The Architecture of Konrad Zuse's Early Computing Machines

Raúl Rojas*

September 29, 1997

Abstract

This paper provides the *first* detailed description of the architecture of the computing machines Z1 and Z3 designed by Konrad Zuse in Berlin between 1936 to 1941. The necessary information was obtained from a careful evaluation of the patent application filed by Zuse in 1941. Additional insight was gained from a software simulation of the machine's logic. The Z1 was built using purely mechanical components, the Z3 using electromechanical relays. However, both machines shared a common logical structure and the programming model was exactly the same. We argue that both the Z1 and the Z3 possessed features akin to those of modern computers: memory and processor were separate units, the processor could handle floating-point numbers and compute the four basic arithmetical operations as well as the square root of a number. The program was stored on punched tape and was read sequentially. In the last section of this paper we bring the architecture of the Z1 and Z3 into historical perspective by offering a comparison with computing machines built in other countries.

1 Early computing machines

Konrad Zuse is popularly recognized in Germany as the "father of the computer" and his Z1, a programmable automaton built from 1936 to 1938, has been called the "first computer" in the world. Other nations reserve this privilege for one of their own scientists and there has been a long and often acrimonious debate on the issue of the "true" inventor of the computer. Sometimes the discussion is preempted by specifying in full detail the technological features of a specific machine. The ENIAC, for example, has been called the first *large-scale general-purpose electronic* computer in the world [Burks, Burks 81]. The ENIAC (acronym for *Electronic Numerical Integrator and Computer*) was built at the Moore School of Electrical Engineering of the University of Pennsylvania from May 1943 to 1945. It solved its first problem in December 1945 and was officially presented in February 1946. Another contender for the title of first computer is the Mark I built by Howard Aiken at Harvard University between 1939 to 1944. The Mark I was an electromechanical machine, a kind of hybrid between the totally mechanical nature of previous computing devices and the electronics available at the time [Aiken, Hopper 46]. The machine built by John Atanasoff (later called the ABC) at Iowa State College from 1938 to 1942 used vacuum tubes, but was restricted to the addition and subtraction of vectors and had an structure inappropriate for universal computation [Burks, Burks 88]. In direct contrast to these three machines, the Z1 was far more flexible and was designed to execute a long and modifiable

*Professor of Computer Science at the Martin Luther University in Halle.

sequence of instructions contained on a punched tape. Zuse's machines were not purely electronic and were of reduced size. Since the Z1 was completed prior to the Mark I, it has been called the first *programmable* calculating machine in the world. Of course the old debate will not be closed with this paper, but we want to show here just how advanced the machines built by Zuse were when considered from the viewpoint of modern computer architecture and compared with other early designs of the time.

The university student Konrad Zuse started thinking about computing machines in the 1930s. He realized that he could construct an automaton capable of executing a sequence of arithmetical operations like those needed to compute mathematical tables. Coming from a civil engineering background, he had no formal training in electronics and was not acquainted with the technology used in conventional mechanical calculators. This nominal deficit worked to his advantage, however, because he had to rethink the whole problem of arithmetic computation and thus hit on new and original solutions.

Zuse decided to build his first experimental calculating machine exploiting two main ideas: a) the machine would work with binary numbers; b) the computing and control unit would be separated from the storage. Years before John von Neumann explained the advantages of a computer architecture in which the processor is separated from the memory, Zuse had already arrived at the same conclusions. In 1936 the memory[1] of the planned machine was completed. It was a mechanical device but not of the usual type. Instead of using gears (as done by Babbage in the previous century), Zuse implemented logical and arithmetical operations using sliding metallic rods. The rods could move only in one of two directions (forward or backward) and were therefore appropriate for a binary machine [Zuse 70]. The processor of the Z1 was completed a few months after the storage unit, using the same kind of technology. It worked in concert with the memory but was never very reliable. The main problem was the precise synchronization that was needed in order to avoid applying excessive mechanical stress on the moving parts. It is interesting to point out that in the same year as the memory of the Z1 was completed, Alan Turing wrote his ground-breaking paper on computable numbers in which he formalized the intuitive concept of computability.

