impossible. However, these machines have the advantage that they can be used at any time and can easily be transported from classroom to classroom or even from school to school.

The Olivetti Programma 101 is one such machine that has been widely used with children and has been found most successful. Many other programmable calculators (from Hewlett Packard, Munroe, IME and a multitude of excellent Japanese manufacturers) will probably serve equally well. It would be necessary to compare the facilities offered by a number of them before choosing which one a school should buy.

The remote terminal provides a facility very suitable for use by children. It has the disadvantage that the time available for use may be limited for financial reasons. This could well be outweighed by the great advantage of a very simple language and the ability to tackle non-numeric and data processing problems. A simple subset of a language like BASIC is very easy for children to learn and the diagnostic assistance given by the computer if the programme contains errors is very helpful. The ability to edit a problem by a simple command is particularly valuable for use with young children. There is no reason why this type of facility should not be shared between schools.

hen choosing the facility to be used it is important to
member that pupil-time is as important as computer-time.
ne must also consider the objectives of the work and the
evelopment of this later on in school and beyond. The
ecise way in which the earlier work with the egg-box
mputer is arranged will depend on the follow-up with the
al machine. It is not at all difficult, for example, to arrange
at exactly the same language is used.

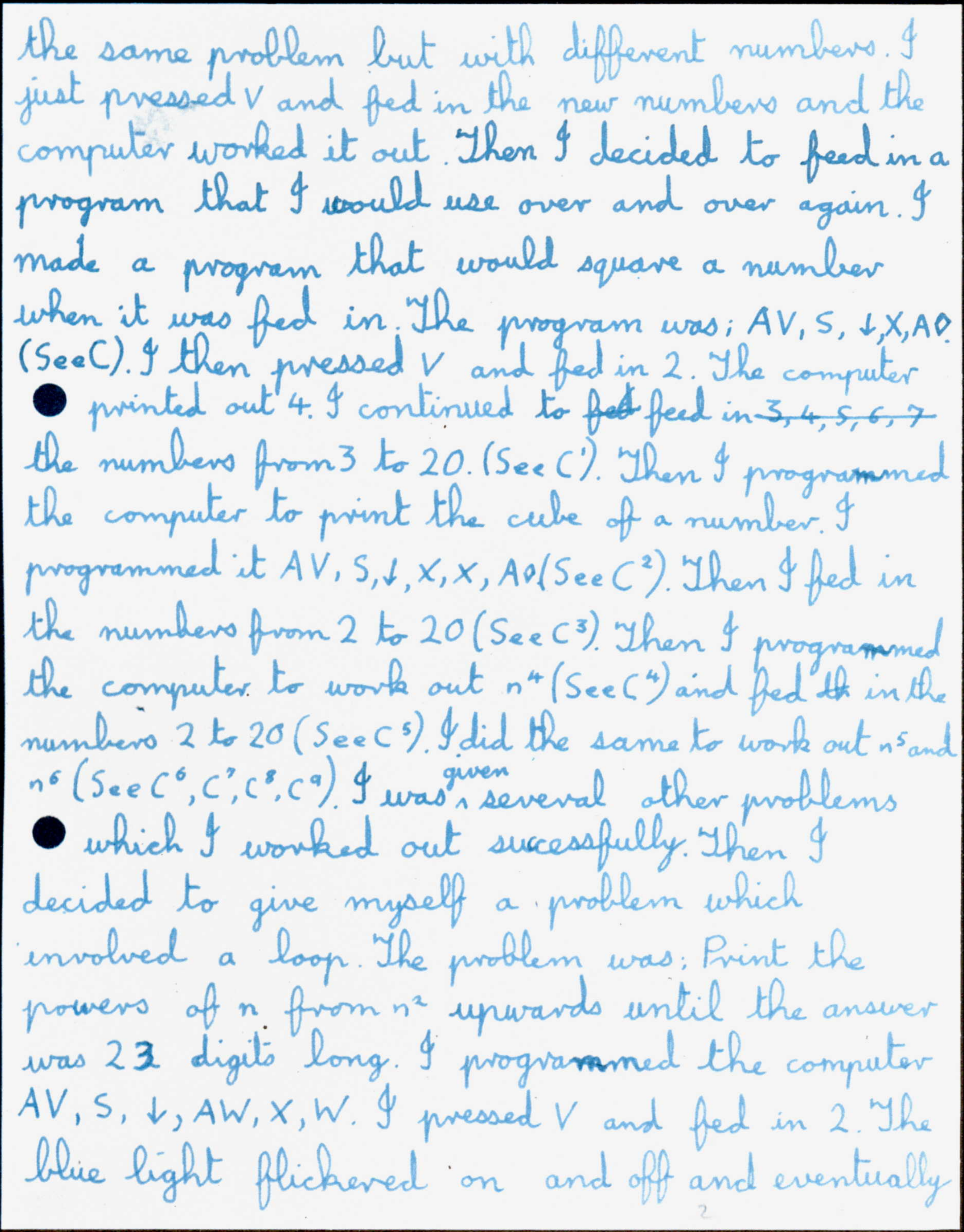
any important points are raised in the sections above and
would be wise to discuss these with experienced teachers
ior to making any decision on equipment. Computing in
hools is the concern of the Schools Branch of the
ational Computing Centre and also of the School
ommittee of the British Computer Society. Both will be
nly too pleased to help and can be contacted by writing to
ne Secretary, BCS Schools Committee, National
omputing Centre Ltd, Quay House, Quay Street,
anchester M3 3HH.



