Experimental computer for schools

D. M. Taub, M.Sc., F.B.C.S., C.Eng., F.I.E.E., C. E. Owen, M.A., F.B.C.S., C.Eng., F.I.E.E., and B. P. Day, B.Sc.

Abstract

The computer is a small desk-top machine designed for teaching schoolchildren how computers work and how to use them. It works in decimal notation and has a powerful instruction set which includes 3-address floating-point instructions implemented as 'extracode' subroutines. Addressing can be absolute, relative or indirect. For input it uses a capacitive touch keyboard, and for output and display, a perfectly normal t.v. receiver is used. Another input/output device is an ordinary domestic tape recorder, used mainly for long-term storage of programs. To make the operation of the machine easy to follow, it can be made to stop at certain stages in the processing of an instruction and automatically display the contents of all registers and storage locations relevant at that time. The paper gives a description of the machine and a discussion of the factors that have influenced its design.

1 Introduction

In a little over 20 years, the digital computer has developed from a promising university project into one of the most important tools in science, industry, commerce and government. The use of computers is now growing at such a rate that within a few years every educated person will need to know something of how they work and how to use them.

Many teachers think that the basis of the subject should be taught while a child is still at school, and they have already taken the first steps in this direction. Children at primary schools already learn the essentials of the binary numbering system and binary arithmetic, but the subject of computers is probably too complex to be tackled in any depth until a child reaches secondary school.

Here, two different approaches have so far been tried. One is to introduce the subject from the physics standpoint through simple logic circuits, binary counters and adders. This is valuable in teaching children how the basic elements of a computer work, but in the time available it would be impossible to cover the organisation of a complete machine in the same detail.

The other approach concentrates on how computers are used, rather than how they work. Children are taught simple mathematical programming using one of the problem-oriented languages such as ALGOL or FORTRAN. Again, this is useful material; it encourages clear thought and precise expression, and gives a great stimulus to the mathematics course.

This approach too has its drawbacks. The main one is that the user remains isolated from the machine. It is possible for him to get results in a problem-oriented language without knowing anything about the machine itself; in other words, the machine remains a 'black box'.

The second drawback concerns expense. Very few schools can afford a machine large enough to process problem-oriented languages. This means sending the children's programs away, perhaps to a university or local-authority computer and, by the time they have been punched onto cards or tape, run on the computer and the results sent back to the school, several days will have passed. A child, like any other beginner, generally needs a few attempts before his program runs correctly, and the succession of delays can easily make him lose interest. A way round the problem is to put a computer terminal into each school and connect it online to a central machine, but this is expensive, and likely to remain so for some time.

In a sense, the two approaches just described represent extremes. The first considers only the fine detail of a machine and does not go as far as looking at the machine as a whole. The second, as far as it considers the machine at all, treats it as a single unit: a means of getting results.

The best approach, when it comes to understanding the computer itself, probably lies halfway between. The computer is first presented as a simple assembly of store, arithmetic unit, input and output devices, and control unit. The student starts to write programs in terms of this machine, and, as he progresses and needs more facilities, the picture of the machine is gradually built up to show how each facility is added. After dealing with very simple programs, branching is introduced, and later, indirect and relative addressing, and subroutine linkage. Essentially, the idea is that the student builds up his knowledge of the machine at the same time as he learns how to use it.

The success of this approach depends very much on having the right kind of machine available. What is needed is a machine which can be made to appear very simple at first, but which has the more advanced features available when the student is ready for them. Perhaps most important of all, however, it must be so designed that the student can follow exactly what the machine is doing; this means paying special attention to the display.

Apart from the use already outlined, a computer of this kind would be a valuable demonstration tool in the mathematics course. There are several important topics that can only be touched on at present because they call for too much computation; e.g. iterative processes, summation of series and numerical integration. A machine with good display facilities would allow these topics to be presented much more vividly.

During the past few years, we have been developing a machine along the lines proposed. The first prototype[1] was built in 1967 and teachers' reactions to it were very encouraging. Since then, six more models have been made and are now on loan to a group of schools for a year's thorough field trial. The aim is to use them in as many different types of school as possible and with children of widely differing abilities. In this way, we hope to find out whether the suggested approach is in fact sound.

The purpose of this paper is to describe the machine and to discuss the factors on which its design is based.

2 Main features

The design of the machine was strongly influenced by three factors:

(*a*) It had to be cheap, or schools would not be able to afford it.
(*b*) It had to be easy to use.
(*c*) It had to have very good display facilities.

One way of keeping the cost down was to restrict the size of the store. This meant that the machine would not be able to run compiler programs, and programming would have to be done either in machine language or in a language that could be translated very easily into machine language.

Bearing this in mind, one of the main factors contributing to (*b*) and (*c*) was to design the machine to work in decimal notation. The input and the display are both in conventional decimal form, though within the machine a binary-coded decimal representation is used.

Paper 6069 E, received 23rd October 1969
The authors are with IBM UK Laboratories Ltd., Hursley Park, Winchester, Hants., England

2.1 Basic instructions and 'extracode' instructions

The main parts of the machine are a store, a group of registers, and an adder/subtractor unit. The instruction set contains instructions for moving data between the store and registers and for carrying out addition, subtraction, shifting and testing operations on the register contents.

Using the basic instructions, any of the more complex processes such as floating-point division can be built up. It would be very tedious though if every program had to be written at such a detailed level; so the machine also has a set of 3-address floating-point instructions. These are implemented in the same way as the 'extracode' instructions used in Atlas;[5] i.e. by automatically calling subroutines of basic instructions held in a part of the store specially provided for the purpose.

When programming is done in extracode instructions, the machine is arranged to appear to the user as a 3-address machine; i.e. it does not display any of the detailed inter-register operations that are taking place. These are displayed only when the machine is programmed in basic instructions. The machine can thus be presented to the student at two quite different levels of detail, which should be a valuable feature. Experience so far suggests that the computer will be used mainly at the extracode level, and that students will not do much programming in basic instructions until they have reached a more advanced stage.

2.2 Keyboard input, t.v. output

The input and output were made as simple and inexpensive as possible. For the initial input of programs and data, the machine has a keyboard using capacitive touch keys. Their main advantage over ordinary contacts is ruggedness, making them more able to stand up to school use.

For output and display, the machine uses a perfectly normal television receiver. This has a number of very attractive features:

(*a*) The t.v. set is mass-produced, and is therefore very much cheaper than a specialised piece of computer hardware.
(*b*) It is cheaper to maintain because skilled people can be found wherever there is a t.v. service.
(*c*) The screen is large enough to be used for demonstration to a whole class.

2.3 Tape recorder

Another input/output device used by the machine is an ordinary domestic tape recorder. When a teacher wants to load a demonstration program at the beginning of a lesson, it would take too much time if he always had to enter it via the keyboard. Therefore, having entered it once, he can record it on tape and reload it into the machine whenever he needs it. The tape recorder can be useful too if a class is doing a piece of work that cannot be completed in a single lesson. At the end of the lesson, the contents of the whole store are recorded, leaving the machine free for use by another class. When the first class is ready to resume work, the store is reloaded. As far as cost and maintenance are concerned, the tape recorder has the same attractions as the television set.

The computer, its t.v. set and tape recorder are illustrated in Fig. 1.

