Chapter 12

ZEBRA, a simple binary computer[1]

W. L. van der Poel

Summary The computer ZEBRA is a computer based on the following ideas:

1. The logical structure of the arithmetic and control units of the machine have been simplified as much as possible; there is not even a built-in multiplier nor a divider.
2. The separate bits in an instruction word are used functionally and can be put together in any combination.
3. Conventional two stage operation (set-up, execution) has been abandoned. Each unit time interval can be used for arithmetical operations.
4. A small number of fast access registers is used as temporary storage; at the same time these registers serve as modifier registers (B-lines).
5. Optimum programming is almost automatically done to a very great extent. The percentage of word times effectively used is usually greater than 60%.
6. An instruction can be repeated and modified while repeated by using an accumulator as next instruction source and the address counter as counter. This can be done without any special hardware.

This has resulted in a machine which has a very simple structure and hence contains only a very moderate number of components, giving high reliability and easy maintenance. Because of the functional bit coding, the programming is extremely flexible. In fact the machine code is a sort of micro-programming. Full-length multiplication or half-length multiplication in half the time are just as easy, only require a different micro-programme. The minimum latency programming together with the effective use of word times lost in other systems results in a very high speed of operation compared to the basic clock pulse frequency.

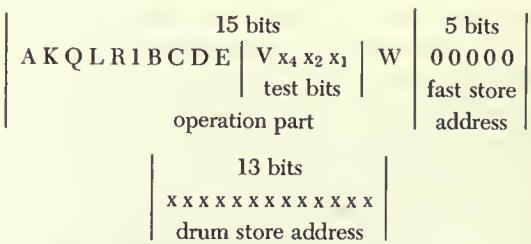
Introduction

In the Dr. Neher Laboratory of the Dutch Postal & Telecommunications Services the logical design of a computer called ZEBRA has been developed, and this computer has been engineered and constructed by Standard Telephones & Cables Ltd, England. The logical system is so different from most computers, that it is worth while to devote a special lecture to it. As time is limited, no technical details nor questions about dimensions or capacity will be discussed. They can all be found in the literature [van der Poel, 1956; van der Poel, 1952].

The main idea of the machine is to economise as far as possible on the number of components by simplifying the logical structure. For example, multiplication and division are not built in but must be programmed. Of course this system can only work with an appropriate internal code which has enough properties to execute basic arithmetic and logical routines effectively. In fact, the internal machine code is more or less a system of microprogramming [Wilkes and Stringer, 1953].

Operation part of the instruction

The most conspicuous, but probably not the most important, characteristic is the functional use of the separate bits in the operation part of an instruction. An instruction word in ZEBRA is composed as follows:



It is a binary, two-address machine with one address of 13 bits for the selection of a location in the main store (a drum of 8192 locations divided into 256 tracks of 32 words each), and a second address of 5 bits for the selection of one of 12 fast access store registers and several permanently wired locations (e.g., input, output, accumulators, constants). The operation part has 15 bits, each one having a separate and independent meaning. The most important of these are the A, K, D and E bits.

A- and K-bits

There are four main components in the machine: the drum store, the fast store, the arithmetic unit and the control. The A-bit in the instruction controls the interconnection of the drum and the

[1]*Proc. ICIP, UNESCO*, pp. 361–365, June, 1959.

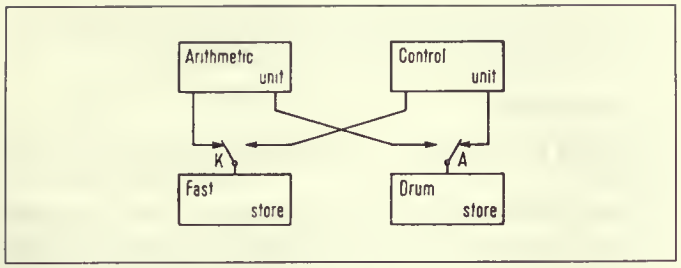


Fig. 1. The main units of the computer.

arithmetic unit or the control. In the same way the K-bit controls the interconnection of the fast store with the arithmetic unit or the control unit. These interconnections can be seen from Fig. 1.