The following eight pages reproduce a Primary child's account of work done with a desk-top computer, the Olivetti Programma 101.

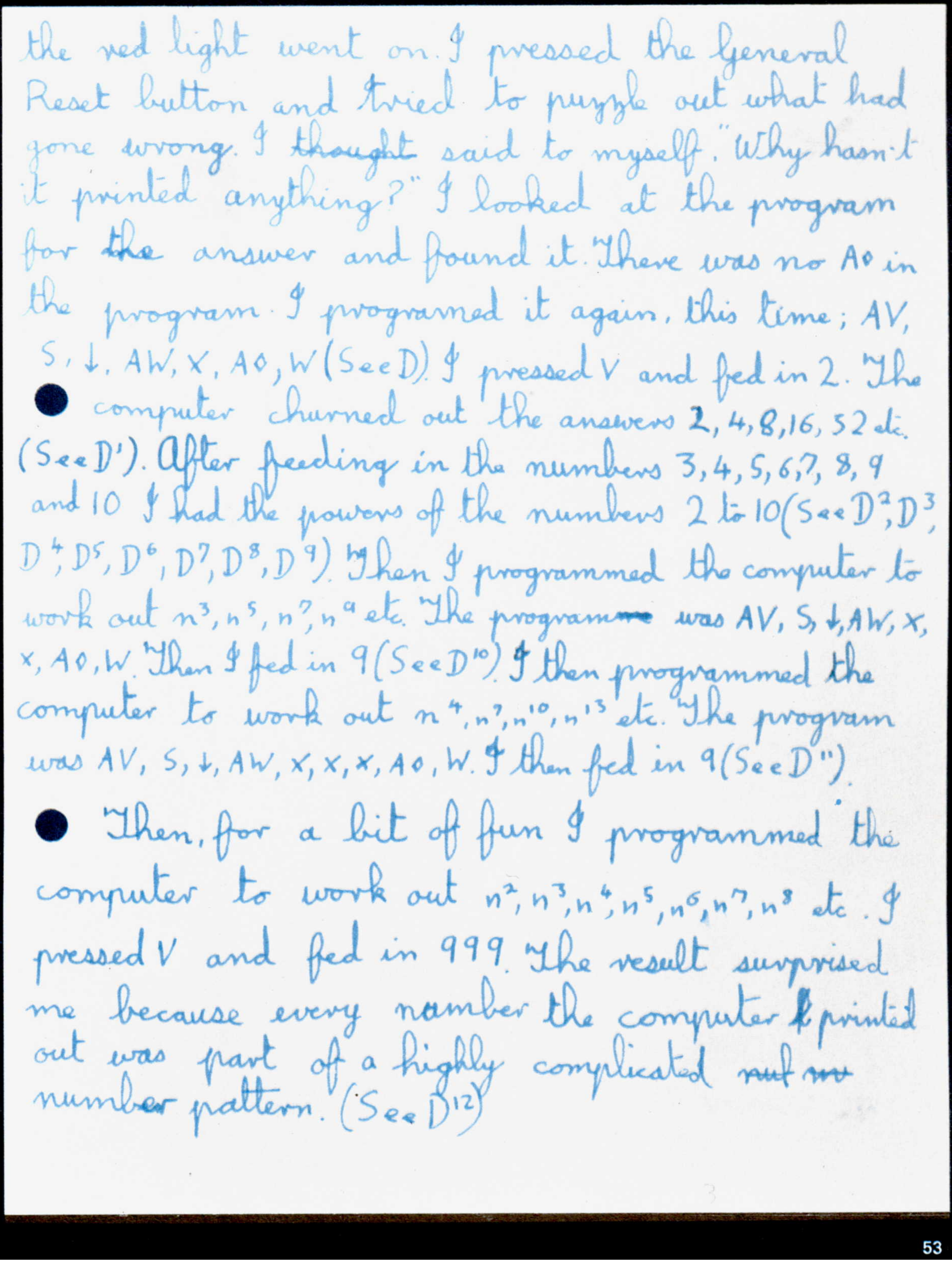
The references, e.g. (See A), are to the print-out from the computer which we have not reproduced as it seemed repetitive and not particularly interesting.