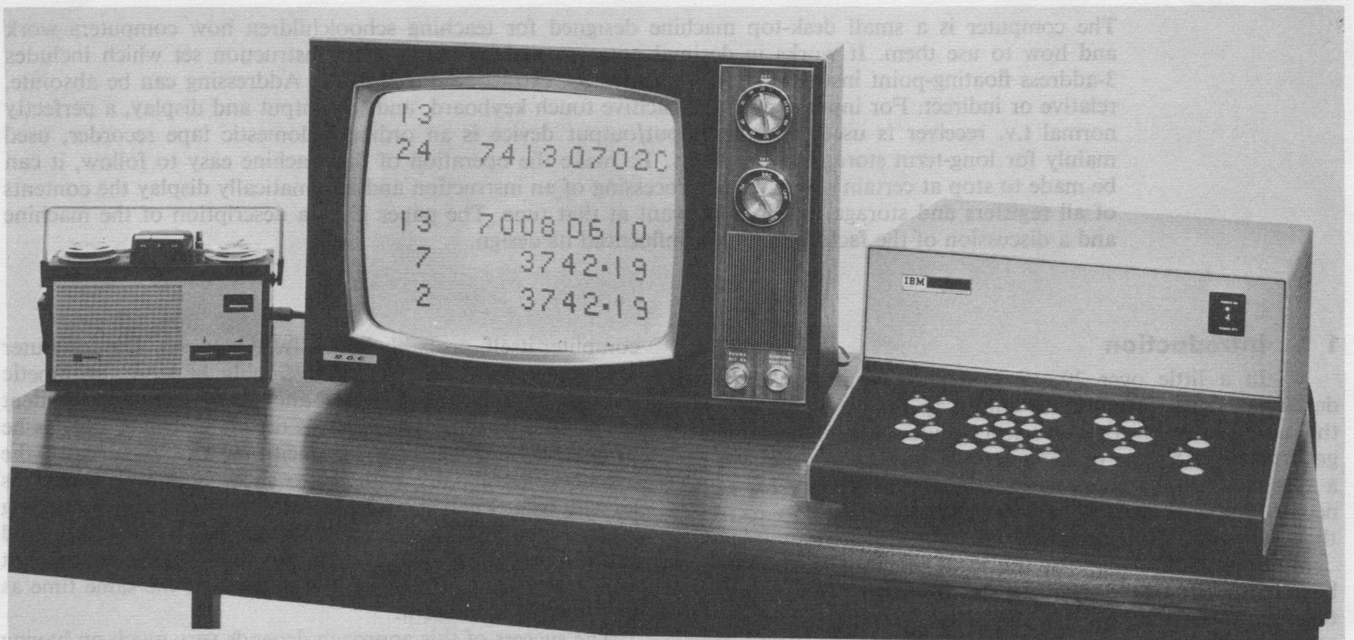

Fig. 1
Complete installation

3 Machine organisation

The information given in this Section relates to the machine as it appears to the user, rather than to the engineering implementation. For example, the main store and the various general-purpose and special-purpose registers are listed separately, though in fact they are all housed in the same matrix of magnetic cores. Also, the word length in the main store is given as eight decimal digits, but, as far as implementation is concerned, only one decimal digit is read or written at a time. Details of the engineering implementation are given in Section 7.

3.1 Main store

The main store has a capacity of 200 words, each being eight decimal digits long. The first hundred words (addresses 0 to 99) are for users' programs and data, and the second hundred (addresses 100 to 199) are for the subroutines that implement the extracode instructions.

3.2 Data format

The way in which the eight digit positions in a word are used depends on whether the word represents an instruction or a piece of numerical data. When it represents an instruction, all eight positions store decimal digits. When it represents a number, the first digit position stores the sign, the second stores the number of decimal places, i.e. the negative of the exponent, and the remaining six store the significant digits. The format is shown in Fig. 2*a*.

The number of decimal places can have any value from 0 to 6. Thus, numbers can normally range from -999999 to $+999999$, the smallest finite number being $\pm 0{\cdot}000001$.

This range is much less than in most computers, but teachers consider, and experience has so far confirmed, that it is quite enough for school use. Numbers outside the range can be handled, but not so conveniently (see subcaption to Table 3). Negative numbers are always handled in 'sign and

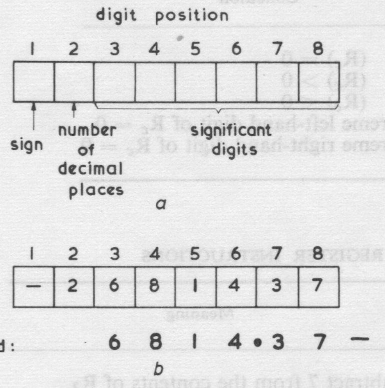

Fig. 2

Data format

a Allocation of digit positions
b Example

modulus' form to save the user from having to concern himself with complement notation.

Although numbers are stored as an integer and exponent, provided they lie within the normal range of the machine, they are entered on the keyboard and appear on the display in conventional form, with the decimal point in the appropriate place. Any negative sign is keyed in after the least significant digit and is displayed at the right-hand end (see Fig. 2*b*). Positive signs are not keyed in and do not appear on the display.

3.3 General-purpose registers

There are 10 general-purpose registers R_0–R_9. R_0–R_7 are single-length registers, each storing a sign and six decimal digits, and R_8 and R_9 are double-length registers, each storing a sign and 12 decimal digits.

3.4 Transfers between main store and registers

The general-purpose registers hold numbers only in integer form. Therefore, when transferring a number from the store to the registers, the integer and exponent portions have to be separated. This is done as follows:

(*a*) The READ STORE instruction causes the integer and sign of the required word to be copied into R_1 and the exponent into R_0.

(*b*) From that point onwards, they are manipulated separately using the register instructions (see Section 4.1).

(*c*) When a number is to be written back into the store, the integer must first be set up in R_1 and the exponent in R_0.

(*d*) The basic STORE DATA instruction combines the contents of these registers and writes them in the specified storage location according to the format of Fig. 2.

The above arrangements apply only to numerical data. When a student reaches a more advanced stage, he may want to carry out operations on instructions, and the machine has been designed to make this possible. The necessary transfers between store and registers still take place via R_0 and R_1, but the storage word has now to be split in a different way. Details of how this is done are given in Table 7.

3.5 Modifier registers

Registers R_3, R_4 and R_5 serve a dual purpose. As well as being general-purpose registers, they can be used as modifier registers. For basic single-address instructions, the instruction code specifies which of them, if any, is to be used. For 3-address instructions with relative addressing, the first address is modified by the contents of R_3, the second by the contents of R_4 and the third by the contents of R_5.

3.6 Keyboard register

All numbers and instructions entered on the keyboard are first held in the keyboard register. From here they can be transferred to various parts of the machine, depending on how they are to be used (see Section 6).

3.7 Instruction address registers

The machine has a slightly more elaborate arrangement than usual for keeping track of instruction addresses. Besides the normal instruction address register (i.a.r.), which holds the address of the next instruction to be fetched from the store, there is an auxiliary register which holds the address of the instruction currently being processed. The auxiliary register plays no direct part in the operation of the machine, but it helps the user when he is following through a program, instruction by instruction.

As soon as an instruction has been fetched from the store, its address is transferred from the i.a.r. to the auxiliary register and remains there until the following one has been fetched. Thus, when an instruction has just been completed, the display can show the address from which that instruction was read (held in the auxiliary register) and the address of the following one (held in the i.a.r.). This is especially valuable for BRANCH instructions.

3.8 Instruction register

The instruction register holds the instruction currently being processed.

