The X–1 Computer

by B. J. Loopstra

Summary: This article describes a small, fast, transistorized computer, designed basically for commercial data-processing, developed by the staff of the Mathematical Centre, Amsterdam. The article is based on a talk given by the author to a Colloquium at the University Mathematical Laboratory, Cambridge, on 27 November 1958.

1 HISTORICAL BACKGROUND

At the Mathematical Centre in Amsterdam several electronic digital computers have been built since 1953. The first of these was the ARRA computer, which was a rather small magnetic-drum machine with an average speed of about 40 operations per second. A copy of this machine, called FERTA, was built for the Fokker aircraft works, and is still being used there. The third machine is called ARMAC, and its average speed is about 1,000 operations per second. All these computers were financed by the Dutch Government, and built in the Mathematical Centre for its own use, and for computing for other people, in much the same way as the machines in the Mathematical Laboratory here in Cambridge. When ARMAC was finished, so was our money. Since we had a group of about ten people fairly well acquainted with computer design and operation, we now had to find a job for them. Now, in England, you have been lucky (or perhaps you would say unlucky), since existing industry has been able to absorb quite a lot of people in this field. No such industry existed in Holland, so our intention was, firstly, to found such an industry, and then to absorb the people into it. Of course the question of finance arose, but we were fortunate in finding an organization who were willing to try the experiment of founding an industry to develop and sell computers. That is why the group at the Mathematical Centre is now being taken over by a commercial firm, called *N.V. Electrologica*. This new firm is a wholly-owned subsidiary of the Dutch life insurance company *Nillmij*.

The arrangements, to begin with, were that the staff of the Mathematical Centre would develop a new type of computer, called the X–1, and that this new firm would pay all the expenses. There was an understanding that, when the development reached a certain point, the firm would take over the personnel, the equipment, and the designs, from the Mathematical Centre, and continue on a purely commercial basis without any help from the Centre.

The present* position is that the prototype model of the X–1 machine (whether you call it a *machine* or a *system* I don't mind—perhaps it is a system in that you can have small or large X–1's) is now 99% complete and working. Several other models are on order, the present total being about 8, of which some are going to Germany and others to Holland.

The idea behind the development of the X–1 system has been that it should be primarily intended for commercial work, and I hope that you will bear that in mind when judging the machine. However, we felt that, even for commercial work, a fairly fast machine would be advantageous, and, of course, input-output facilities would play an important part.

* November 1958.

2 BRIEF SPECIFICATION

1. The machine is transistorized.
2. Its internal operation is in binary. (Perhaps this is rather surprising since the machine is intended primarily for commercial use.)
3. It works in fixed point.
4. The word length is 27 binary digits. (This is rather small; maybe it was based on the assumption that people have not so much money nowadays!)
5. The memory consists entirely of magnetic cores; there is no drum.
6. Regarding input-output equipment, I shall only mention what has already been realized: punched tape, typewriter, and punched cards.

3 THE STORE

The only memory is of magnetic cores, and there is a single-address order code. The address length, which, of course, fixes the maximum capacity of the store, is 15 binary digits, giving us a maximum capacity of 32,768 words, each of 27 bits. A fast memory of that size is fairly expensive, so it is built in blocks of 512 words. The minimum machine has a fast store of 512 words, which can be increased by multiples of 512 words up to the maximum, if this is required. Actually 8 blocks of memory (4,096 words) are housed in one cabinet; after that a second memory cabinet is needed, with its own selection, read-in, and read-out circuits.

The memory is of two different kinds, known as the *active* and *passive* memories. The passive memory is a kind of *wired store* (see Renwick, 1957), which is useful for storing such information as input routines, fixed subroutines and so on. It is considerably less expensive than the active memory, which is the normal coincident-current type of magnetic-core matrix. The passive store may be partly specified by the customer if he so desires, and in that case it comes in blocks of 64 words. He may build up any set of subroutines he chooses, or any input routine he likes, though obviously the latter is not to be encouraged. The cores we use are not the very

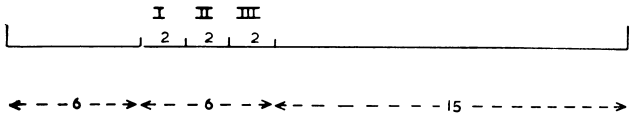

FIG. 1.

small size with a turn-over time of 1–1½ μsec, but a very much slower type with a turn-over time of about 8 μsec; therefore the cycle-time of the memory, which is fixed by the read-write cycle, is 32 μsec. When you know this fact, some of the operation times which I shall give you presently will be more clearly understood.