First of all, I was show~~ed~~n how to add, subtract, multiply, ~~and~~ divide and to find a square root using the computer as a calculator. I was then show~~ed~~n how to use the memory of the computer. Then I was given the instruction booklet and I read that and answered most of the questions in it correctly. I learnt how to program the computer and to use unconditional jumps and conditional jumps from the ~~ins~~ instruction booklet. I was then given my first problem. It was; $y = \frac{a + b \times c}{d}$ find y if $a = 4, b = 6, c = 2, d = 5$. I programmed the computer AV, S, ↓, S, +, S, X, S÷, A◊ (See A). I pressed V and fed in the numbers and the computer gave the correct answer, 4. Then I was given another problem, this time using the memory. $y = \frac{a(b+c)}{d+e}$ find y if $a = 2, b = 5, c = 8, d = 9, e = 4$. I programmed the computer; AV, S, B↑, S, ↓, S, +, BX, B/↕, S, ↓, S, +, C↕, B/↓, C÷, A◊ (See B). I pressed V and fed in the numbers and the computer gave the correct answer, 2. Later I realised that I could have programmed it AV, $\overset{(b)}{S}$, ↓, $\overset{(c)}{S}$, ~~X, B↕~~, ~~S~~, +, $\overset{(a)}{S}$, X, B↕, $\overset{(d)}{S}$, ↓, $\overset{(e)}{S}$, +, C↕, B↓, C÷, A◊ (See B') which is slightly shorter but means feeding in the numbers b and c before a. I was then given



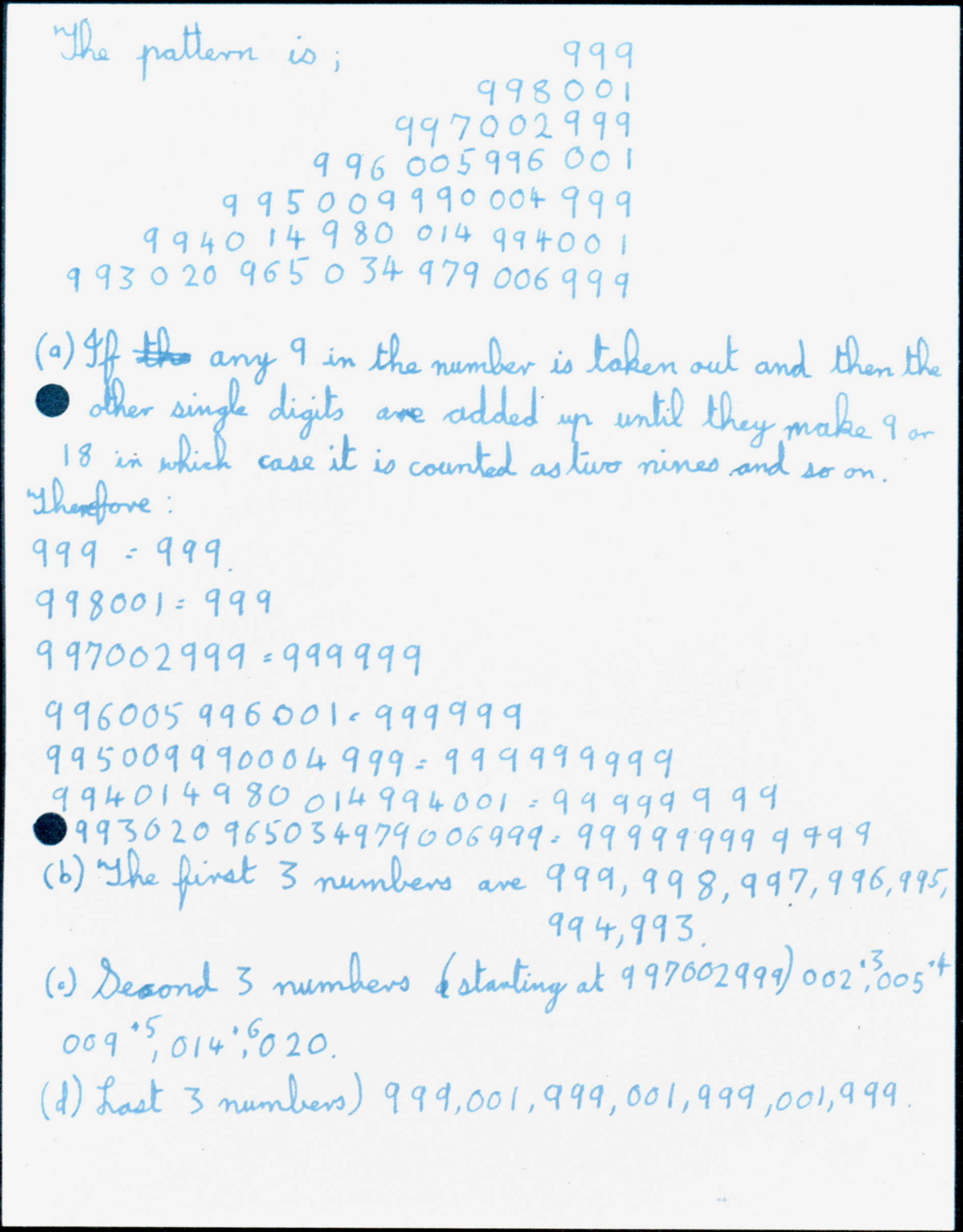
the same problem but with different numbers. I just pressed V and fed in the new numbers and the computer worked it out. Then I decided to feed in a program that I would use over and over again. I made a program that would square a number when it was fed in. The program was; AV, S, ↓, X, A0 (SeeC). I then pressed V and fed in 2. The computer ● printed out 4. I continued to ~~fet~~ feed in ~~3, 4, 5, 6, 7~~ the numbers from 3 to 20. (See C^1). Then I programmed the computer to print the cube of a number. I programmed it AV, S, ↓, X, X, A0 (See C^2). Then I fed in the numbers from 2 to 20 (See C^3). Then I programmed the computer to work out n^4 (See C^4) and fed ~~th~~ in the numbers 2 to 20 (See C^5). I did the same to work out n^5 and n^6 (See C^6, C^7, C^8, C^9). I was given several other problems ● which I worked out successfully. Then I decided to give myself a problem which involved a loop. The problem was; Print the powers of n from n^2 upwards until the answer was 23 digits long. I programmed the computer AV, S, ↓, AW, X, W. I pressed V and fed in 2. The blue light flickered on and off and eventually