3.9 Address registers

When relative or indirect addressing is used, the absolute address (or addresses, in the case of 3-address instructions) will be different from the address portion of the instruction; hence the need for extra address registers. There are three of these to cater for the 3-address case.

3.10 Link register

This is used for temporary storage of the link address in BRANCH AND LINK instructions. The basic instruction set includes an instruction for transferring its contents into the main store (Table 7).

3.11 Control latch

The control latch is a single-bit register that is set to 0 or 1 in response to TEST instructions on the contents of registers. The basic CONDITIONAL BRANCH instructions test the state of this latch to determine whether or not branching will take place.

4 Instruction set

As already mentioned, there are two main types of instruction:

(*a*) basic instructions
(*b*) extracode instructions

The basic instructions consist of four decimal digits each, and are stored two to a word as shown in Fig. 3(i). The extracode instructions have eight digits each and are stored one to a word as shown in Fig. 3(ii).

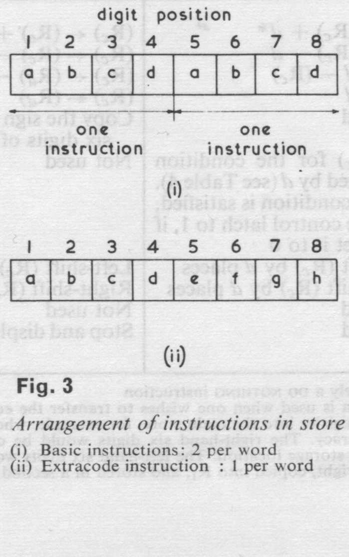

Fig. 3

Arrangement of instructions in store

(i) Basic instructions: 2 per word
(ii) Extracode instruction : 1 per word

The basic instructions can be subdivided into

(i) single-address instructions for moving data into and out of the store and for branching

(ii) register instructions for operating on the contents of registers.

The type of instruction is determined by the first digit *a* as shown in Table 1. This digit also shows how the remaining digits are to be interpreted, and, for single- and 3-address instructions, the type of addressing to be used; i.e. whether absolute, relative or indirect.

Table 1

INTERPRETATION OF DIGIT *a*

a	Type of instruction
0	Register instructions
1	
2	Single-address instructions
3	
4	
5	
6	
7	3-address extracode instructions
8	
9	

Details of the different types of instruction are given in Sections 4.1–4.3 and Tables 1–10. For brevity, the contents of a particular register or storage address are indicated by placing brackets round the register or address designation. Thus (R_3) means 'the contents of R_3', and (25) means 'the contents of storage location 25'. The arrow $\leftarrow$ is used as an assignment symbol. Thus, $(R_c) \leftarrow (R_c) + d$ means: set the contents of R_c to the sum of its existing contents and *d*.

4.1 Register instructions

In register instructions, digit *b* always specifies the operation and *c* a register number. The interpretation of digit *d* depends on whether *a* is 0 or 1, as shown in Table 2. The various operations are listed in Tables 3 and 4, and some examples are given in Table 5.

Table 2

INTERPRETATION OF DIGITS IN REGISTER INSTRUCTIONS

a	*b*	*c*	*d*
0	Operation	Register number	$b \neq 5$: literal* $b = 5$: condition to be tested (see Table 4)
1			register number

* In this case *d* is used simply as a numerical digit, it is not used as a code

Table 3

REGISTER INSTRUCTIONS: OPERATIONS

	Operation	
b	$a = 0$	$a = 1$
0	$(R_c) \leftarrow (R_c) + d$*	$(R_c) \leftarrow (R_c) + (R_d)$
1	$(R_c) \leftarrow (R_c) - d$	$(R_c) \leftarrow (R_c) - (R_d)$
2	$(R_c) \leftarrow d - (R_c)$	$(R_c) \leftarrow (R_d) - (R_c)$
3	$(R_c) \leftarrow d$	$(R_c) \leftarrow (R_d)$
4	Not used	Copy the sign and right-hand six digits of R_d into R_c†
5	Test (R_c) for the condition specified by *d* (see Table 4). If the condition is satisfied, set the control latch to 1, if not, set it to 0	Not used
6	Left-shift (R_c) by *d* places	Left-shift (R_c) by (R_d) places
7	Right-shift (R_c) by *d* places	Right-shift (R_c) by (R_d) places
8	Not used	Not used
9	Not used	Stop and display (R_c) and (R_d)

* 0000 is effectively a DO NOTHING instruction

† This instruction is used when one wishes to transfer the complete contents of a double-length register into the main store, for example, when working to more than 6-digit accuracy. The right-hand six digits would be copied into R_1 and from there into a storage location. The left-hand six digits would then be shifted six places to the right, copied into R_1, and stored in a second location

Table 4

TEST CONDITIONS USED IN REGISTER INSTRUCTIONS

d	Condition
0	$(R_c) = 0$
1	$(R_c) > 0$
2	$(R_c) < 0$
3	Extreme left-hand digit of $R_c = 0$
4	Extreme right-hand digit of $R_c = 0$

Table 5

EXAMPLES OF REGISTER INSTRUCTIONS

Instruction code	Meaning
0137	Subtract 7 from the contents of R_3
1351	Copy the contents of R_1 into R_5
0542	If the content of R_4 is less than 0, set the control latch to 1, otherwise set it to 0

4.2 Single-address instructions

The format of single-address instructions is shown in Fig. 4(i). Digit *a* specifies the type of addressing (discussed later), *b* the operation, and *c* and *d* an address. Digits *c* and *d* cannot specify an address completely, however, as they can define only 100 locations, whereas the main store has 200 (see Section 3.1). Therefore *c* and *d* are taken, respectively, as the tens and units digits of the address, and the hundreds digit is taken from the hundreds digit position of the i.a.r.

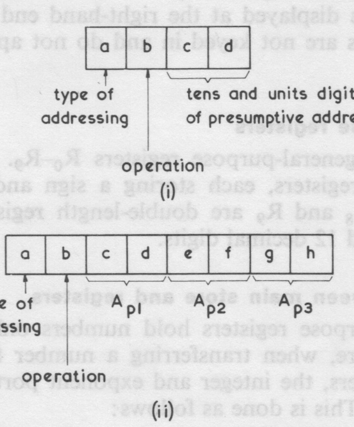

Fig. 4
Instruction formats
(i) Single-address
(ii) 3-address

This effectively divides the store into two areas, locations 0–99 and 100–199, and an instruction stored in either of these areas will always refer to an address in the same area. Users' programs are restricted to locations 0–99, and so they always refer to an address in which the hundreds digit is zero. To the user, therefore, the *c* and *d* digits of an instruction represent the complete address.