The Z1, although unreliable, showed that the architectural design was sound and compelled Zuse to start investigating other kinds of technology. Following the advice of his friend Helmut Schreyer, he considered using vacuum tubes, but gave up the idea in favor of electromechanical relays which were easier to obtain before and during the war. An "intermediate" simpler model (the Z2) was built using a hybrid approach (a processor built out of relays and a mechanical memory). Immediately afterwards Zuse started building the Z3, a machine consisting purely of relays but with the same logical structure as the Z1. It was ready and operational in 1941, four years before the ENIAC.

This paper offers the first detailed discussion of the architecture of the Z1 and Z3. The Z1 was reconstructed by Zuse himself in Berlin during the 1980s, and is now one of the exhibition attractions at the Berlin Museum of Transportation and Technology. However, the information available describes only the design of the mechanical memory [Schweier, Saupe 88]. The Z3 was documented by Zuse in his patent application Z-391 of 1941 which is rather difficult to decipher due to the non-standard notation and terminology [Zuse 41]. Czauderna's book about the Z3 is a good source to understand the historical environment surrounding Zuse's inventions, but does not describe

[1] Zuse called it the "Speicherwerk" (storage mechanism). The term "Speicher" is still used in German instead of the antropomorphic term "memory" introduced by John von Neumann.

the Z3 in detail [Czauderna 79]. In what follows, since Z1 and Z3 were practically equivalent from the logical point of view, we refer only to the Z3. The main architectural difference between the Z1 and Z3 was the fact that the square root operation was left out of the Z1. There were also minor differences in the number of bits used for arithmetical operations in the processor (the Z1 used one bit less for the mantissa of floating-point numbers) and the number of cycles needed for each instruction. With this minor caveat and taking only the architectural features into account we can speak of Z1 and Z3 as virtually the same machine.

2 Architectural overview of the Z1 and Z3

This section summarizes the most relevant architectural features of the Z3. The paper goes from the simple to the complex: first we provide an overview of the architecture and in Section 3 we go into more detail. In order to avoid awkward sentences, we refer to the Z3 in the present tense.

Block structure

The Z3 is a floating-point machine. Whereas other early computing automata like the Mark I, the ABC and the ENIAC worked with fixed-point numbers, Zuse decided very early on to adopt what he called the "semi-logarithmic" notation, which corresponds to the modern floating-point representation.

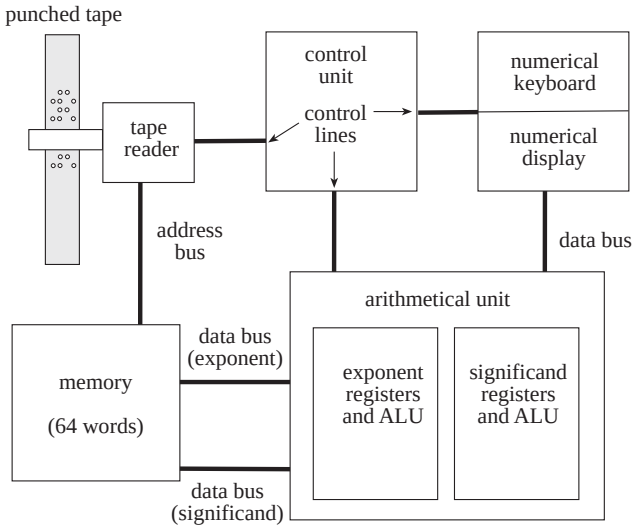


Figure 1: The building blocks of the Z3

Figure 1 is an overview of the main building blocks of the Z3. The first relevant feature is the separation between processor and memory. The Z3 consists of a binary memory unit (capable of storing 64 floating-point numbers), a binary floating-point processor, a control unit and input-output devices. Memory and arithmetical unit are connected through a data bus, which transmits the exponent and significand of the floating-point representation. The control unit contains the microsequencers needed for each instruction. Control lines going from the control unit to the processor, the memory, and the I/O devices enforce the correct synchronization of all units. The

tape reader provides the opcode of each instruction as well as the address for memory accesses. The I/O devices are connected through a data bus to the computing unit.

