U2778

Appl. sci. Res. Section B, Vol. 2 1952

A SIMPLE ELECTRONIC DIGITAL COMPUTER

by W. L. VAN DER POEL

Central Laboratory of the Netherlands Postal and Telecommunications Services, 's-Gravenhage

Summary

In this article will be described the logical principles of an electronic digital computer which has been simplified to the utmost practical limit at the sacrifice of speed. Essential in this computer is that the control registers form part of the arithmetic unit, which results in a very simple method for using sub-programmes. The arithmetic unit consists of two non-calculating registers and an adding unit for adding three digits together at a time. There is only one instruction register that consists of a number of flip-flops. The seven possible instructions of this machine have four digits, which are used functionally. Thus no decoder is required. Multiplication and division must be programmed. Examples are given. A small number of short registers containing one number each is supplied to serve as working positions. The long registers are so constructed that when the short registers are used, the next instruction just comes out when the former instruction has been executed. Advantages and drawbacks of this machine are mentioned. A machine based on these principles has been built out of the elements of a larger machine which is in course of construction at the Mathematical Department of the Central Laboratory of the Netherlands Postal and Telecommunications Services. This machine has run very satisfactorily for two months.

§ 1. *Introduction*. Lately the logical principles of various machines have been described by several authors. They are all devised on the same basic principles but differ in features and nature of elements for the technical realisation of the machine [1]–[9]. All these machines have in common the following basic parts:

a) A memory. Here will be stored the numbers which must be processed as well as the instructions which indicate what these processes will be.

b) An arithmetic unit. In the arithmetic unit the elementary arithmetical operations take place. In this machine these operations will only be addition and subtraction.

c) A control. The control knows at a certain moment where the next instruction that must be executed is to be found. After extracting that instruction it controls the action of the arithmetic unit, i.e. it directs the execution of that instruction.

d) The input device. At the beginning of a calculation the necessary numbers and instructions must be fed into the memory. As the speed will be too great for manual input, the information must be recorded in code on a medium, a teletype tape, for example. From this medium it can be transferred into the machine.

e) The output device. To produce the results in a legible form the machine must have a device by which it can communicate with the outside. In most cases this will be accomplished by an electrically controlled typewriter that can be fed, directly or indirectly, from the machine.

During the construction of an electronic computer at the Mathematical Department of the Central Laboratory of the Netherlands Postal and Telecommunications Services out of the already available parts a small machine has been built. This small machine contains all the afore-mentioned five units, but in one respect it differs appreciably from most other machines: the arithmetic unit and the control have so many parts in common that they are not distinguisable. Although this small machine has no built-in multiplier, it has most of its logical principles in common with the large machine, so it offers an excellent opportunity to describe these principles without undesirable complication. It must be kept in mind, however, that this machine is not meant as a practical computer, but only serves the purpose of gaining experience with the large computer.

§ 2. *Memory.* The construction of a computer will be greatly dependent upon the financial funds, the technical possibilities, the desired speed and the available time and space, so it is not possible to discuss the logical principles in general without making a choice for certain types of components or certain basic properties. The system which we shall describe applies only to those types of memories which operate in a serial form, i.e. the digits of a number go into, or come out of the memory as a time sequence. Not only the digits of a number, but also the different numbers come out of the memory consecutively. We shall confine ourselves to two particular examples of these memories: the mercury delay-line and the magnetic drum. Both types have been described in literature, so I shall only summarise their basic properties.

The mercury delay-line consists of a tube which contains mercury. Both ends are terminated by a quartz crystal. One of these crystals transmits a pulse-modulated carrier into the mercury in the form of an acoustical vibration, the other crystal receives the signal after a certain delay. The electrical pulse-train can then be amplified, detected, reshaped and recirculated. It is possible to have delay-lines of 30 or 1000 pulses length, with a pulse frequency of 1 Mc [10]) [11]) [12]).

The magnetic drum consists of a drum of non-magnetic material, coated with a layer of a ferromagnetic material. By a magnetic system, a so called "head", it is possible to write upon the drum short pulses which remain on that drum by virtue of the remanence. The same head can detect the pulses again and thus read the drum. The pulse-frequency can be 50 kc and one revolution of the drum can take 20 ms, so about 1000 pulses can be allocated on a circumference, a "track". It is possible to have a large number of tracks on the same drum [13]).

§ 3. *Arithmetic unit.* Also for the arithmetic unit a choise must be made in advance. Inside the machine we shall use the binary system throughout. Numbers will consist of 30 digits and a sign digit each (see appendix I). There is a fixed binary point, the position of which remains to be chosen. We shall work here only with two particular positions of the binary point: on the extreme right of a number and between sign digit and first digit. Thus we get integers or numbers between 1 (not incl.) and —1 (incl.), respectively. Negative numbers will be represented by complements. The instructions which indicate the elementary steps in a calculation will only consist of two parts:

a) an address, i.e. a number which refers to the location in the memory where the operand is standing or to where the operand must be transferred.

b) an operation which tells the arithmetic unit what to do with the operand.

Such a system is called a one-address-code. This is the simplest possible code, because it always performs one basic operation at a time. We try to make the machine as simple as possible, so we decide that no built-in multiplier or divider will be incorporated. The only

arithmetic operations that the machine must be able to perform are addition and subtraction. Multiplication consists of addition and shifting, and shifting can be done by doubling the partial result or adding this to itself in the binary system. So multiplication can always be broken down into addition only. In a programmed computer the storing of the results of an addition or subtraction is an operation which must be explicitly stated in the programme, so we also get storing as an essential operation. Let us now analyse the function of the control. The basic cycle of operations consists of two parts:

a) Take in an instruction from register *n* of the store and add 1 to *n* so that the next instruction can be taken in from the next register.

b) Execute the just extracted instruction.

So we have in addition to the operations "add" (or "subtract") and "store" the operation: take in an instruction from register *n*.

Now we observe that

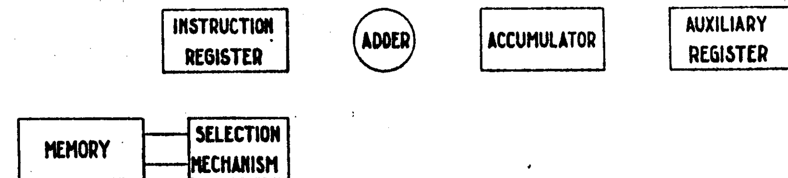
a) during the extraction of an instruction we do not perform arithmetic operations on numbers, but we have to add a "one" to the address of the extraction instruction.

b) During the execution of an instruction we need three registers, 1°) one for the instruction which must be obeyed, 2°) one for the extraction instruction for the next instruction, 3°) one for the number to which we must add the number to which the current instruction refers, or the number which must be stored.

So we see that we only need three registers in arithmetic unit and control together. We can then use the same adding mechanism for adding two numbers and for adding 1 to the extraction-instruction.

§ 4. *Action of the instructions.* Let us now follow in detail how this is accomplished. We shall denote a register for a number by a rectangle and the adding mechanism by a circle. This circle is so constructed that out of every two incoming pulses, representing the digits of the two numbers, it constructs an outgoing pulse, representing a digit of the sum. (It also produces a carry-over which must be delayed one pulse time and must then be fed in again at the input of this box together with the next digits of the numbers which must be added. See appendix II).

We shall give names to these registers and call them



We need the following types of instructions:

Take in an instruction from register *n*, abbreviated — $X\ n$

Adding is also a basic operation. For convenience we have included four variants.

Add the contents of register *n* into the cleared accumulator — $P\ n$

Subtract the contents of register *n* from the cleared accumulator — $Q\ n$

Add the contents of register *n* to the number already in the accumulator — $O\ n$

Subtract the contents of register *n* from the number already in the accumulator — $A\ n$

All these instructions leave the contents of the memory undisturbed.