When relative or indirect addressing is used, the address

Table 6

FORMATION OF ABSOLUTE ADDRESS IN SINGLE-ADDRESS INSTRUCTIONS

a	Type of addressing	A_a
2	Absolute	A_p
3	Relative	$A_p + (R_3)$
4		$A_p + (R_4)$
5		$A_p + (R_5)$
6	Indirect	(A_p)

A_p is the presumptive address; i.e. the address specified in the instruction
the absolute address

Table 7

SINGLE-ADDRESS INSTRUCTIONS: OPERATIONS $a = 2$–6 INCLUSIVE

b	Operation
0	Copy (A_a) into R_0 and R_1 as follows (see Fig. 1). If A_a contains data (recognisable by a sign in digit position 1), copy the exponent digit into R_0 and the sign and significant digits into R_1. If A_a contains an instruction (recognisable by a decimal digit in position 1), copy the left-hand four digits into R_0 and the right-hand four digits into R_1
1	Store (R_0) and (R_1) in location A_a in data format; i.e. copy (R_0) into the exponent position, and copy (R_1) into the sign and significant digit positions. If the number is outside the range that can be held in a storage location, set the ERROR indicator and stop
2	Store (R_0) and (R_1) in location A_a in instruction format; i.e. copy (R_0) into the left-hand four digit positions and (R_1) into the right-hand four digit positions
3	Not used
4	Unconditional branch to A_a. Store link address in link register*
5	Branch to A_a if control latch is set to 1. Store link address in link register
6	Branch to A_a if control latch is set to 0. Store link address in link register
7	Store the contents of the link register in A_a
8	Input Stop to allow the operator to enter data. When the machine is restarted, transfer the contents of the keyboard register into A_a
9	Display Stop and display (A_a)

* When two basic instructions are stored in one storage location, branching can take place only to the first of them [see Fig. 3(i)]

Table 8

EXAMPLES OF SINGLE-ADDRESS INSTRUCTIONS

Instruction code	Meaning
2179	Store (R_0) and (R_1) in location 79 in data format
4513	Form the absolute address A_a by adding (R_4) to 13. Branch to A_a if the control latch is set to 1
6984	Read (84) to determine A_a. Stop and display (A_a)

given in the instruction is not the same as the absolute address of the operand. The procedure for forming the absolute address is shown in Table 6. The address given in the instruction (i.e. the c and d digits and the hundreds digit of the i.a.r.) is known as the presumptive address A_p and the absolute address is denoted by A_a.

There is a safeguard built into the machine to prevent presumptive addresses in the range 0–99 from giving rise to absolute addresses greater than 99. The reason is that users' programs must not be allowed to write into locations 100–199; they could otherwise overwrite the extracode subroutines stored there. For relative addressing, the safeguard is provided by ignoring any carry from the tens digit position when address modification takes place, and for indirect addressing, the safeguard is provided by taking only the tens and units digits of (A_p).

The single-address operations are listed in Table 7 and a few examples given in Table 8.

4.3 3-address extracode instructions

The format for the 3-address instructions is shown in Fig. 4(ii). The digit a specifies the type of addressing as shown in Table 9, and b specifies the operation. These instructions

Table 9

FORMATION OF ABSOLUTE ADDRESSES IN 3-ADDRESS EXTRACODE INSTRUCTIONS

a	Type of addressing	A_{a1}	A_{a2}	A_{a3}
7	Absolute	A_{p1}	A_{p2}	A_{p3}
8	Relative	$A_{p1} + (R_3)$	$A_{p2} + (R_4)$	$A_{p3} + (R_5)$
9	Indirect	(A_{p1})	(A_{p2})	(A_{p3})

A_{p1}, A_{p2} and A_{p3} are the three presumptive addresses
A_{a1}, A_{a2} and A_{a3} are the corresponding absolute addresses

and their operands may be held only in locations 0–99 of the store; therefore only two digits are needed to specify eacn address. The digits c and d are the tens and units digits of the first address, e and f the second address, and g and h the third. Absolute addresses greater than 99 are barred, as in single-address instructions.

The way the machine carries out an extracode instruction is by automatically branching to a subroutine held in storage locations 100–199. Before branching takes place, the absolute addresses are determined and stored in locations 100, 101 and 102, respectively, as shown in Table 10, and the link

Table 10

ENTERING AND RETURNING FROM EXTRACODE SUBROUTINES

Address	Contents
100	A_{a1}
101	A_{a2}
102	A_{a3}
103	Link address
—	
—	
—	
—	
110	2703 2420
111	2703 2421
112	2703 2445
etc.	
—	
—	
—	
120	First instructions in routine called by $b = 0$
121	First instructions in routine called by $b = 1$
—	
—	
—	
145	First instructions in routine called by $b = 2$
—	
—	
—	
190	6403 0000; i.e. return to main program

address is stored in the link register. Branching then takes place to an address formed by adding 110 to the b digit of the instruction. Thus, if $b = 0$, branching takes place to 110, if $b = 1$ branching takes place to 111 etc. All the above operations take place automatically; they do not have to be programmed by the user.

The first instruction in the branch location moves the link address into a storage location. In the example in Table 10, the link address is moved into location 103, but the person writing the routines could choose any location from 103 to 199. The second instruction branches to the beginning of the appropriate routine. During the routine, the operands are obtained by indirect addressing using the presumptive addresses 100, 101 and 102, and when returning to the main program the link address is obtained in a similar way.

The subroutines can be written into the store from the

Table 11

SUGGESTED EXTRACODE OPERATIONS FOR FLOATING-POINT CALCULATIONS

b	Operation
0	$(A_{a1}) \leftarrow (A_{a2}) + (A_{a3})$
1	$(A_{a1}) \leftarrow (A_{a2}) - (A_{a3})$
2	$(A_{a1}) \leftarrow (A_{a2}) \times (A_{a3})$
3	$(A_{a1}) \leftarrow (A_{a2}) \div (A_{a3})$
4	Branch to A_{a1} if $(A_{a2}) = (A_{a3})$
5	Branch to A_{a1} if $(A_{a2}) > (A_{a3})$
6	Branch to A_{a1} if $\lvert(A_{a2})\rvert > \lvert(A_{a3})\rvert$*
7	If $(A_{a3}) \neq 0$, branch to A_{a1} and store link address in A_{a2}
8	Input and display Stop, to allow the operator to enter data. When the machine is restarted, transfer the contents of the keyboard register into A_{a1}. While the machine is stopped, it displays (A_{a1}), (A_{a2}) and (A_{a3})
9	Display Stop and display (A_{a1}), (A_{a2}) and (A_{a3})

$a = 7$ 8 or 9
* This instruction is useful for determining whether a computation has been carried to a required degree of accuracy

keyboard; therefore the user is perfectly free to choose his own extracode operations as long as the subroutines will fit into the available storage space. At a more advanced stage, users will sometimes want to do this, but, for the general run of mathematical work, practically every need can be met by the suggested set of floating-point operations listed in Table 11.

5 Data-flow diagrams

Figs. 5 and 6 show the main parts of the machine and the data-flow paths. They represent the machine as it appears to the user; in other words, the paths shown do not necessarily correspond to electrical connections.

Figs. 5 and 6 show how the machine appears when using basic and 3-address instructions, respectively. Fig. 6 is the simpler of the two, but even this would appear complicated to a schoolchild meeting it for the first time. The suggestion is, therefore, that the diagram should be built up step by step as the child's knowledge of the machine increases. This is the approach taken in Reference 2.