It will be seen that A and K can have 4 possible combinations:

Case 1. A = 0, K = 0. This is called the adding jump (Fig. 2*a*).

While a new instruction is coming into the control from the drum, the arithmetic unit can at the same time do an operation with the operand coming from the fast store. This is the fastest type of operation. When the following instruction is placed in the next location on the drum there is no waiting time, and 32 instructions of this type can be executed per revolution. (One revolution = 10 ms, one word time = 312 μs.)

Case 2. A = 0, K = 1. This is called the double jump (Fig. 2*b*).

Both stores are now used for giving information to the control, i.e., making a jump. Since the fast store is used for the control, the instruction coming in from the drum is modified by the contents of a fast register. In this way the B-line facility, as it is often called, is realised.

Case 3. A = 1, K = 0. This is called the double addition (Fig. 2*c*).

Both stores are now connected to the arithmetic unit. The control must take care of itself using the address counter which is *stepped up by 2 at a time*, thus enabling this type of instruction to reach the number lying between the two successive instructions without any waiting time. Constants in particular will always be taken from optimum places on the drum.

Case 4. A = 1, K = 1. This is called the jumping addition (Fig. 2*d*).

While the drum is used for the arithmetic unit the address counter is modified by a fast register. Control may thus be passed to any instruction, and not only to the next instruction.

D- and E-bits

The functional bits D and E control the direction of flow of information.

D = 0 means: read from the drum.

E = 0 means: read from the fast store.

D = 1 means: write to the drum.

E = 1 means: write to the fast store.

A few possible instructions will be given below. In the written code a drum address will always be written with 3 or more digits and the absence of the A-bit will be indicated by the letter X. (This is necessary for the input programme to recognize the beginning of a new instruction.)

A200.5	Add ⟨200⟩ (the contents of address 200) and ⟨5⟩ to the accumulator. Step the address counter by 2.
X200E5	Take next instruction from 200 (= jump to 200) and store contents of accumulator in 5.
X200KE5	Jump to 200 and store previous contents of address counter in 5. This amounts to placing a link instruction for return from a sub-routine.
X200K5	Take next instruction from 200 but modify it with ⟨5⟩ thus making a variable instruction.

Arithmetic bits

The remainder of the function bits have arithmetic meanings. We shall only briefly indicate their different actions.

B: Do not use the A accumulator (most significant accumulator) but the B accumulator.

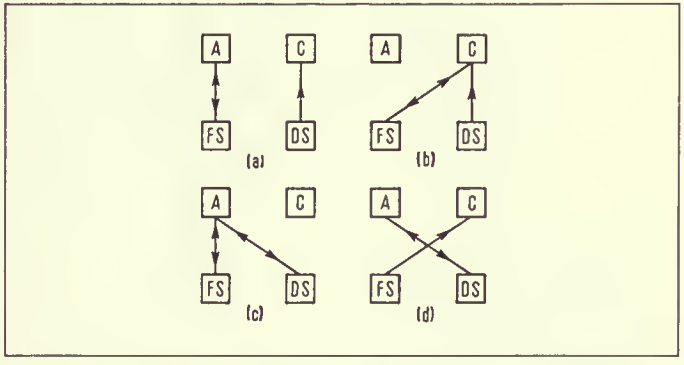


Fig. 2. The possible combinations of the A- and K-bits.

C: Clear the accumulator specified by B after storing, or before addition. (In a serial machine like ZEBRA this is automatically the case, cf. Fig. 3.)

I: Subtract instead of add.

Q: Add one (unit in the least significant place) to the B-accumulator.

L: Shift both accumulators one place to the left.

R: Shift both accumulators one place to the right. The accumulators are always coupled together in shifting except when C is present.

A few more examples will be given.