The third basic operation is storing. We have two variants.

Store the number from the accumulator in register *n* and clear the accumulator — $S\ n$

Store the number from the accumulator in register *n* and do not clear the accumulator — $R\ n$

Here the contents of register *n* are written over by a new number.

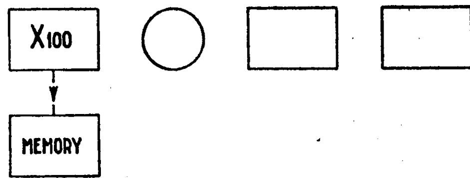
The logical system of coding these instructions will be discussed in § 5.

Let us follow in detail what happens when we execute a simple programme for adding *a* to *b* when *a* is stored in 20, *b* in 21 and the result must be stored in 22. Let the programme begin at 100. The programme then looks as follows:

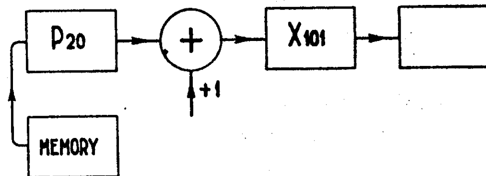
→ 100	*P* 20		20	*a*
101	*O* 21		21	*b*
102	*S* 22		22	(sum)

In front of the vertical line are the numbers of the registers, behind this line the contents of these registers.

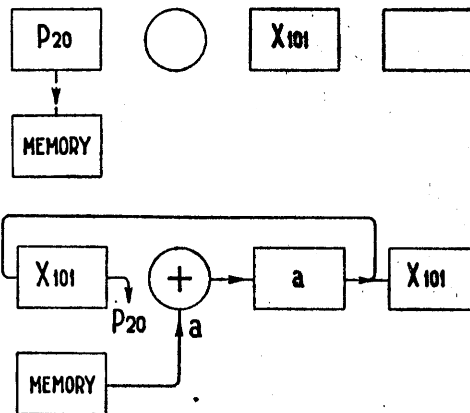
The contents of the instruction register at the beginning of the programme are



The address of X 100 goes to the selection mechanism of the memory, whereupon this issues the contents of register 100.

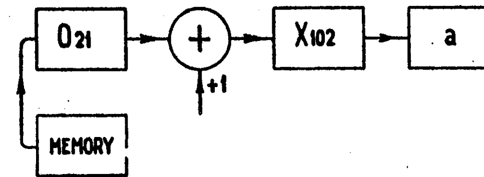


At the same time the adder is used for adding a one to the address of X 100, so it becomes X 101, meaning: the next instruction is in register 101. Now in the instruction register stands P 20. The address goes again to the selection mechanism and P alters the interconnecting paths between the registers of the arithmetic unit, so there follows



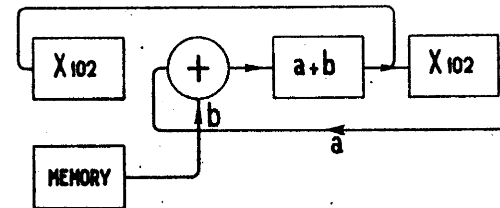
Because of the change in interconnection X 101 is pushed to the instruction register by a. X 101 pushes out the previous contents of the instruction register.

The next operation is X 101: take in an instruction from register 101.

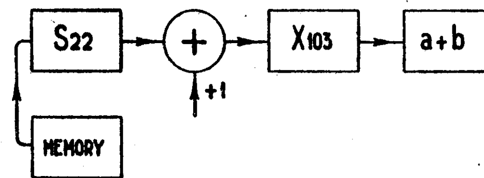


a is pushed into the auxiliary register. The adder is again used for adding one to X 101.

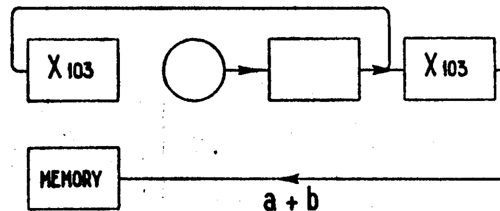
The next operation is O 21: add contents of 21 to the accumulator.



X 102 takes in the last instruction of the example



S 22 means: store the sum in 22 and clear the accumulator.



The machine proceeds with X 103.

§ 5. *Functional use of the instructions*. Before entering into details about programming we first discuss the effects of the different operations upon the routing of numbers through the registers of the arithmetic unit.

We can divide the operations into two classes:

a) The instruction X. Upon this instruction no arithmetic operation takes place, i.e.: no transfers or calculations with numbers are effected.

b) All other instructions. They all indicate a transfer or an addition (subtraction).

So we can reserve one binary digit for the denomination X—*non-X*. 0 means X, 1 means *non-X*. Herefore we use the first digit.

The next classification of the instructions is:

a) Reading instructions. Here a number is extracted from the memory. They are X, P, Q, O and A.

b) Writing instructions. Here a number is transferred to the memory. The instructions are S and R.

The second digit can be used for denoting this. 0 means read; 1 means write.

Then we have instructions upon which the accumulator is cleared or not. In clearing, a path for leading the number back to the accumulator is simply interrupted. The third digit of the operation is denoting "clear" or "not clear". 0 means clear, 1 means not clear.

The fourth digit remains for the indication "positive" or "negative". This only applies for adding instructions P, Q, O and A. 0 means add, 1 means subtract.

We now get the following table of instructions and their actual coding in the machine.

TABLE I

		X		NON-X	
		POS.	NEG.	POS.	NEG.
READ	CLEAR	X		P	Q
	NOT CLEAR			O	A
WRITE	CLEAR			S	
	NOT CLEAR			R	

TABLE II

	X—NON X	READ—WRITE	CLEAR—NOT CL.	POS.—NEG.
X	0	0	0	0
P	1	0	0	0
Q	1	0	0	1
O	1	0	1	0
A	1	0	1	1
S	1	1	0	0
R	1	1	1	0

There are 9 other possible instructions with 4 digits. The negative version of storing would not be possible because making a number negative is an arithmetical operation which can only be done in the adder. The operations 1101 and 1111 act just the same as $S = 1100$ and $R = 1110$. On an X-instruction the third and fourth digit have no influence. Only when the second digit is 1, it causes the X-instruction to read no new instruction, but to write a number. At the same time it reads 0, so the next instruction becomes X 0, the latter stopping the machine (see § 6 and § 10).

Considerations of a practical nature led to the positioning of the four operation digits at the extreme left of a register, so the X—*non-X* digit occupies the same place as the sign digit. The address is written in the extreme right of a register, so the address can also serve as a natural number. Futhermore a one must be added to the address. This can be the least significant one which simplifies the construction of the one-generator. The one-generator always emits a one on the first pulse time and can then be shut off to zero. If it had to emit a one on another place, a counter would be needed.

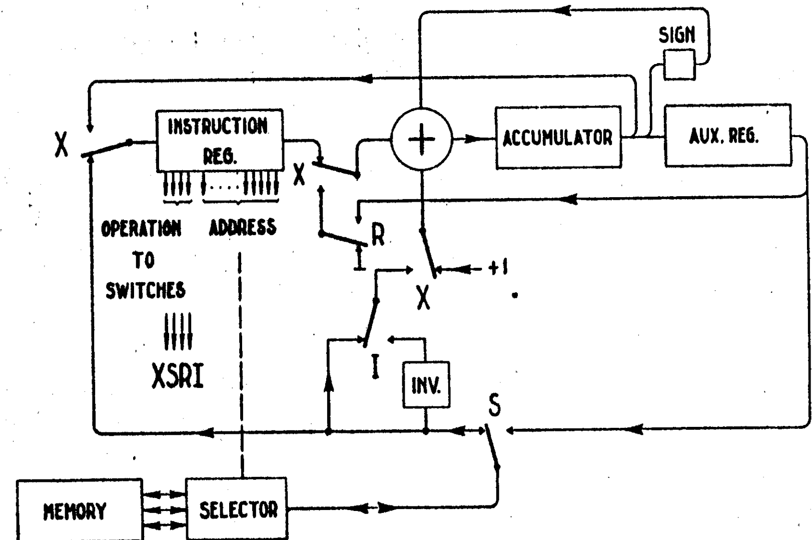


Fig. 1. Schematic diagram of the arithmetic unit and the control unit.