4 ORDER STRUCTURE

An interesting point about this machine is its order structure. I have already mentioned that the word length is 27 bits, of which 15 bits are used for the address, leaving 12 bits for the functional part of the order. As you will see from Fig. 1, there is only one instruction per word. The 12 bits of the function may be further subdivided into two groups of 6, the second of which is subdivided again into 3 groups of 2 bits each. Now the first group of 6 has 64 values, giving 64 possible main functions for the instruction. The other 6 bits are used for modification of each of these instructions, and each group of 2 bits must be considered separately. Considering the possibility 00 in each of the 3 positions as indicating no variation, we see that each group gives 3 possible variations, so that, in principle, each instruction can have 27 different forms, although many of them are meaningless.

Before mentioning the more important of these functions, perhaps I should say something about the internal organization of the machine. This is indicated diagrammatically in Fig. 2. The machine has two 27-bit accumulators called A and S; it has a 15-bit register called OT which is the order counter (I use the Dutch abbreviations, for simplicity's sake); there is an instruction register OR of 27 bits; there is a single B-register of 16 bits, which may be considered as 15 for the address, and a sign digit; there are 3, 1-bit registers about which I shall speak later on.

5 ARITHMETICAL AND LOGICAL ORDERS

Now let us consider the first 6 bits of an instruction. We usually group our instructions in sets of 8, for example:

$$\begin{array}{ll} 0 & (n) + (A) \to (A) \\ 1 & -(n) + (A) \to (A) \\ 2 & (n) \to (A) \\ 3 & -(n) \to (A) \\ 4 & (A) + (n) \to (n) \\ 5 & -(A) + (n) \to (n) \\ 6 & (A) \to (n) \\ 7 & -(A) \to (n) \end{array}$$

where (X) stands for "content of X" and $\to$ indicates replacement.

This is quite usual, and you will see that the store has full accumulative properties since you can add or subtract from A into any storage location. Similar sets of instructions exist for the S-register (which is thus a second accumulator) and for the B-register. The B-register therefore has quite a wide range of facilities.

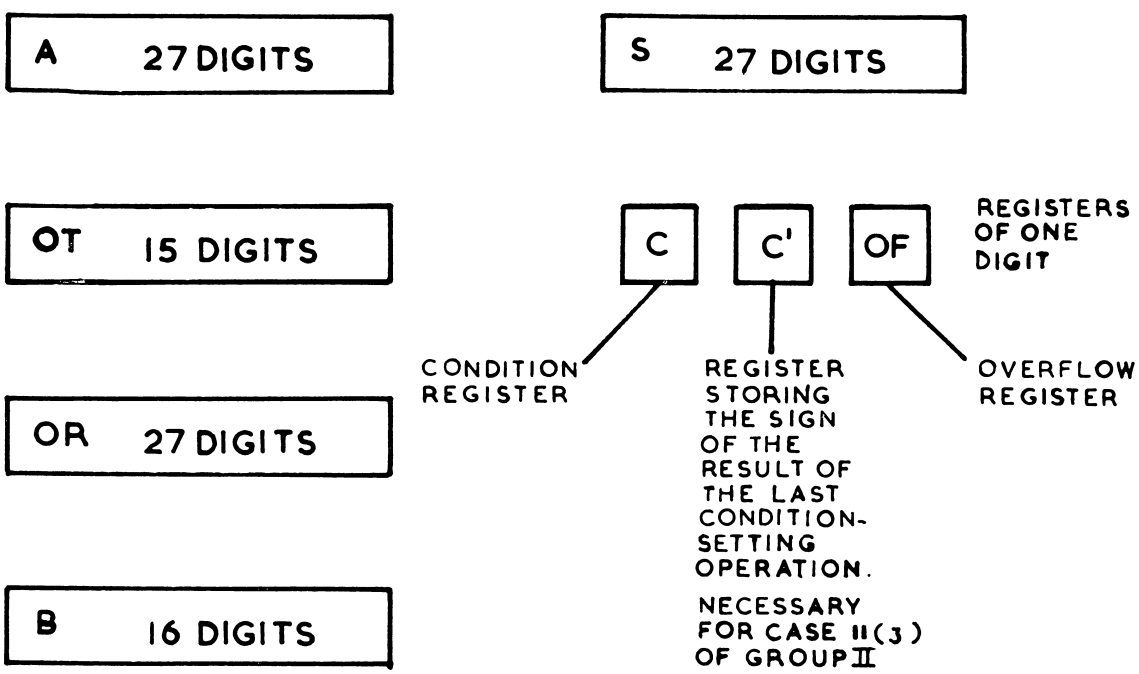

FIG. 2.