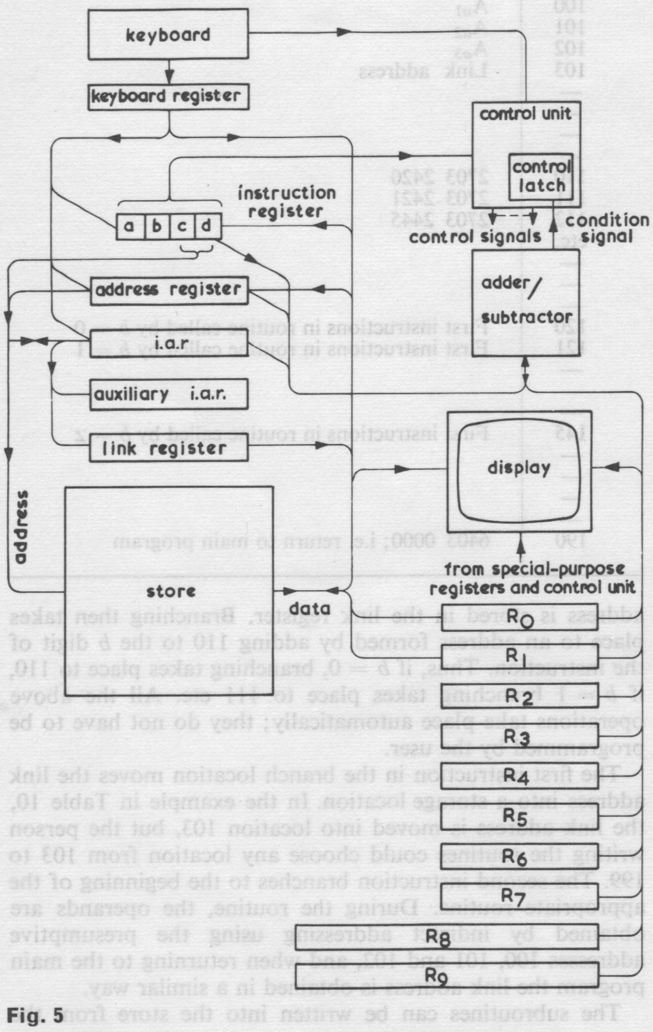

Fig. 5
Main data-flow paths used in basic instructions

6 Operating the machine

In operating the machine, the keyboard and display play a dominant part. The keyboard is used for controlling the machine and for loading programs and data. The display serves as the main output device and also enables the user to follow in detail what the machine is doing.

The keyboard layout illustrated in Fig. 7, shows that the keys fall into four groups. The left-hand group consists of control keys used for inspecting and loading the store, and for loading the i.a.r. with the starting address of a program. The second group from the left consists of numeral, sign, decimal point and 'clear' keys for setting up numbers in the keyboard register. The third group is used when running programs, and the fourth is used when recording or playing back from a tape recorder.

The display consists of six rows, each containing 15 character positions. The format is variable and depends on the

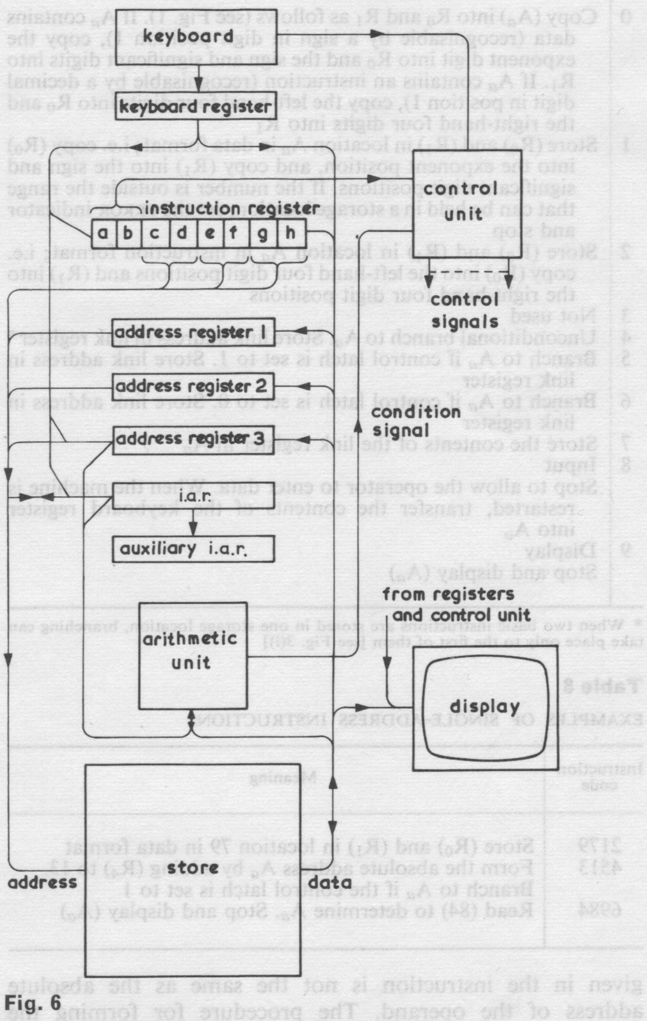

Fig. 6
Data-flow paths used in 3-address instructions

operation being carried out. In designing it, the principle was to show everything that is relevant at the time and no more.

Sections 6.1–6.7 describe how the keyboard and display are used.

6.1 Inspecting contents of the store

To see what is in a particular storage location, the operator first touches the NORMAL RESET key which clears the display, apart from the i.a.r. which is shown at the left of the top row (Fig. 8*a*). The operator then keys in the address whose contents he wishes to know; this appears in the keyboard register shown at the right of the top row (Fig. 8*b*). He then touches the LOAD ADDR key which transfers the keyboard register contents into an address register. This register is displayed at the left of the bottom row, and the contents of the required location is shown at the right, as shown in Fig. 8*c*.

A second address may now be keyed in the same way, or if the operator wants to see what is in a neighbouring location, he can use the INCR ADDR and DECR ADDR keys. These respectively increase and decrease by one the contents of the address register.

6.2 Loading the store

Loading a storage location is a simple extension of the procedure given in Section 6.1. The existing contents of the location are displayed first (Fig. 8*c*), and then the number or instruction to be loaded is keyed into the keyboard register (Fig. 8*d*). Touching the LOAD STORE key transfers this number into the required location replacing what was there before (Fig. 8*e*).

To load consecutive locations, as for example, when loading a program, the first location is loaded as above. The INCR ADDR key is then touched and the second location loaded etc.

6.3 Loading the i.a.r.

When a complete program has been loaded, the i.a.r. has to be set to the address of the first instruction. To do