We can now draw a schematic diagram of the arithmetic unit and the control unit (fig. 1). The address part of the instruction goes to the selection mechanism of the memory. The operation part controls

the routing of numbers through the various parts of the machine. The switching in these paths is symbolically designated by ordinary contacts which are drawn in the quiescent state. The four digits of the operation control the following switches:

The first digit	X—non-X	by contact X,
The second digit	read—write	by contact S,
The third digit	clear—not clear	by contact R.
The fourth digit	positive—negative	by contact I.

The box in which has been written inv. is an inversion box making a number negative, i.e. replacing zeros by ones and vice versa (see appendix I). As can be verified, the interconnections in the small diagrams just result, in each case, from the setting of the switches belonging to the particular instruction. The box "sign" will be explained in the next paragraph.

Of the seven instructions that are possible only three are strictly necessary. These are X, A and S. We can clear the register by means of S. Addition can be effected by two subtractions because $-(-a)=$ $=+a$. Of course many more instructions, even for quite simple programmes, are then required. The operation of the instruction contains in this case only the X—*non*-X digit and the read—write digit. The contact I is always on the other side, the contact R must be operated together with S. Actually the machine worked in this fashion during the first experiments. We note here that it is also possible to build up subtraction from addition only.

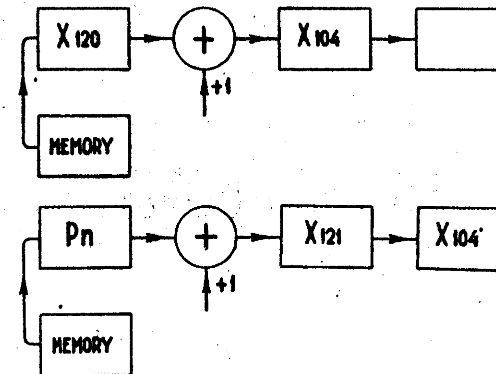
§ 6. *Unconditional and conditional transfers. Sub-programmes.* We must first discuss some additional facilities of the arithmetic unit without which a calculating machine is not a universal machine in the Turing sense [14]). Perhaps this can best be shown by a few examples.

When we want to perform a programme, we cannot execute all instructions in one unbroken sequence, but such a programme can contain repetition cycles or sub-programmes which can be called into action at different points of the main programme. So we must have means for breaking the normal course of obeying instructions sequentially. This can be done by the instruction X.

Suppose the last example continued with

	103	$\underline{X\ 120}$
103 →	120	$P \quad n$
	121	etc.

In register 103 now stands the instruction: take in an instruction from 120. The following happens:



Instead of the normal alternation of X-instructions with *non*-X instructions, here two X-instructions are executed one immediately after another. In the accumulator now stands X 121, so the next instruction will be taken in from 121: the machine continues to proceed at 120 etc. In the auxiliary register X 104 is retained, giving an indication of the place where we came from. Now repetitive processes are possible in the programme, for example

120 →	100	. . .
	101	
	102	
	⋮	
	120	$\underline{X\ 100}$

When coming to the instruction in 120, the programme goes back to 100.

A second application of the X-instruction is the use of sub-programmes. Suppose we have to perform as a part of a larger calculation an often recurring process, for example in this machine a multiplication. We do not want to programme it explicitly each time a multiplication is needed, but we make once for all a multiplication

programme which we can use each time. At different places in the main programme the multiplication programme is called into action and at the end of this sub-programme it must proceed in the place where it came from (fig. 2).

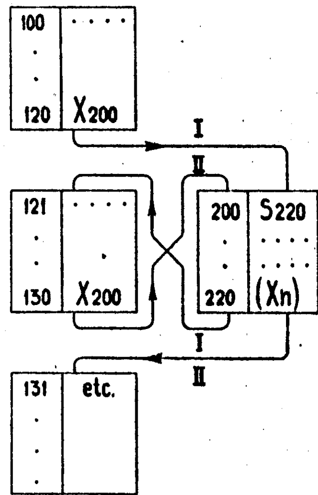
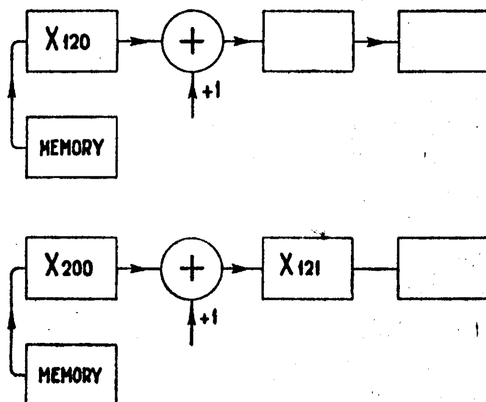


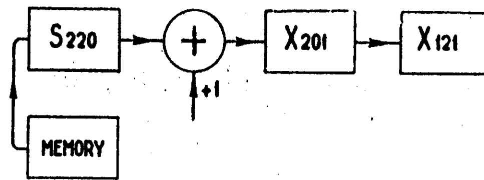
Fig. 2. Use of a sub-programme.

At 120 the sub-programme is called in. It must go back to 121. At 130 the sub-programme is called in again. Then it must go back to 131. This can be accomplished by simply making the sub-programme begin with 200 | *S* 220.

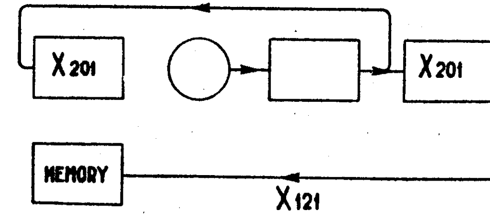
The subsequent stages are



Go to 200
X 121 is held in the accumulator.

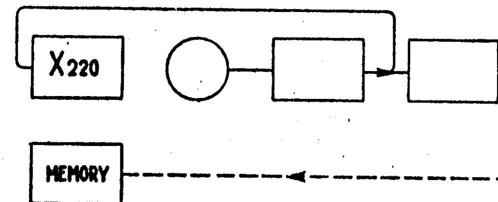


Store return instruction at the end of the sub-programme.

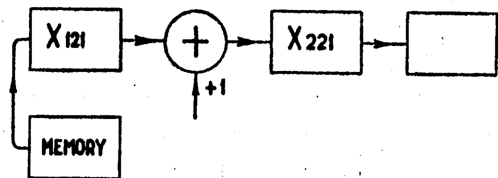


Proceed with 201.

Afterwards



Take in the return instruction.



Proceed with 121.

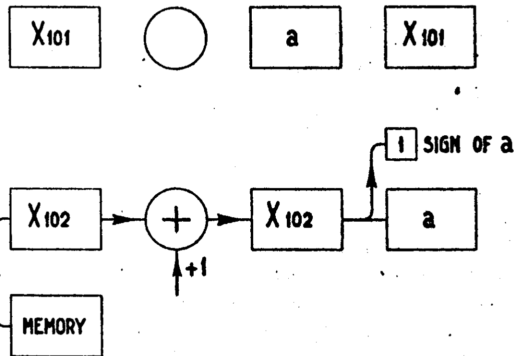
The second time the sub-programme is called in *X* 131 is stored as return instruction.