These three groups take care of 24 of the possible orders. There are also sets of multiplication and division orders. A typical multiplication order multiplies the content of n by the content of S, and writes the result in A and S combined together to hold a double-length number. If you wish, you can, at the same time, add the content of A to the least significant half. A typical division instruction divides the double-length number, held in A and S together, by the content of n, replacing S by the quotient and A by the remainder.

There are two logical operations: a *collate* instruction (*logical multiplication*) collates the content of n with the content of A or S; and a *logical addition* order adds the content of n to A or S, suppressing any carry.

6 JUMP ORDERS

Perhaps rather more interesting are the jump instructions. The normal one has the following form:

$$(n) + (OT) \rightarrow (OT)$$

OT is the instruction counter, so this could almost be called an "addressed" jump. It is *not* the usual kind of jump instruction where the content of the instruction counter is *replaced* by the *address part* of the order; in this case the content of the instruction counter is *increased* by the *content* of a specified memory location. A similar instruction is:

$$-(n) + (OT) \rightarrow (OT)$$

A further instruction is:

$$(n) \rightarrow (OT)$$

This is more like a normal jump apart from the fact that we now transfer the address, to which we want to jump, from the memory to the instruction register. There is no need to say that, in many cases, this can be rather awkward. We do not want this at all but, as we shall see presently (Section 7), there is a way around the difficulty. Actually there is even the order:

$$-(n) \rightarrow (OT)$$

If anyone can give me an application for it I shall be very glad; so far we have not been able to find one!

Apart from these common types of jump instructions, there are two other kinds, a *subroutine jump*, and a *counting jump*. To consider first the counting jump, this has the form:

$$n \rightarrow (OT)$$
$$(\tau_p) - 1 \rightarrow (\tau_p)$$

where the $\tau_p (p = 0, 1, \ldots 7)$ are a set of 8 memory registers, and the value of p is specified in the counting-jump instruction (how, we shall consider later). Note that in this instruction it is n, not (n), that is placed in OT. Each time this operation is performed, unity is subtracted from the specified counting register. As it stands here, this instruction is not yet a counting jump in the true sense of the word; there needs to be some test of conditionality, but we shall discuss this later on.

The subroutine jump acts as follows:

$$n \rightarrow \mathrm{OT}$$
$$(OT)^* + 1 \rightarrow (\lambda_p)$$

where $(OT)^*$ is the content of OT *before* the instruction is obeyed, and the $\lambda_p (p = 0, 1, \ldots 15)$ are a further set of 16 memory registers which are set apart solely for the purpose of storing the return addresses from subroutines. Now a 4-bit indication (described in Section 7) is required in the instruction to specify p, that is, to determine the storage register which is to hold the return address. One thing that now becomes clear is the reason why we have the addressed type of jump-instruction. The address to which we wish to return has been stored, so that, at the end of a subroutine, we want to jump to the location specified by the content of one of the λ's. So, with these jump orders, one can have a fixed λ for a specified subroutine, and a fixed jump instruction at the end of the subroutine, such as "jump to the content of λ_6", say.

That is all I shall say about the order code except, perhaps, to mention that there are also instructions for shifting, for operating the tape readers, tape punches, etc. I shall not discuss these at length since they are quite ordinary and have no special features.

7 ORDER MODIFICATION

What are perhaps worthy of more elaboration are the 3 pairs of digits of the instruction, which I have labelled I, II, and III in Fig. 1.

In Group I, there are 4 possibilities:

00
01 *A*
10 *B*
11 *C*

of which we shall ignore possibility 00. Possibility 1, which we designate A, we call *absolute*. In instructions with this form, instead of, say, adding (n) into the accumulator, the number n is added into the accumulator. The number n can therefore take any value between 0 and $2^{15} - 1$; n cannot be negative but, as there is a complete set of complementary orders, it is just as easy to subtract as to add. There are the same facilities with multiplication and division; it is thus possible to, say, divide by 2, 3, or 6, or to multiply by 37, in a single instruction. As a matter of fact, we have found this a very powerful facility, especially in commercial programs. We have found program sheets where half the instructions have had an A added.