A200BCE25	Store ⟨B⟩ in 5, clear B and add ⟨200⟩ to B.
X200QLIBCE6	Jump to 200. Store ⟨B⟩ in 6, put −1 in B (because of QIBC) and shift the A accumulator one place to the left. Shifting from B into A is prevented by the presence of C.
X200RBC3	Jump to 200. Shift A to the right. Copy ⟨3⟩ into B. As register 3 is just an address for the B accumulator itself, this means that A is shifted while B is static.
X200K3QIBC	Take the instruction from 200 and modify it with the contents of the B accumulator (= register 3). Put −1 in B afterwards.

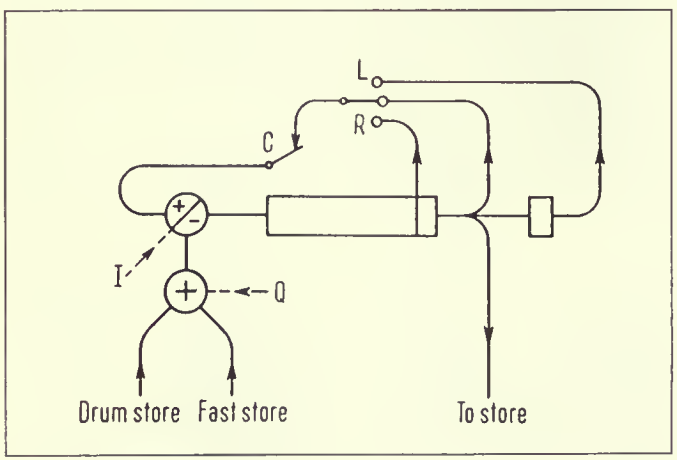


Fig. 3. Accumulator.

As can be seen, many complicated operations can be composed by the elementary possibilities of the separate bits.

The accumulator

A simplified block diagram of one of the accumulators is shown in Fig. 3.

Shifting is effected by looping the accumulator over one place less or one place more. In a double addition the contents of the drum store and the fast store are first added together in the pre-adder (possibly augmented by unity in the B accumulator, if Q is present) and this result is added into the accumulator (or subtracted in case of I). A clearing gate controlled by C interrupts the recirculation of the previous contents.

The control unit

The control unit has two shifting registers, the C-register which receives the next instruction to be executed and the D-register or counter. The block diagram is shown in Fig. 4. After a new instruction has come into C, it is taken over in parallel form into E in the interword time. It remains in E while the next instruction is coming into C. Let us explain the action of this control with a short programme.

Examples of programmes

100 X101E5
101 ACI02
102 constant
103 etc.

The actions in the several registers are now:

⟨A⟩	⟨C⟩	⟨D⟩	
	X100		Suppose X100 is in C at the start. This will take ⟨100⟩ into C. ⟨C⟩ + 2 → D.
	X101E5	X102	Another jump comes into C taking in ⟨101⟩ and storing ⟨A⟩ → 5. ⟨C⟩ + 2 → D gives X103E5.
AC102	X103E5		Note that the operational part is kept in the counter. The necessary constant from 102 is just becoming available.
const.	X103E5		The next instruction is taken from 103 which is immediately following. The constant in A is stored to 5 by E5, and is still active after coming back from D.

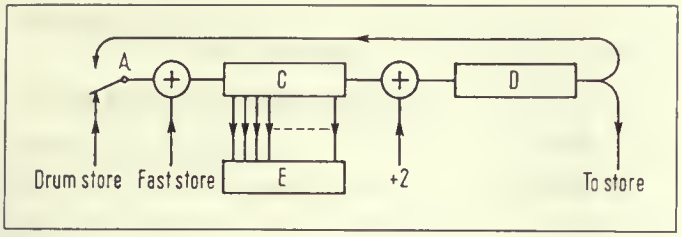


Fig. 4. Control unit.

This is the most important aspect of the machine. An instruction in the address counter comes back after an A-instruction and can do something useful. To our surprise we found that in many more cases than we first suspected, the second action could be used effectively. In most other computers the time of access to the next instruction is lost because nothing can be done concurrently in the arithmetic unit.