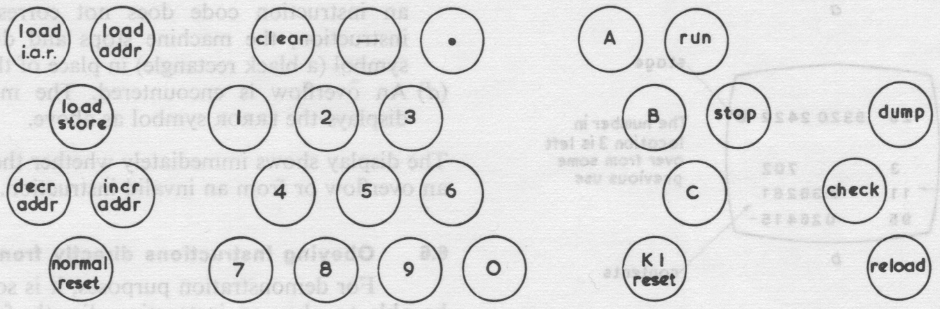

Fig. 7
Keyboard

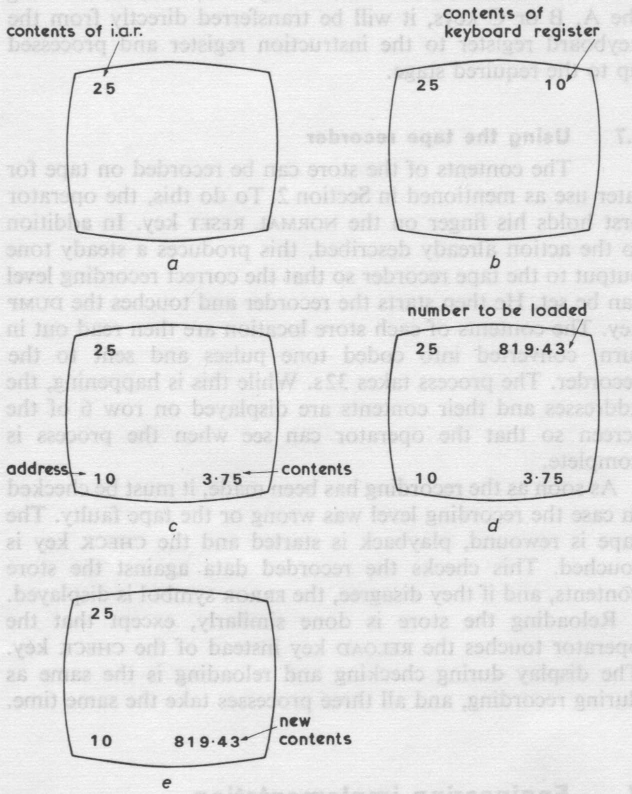

Fig. 8
Displaying and replacing contents of a storage location
a After resetting
b After keying in address
c After touching LOAD ADDR key
d After keying in number to be loaded
e After touching LOAD STORE key

this, the operator enters the address into the keyboard register (Fig. 9*a*) and then touches the LOAD IAR key to transfer it to the i.a.r. as shown in Fig. 9*b*.

6.4 Displaying individual instructions

One of the main uses of the display is to allow the operator to follow the operation of the machine as each instruction is processed. Designing a suitable display presented problems; the main one was how best to accommodate the large amount of relevant information. For instance, one has to display the instruction itself and the address from which it was read, and, in relative and indirect addressing, the data from which the absolute addresses are determined; e.g. the contents of modifier registers. There are also the absolute addresses themselves and their contents before and after the instruction is carried out.

To put all this information on the screen at one time would be far too confusing. What is done, therefore, is to make the machine stop at certain stages during the processing of an instruction, and display what is relevant at that stage. At

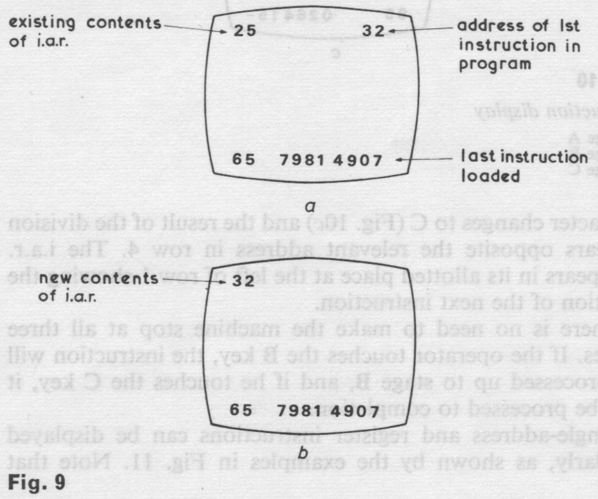

Fig. 9
Loading i.a.r.
a After keying in address of first instruction in program
b After touching LOAD IAR key

stage A, the instruction has been fetched from the store and loaded into the instruction register. At stage B, the absolute addresses have been determined, and at stage C the instruction has been carried out and the i.a.r. set to the address of the next one.

As an illustration, consider the instruction 9320 2422—a floating-point divide instruction using indirect addressing. To process the instruction up to stage A, the operator touches the A key. The display then appears as in Fig. 10*a*. The i.a.r. is not shown at this stage, and so the top row is blank. At the left of the second row is the auxiliary i.a.r. showing the location from which the instruction was read. To the right of this is the instruction register showing the instruction itself, and to its right, the stage character A indicating the stage reached. Row 3 is blank. At the left of rows 4, 5, and 6 are the presumptive addresses, and to the right of each are the contents of the respective locations, in other words, the absolute addresses. If the instruction had called for relative addressing, the modifier register designations R3, R4, and R5 would have appeared at the left of these three rows, and their contents to the right. With absolute addressing, the three rows would have been blank.

Touching the B key then processes the instruction up to stage B (Fig. 10*b*). The stage character changes to B, but, apart from this, the top two rows remain as they were. At the left of rows 4, 5, and 6 are the absolute addresses, and to the right of each the existing contents of that location; i.e. before the instruction is carried out.

To complete the instruction, key C is touched. The stage

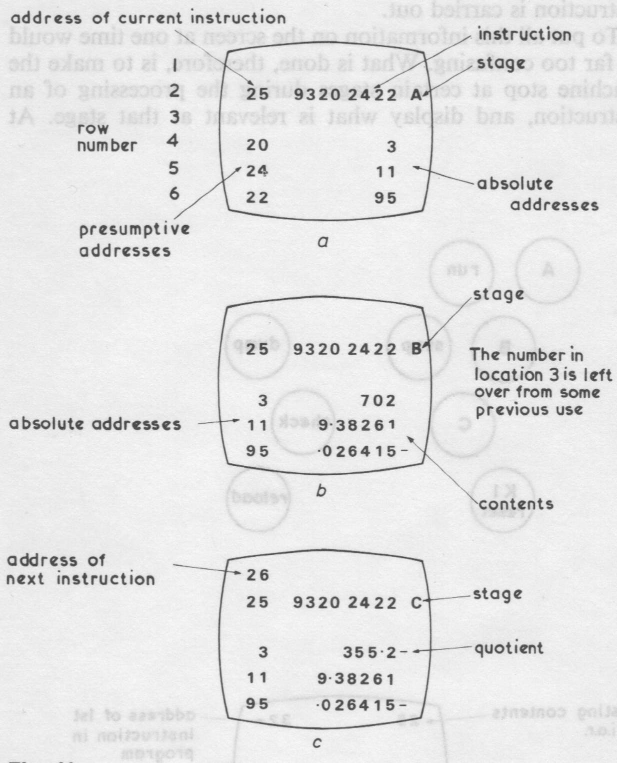

Fig. 10
Instruction display
a Stage A
b Stage B
c Stage C

character changes to C (Fig. 10*c*) and the result of the division appears opposite the relevant address in row 4. The i.a.r. reappears in its allotted place at the left of row 1 showing the location of the next instruction.

There is no need to make the machine stop at all three stages. If the operator touches the B key, the instruction will be processed up to stage B, and if he touches the C key, it will be processed to completion.

Single-address and register instructions can be displayed similarly, as shown by the examples in Fig. 11. Note that

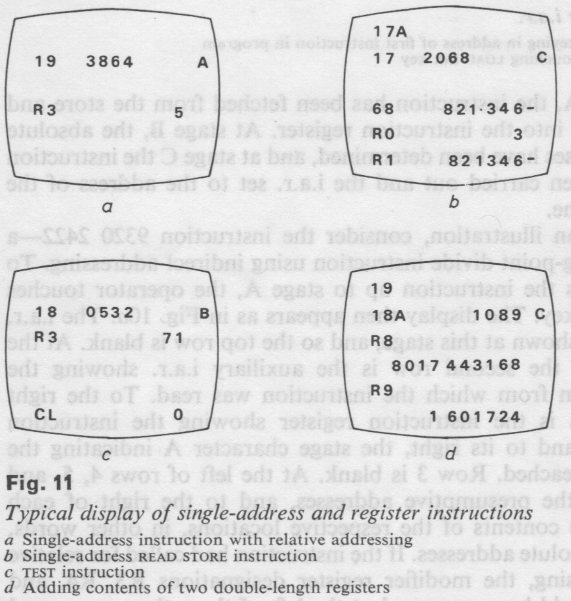

Fig. 11
Typical display of single-address and register instructions
a Single-address instruction with relative addressing
b Single-address READ STORE instruction
c TEST instruction
d Adding contents of two double-length registers

with register instructions, row 3 is used as well as rows 4, 5, and 6. With single-length registers, the register designation and contents always appear on the same row, but with double-length registers there would not be enough room; the contents are therefore shown on the row below the designation (Fig. 11*d*).