Possibility 2, which we designate B (since it concerns the B-register) simply means that the content of the B-register is added to the address before the instruction is obeyed; in fact, this is the conventional use of a B-register. It is now logical to designate by C the third possibility. This has the same effect as B, except that the address as it stands in the store is also modified.

These uses of Group I also apply to the jump orders,

and it is by means of the first possibility that we get over the difficulty mentioned in Section 6, and thus achieve a set of more conventional jump orders.

The pair of digits in Group II is concerned with *condition setting*. Perhaps I should mention here that we do not have conditional orders in the normal sense of the word. What we do have is a register C (one of the 1-bit registers I mentioned earlier) which can hold 0, when the conditionality is said to be positive, or 1, when the conditionality is said to be negative. As will be seen later, it is the value of C which is tested for conditionality, and not, for instance, the sign of the content of the accumulator. This means that we must have facilities for setting this register, and it is the pair of digits in Group II which determines how this is done. As before, there are 4 possibilities: 00, 01, 10, and 11, of which we shall ignore the case 00.

Possibility 1 sets register C to 0 if the result is positive, always supposing that the "result" is something well-defined. It is not so in every case; for instance, you have to define what you mean by the result in a division, or a multiplication, or a shift instruction, and, in fact, a jump order cannot be said to have a result at all. But more of that later; normally, in the case of an order that one would expect to have a result, it is just what one would suppose it to be.

Possibility 2 sets register C to 0 if the result is zero.

Possibility 3 is rather more complicated. It sets register C to 0 if the result has the same sign as the result of the previous condition-setting instruction. Another of the 1-bit registers stores the signs of the results of the condition-setting instructions, and the sign of the present result is compared with the previous value. With this group of digits, as you will see, it is possible to have a result, and to set a condition dependent on this result, but not to act on the condition until some later stage.

The last 2 digits, Group III, are concerned with *condition reaction*. As before, we shall ignore the case 00, of the 4 possibilities 00, 01, 10, and 11.

Possibility 1 deserves special attention; it really has nothing to do with conditional reaction. Its effect is that there is no change in any register; although there is a result, it is merely formed in a special register, not included in Fig. 2, and is then discarded, although possibly being used for condition setting. It is very useful for this purpose; for instance, we can form a sum and note its sign in the condition register, without making any changes anywhere else. For example, we may wish to compare a number with a list of numbers to discover whether it is equal to any one in the list. In most machines it is necessary to subtract and test whether the result is 0, by which time the original number is lost, but with these facilities we can perform the subtraction and test for zero without changing anything.

Possibilities 2 and 3 determine whether the instruction is to be performed or skipped. In Case 2 the instruction will be skipped if the content of the condition register is 1, in Case 3 if it is 0.

With jump instructions, the use of these facilities transforms them into rather conventional conditional jump instructions. In the case of counting jumps the 2 bits of Group I are used, together with the least significant bit of the main instruction group, to specify τ_p (3 bits giving 8 possible τ_p's). The bits of Group II then have a different meaning:

01 means carry out the jump if new $(\tau_p) > 0$; if not, skip the instruction;

10 means carry out the jump if new $(\tau_p) = 0$; if not, skip the instruction;

11 means carry out the jump if new $(\tau_p) \geqslant 0$; if not, skip the instruction.

In the case of subroutine jumps the 2 bits of Group I are used for the specification of λ_p, as is also the first bit of Group II, and the least significant bit of the main function group. Both Groups I and II, therefore, cannot have their normal meanings. In Group III possibility 01 becomes:

Carry out the jump if the content of the overflow register $OF = 1$; if not, skip the instruction.

The two remaining possibilities in this group have their normal significance.

8 OPERATING SPEEDS

The following is a short survey of some of the operating speeds of these instructions (in μsec).

Normal instructions operating on registers A, S or B:

Function	*Normal*	*Variation A*	*Variation C*
0	64	44	72
1	64	44	72
2	64	36	72
3	64	36	72
4	76	—	84
5	76	—	84
6	64	—	72
7	64	—	72
Logical operations	64	36	72
Jump instructions	64	36–44	72
Counting jumps	76	—	—
Subroutine jumps	72	—	—

Shift instructions: $40 + 8n$ μsec, where n is the number of places shifted.

Multiplication and division: 500 μsec in all cases.

For binary-decimal conversion processes, a fast multiplication by 10 is available, operating in 64 μsec.

9 INPUT AND OUTPUT

Finally I should like to mention something about the input-output arrangements, especially punched cards. The X–1 system has what are called *complete interruption facilities*. This means that the input and output equipment can interrupt the operation of a program, and force the machine to take some other action.