2



the red light went on. I pressed the General Reset button and tried to puzzle out what had gone wrong. I ~~thought~~ said to myself, "Why hasn't it printed anything?" I looked at the program for ~~the~~ answer and found it. There was no A◊ in the program. I programed it again, this time; AV, S, ↓, AW, X, A◊, W (See D). I pressed V and fed in 2. The computer churned out the answers 2, 4, 8, 16, 52 etc. (See D'). After feeding in the numbers 3, 4, 5, 6, 7, 8, 9 and 10 I had the powers of the numbers 2 to 10 (See D^2, D^3, D^4, D^5, D^6, D^7, D^8, D^9). ~~My~~ Then I programmed the computer to work out n^3, n^5, n^7, n^9 etc. The program~~me~~ was AV, S, ↓, AW, X, X, A◊, W. Then I fed in 9 (See D^{10}). I then programmed the computer to work out n^4, n^7, n^{10}, n^{13} etc. The program was AV, S, ↓, AW, X, X, X, A◊, W. I then fed in 9 (See D'').

Then, for a bit of fun I programmed the computer to work out n^2, n^3, n^4, n^5, n^6, n^7, n^8 etc. I pressed V and fed in 999. The result surprised me because every number the computer ~~I~~ printed out was part of a highly complicated ~~nuf nu~~ number pattern. (See D^{12})



The pattern is ;

999

998001

997002999

996 005996 001

995009990004 999

9940 14 980 014 994001

993020 965 034 979 006999

(a) If ~~the~~ any 9 in the number is taken out and then the other single digits are added up until they make 9 or 18 in which case it is counted as two nines and so on.

Therefore :

999 = 999.

998001 = 999

997002999 = 999999

996005 996001 = 999999

995009990004 999 = 999999999

9940149 80 014 994001 = 99999999

993020 965034979006999 = 999999999 9999

(b) The first 3 numbers are 999, 998, 997, 996, 995, 994, 993.

(c) Second 3 numbers (starting at 997002999) 002^{+3}, 005^{+4} 009^{+5}, 014^{+6}, 020.

(d) Last 3 numbers) 999, 001, 999, 001, 999, 001, 999.

(e) Next to last 3 numbers $\underbrace{\overset{-2}{998},\overset{+2}{002}}_{998+002 = 1000}, \underbrace{\overset{-2}{996},\overset{+2}{004}}_{996+004 = 1000}, \underbrace{\overset{-2}{994},\overset{+2}{006}}_{994+006 = 1000}$

(f) Third set of 3 numbers

$-3(+3) = -6(+4)\ -10(+5)\ -15$

999, 996, 990, 980, 965

(g) Starting at the bottom right hand corner working diagonally to the opposite side the numbers are

909609

Starting at the top right hand corner working diagonally to the opposite side the numbers are

909604(9)

When the pattern is written out backward there are more noticable pattern:

```
                    999
                 100899
              999200799
           100699500699
        999400099900599
     100499410089410499
999600979430569020399
```

I looked at the ~~se~~ powers of other numbers and found there were patterns.