One of the most important functions of an automatic computer is its ability to make decisions. When we are doing a repetitive calculation, for example counting, we want the machine to be able to detect when it has reached a certain count and then alter the course of the programme. This machine also has means to make a discrimination on the sign of a number, put in the accumulator for this purpose. For doing a so-called conditional instruction we have a few additional components, so constructed that no special instruction is needed,

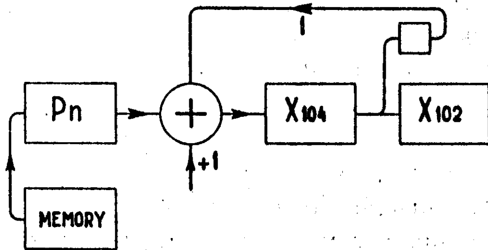
but only a special arrangement of the already existing instructions. This is done as follows. We want to go to the instruction in register 200 when the contents of register 30 are positive, but we want to proceed normally, i.e. sequentially, if this number is negative. The programming is

	100	P 30		
	101	X 102	30	a
	102	$\underline{P \quad n}$		
	103	$\underline{X \ 200}$		
102 →	104	etc.		

After P 30 we have taken in the number in 30, so the contents of the registers are



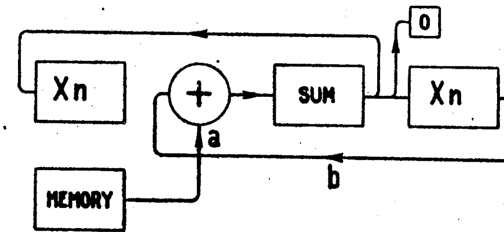
Now the dummy X-instruction directs the programme to the next instruction. The extra apparatus consists of a single flip-flop, receiving the same input as the auxiliary register. After it has gone through all digits of a which is pushed in, it contains at the end of the number cycle the last digit of a. This is just the sign digit. Suppose this sign to be 1. Now the next step is again an X-instruction.



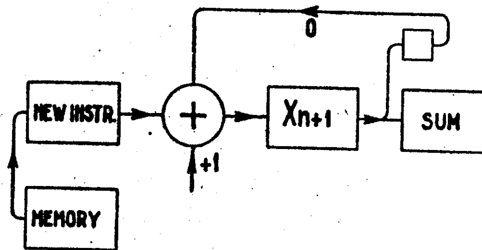
This causes the contents of the sign flip-flop to be added into X 102, together with the 1 which is always added during an X-instruction. Thus X 102 becomes X 104 and the instruction in 103 is skipped. (For adding the sign digit the already mentioned third input of the adder is used. See appendix II).

When a is positive, the sign flip-flop contains a zero. Then during instruction X 102 only 1 is added to X 102, the latter becoming X 103. After executing $P\,n$, it jumps to 200.

Of course, this part of the machine is always in action, but in the normal course of events every X-instruction is followed by a *non-X* instruction. During a *non-X* instruction an X-instruction is pushed into the auxiliary register and the sign flip-flop is filled with the



sign of X. As the most significant digit of an X-instruction is always zero ($X = 0000$), the following X-instruction always adds the normal one and a zero.



The same reasoning holds for the use of an X-instruction as unconditional transfer after clearing of the accumulator. In this case the accumulator contains only zeros which are pushed into the auxiliary register and the sign flip-flop. So this also receives a zero.

Recapitulation: An X-instruction in the programme operates as an unconditional transfer when the accumulator has been cleared by an instruction S or by P 0 or when it is certain that the accumulator contains a positive number. An X-instruction inserted after not clearing the accumulator operates equally as an unconditional transfer, but then the next instruction is conditional, i.e. the instruction

after this conditional instruction is skipped when the number taken in before the extra X-instruction was negative.

These facilities of the X-instruction can be combined, of course. For example with the aid of an X-instruction we can go to a sub-programme storing the return instruction. The same X-instruction can make this storing instruction conditional (fig. 3).

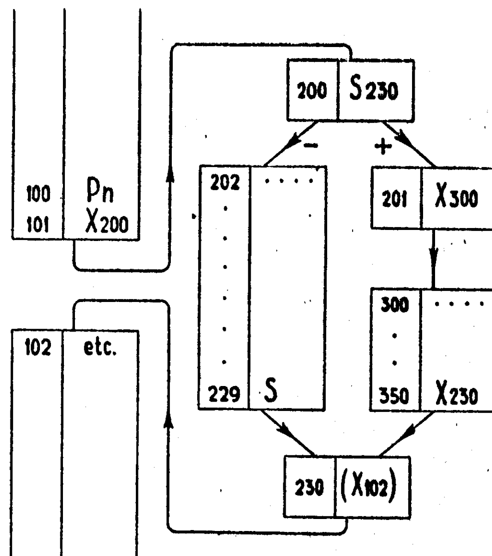


Fig. 3. Sub-programme with conditional entrance point.

In the following paragraphs we shall denote every unconditional transfer by an underlined X-instruction and every conditional transfer by a broken underlining. Entrance points are indicated in the programmes by an arrow together with the number of the register where we come from.

§ 7. *A simple multiplication programme.* We shall now give a programme for multiplication of two unsigned numbers of which the product does not exceed a single length register. The programme will be written in the form of a sub-programme.

The working positions will be allocated as follows:

0	X 0 = zero
1	X 1 = 1 = one-generator
4	Multiplier
5	Multiplicand
6	Product. At the start: multiplicand
9	Count for testing completion of operation

Before we are going to use the multiplication programme, we place multiplier and multiplicand in registers 4 and 6, respectively. The product will be formed in 6. The main programme can be for example

Entry	Reg.	Instr.		Description
→	200	P	20	Put a into 4
	201	S	4	
	202	P	51	Put b into 6
	203	S	6	
	204	X	100	Go to multiplication programme
121 →	205	P	6	Put result into 72
	207	S	72	

20	a
51	b
72	(ab)

The multiplication sub-programme now becomes:

Entry	Reg.	Instr.		Description	Remarks
→	100	S	121	Store return instruction at end of programme	Contents of 121 thus becoming X 205.
	101	P	6	Put multiplicand into 5	It is an advantage to have the product again in 6 so as to be able to form easily $a \times b \times c$.
	102	S	5		
	103	S	6	Clear 6 for product	After S 5 the accumulator is empty.
	104	P	1	Put count into 9	A one is put into reg. 9 as a count. This count is shifted at each digit of the multiplier till it reaches the sign digit.
	105	S	9		
120 →	106	P	4	Shift the multiplier by doubling it	By doubling the multiplier the next digit is taken into account each time. The sign digit is neglected by doubling the multiplier at the beginning already.
	107	O	4		
	108	R	4	Store without clearing	
	109	X	110	Prepare conditional transfer	When the most significant digit is 0, nothing must be added to the product, but when this digit is 1 the multiplicand must be added.
	110	P	5	Add multiplicand	
	111	A	5	If multiplier positive subtract multiplicand	
110 →	112	O	6	Add double the previous partial product	
	113	O	6		
	114	S	6	Store partial product in 6	
	115	P	9	Shift count to the left by doubling it	
	116	O	9		
	117	R	9	Store without clearing	

118	X 119	Prepare conditional transfer	When the count becomes negative it indicates that all digits are used.
119	$\underline{P}$ _ $\underline{0}$	Dummy instruction taking in 0	
120	$\underline{X\ \ 106}$	If count is positive repeat calculation.	
119 → 121	$\underline{(X\ \ n)}$	If count is negative go back to main programme	In this register the return instruction has been placed by 100 | S 121.

§ 8. *Arrangement of the registers.* When we would use a serial type of a memory, as proposed, with about 30 numbers per revolution, all programmes would be exceedingly slow because every instruction would require a waiting time of half a revolution for the necessary number. Together with the selection of this instruction this would become just one revolution per instruction. For a machine which must do a multiplication by a fairly long programme this would be too slow. The sequence 106—120 consits of $\approx$ 15 instructions. This sequence is executed 30 times for one multiplication requiring $\approx$ 450 revolutions. There is a method, however, enabling to perform normal consecutive instructions in only two number cycles. The normal sequence of events is that always an X-instruction alternates with a *non*-X instruction. If the number which must be operated upon could be taken from the memory without a waiting time and if instructions were not placed in consecutive registers but in every second register, we could obey one instruction in two number times. Now there is in general only a small quantity of numbers in a calculation which must be always immediately available. Therefore we install a small quantity of short registers containing only one number. Then there is no searching time for these short registers.