Suppose we have a card reader and a central computer, as in Fig. 3. At some instant, the central computer may send a signal to the card reader calling for a card to be started to be read into the machine. The time taken for the card to be completely read may be, say, half a second, or perhaps even ten seconds, during which time the computer can continue with the original program. The information from the card is read into the buffer store associated with the reader and, only when the card is completely read and the buffer store filled, is a signal sent from the control equipment of the buffer back to the central computer. When this happens, the central computer interrupts the operation of the program, and sends control to a subroutine which is determined by the group of input units to which the reader belongs. Every input unit is in a certain group of units, and there can be as many as 6 of these groups, each containing any number of pieces of equipment. There is no theoretical limit to the number of units in any group. For instance, one group may contain all the punched-tape mechanisms, another the punched-card reproducers, another the punched-card sorters, and yet another may consist of tabulating machines. In the future there will be a group for magnetic-tape units.

Whenever a particular unit has data available, it immediately sends a signal to the central computer as a warning that the information is waiting. The operation of the program is interrupted by a subroutine which has to decide on the next step to be taken. Of course, it may decide to ignore the information altogether, although normally it arranges that the contents of the buffer are transferred to a particular section of the main store.

The advantages of this system are obvious. There is no time wasted in waiting for data, since the machine is called upon to deal with information as soon as it is available, and can continue computing normally in the meantime, with no need to wait for a single microsecond. Secondly, this is one of the few ways in which a large number of non-synchronous input and output units may be connected to one central computer.

Of course, some extra facilities are required; for instance, several units from a certain group may produce a signal simultaneously. In this case the group subroutine must be able to determine which of the units to deal with. For this purpose a special register is associated with each group. As another example, two groups may produce a signal at the same instant, and the machine must know which group is to be dealt with first. Some order of priority has to be decided. Normally the fastest piece of equipment is dealt with first, otherwise a considerable amount of time may be lost.

Another outcome of the interruption facility is that it is quite easy to run two programs at the same time. This was rather a strange sight when we tried it the first

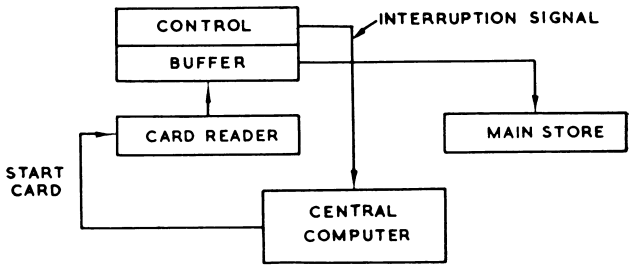

FIG. 3.

time, and we were very astonished to find that the system worked. We had a group of visitors, some of whom were interested in the scientific aspect of the machine, while the rest wanted to see applications to commercial work, and we set out to give a demonstration which would interest both groups of people. We thought of running a more or less scientific problem—computing primes—on the one hand, and a commercial (actually a life insurance) problem on the other. All we needed was a very small co-ordinating program informing the machine whenever data were available. Actually, this is a simple program, since we know that data for computing primes are always available—there are always more primes to be found! For the commercial problem, cards had to be fed into the machine, and information was only available at specific times. Actually the commercial problem took 10 msec computing time to the 500 msec taken to read a card, so that this program was obeyed only at comparatively long intervals. The result was rather astonishing to watch. The prime-number program operated almost continuously, its speed hardly diminishing when the card-operated program ran. Only now and then was there a barely-discernable wait of the typewriter, between two decimal digits of a prime number, while the contents of a card were dealt with. This seems rather an interesting way of organizing life!

DISCUSSION

In reply to a question the author gave the following additional explanation: Mention that a punched-card sorter is attached to the machine is, perhaps, rather surprising. The sorter is primarily used as a card input device. Cards are read, stored in a buffer, and subsequently checked, by two complete sets of 80 brushes. An interruption signal is sent to the machine after the read-in of each card. The sorter can be stopped between cards (by means of a special type of brake).

The sorting mechanism itself can be operated by the central computer by means of a special sorting instruction. This instruction gives the programmer the facility to deposit the card in any of the available reception pockets. It has been arranged that any card showing a faulty reading will automatically go into a predetermined pocket.

REFERENCES

RENWICK, W. (1957). "A Magnetic-Core Matrix Store with Direct Selection Using a Magnetic-Core Switch Matrix," *Proc. I.E.E.* Vol. 104, Part B, Supplement No. 7, p. 436.