Another example of the action of the control is the jump to a sub-routine. Suppose that we have the following piece of programme:

100 X200KE5	Jump to sub-routine starting in 200. Place return jump in 5.
102 etc.	Sub-routine returns here.

The action is as follows:

⟨C⟩	⟨D⟩	
X100		The instruction is taken from 100.
X200KE5	X102	X200KE5 → C and X100 + 2 → D. Now KE5 stores D in 5. Thus ⟨5⟩ = X102.
(200)		The subroutine at 200 is executed and ends with XK5: jump to 5.
XK5		Take instruction from 5.
X102		Now the main programme proceeds to 102
(102)		etc.

By ending the sub-routine:

220 X221K5
221 − 1

we can return not two but one location further on, i.e., X221K5 takes as next instruction ⟨5⟩ − 1 = X101. Here 5 contains the instruction and the drum modifier.

The test bits

The digits V x_4 x_2 x_1 will not be dealt with extensively but the different combinations of these 4 digits represent different types of test. When for example V1 is attached to an instruction, this instruction will be executed when ⟨A⟩ is negative, but will be skipped altogether when ⟨A⟩ is positive or zero. The harmless A-instruction will then be executed instead. The test can be attached to a jump, giving a conditional jump, as well as to an A-instruction, giving a conditional addition.

The W-bit

So far the digit W has not been mentioned. When W is present in an instruction the drum address is not used. The instruction is not kept waiting but is immediately executed and the drum is completely disregarded. With the help of this digit W, jumps can be made to instructions in the fast store, e.g., XK5W takes the instruction from 5 only, and the drum does not deliver any number. The use of this type of instruction has very peculiar consequences. Let us take the following example:

100 X101KE6 ⟨5⟩ = ARW
101 X8186K5RW ⟨6⟩ = filled with return instruction
102 etc.

The action is as follows:

	⟨A⟩	⟨C⟩	⟨D⟩	
a		X100		Take instruction from 100.
		X101KE6	X102	Jump to 101 and store return instruction X102 in 6.
		X8186K5WR		Do 1 right shift.
½ a	ARW	X8188K5RW		Do another right shift by ARW. The drum address in D is counted up but is not active. The register address remains the same. Hence the instruction in 5 is repeated.

$2^{-2} \cdot$ a X8188K5RW	The repeating instruction as well as the repeated instruction are both shifted one place to the right.
$2^{-3} \cdot$ a ARW X8190K5RW	
$2^{-4} \cdot$ a X8190K5RW	
$2^{-5} \cdot$ a ARW X000K6RW	As the drum address overflows into the fast store address the repeating instruction becomes X8192K5RW = X000K6RW taking the next instruction from 6.
$2^{-6} \cdot$ a X000K6RW	
$2^{-7} \cdot$ a X102	As ⟨6⟩ = X102 the repetition returns to the main programme and the A accumulator is shifted over 7 places.

The instruction ARW has thus been repeated p times when the drum address of the repeating instruction is 8192—2p. This way of repeating an instruction has made it possible to do multiplication, division, block transfers, table look up and many other small basic repetitive processes in a very simple way. There is no special hardware present in the machine to do the counting necessary for the repetition, as this counting is done by the normal address counter.

As a last example we shall give a programme for the summation of a block of locations from 200 to 300 in the store. This involves 101 locations. The programme reads:

100 A101BC 101 A200Q	Put A200Q in B (B has address 3).
102 X103KE4C	Put return jump X104 in 4. Clear A in advance.
103 X7990K3W	Repeat A200Q 101 times. Because A200Q is standing in B the Q augments the instruction itself at every repetition. Hence successively ⟨200⟩, ⟨201⟩ etc. are added to A. At the end the sum is left in A and the programme proceeds at 104.
104 etc.	

It is left to the reader to work out the action diagram.

This example is not programmed for minimum waiting, but by supplying the repeating instruction X7990K3W with a Q it will step up the repeated instruction A200Q by 2 every time. Now, once the first instruction has been located, all even locations following are emerging from the drum just at the right time. The odd numbered locations must be summed in a second, similar repetition.

References

VandW59; VandW52, 56; WilkM53*a*.