After that I programed the computer so that when button V was depressed the computer ~~so that when~~ would print n^2, n^3, n^4, n^5, n^6, etc. and when button Y was depressed the computer would print $n^3, n^5, n^7, n^9, n^{11}$ etc. The program read; AV, S, ↓, AW, X, A0, W, AY, S, ↓, AZ, X, X, A0, Z.. (See E). Then I tested the program (See E'). It printed the powers ~~of~~ correctly.

Then I decided that I would ~~find~~ work out a program that would work out any powers of any numbers. The program was; AV, S, ↓, AW, BX, A0, W. (See F) Into B ~~&~~ a power of n is fed. Then ~~n is feed fed~~ V is depressed. Then n is fed in. If n was fed into B the computer will print out n^2, n^3, n^4, n^5 etc. (See F^2) If n^2 ~~was~~ is fed into B the computer will print out n^3, n^5, n^7, n^9 etc (See F^1, F^3). If n^3 is fed into B the computer will print out n^4, n^7, n^{10}, n^{13} etc. and so on (See F^4).

I then tried to work~~ing~~ out $\sqrt{n}, \sqrt[3]{n}, \sqrt[4]{n}, \sqrt[5]{n}$ etc. but after much trying I decided it was impossibl

Then I programmed the computer to work out the area of a circle. The program was; AV, S, ↓, X, BX, A0. Then I fed in 3.14285 B↑. Then I fed in the radius of a circle, in this case 4 ins. The computer typed out 50·28560 (See G) which I presume to be the correct answer. Then I programmed the computer to calculate the volume of a hollow cylinder ~~st~~ such as one of the rolls of paper the computer uses. The program was; AV, S, ↓, X, BX, S, C↑, CX, D↕, S, ↓, X, BX, CX, E↕, D↓, E -, A0 constants 3·14285B↑

I fed in the dimensions of one of the rolls of paper the computer uses; radius 1·75 ins
length 3·5 ~~ing~~ ins.
radius of the hole and piece of cardboard or plastic in the middle 0·4 ins

The computer gave the answer 31·92742 cub. ins (See H).

Then I decided I would work out a program that would work out the squares of 1 to whenever the computer was stopped. The program read AV, D↓, B +, D↕, D↓, A0, X, A0, A *, ~~H~~V. constants 1 B↑. I pressed V and the computer printed 1, 1, 2, 2, 3, 3, 4, 4, 5, 5 etc. I pressed the General Reset and tried to puzzle out what had gone wrong. The counter was working but the computer was not squaring the numbers. Then I rewrote the program with a slight alteration; AV, D↓, B +, D↕, D↓

A◊, DX, A◊, A∗, MV. constants 1 B↑. (See I) I pressed V and the computer printed out the correct answers (See I'). I Then I decided to program the computer to ~~do a~~ work out a simple problem, work out the square roots of the numbers from 20 descending. I programmed the computer: AV, C↓, B-, C↕, C↓, A◊, C√, A◊, A∗, MV. constants 21 C↑, 1 B↑. (See J). I pressed V and the computer printed the answers correctly until it came to negative numbers. The computer then gave the answer as if ~~it~~ the number was a positive number but I did not excpect the computer to give any correct square roots of negative numbers as negative number cannot have square roots. (See J'). I then programmed the computer so that it would work out the cubes of the numbers from 1 upwards. The program ~~was~~ is; AV, C↓, B+, C↕, C↓ A◊, CX, CX, A◊, A∗, MV. constants 1 B↑. (See K). The computer worked out the problem successfully.

n account from three schools in Cambridgeshire

ne Olivetti Programma 101 was used for a period of six eeks. During this time the computer was shared by three hools. Although this necessitated a great deal of carrying out, the machine gave no trouble and proved to be robust nd reliable.

uxford Primary School

ne children were shown the simplest operations of the mputer as an electronic calculator and were quickly able to e this aspect of the machine to full advantage.

ater they were introduced to the simple whys and hows of ogramming and they programmed $A + B$, $A - B$, $A \times B$ nd $A \div B$. This led to $(A + B) \times C$ and then to $(A + B) \times C$ here C is a constant. By now the children were programming ithout any assistance whatsoever, and using the operations $+$, $-$, $\times$, and $\div$ in combination. The next stage was to ing in decisions i.e. if the answer is positive print 999, if egative print 666, and looping techniques. This was quickly asped and presented little difficulty. Finally each child xplained the fundamentals of the machine to a friend and a art was made by them in using the machine for simple ogramming.

onclusion : No piece of equipment has motivated more gument and discussion than the Olivetti Programma 101.

ne logical arguments and reasoning of the group were urprisingly profound – they quickly realised the need for a gical step-by-step program if correct results were to be und and this really set them thinking, often aloud, and onstructively criticising another child's argument.

hese young children can master all the techniques built into e machine. Their use of these techniques is only confined by eir lack of mathematical knowledge.

he machine proved to be a painless – indeed pleasurable – troduction to algebra. They were working on sound athematical lines without resort to numbers in order to rogram the machine. Then they fed numbers into the algebra generate arithmetic.

he glimpse into the computer age in which they will live as dults had a marked effect on the group. They could easily oresee the age of instant information.