6.5 Running at full speed

When the operator wants to run the machine at full speed without displaying individual instructions, he touches the RUN key. The machine then runs until one of the following happens:

(*a*) An INPUT or DISPLAY instruction is reached. With an INPUT instruction the machine stops and displays at stage B to allow the operator to enter data on the keyboard; with a DISPLAY instruction, it stops at stage C.
(*b*) The operator touches the STOP key. The machine stops and displays at stage C of the current instruction.
(*e*) The machine tries to carry out an invalid instruction. If an instruction code does not correspond to a valid instruction, the machine stops and displays the ERROR symbol (a black rectangle) in place of the stage character.
(*d*) An overflow is encountered. The machine stops and displays the ERROR symbol as above.

The display shows immediately whether the error arose from an overflow or from an invalid instruction.

6.6 Obeying instructions directly from keyboard

For demonstration purposes, it is sometimes useful to be able to obey an instruction directly from the keyboard rather than first putting it into a storage location. To do this, the operator touches the KI RESET key; this clears the display in the same way as the NORMAL RESET key, except that the letter K appears on row 2 immediately below the i.a.r. An instruction can then be keyed in, and, on touching the A, B or C keys, it will be transferred directly from the keyboard register to the instruction register and processed up to the required stage.

6.7 Using the tape recorder

The contents of the store can be recorded on tape for later use as mentioned in Section 2. To do this, the operator first holds his finger on the NORMAL RESET key. In addition to the action already described, this produces a steady tone output to the tape recorder so that the correct recording level can be set. He then starts the recorder and touches the DUMP key. The contents of each store location are then read out in turn, converted into coded tone pulses and sent to the recorder. The process takes 32s. While this is happening, the addresses and their contents are displayed on row 6 of the screen so that the operator can see when the process is complete.

As soon as the recording has been made, it must be checked in case the recording level was wrong or the tape faulty. The tape is rewound, playback is started and the CHECK key is touched. This checks the recorded data against the store contents, and if they disagree, the ERROR symbol is displayed.

Reloading the store is done similarly, except that the operator touches the RELOAD key instead of the CHECK key. The display during checking and reloading is the same as during recording, and all three processes take the same time.

7 Engineering implementation

7.1 Store

One of the main ways of keeping down the cost of the machine was, wherever possible, to locate the various general-purpose and special-purpose registers in the same core matrix as the main store. This tends to limit the speed, but is very much cheaper than using transistor or integrated-circuit registers.

The matrix stores a total of 1760 4-bit characters, of which 1600 are used as the main store and the remainder as registers. Characters are read and written one at a time, the four bits of each character being handled in parallel.

The cycle time of the store depends on the particular t.v. standard used for display (see Section 7.3). The field-trial models use the 405 line, 50field/s standard which calls for a cycle time of $4\cdot92\,\mu$s. The other commonly used standards, 525 line, 60field/s and 625 line, 50field/s, would need a cycle time of $3\cdot16\,\mu$s.

7.2 Data-flow diagram

A simplified data-flow diagram is shown in Fig. 12. With the main store and registers sharing a single matrix, it can be seen that many of the basic operations consist of copying digits from one location in the matrix to another, or carrying out arithmetic on the contents of two locations. This is the reason for the two address registers A and B in the diagram.

All characters read from the store pass into the 4-bit register D. If they need to be saved for use during a later storage cycle, as for instance in arithmetic operations, they are held in a second 4-bit register E. Register F is an 8-bit register that holds the first two digits *a* and *b* of an instruction. The output of this register is gated with the timing waveforms to produce the internal control signals and the signals needed to control the display. The contents of the F register determine the display format, and the timing pulse circuits supply the line and field synchronising pulses for the t.v. receiver.

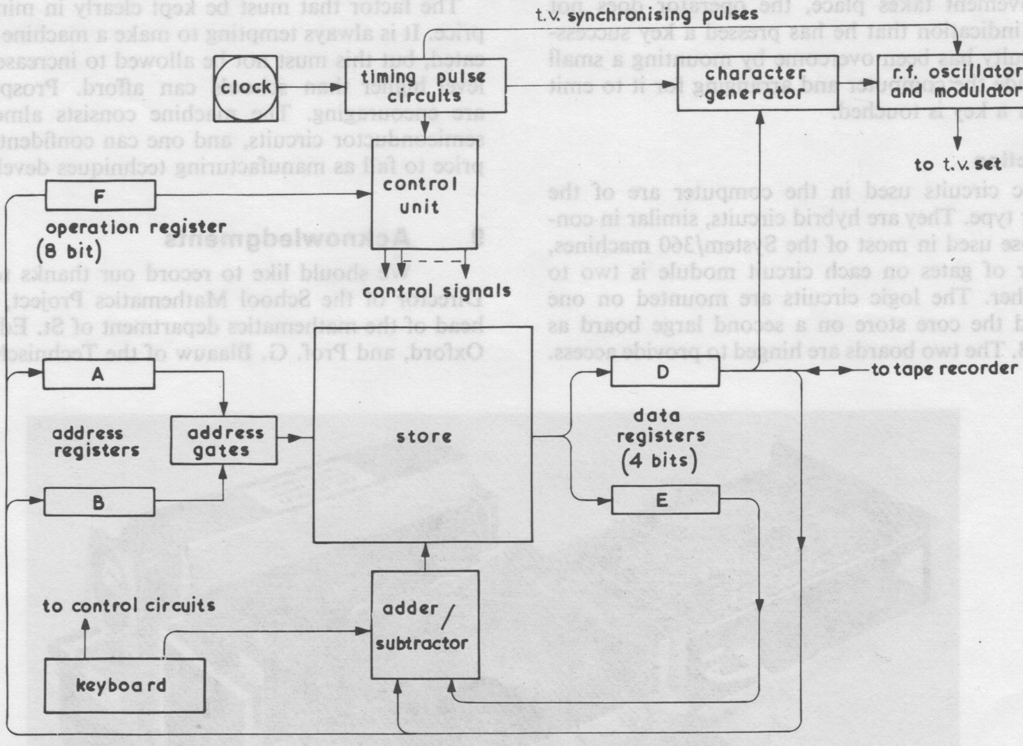

Fig. 12
Simplified data-flow diagram

7.3 Display system

A basic requirement of the display system was that it should use a perfectly normal t.v. receiver with no modification whatever. This means that the computer has to supply an r.f. signal conforming to the t.v. standard of the country where it is used.[3] The field-trial models have been built to work with the 405-line, 50field/s standard, but the design is such that they could be changed to the 525- or 625-line standards without much difficulty.

The only departure from normal practice is that no interlace is used; it was found to be an unnecessary expense. Thus, on the 405-line standard, the number of lines in the transmitted picture allowing for the field-blanking period is about 178.

The displayed characters are built up from a pattern of elements in a 7×5 array. On the 405-line standard, each element has a height corresponding to 2 line scans; on the 525- and 625-line standards the height would correspond to 3 line scans.