Floating-point representation

Figure 2 shows the representation used in the memory of the Z3. The first bit is used to store the sign of the number, the following 7 bits for the exponent, and the last 14 bits for the significand (only the 14 places to the right of the decimal point). The bits of the exponent are called part "A" of the number and are denoted by $a_6, \ldots, a_0$. The bits of the significand are called part "B" of the number and are denoted by $b_0, b_{-1}, \ldots, b_{-14}$. The exponent is coded as a two's complement number. The range of possible values runs therefore from -64 to 63. The significand is stored in *normalized* form [2], that is, the first digit before the decimal point (b_0) must always be a 1. This digit does not need to be stored (and therefore does not appear in Figure 2) so that the effective range of the numbers in the memory unit is equivalent to a significand of 15 bits. However, there is a problem with the number zero, which cannot be expressed using a normalized significand. The Z3 uses the convention that any significand with exponent -64 is to be considered equal to zero. Any number with exponent 63 is considered infinitely large. Operations involving zero and infinity are treated as exceptions and special hardware monitors the numbers loaded in the processor in order to set the exception flags (see Section 4). With this convention

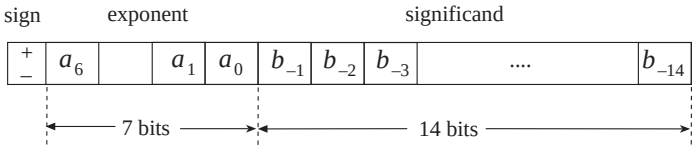


Figure 2: The floating-point representation in memory

the smallest number representable in the memory of the Z3 is $2^{-63} = 1.08 \times 10^{-19}$ and the largest is $2^{62} = 4.61 \times 10^{18}$. The arguments for computations can be entered as decimal numbers on the keyboard of the Z3 (four digits). The exponent of the decimal representation is entered by pushing the appropriate one in a row of buttons labeled $-8, -7, \ldots, 7, 8$. The original Z3 could only accept input between 1×10^{-8} and 9999×10^{8}. The reconstruction of the Z3 built by Zuse for the Deutsches Museum in Munich provides enough buttons for larger exponents. With this arrangement the whole numerical capacity of the machine can be reflected on the acceptable input. The same can be said of the output. However, the Z3 does not print the numerical results produced by the program. A single number is displayed on an array of lamps representing the digits from 0 to 9. The largest number that can be displayed is 19999. The smallest is 00001. The largest exponent that can be displayed is +8, the smallest -8.

Instruction set

The program for the Z3 is stored in punched tape. One instruction is coded using 8 bits for each row of the tape. The instruction set of the Z3 consists of the 9 instructions shown in Table 1. There are three types of instructions: I/O, memory, and arithmetical operators. The opcode has a variable length of 2 or 5 bits. Memory

[2] Donald Knuth attributes the invention of normalized floating-point numbers to Zuse [Knuth 81].

operations encode the address of a word in the lower six bits, that is, the addressing space has a maximum size of 64 words, as we mentioned before.

Table 1: Instruction set and opcodes of the Z3

Type	Instruction	Description	Opcode
I/O	Lu	read keyboard	01 110000
	Ld	display result	01 111000
memory	Pr z	load address z	11 $z_6z_5z_4z_3z_2z_1$
	Ps z	store address z	10 $z_6z_5z_4z_3z_2z_1$
arithmetic	Lm	multiplication	01 001000
	Li	division	01 010000
	Lw	square root	01 011000
	Ls_1	addition	01 100000
	Ls_2	subtraction	01 101000

The instructions on the punched tape can be arranged in any order. The instructions Lu and Ld (read from keyboard, display result) halt the machine, so that the operator has enough time to input a number or write down a result. The machine is then restarted and continues processing the program.

The instruction most conspicuously absent from the instruction set of the Z3 is conditional branching. Loops can be implemented by the simple expedient of bringing together the two ends of the punched tape, but there is no way of implementing conditional sequences of instructions. The Z3 is therefore no universal computer in the sense of Turing.

Number of cycles

The Z3 is a clocked machine. Each cycle is divided into five "stages" called I, II, III, IV, and V. The instruction in the punched tape is decoded in stage I of a cycle. The two basic arithmetical operations of the machine are addition and subtraction of exponents and significands. The operations can be executed in the first three stages of each cycle. Stages IV and V are used to prepare arguments for the next operation or to write back results.