Cottenham Village College

Three maths sets were involved and an attempt was made to assess the possibilities of using the 101 in a class situation as opposed to small group work.

It was decided to treat the experiment fairly formally and each of the sets followed roughly the same approach as detailed below :

Stage 1 Introduction to computer. A look at the five operations. Practice in working these operations ($+$, $-$, $\times$, $\div$, $\sqrt{}$) in decimals, using pencil, paper and hand calculators.

Stage 2 Useof the Programma 101 as an electronic calculator.

Stage 3 Learning to write simple programs to carry out straightforward calculations such as a $\frac{(b + c)}{c}$, $a^2 - b^2$ $(a + b)^2 + c$, etc.

Stage 4 Running these simple programs on the 101 and also working the calculations on hand calculators.

Stage 5 Learning to write programs with unconditional jumps. Here the youngsters were left to work on their own or in small groups. The emphasis here was to produce series of numbers e.g. $y = kx$, $y = x^2$, $y = x^3$,
Fibonacci series, $\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{2}{5}$ . . . . . etc.

Stage 6 Running these programs. Correcting program faults. Attempting to improve and shorten successful programs.

Stage 7 Introduction to conditional jumps. At this stage all the work was on an individual or small group basis.

Here are extracts from a few of the pieces of writing produced by pupils at the end of the experiment.

Angela I pictured the computer as a great huge thing with flashing lights and a voice coming from somewhere inside . . . . . At first I found programs hard to understand but I now feel able to write one without having to worry about it. If I had another chance to work on a computer I would quickly take it.

Dianne Dianne wrote a program to give this series
1, 2, 6, 24, 120 1124000727777607680000,
'and it got that big after only 22 terms'.

David I thought that the computer that was coming would be in the back of a lorry and that there would be hundreds of knobs and levers, but when it did come it was about 3 ft. by 3 ft. and 1 ft. and it was much smaller than I expected. You could work out any of the tables you did not know and it would go on writing it out until you press a button to stop it. I worked out the 671 times table and we let it give us twenty answers.

John John wrote a program to print out 1, 11, 111, 1111, 11111, etc. He also wrote the following:

With a gentle purr and a clickety click,
We put the program in.
The computer got to finish its work
Before the clock struck ten.
But Oh dear! What's gone wrong?
The little red light comes flashing on.

(The red light appears when the stores are overloaded or when the machine is asked to do the impossible e.g. divide by zero.)

Geraldine The best part of working with the computer was when we wrote our own programs. It was good putting the programs into the computer and it was amazing how fast the computer worked out the problems. When the computer could not take any more numbers it went red.

We are certain that there is a real place for computer work in school, and we are certain that a computer like the 101 is the type of machine best suited to our needs; we are not certain that we made the best use of it during this short experiment. There is one other uncertainty with far-reaching implications – what effect will the machine have on what we do and when we do it? There is little doubt that algebraic methods are implicit in using the machine, and that geometry need no longer be the branch of mathematics where logic is used.

The Grammar School for Boys

Programs, in which pupils have taken a lively interest, were made for a wide and varied list of computations. Very little time was given to using the machine as a calculator, our principal aim being to use it as a computer (i.e. with a program stored).

There was an attempt to get elegance into our programs. Competitions were held to see which student could solve a problem using the smallest number of instructions.

The following list of programs gives some indication of the kind of ideas which interested the pupils.

Square roots, reciprocals with division, solving general quadratics by formula and by iteration, evaluation of series for the exponential, circular and hyperbolic functions to a required degree of accuracy, solution of cubics, quartics, etc. by various methods, graph plotting, solution of differential equation, evaluation of highest common factor, Fibonacci sequence with ratio of $\frac{U_r}{U_{r+1}}$, triangular numbers, Eudoxus' numbers leading to an evaluation of $\sqrt{2}$ (written by a 13-year old boy).

The machine was taught to play 'Nim' by writing a program for the game, and programs were written to solve engineering problems and to write out prime numbers and cancel fractions. Problems involving calculations in astronomy were solved and numbers were sorted in order of magnitude. An interesting program was to feed in three numbers and ask the machine to determine whether or not a triangle can be drawn and, if so, to state whether it was acute, obtuse, or right angled.

Pupils were successful in producing programs to convert numbers from one base to any other base and £ s. d. to any foreign currency.