The instructions are interspersed in the memory. The positions in one delay-line of 32 positions are then numbered for example

32	48	33	49	34	50	35	51	36	52	37	53	38	54	39	55	40	56	41	57	42	58	43	59	44	60	45	61	46	62	47	63	A

The selection is done by counting each second register until this count coincides with the address in the instruction register. On the first revolution the first, third, fifth, etc. registers are selected. On the second revolution the second, fourth, sixth etc. registers are selected. An X-instruction for unconditional transfer just waits till

the contents of the desired register come out. Note that jumping from 40 to 56 requires no waiting time, as 56 follows immediately after 40. A programme

60	P 45
61	etc.

also requires no waiting time, because 45 lies between 60 and 61, but

60	P 46
61	etc.

would require a whole revolution. In general p lies between n and $n + 1$ when $p = n + 16k$ where k is an integer. Of course we also want to reach 32 from 63 with one number time between them. That is why we insert an extra space (numbered A) which allows for this time. During this space the counter is reset and prepared for counting the other 16 number times of the next revolution. This results in a waiting time of three number times instead of one between 47 and 48. (It would have been practical to have 31 registers in one revolution. Then we should not need the extra position A and there would be no waiting time between 47 and 48. However, difficulties would then arise with the use of the five least significant digits of the address for selecting only 31 registers. See appendix III. The best remedy would be to omit register 63, 95 etc. altogether, but then an unconditional transfer would always be needed in 62, 94 etc. to preserve the continuity of the programmes).

The idea of using the instructions at the moment when they come out of the memory has been used already in several other machines, for example in England for the ACE. (National Physical Laboratories, Teddington) [7] [8]) and in America for the EDVAC (Moore School of Electrical Engineering, Univ. of Pennsylvania) [15]). Both machines have three address codes while a fourth address indicates the location of the next instruction. In the ACE this is done by a relative timing number, in the EDVAC by an absolute register number.

We shall postulate the following arrangement of registers: reg. 0—31 inclusive are short registers, which are always immediately available, reg. 32—63, 64—95 etc. are delay-lines of 32 numbers each (or tracks on a drum). Of the registers 0—31 we shall give a special purpose to 0—3 and to 30 and 31.

0	X 0	"zero"
1	X 1	"one" in the least significant place
2		auxiliary register
3	P 0	"one" in the most significant place
30	Output register	
31	Input register	

0, 1 and 3 contain constants. These registers do not consist of a delay-line but are wired in in the control. It is not possible to write in them.

Register 2 deserves a little explanation. Only the instruction register needs to consist of a series of flip-flops, the accumulator and the auxiliary register consist of short delay-lines. Of these registers the auxiliary register has received an ordinary location number. Now a few special instructions are possible.

X 2 Take in as a new instruction the number just calculated in the accumulator. This calculated instruction must always be an X-instruction, otherwise the programme proceeds with X 3, which makes no sense.

Example:

	32	P	20	20 contains 1, 2 or 3 dependent upon the calculation.
	33	O	35	Add the constant X 35. The result thus becomes X 36, X 37 or X 38.
	34	$\underline{X}$	$\underline{2}$	Constant.
	35	X	35	
34 →	36	X	a	The programme is transferred to a, b or c.
34 →	37	X	b	
34 →	38	X	c	

Observe that this process makes the conditional transfer theoretically superfluous.

P 2 No useful instruction. Equivalent to O 0 or R 0.
Q 2 This changes the sign of the contents of the accumulator.
O 2 This doubles the contents of the accumulator.
A 2 No useful instruction. Equivalent to P 0 or S 0.
S 2, R 2 } Not possible. Reg. 2 has no writing entrance.

The functioning of 30 and 31 will be described in a separate paragraph.

§ 9. *Example of a complete multiplication programme.* In general the programmer does not need to worry about timing of programmes. Only in standard sub-programmes it is worth-while to make them most economic in time. We must try to take advantage of the short registers and of free registers which are just in such positions that it takes no waiting time to reach them.

As an example we shall give a programme for signed multiplication which gives the complete double-length product. A short supplementary programme serves for rounding off the most significant part of the product. Also the already given short multiplication programme will be brought into the new form. For the arithmetical process used, see appendix I.

The allocation of the working positions is:

4	a	multiplier
5	h	„head" of result
6	b, t	first: multiplicand, afterwards: "tail" of result
7	b_h	"head" of long form of multiplicand
8	b_t	"tail" of long form of multiplicand
9	c	count
10	(—)	return instruction

When using the programme for signed, long multiplication, multiplier and multiplicand must be put into registers 4 and 6. Result appears in 5 and 6. Programme begins at 32.

When using the same long multiplication programme but rounded off, the rounded-off head of result appears in 6. The programme begins at 45. For short multiplication, multiplier and multiplicand must be put into 4 and 6, result appears in 6. This part begins at 128.

We now give the complete programmes:

→	32	S	112	Put return instruction into 112	There is no waiting time for 112 as $112 = 32 + 5 \times 16$
	33	P	49	Put A 1 in carry-over sub-programme	A special carry-over programme serves normally for bringing digits from tail to head. However, for producing long form of b and for complementing long form of b, carry-over must be negative which is done by A 1. P 49 takes no waiting time as $49 = 33 + 16$.
	34	S	104		
	35	S	5	Clear reg. 5	
	36	$\underline{X}$	$\underline{83}$	Go to sub-programme to produce long form of b	Sub-programme can always be back within one revolution.
10 →	37	P	5	Put b_h into 7	
	38	S	7		
	39	P	6	Put b_t into 8	
	40	S	8		
	41	S	5	Clear 5 and 6	
	42	S	6		

From	Addr.	Instr.	Description	Remarks
	43	P 4	Test sign digit of multiplier	The dummy X-instruction preparing the conditional transfer is at the same time an unconditional transfer to 61 (one waiting time).
	44	X 61	and go to 61	
→	45	S 59	Begin of rounded-off multiplication progr.	
	46	X 32	Store return instruction and go to 32	Storing takes one revolution.
10 →	47	P 6	Test sign of tail and prepare conditional transfer	
	48	X 116	and go to 116	
	49	A 1	Constant.	
97 →	50	P _ 0	Dummy instr. Conditional transfer	
	51	X 71	When last digit is not treated, go to 71	
50 →	52	P 112	When last digit is treated, take in return instruction	
	53	S 10	Store this in 10	
	54	Q 5	Take in tail and prepare test for sign of tail	Because zero would still be positive when taken in negatively, a one is subtracted, thus forming pseudo-complement.
	55	A 1		
	56	X 106	Go to 106	Here a part of the normal carry-over programme serves for providing the tail with the sign of the head.
126 →	57	P 5	Put rounded-off product into 6	Product is placed into 6, so repeated products can be formed easily (e.g. $a \times b \times c$).
	58	S 6		
	59	(— —)	Return instruction	
	60			
44 →	61	Q _ 7	Take in b_k negatively	The long form of b is made negative here.
	62	X 67	If multiplier pos.: go to 67	If sign of multiplier is positive the negative long form needs not be formed so programme goes to 67
61 →	63	S 5	Otherwise: store $-b_k$ into 5	
	64	Q 8	Take in $-b_t$ and store into 6	The carry-over programme serves here to make long form of $-b$ out of b by carrying over the sign digit negatively.
	65	S 6		