Compared with other methods of display, the essential point about using a t.v. raster is that the electron beam does not 'paint' one character completely before moving on to the next. When painting a row of characters, it first forms the topmost row of elements in all the characters along the row, then the second row of elements etc. Just before the beam reaches a particular character position, that character is read from the store and held in register D (Fig. 12). The timing-pulse circuits indicate which row of elements is being painted, and their output and that of the D register are sent to the character generator which produces the appropriate modulating waveform. It takes two line scans to paint a row of elements completely and the characters are 7 elements high; therefore every character on the display has to be read from the store a total of 14 times during each field period.

The output from the character generator and the synchronising pulses are combined with the output from a crystal-controlled oscillator using a Schottky-barrier diode modulator. The resulting signal is attenuated to a level of about 2mV in 75Ω, corresponding to the signal that would be obtained about halfway between a t.v. transmitter and the fringe of its service area.

7.4 Tape-recorder link

When the contents of the store are being recorded, characters are sent to the tape recorder at the rate of one per field period. At the beginning of each field period, a tone pulse of about 2kHz is transmitted which lasts for $\nu/8$ (where ν is the field period). This serves as a 'start' pulse. During the next four $\nu/8$ periods, the four bits of the character are transmitted serially starting with the most significant. If the bit is 1, a tone pulse is transmitted, and if it is 0, no pulse is transmitted. During the final three $\nu/8$ periods nothing is transmitted.

This system tolerates quite a wide variation in tape-recorder speed. Allowing for the tone pulse rise and fall times of 0·6ms, the tape speed during playback must be within ±7·5% of that during recording. The tape speed tends to vary asymmetrically about its nominal value owing to the increase of tape slip as the recorder gets older, and so, expressed in terms of the nominal value, the allowable speed variation is from +3% to −4·2%. This seems reasonable, even for battery-driven machines.

7.5 Touch keyboard

The touch keys use the principle of the transistor pump circuit.[4] When the operator touches a key, he introduces a capacitance between the key and earth. This capacitance is repeatedly charged through a resistor and discharged through the base–emitter junction of a transistor, giving rise to pulses of collector current. These build up a charge on a capacitor in the collector circuit. The circuit is so designed that a finger capacitance below 3 pF produces no output signal, but 12 pF or more produces a full output signal. Details of the touch-key circuits will be published separately.

A criticism that has been levelled against touch keys is that, as no movement takes place, the operator does not have a positive indication that he has pressed a key successfully. This difficulty has been overcome by mounting a small loudspeaker inside the computer and arranging for it to emit a tone whenever a key is touched.

7.6 Construction

The logic circuits used in the computer are of the diode–transistor type. They are hybrid circuits, similar in construction to those used in most of the System/360 machines, but the number of gates on each circuit module is two to three times higher. The logic circuits are mounted on one large board and the core store on a second large board as shown in Fig. 13. The two boards are hinged to provide access.

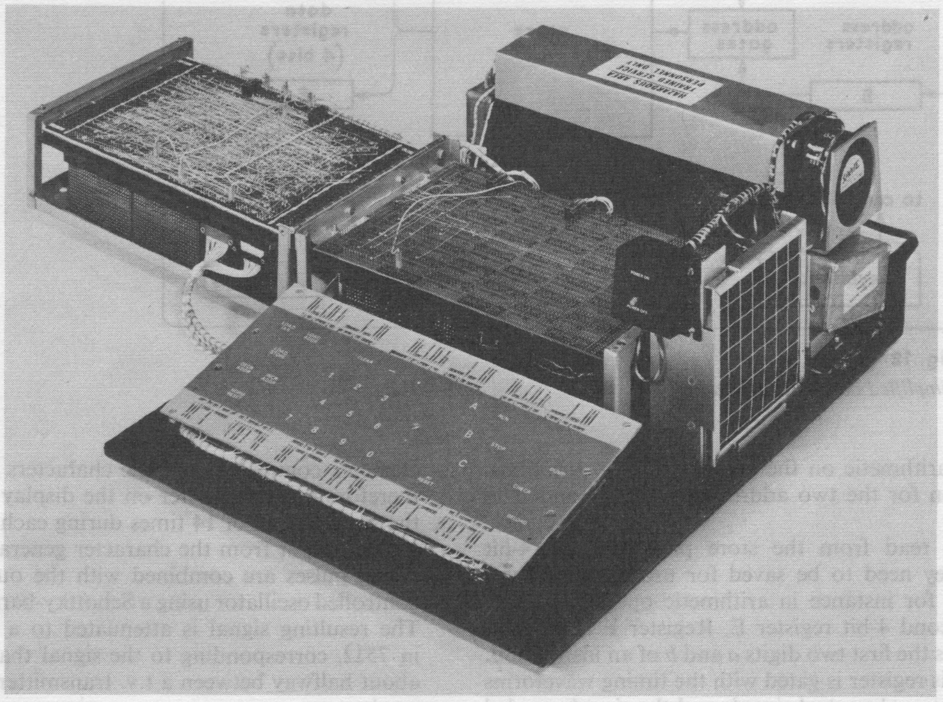

Fig. 13

General view with cover removed

The field-trial models are rather larger than a final production version would be. This is mainly because a core store of the correct size was not immediately available, and so the one fitted is much larger than the machine really needs. The machine measures 18 in wide by 22 in long by 9·5 in high and weighs 56 lb. The power consumption is 240 W.

8 Future work

Field trials of this computer are being carried out during the school year 1969–70. The trials have two objectives; to learn more about the aspects of computers and computing that can usefully be taught to schoolchildren and to assess the suitability of this particular machine. Initial enthusiasm has been great, but one must try to discount the element of it that can be put down to novelty.

The machine must be regarded as a first experiment, and there are no plans to put it on the market. But, if experience shows our approach to be sound, there are several ways in which a machine of this type could be developed. For instance, the range of applications could be extended by making it handle alphabetic characters as well as numbers.

The factor that must be kept clearly in mind, however, is price. It is always tempting to make a machine more sophisticated, but this must not be allowed to increase the price to a level higher than schools can afford. Prospects, however, are encouraging. The machine consists almost entirely of semiconductor circuits, and one can confidently expect their price to fall as manufacturing techniques develop.

9 Acknowledgments

We should like to record our thanks to B. Thwaites, Director of the School Mathematics Project, J. D. Tinsley, head of the mathematics department of St. Edward's School, Oxford, and Prof. G. Blaauw of the Technische Hogeschool, Twente, Enshede, Netherlands for their contributions to the specification of the machine. We should also like to thank J. Travis, P. Johnson, A. Patten, J. Pike and E. Chamberlain, who carried out much of the detailed design of the prototype and field-trial models, and the directors of IBM UK Laboratories Ltd. for permission to publish this paper.

10 References

1 TAUB, D. M., and OWEN, C. E.: British Patent Specification 1 143 327, Feb. 1969

2 TAUB, D. M., and TINSLEY, J. D.: 'A first course on the schools computer' (IBM UK Laboratories, 1969)

3 Documents on the 11th plenary assembly of the CCIR, Oslo, report 308–1 (ITU, 1966), pp. 217–230

4 HEMINGWAY, T. K.: 'Electronic designer's handbook' (Business Publications, London, 1966), p. 219

5 SUMNER, F. H., HALEY, G., and CHEN, E. C. Y.: 'The central control unit of the Atlas computer', *in* POPPLEWELL, C. M. (Ed.): 'Information processing 1962' (North Holland, Amsterdam, 1963), p. 657