Before passing on to details of other approaches to computer practice in schools it should be said that the programs for the Olivetti Programma 101 mentioned in this section do not in any way show the full capabilities of the machine. Although its storage capacity is very limited in comparison with a large installation, there are probably few examples of problems likely to appear in school work where the capacity of the machine would be inadequate.

In many instances the limited storage will be an advantage in that the student will be obliged to plan his program with great care so as not to exceed the capacity of the machine. By so doing he will be writing programs with the maximum elegance and simplicity.

Conclusion

t seems likely that for some time to come computer education n schools will be the responsibility of the mathematics teachers but it is to be hoped that, as progress is made, it will soon cease to be a subject in itself, and will be integrated with many of the other subjects.

Already, in Universities, the computer is used by scientists, engineers, economists and sociologists, more than it is by the mathematicians. There are, in fact, few disciplines in which use is not being made of the computer. The very name – computer – is perhaps inappropriate since the greater part of he work done on computers today is some kind of information processing in which the mathematical content is small.

At the level of this book it may readily be seen that the preparation of flow charts can form a valuable stage in many different subjects. The formation of these ordered sequences of explicit instructions demands both imaginative and logical thinking. A high level of understanding is also required, and any deficiencies will often be quickly seen.

Publicity is often given to 'mistakes by computers', and these are held up as examples of their fallibility. The truth is, in nearly all cases, that the error is due to a fault in the instructions to the machine: it is a human error. The computer must have precise instructions on every detail of its assignment. A computer programmed to print accounts will quite happily send out a bill for £0-00 (and reminders about non-payment) unless it is specifically instructed not to do so.

t should be remembered that this book is concerned with computer understanding for pupils of a wide range of ability, not just a small selected set at the top end of the secondary school stage. Many pupils may not have the opportunity to progress beyond the level of this book but even then they should have some understanding of what a computer can do, or perhaps more important, what it can not do. It should be clear to them that the computer is nothing more than an instrument in the hands of man, totally dependent on human intelligence.

n the future, it is likely that computers will be used more and more to gain access to knowledge; to organise and abstract nformation; and to control routine mechanical procedures. This does not, however, point the way to a mechanical age in which the computer 'takes over'. The results produced by a computer will always be dependent on the instructions of people.

The following publications of the Nuffield Mathematics Project have appeared in 1967–72:

Introductory Guide

I do, and I understand ●■▼ (1967)
This Guide explains the intentions of the Project, gives detailed descriptions of the ways in which a changeover from conventional teaching can be made and faces many of the problems that will be met.

Teachers' Guides

Pictorial Representation ■1 (1967)
Designed to help teachers of children between the ages of 5 and 10, this Guide deals with graphical representation in its many aspects.

Beginnings ▼ (1967)
This Guide deals with the early awareness of both the meaning of number and the relationships which can emerge from everyday experiences of measuring length, capacity, area, time, etc.

Mathematics Begins ❶ (1967)
A parallel Guide to *Beginnings* ▼, but more concerned with 'counting numbers' than with measurement. It contains a considerable amount of background information for the teacher.

Shape and Size ▼2 (1967)
The first Guide concerned principally with geometrical ideas. It shows how geometrical concepts can be developed from the play stage in *Beginnings* ▼ to a clearer idea of what volume, area, horizontal and symmetrical really mean.

Computation and Structure ❷ (1967)
Here the concept of number is further developed. A section on the history of natural numbers and weights and measures leads on to the operation of addition, place value, different number bases, odd and even numbers, the application of number strips and number squares.

Shape and Size ▼3 (1968)
Continues the geometrical work of ▼2. Examination of two-dimensional shapes leads on to angles, symmetry and patterns, and links up with the more arithmetical work of ❷

Computation and Structure ❸ (1968)
Suggests an abundance of ways of introducing children to multiplication so that they will understand what they are doing rather than simply follow rules.

Graphs Leading to Algebra ■2 (1969)
This Guide develops the use of co-ordinates and introduces open sentences and truth-sets. It goes on to deal with the graphical aspect of these mathematical statements, introducing graphs of inequalities, intersection of two graphs and graphs using integers.

Computation and Structure ❹ (1969)
The main concern here is with the introduction of the integers {.... $^-3$, $^-2$, $^-1$, 0, $^+1$, $^+2$, $^+3$, ...}. This Guide builds up the idea of the integers in terms of ordered pairs of numbers before introducing the number line and other applications: this lays a sound foundation for operations on integers. It ends with a short section on large numbers and indices.

Shape and Size ▼4 (1971)
This Guide introduces the idea of a vector, not as something purely abstract, but as a simple and effective aid to developin geometrical insight in young children. The concept of addition of vectors is built up very slowly, being abstracted (in the tradition of the Project) from a variety of practical experiences. The link is shown between vectors and translations.

Weaving Guides

Desk Calculators ○□▽ (1967)
Points out a number of ways in which calculators can be used constructively in teaching children number patterns, place value and multiplication and division in terms of repeated addition and subtraction.