From	Addr.	Instr.	Description	Remarks
	66	X 83	Go to carry-over subprogramme	
62 ↘ 10 ↗	67	P 115	Alter the carry-over subprogramme for carrying over +1 instead of —1.	No waiting time for 115 as $115 = 67 + 3 \times 16$.
	68	S 104		
	69	P 1	Store shifting count in 9	Counting is again effected by shifting (doubling) 1 until at last it occupies sign digit.
	70	S 9		
51 →	71	P 5	Double head of partial result	In doubling the long form of the partial result, a carry-over from tail to head can arise.
	72	O 5		
	73	S 5		
	74	P 6	Double tail of partial result	This digit occupies the place of the sign digit and is carried to the head by carry-over sub-programme which takes one revolution.
	75	O 6		
	76	S 6		
	77	X 83	Go to carry-over subprogramme	
10 →	78	P 4	Double multiplier, so next digit occupies place of sign digit	
	79	O 4		
	80	R 4	Store without clearing	
	81	X 86	Prepare test of sign of multiplier and go to 86	
	82			
36, 66, 77, 93, 125 →	83	S 10	Begin of carry-over subgrogramme. Store return-instruction, take in tail for testing carry-over.	Return instruction is stored in short register to make possible return within one revolution.
	84	P 6		
	85	X 102	Prepare conditional transfer and go to 102	
81 →	86	P _ 8	Take in b_t	Tail of long form has no sign digit, thus X-instruction in 87 cannot work conditionally.
	87	X 94	If multiplier pos.: go to 94	
86 →	88	O 6	If multiplier neg.: add multiplicand to partial result. First tail	
	89	S 6		
	90	P 7	Then head	
	91	O 5		
	92	S 5		
	93	X 83	Go to carry-over progr.	Carry-over programme returns within one revolution.

10 ↘ 87 ↗	94	P 9	Shift count by doubling and store again in 9 without clearing	
	95	O 9		
	96	R 9		
	97	X 50	Prepare test for sign of count	
	98			
146 →	99	P _ 0	Dummy instruction Conditional instruction	In any case a positive number is in the accumulator. So X 134 can never work as a conditional instr. This part belongs to short multiplication programme.
	100	X 134	If count pos.: go back to 134	
99 →	101	X 7	Otherwise go to return instruction	
85 →	102	P _ 5	Take in head of result	
	103	X 10	If tail is pos.: no carry-over	Sub-programme immediately goes back to normal progr.
102 →	104	(— —)	If neg.: Add (or subtract) carry-over to tail and store in 5	This instruction can be O 1 or A 1.
	105	S 5		
56 →	106	P _ 6	Dispose of sign digit of tail by adding P 0 = most significant one	Same instruction can be used conditionally at the end of multiplication programme to provide tail with sign digit of head.
	107	O 3		
106 →	108	S 6		
	109	X 10	Return instruction	78, 94 and 126 can be reached within one revolution.
	110			
	111			
	112	(— —)	Temporary location of return instruction of multiplication programme	
	113			
	114			
	115	O 1	Constant	
48 →	116	P _ 0	Dummy instruction. Take in zero	To round off a number, the digit following the sign digit of the tail, which has been made positive, must be added to or subtracted from the head as round-off digit, if the head is positive or negative, respectively.
	117	X 122	If sign of tail is pos.: go to 122	
116 →	118	P 49	A 1 is put again in carry-over sub-programme	
	119	S 104		
	120	Q 6	Make tail positive	
	121	S 6		
117 →	122	P 6	Double tail	
	123	O 6		
	124	S 6		
	125	X 83	Go to carry-over sub-programme	The same carry-over sub-programme is used now.

10 →	126	X 57	Go to 57	When head is positive, the instruction O 1 is left in 104, but when head is negative, A 1 is substituted.
	127			

The total time for one multiplication depends on the quantity of ones in the number. On the average the time for one multiplication, not rounded-off, is 113 revolutions and for multiplication, rounded-off, is 116 revolutions.

No explanation will be given of the following short multiplication programme as the action is the same as in § 7. Also the same working positions are used. The time needed is 31 revolutions.

→	128	S 7	100 →	134	P 4	138 →	140	O 6
	129	P 6		135	O 4		141	O 6
	130	S 5		136	R 4		142	S 6
	131	S 6		137	X 138		143	P 9
	132	P 1		138	P _ 5		144	O 9
	133	S 9		139	A 5		145	R 9
							146	X 99

§ 10. *Input and output.* Input and output is provided for by two special registers 30 and 31. We assume having a teletype tape as input medium. Each line of the tape consists of five binary digits, a symbol. Register 31 is a flip-flop register of only five flip-flops. When the programme encounters an instruction P 31, Q 31, O 31 or A 31, the contents of this register of five positions are pushed into the accumulator of 31 digit positions. This causes the five digits of register 31 to be put into the least significant five places of the accumulator. Immediately after being read off, register 31 makes the tape step one line and takes in the new line of five digits. When reading a new symbol before the tape has completed a step after reading the previous symbol, the machine waits. It is not possible to write into register 31.

Output is arranged in the same manner as the input. Register 30 also consists of five digits only. When an instruction R 30 or S 30 writes the contents of the auxiliary register of 31 digits into register 30, only the five most significant digits remain in 30 after shifting through all other digits. Each time a number is written in 30, the five digits are typed on an automatic typewriter or are punched on a teletype punch. Register 30 cannot be read off by a read-instruction. (It would be possible to use the same location number for input and output and make the read-write digit choose between the actual input and output register).

As a last example we shall give an input programme for putting in instructions from the tape. This programme is by no means the best possible (this always depends on the situation), but only serves as an example.

Normally instructions are put in consecutive registers. Each instruction consists of 6 lines of coding on the tape. The first tape symbol is normally 1. The second symbol denotes the operation. The coding for this digit is:

$$X=0 \quad P=1 \quad Q=2 \quad O=3 \quad A=4 \quad S=5 \quad R=6$$

The third to sixth symbol give the address in decimal form. During input conversion to binary code takes place.

The first symbol can assume other values than 1. In general this symbol indicates the number which must be added to the address in which the last instruction has been put in. So we can skip a number of registers before putting in the next instruction. The number 0 serves for the purpose of replacing the store instruction which every time stores the instruction just put in. By virtue of this we can begin to put in instructions from a pre-determined memory location. We can also leave the input programme by replacing the store instruction by an X-instruction.

We shall give a small example. Suppose we want to put in the programme

	Reg.	Instr.	Addr.	
	300	*P*	15	} Put *a* into reg. 6
	301	*S*	6	
	302	*Q*	16	} Put —*b* into reg. 4
	303	*S*	4	
	304	<u>*X*</u>	<u>45</u>	Go to multiplication programme
59 →	305	<u>*X*</u>	<u>310</u>	Go to 310
305 →	310	*P*	6	} Put —*ab* into reg. 15
	311	*S*	15	
	312	*X*	0	Stop (go to 0)

Reg.	Contents
15	*a*, later —*ab*
16	*b*

(X 0 serves as a stop instruction, because the machine then always takes in X 0 for the rest of the time. A simple integrator network detects the permanent absence of pulses in the accumulator and rings a bell.)

We want to put in this programme and begin at 300. So on the tape comes

	0	5	0300	Begin to put in at 300
300	1	1	0015	*P* 15
301	1	5	0006	*S* 6
302	1	2	0016	*Q* 16
303	1	5	0004	*S* 4
304	1	0	0045	*X* 45
305	1	0	0310	*X* 310
310	5	1	0006	*P* 6 Skip 306—309
311	1	5	0015	*S* 15
312	1	0	0000	*X* 0
	0	0	0299	Replace the store instruction by X 299.
	1	0	0000	Add 1 to the store instruction, placing 300 there. Then the programme begins to be executed in 300.