How to Build a Pond ○□▽ (1967)
A facsimile reproduction of a class project.

Environmental Geometry ○□▽ (1969)
This Guide concentrates on making children more critically aware of shapes in their environment and the interrelationship of them. It is intended mainly for Infants and lower Juniors.

Probability and Statistics ○□▽ (1969)
Designed to build up, in a very practical way, a critical approach to statistical information and assertions of probability. It demonstrates the many ways in which data can be collected and organised. Probability is introduced largely through games, but ways of predicting probable outcomes are investigated in detail.

Computers and Young Children ○□▽ (1972)
This is an introduction to the thinking behind computers rather than to the mechanics of them. It covers flow diagrams, punched cards and games in which children simulate a computer and ends with a description of work done in a few schools with an actual computer. It includes work for Juniors and lower Secondary children and is intended to be used from time to time rather than as a concentrated course.

Check-up Guides

Checking up 1 (1970)
This book has been prepared in co-operation with Piaget's Institut des Sciences de l'Education in Geneva. It deals with the various concepts leading towards the idea of number which are covered in the Teachers' Guide *Mathematics Begins* ❶. It explains the relevance of these concepts and gives the teacher guidance on how to check up on whether or not a child has acquired each concept.

Other publications

The Story So Far (1969)
The booklet is an outline of, and index to, the ground covered by the first nine Teachers' Guides of the Project. Its purpose is twofold: to provide easy reference to topics in these Guides for those using them day by day (making a straight index proved an impossible task); and to save teachers of older children from having to read through all the early Guides to find out 'what had happened previously'.

Into Secondary School (1970)
This booklet is intended for teachers of children from 11 upwards, whether in Secondary or Middle schools. Illustrated with stills from the film of the same name, it describes the aims of the Nuffield Mathematics Project as it affects these children, and so is complementary to the Introductory Guide *I do, and I understand*, which explains the philosophy of the Project with special reference to Primary schools.

Problems – Green Set (1969)
This publication consists of a Teachers' Book accompanied by a set of fifty-two cards for distribution to the children.

The set of Problems is intended for use with young Secondary pupils. The problems on the cards are reprinted in the Teachers' Book, with solutions and a considerable amount of background material and suggestions for follow-up work. All the topics covered by these cards are included in the Teachers' Guides already published, but they are presented in such a way that children who have not followed a 'Nuffield-type' course can do the problems and enjoy them.

Problems – Purple Set (1971)
These problems, like the Green Set and the Red Set, are intended for use with young Secondary children. Although this is the third Set to be published, in mathematical sophistication it should rightly come between the Green Set and the Red Set. As in the other two Sets, the problems have been printed on cards. Commentary on the problems and suggestions for follow-up work are included in the Teachers' Book.

Problems – Red Set (1970)
This second set of problems is designed for lower Secondary children. Like the Green Set it consists of a pack of cards for the pupils and a Teachers' Book in which the cards are reproduced. The mathematics covered by the Red Set is rather more sophisticated than that in the Green Set, and many of the cards could well be used with older children.

The Teachers' Book contains the solutions to the problems.

Maths with Everything (1971)
This booklet has the same title as a film made for the Nuffield Mathematics Project about children aged 5 to 7. The purpose of the booklet, and of the film, is well summed up in the commentary: 'It's a question of knowing where to look', and what to look for. The teacher who can be aware of the many opportunities for mathematical experiences and can make the most of those within her reach, will be doing her very best for the children; and 'maths with everything' will help them forward in their development as active and thoughtful people.

Consultative committee

Chairman Professor W H Cockcroft
J W G Boucher
R C Lyness
Miss B M Mogford (1964–1966)
H S Mullaly (from 1966)
R Openshaw
N Payne (from 1967)
D R F Roseveare
J Shanks (from 1966)
A G Sillitto (died 1966)
P F Surman
Dr D R Taunt
Mrs D E Whittaker (from 1967)
F Woolaghan
Professor J Wrigley

Organiser

Professor G. Matthews

Team members

1964 – 1966
J W G Boucher
G B Corston
H Fletcher
Miss B A Jackson
D E Mansfield
Miss B M Mogford

1966 – 1967
D R Brighton
Miss I Campbell
H Fletcher
D E Mansfield
J H D Parker
Miss R K Tobias
A G Vosper

1967 – 1968
E A Albany
D R Brighton
Miss I Campbell
Miss R K Tobias
A G Vosper

1968 – 1969
E A Albany
D R Brighton
A G Vosper

1969 – 1970
E A Albany
D E Jones
J H D Parker
A G Vosper

Designers

Dodd & Dodd

0 550 77023 2 (Chambers)
0 7195 2653 1 (Murray)
0 471 65179-6 (Wiley)