Now we can give the actual coding of the input programme.
The working positions are

	Reg.	Contents	Remarks
185 →	7	*P* 6	This is placed here at the beginning of the input programme. Take in address of instruction.
	8	(*O* 180 + *x*)	Instruction which adds the operation digits to address. Takes one revolution.
	9	(*S* *n*)	Store instruction. Takes also one revolution in general.
	10	(<u>*X* *m*</u>)	Return instruction. The instructions between braces are substituted by the machine itself.

The programme begins at 160

	Reg.	Instr.	Addr.	Explanation	Remarks
→	160	*P*	148	} Put *P* 6 into reg. 7	Reaching 148 takes one revolution. For reg. 148 see page 394.
	161	*S*	7		
179 →	162	*Q*	31	Take in a symbol from the tape negatively	
	163	*R*	5	Store symbol in 5	
	164	*X*	165	Test sign of symbol	Only when symbol is 0, it proves to be positive.
	165	<u>*P*</u>	<u>166</u>	Bring *S* 9 in accumulator	This instruction takes one revolution.
	166	*S*	9	If symbol was neg.: store suicide instr.	*S* 9 stored in 9 causes suicide of this instruction, thus putting in new store instruction.
165 →	167	*P*	9	} If positive: add symbol to store instruction	Properly speaking: subtract negative symbol.
	168	*A*	5		
	169	*S*	9		
	170	*P*	31	} Take in next symbol (operation of instruction)	
	171	*O*	187	} Add *O* 180 to this	As 187 = 171 + 16 there is no waiting time. Out of the operation symbol the machine calculates its own instruction for finding the proper operation be added.
	172	*S*	8	} Store in 8	
	173	*P*	31	} Take in first digit of address	
	174	S	6	} Store in 6	
	175	<u>*X*</u>	<u>147</u>	Go to sub-progr. forming the tenfold of contents of 6 and add next symbol	Sub-programme requires one revolution.

From	Reg.	Instr.	Addr.	Comment	Remark
10 →	176	$\underline{X}$	$\underline{147}$	Ditto	
10 →	177	$\underline{X}$	$\underline{147}$	Ditto	
10 →	178	$\underline{X}$	$\underline{188}$	Go to sub-progr. to store formed instruction	Requires 3 revolutions in general.
10 →	179	$\underline{X}$	$\underline{162}$	Go back to 162 and take in next instruction	
	180	X	0	Pre-fabricated operation digits (180–186)	
	181	P	0		
	182	Q	0		
	183	O	0		
	184	A	0		
	185	S	0		
	186	R	0		
	187	O	180	Constant for add. instr. in 8	
178 →	188	S	10	Store return instruction	
	189	$\underline{X}$	$\underline{7}$	Go to 7	

The sub-programme for forming the tenfold just fits in after short multiplication programme.

From	Reg.	Instr.	Addr.	Comment
175, 176, 177 →	147	S	10	Store return instruction
	148	P	6	Take in part of address already formed
	149	O	2	Double this number
	150	R	5	Store in 5 without clearing
	151	O	2	Make fourfold
	152	O	2	Make eightfold
	153	O	5	Add twofold resulting into tenfold
	154	O	31	Add next symbol
	155	S	6	Store in 6
	156	$\underline{X}$	$\underline{10}$	Go to return instruction in 7.

The time taken by this programme is 9 revolutions per instruction.

The machine cannot start work with a completely cleared memory. We shall not give details, but we mention that a manual input is provided. With this manual input it is possible to fill a prescribed register with a number or instruction. An extremely simple input programme can then be put in to take in a larger and more perfect programme in its turn. Of course a machine with a magnetic drum retains its information in the memory indefinitely.

§ 11. *Advantages and drawbacks.* There are several advantages worth mentioning.

a) The machine is logically simple.

b) It has a flexible system of programming that can easily be learnt. Straight-forward programming is simple because there are only seven instructions. More intricate programmes can be built up

of standard sub-programmes, which can be made once and for all. So only these programmes need special care with respect to their timing to make them most economical. As the user can make the programmes just as he wants them, use of the machine is very flexible.

c) There are only a few parts. The memory including accumulator and auxiliary register, and the instruction register consist of a large number of identical elements. This makes fault location very easy.

d) As the memory constitutes the bulk of the machine, the price is mainly determined by this part.

Drawbacks are:

a) The loss of speed, especially for multiplication, is very serious. In this respect a machine with a magnetic drum is worse than one with mercury delay-lines. Although the largest part of the calculations in an automatic computer are concerned with addition, counting, altering programmes, conditional transfers etc. which are executed very quickly, its overall speed is mainly determined by the multiplication time.

b) A considerable number of registers is needed for multiplication programmes. Complexity of the circuits has been exchanged for capacity of the memory.

c) In one register only one instruction can be stored. Although there would have been room for two instructions in one register, no simple arrangement can be made for that purpose without upsetting the logical structure of the machine.

§ 12. *Results.* The machine as it has been built at the laboratory successfully solved a number of problems. Its storage capacity was rather limited (64 registers), but we can mention the following programmes:

Simple multiplication programmes of various types.

Unsigned multiplication of double-length product.

Summing of a few simple power series.

Division programmes.

Square rooting.

Conversion and deconversion.

An iterative process for the calculation of e^x.

Evaluation of some algebraic forms.

A programme for playing NIM [17]).

This last programme could play NIM with three heaps of articles

of any number, not exceeding capacity. When the machine was given the initial position, it could calculate the right move. This right move was indicated: *a*) by the number of the heap from which objects ought to be removed, *b*) by the number of objects this heap should have to contain. When the machine was offered a winning position, it lost the game by dictating the impossible move: "remove from heap number 4".

Acknowledgement. I wish to express my thanks to Dr. L. Kosten for his constant interest and his many helpful suggestions. The ideas for programming NIM were due to him. For the realisation of the machine the technical skill of Mr. G. J. de Zwart was indispensable.

APPENDIX I

Subtraction is based upon the complement system of representing negative numbers. For example —0.1011 is written as 1.0101. We can also say that all numbers are reduced modulo 2. The digit left of the binary point serves as sign digit $(0 = +,\ 1 = -)$. The numbers which can be represented lie always in the range $1 > x \geqslant -1$. The process for obtaining a complement in the machine is to replace all zeros by ones and vice versa and then to add the complement one.

The process used for multiplication is straight-forward as to positive numbers. When the multiplicand is negative, all partial products become negative (all numbers must be filled out on the left with the sign digit). The only complication arises when the multiplier is negative. The sign digit cannot be treated as the other digits of the multiplier, for then we would assign the value $+1$ to the sign digit. We can solve this difficulty by interpreting the sign digit negatively, thus 0 reading as $-0\ (= +0)$ and 1 reading as -1. So we read

$$1.0101 = 1.0000 + 0.0101.$$

$$-\tfrac{13}{16} = -1 + \tfrac{5}{16}.$$

```
Example  1.1001
         1.0101
         ------
         0.0111
         1.111001
         1.11111001
         ----------
         0.01001101    Carries on the left are lost.
```


Another difficulty arises with the multiplication of two numbers of single length, giving a product of double length. As we have only registers of single length in the memory and in the arithmetic unit, we are compelled to handle the result in two parts which we shall call "head" and "tail" (for most significant part and least significant part, respectively). In multiplication two elementary processes occur: doubling the partial result and adding the next partial product. In both processes a carry-over from tail to head can originate. For this carry-over the sign digit of the tail is used.

```
For example:  0.0011   0.1101
    shifted:  0.0110   1.1010
                   1        1   Add carry and remove sign
              ------   ------     of tail
              0.0111   0.1010
```

When the multiplicand is negative, also a double length version of this number must be made in advance, before multiplication can start. For example

```
short form             1.0101
long form     1.1111   0.0101
```

As we can see, this long form can be made by nearly the same process as for carrying over at shifting, with the difference that the carry-over must be subtracted from the head (which is zero in this case).

The first digit in the multiplier we encounter is the sign digit. This digit must be interpreted negatively, so when it is 1, we have to subtract the long version of the multiplicand. We can form that by complementing both parts separately and we must carry-over negatively then. For example

```
                            1.1111   0.0101
both parts made negative    0.0001   1.1011
                                 1        1   Subtract carry and
                          - ------   ------
                            0.0000   0.1011   remove sign of tail
```

In the multiplication programme of § 9 the same sub-programme is used for carrying over at shifting or adding double-length numbers, but also for manufacturing the long form of the multiplicand and for complementing this.

We have adopted the convention of always providing the tail of the product with the sign of head. Part of the same carry-over programme is now used for adding this sign digit instead of removing it.

Rounding off takes place when the modulus of the tail is greater than a half, which can be seen from the digit following the sign digit. When this digit differs from the sign digit, the number is greater than a half. This round-off digit is added to the head when the number is positive, or subtracted when the number is negative. Again the same carry-over programme is used for adding or subtracting this round-off digit, which has been put in the place of the sign digit of the tail by doubling this tail.

Appendix II

The circle on the drawings which denotes the adder has some functions to be effected which could only be briefly mentioned in the article. As we have seen already, the adder must be capable of adding three digits at a time: *a*) the digit of the first number, *b*) the digit of the second number, *c*) the carry-over from the previous addition of two (three) digits. This carry-over must be delayed one digit time.

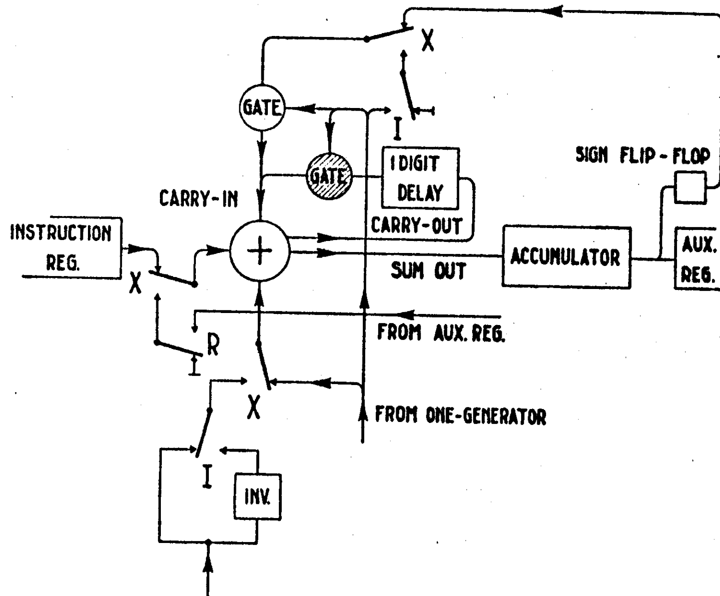


Fig. 4. Adder.

When an addition begins, the first carry-in requires a special treatment. For a *non-X* instruction this carry-in must be a 0 for addition and a 1 for subtraction to comply with the rules of the complement system for subtraction. For an *X*-instruction the first

carry-in must be the contents of the sign flip-flop. The complete schematic diagram of this part is shown in fig. 4. The switches are again meant symbolically. The unhatched gate only passes a pulse when both inputs receive pulses. The hatched gate is normally open and passes the carry-over pulse when the one-generator does not emit a pulse. The one-generator serves also for supplying the complement one for subtraction.

Appendix III

The selection mechanism also deserves a more detailed discussion. Before the address has reached 32, one of the short registers must be selected, and as these registers are in parallel, they require a selection tree. The five least significant digits of the address go to this tree. When the address is a number greater than 32, the least significant five digits must indicate the position of a number in a delay-line (or track) and the next digits must go to a selection tree for switching the inputs and outputs of the different delay-lines (tracks) (fig. 5).

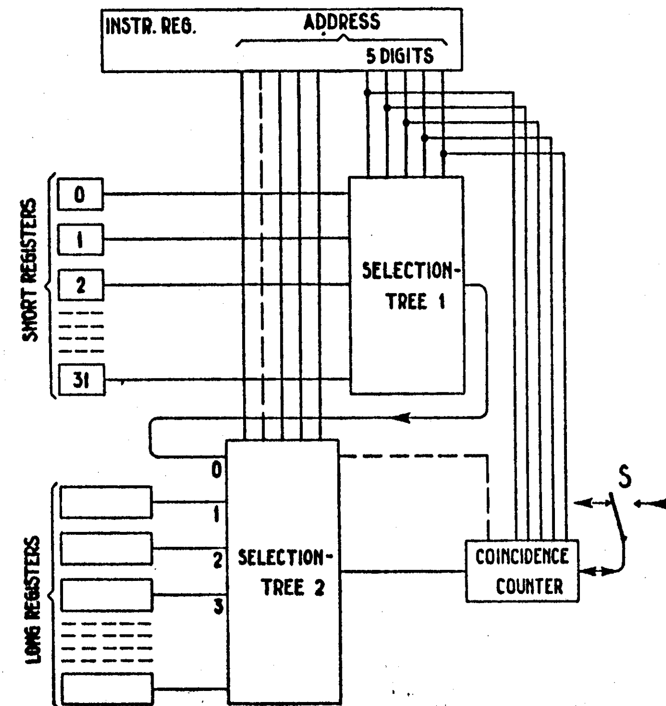


Fig. 5. Selection mechanism.

A counter detects the coincidence between the least significant five digits of the address and the number of the register just coming out. When this coincidence takes place, it opens the register for one number time. When tree 2 selects 0, the coincidence counter is switched off as the contents of a short register are always immediately available. This is indicated by a broken line in the figure.

We can observe another fact from the figure. When selection of a certain register takes place, the contents of the instruction register remain the same until coincidence takes place. Then the chosen register must be opened during one number time. During that time the selection tree must still select the same register, although the instruction in the instruction register is being replaced by another instruction. So it is clear that the selection tree must remember the selected register during extraction. In this way it serves a memory function. The same can be said of the switches *X*, *S*, *R* and *I*. These switches also memorize the operation which is just being executed.

Received 17th December 1951.

REFERENCES

1) Neumann, J. v., H. H. Goldstine and A. W. Burks, Preliminary discussion of the logical design of an electronic computing instrument. Ord. Dept. U.S.A. (1947).

2) Hartree, D. R., Calculating instruments and machines; Univ. Ill. Press (1949).

3) Proceedings of a symposium on large scale digital calculating machinery. Ann. Harv. Comp. Lab. **16** (1948).

4) Huskey, H. D., Report on the ENIAC Univ. of Pennsylvania (1946).

5) Report of a conference on high-speed automatic calculating machines. Univ. math. Lab. Cambridge (1950).

6) West, C. F., and J. E. De Turk, Proc. Inst. Radio. Engrs **36** (1948) 1452.

7) Wilkes, M. V., J. sci. Instr. **26** (1949) 385.

8) Wilkinson, J. H., Proc. Roy. Soc. **195** (1948) 265.

9) Automatic computing equipment at the N.P.L., Engineering **171** (1951) 6.

10) Sharpless, T. K., Electronics **20** (1947) Nov. 134.

11) Wilkes, M. V., and W. Renwick, Electronic Engns **20** (1948) 208.

12) Auerbach, I. L., J. P. Eckert, R. F. Shaw and C. B. Sheppard, Proc. Inst. Radio Engrs **37** (1949) 855.

13) Booth, A. D., Electronic Engng **21** (1949) 234.

14) Turing, A. M., Proc. Lond. math. Soc. **42** (1936) 230.

15) Koons, F., and S. Lubkin, Math. Tables Aids Comput. **3** (1949) 427.

16) Booth, A. D., Electronic Engng **22** (1950) 492.

17) Stuart Williams, R., Electronic Engng **23** (1951) 344.