The instructions implemented in the Z3 require the following number of cycles:

Multiplication:	16 cycles
Division:	18 cycles
Square root:	20 cycles
Addition:	3 cycles
Subtraction:	4 or 5 cycles, depending on the result
Read keyboard:	9 to 41 cycles, depending on the exponent
Display output:	9 to 41 cycles, depending on the exponent
Load from memory:	1 cycle
Store to memory:	0 or 1 cycle

According to Zuse, the time required for a multiplication was 3 seconds. Considering that a multiplication operation needs 16 cycles, we can estimate that the operating frequency of the Z3 was $16/3 \approx 5.33$ Hz [3].

The number of cycles needed for the *read* and *display* instructions is variable, because it depends on the exponent of the arguments. Since the input has to be converted from decimal to binary representation, the number of multiplications needed with the factor 10 or 0.1 is dictated by the decimal exponent (see Section 4).

Addition and subtraction require more than one cycle because in the case of floating-point numbers, care has to be taken to set the size of the exponent of both arguments to the same value. This requires some extra comparisons and shifting.

A number can be stored in memory in *zero* cycles, when the result of the last arithmetical operation can be redirected to the desired memory address. In this case the cycle needed for the store instruction overlaps with the last cycle of the arithmetical operation.

Programming model

It is very important to describe the programming model of the Z3, that is, the part of the machine visible for the programmer. From the point of view of the software, the Z3 consists of 64 memory words that can be loaded into two floating-point registers, which we simply call R1 and R2. These two registers contain the two arguments of arithmetical operations requiring them. The programmer can write any sequence of instructions, but has to keep in mind the state of the machine's registers.

The important point to remember is the following: the first load operation in a program (Pr z) transfers the contents of address z to R1. Any other subsequent load operation transfers a word from memory to R2. A read keyboard instruction loads the numerical input into register R1 and *destroys* register R2.

Arithmetical operations do not specify their arguments in the opcode. Their implicit semantics is the following:

Multiplication:	R1:=R1×R2
Division:	R1:=R1/R2
Addition:	R1:=R1+R2
Subtraction:	R1:=R1−R2
Square root:	R1:=sqrt(R1)

R2 is set to zero after an arithmetical instruction whereas the result is stored in R1. Subsequent load operations refer to R2. The store and display instructions refer always to register R1, which also contains the result of the previous arithmetical operation. After a store or a display operation R1 is set to zero. The next load operation refers then to R1.

An example is better than many additional remarks to clarify the programming model of the Z3. Assume that we want to compute a polynomial using Horn's method:

$$x(a_2 + x(a_3 + xa_4))) + a_1.$$

Assume further that we have stored the constants a_4, a_3, a_2, a_1 in the addresses 4, 3, 2, and 1 of the memory unit. The value x is stored in address 5. The program that performs the desired computation is the following:

[3] It is a curious fact of life that the gate-level simulation of the Z3 implemented by my students using a personal computer also required around 3 seconds for a multiplication!

Pr 4	load a_4 in R1
Pr 5	load x in R2
Lm	multiply R1 and R2, result in R1
Pr 3	load a_3 in R2
Ls_1	add R1 and R2, result in R1
Pr 5	load x in R2
Lm	multiply R1 and R2, result in R1
Pr 2	load a_2 in R2
Ls_1	add R1 and R2, result in R1
Pr 5	load x in R2
Lm	multiply R1 and R2, result in R1
Pr 1	load a_1 in R2
Ls_1	add R1 and R2, result in R1
Ld	display result

After the last instruction has been executed the processor is reset to its initial state. A new program sequence can be started.

3 Block diagram of the Z3

In this section we take a closer look at the structure of the Z3 and describe its main building blocks in more detail. The main issue is how to enforce the correct synchronization of the available components.

The processor

Figure 3 shows a simplified representation of the arithmetical unit of the Z3. There are two parts: the left side is used for operations with the exponents of the floating-point numbers, the right side for operations with the significands. Af and Bf are registers used to store the exponent and significand of what, from the programmers point of view, is register R1. We will refer to R1 as the register pair <Af,Bf>. The register pair <Ab,Bb> stores the exponent and significand of R2. The pair <Aa,Ba> contains the exponent and the significand of a third temporal floating-point register invisible to the programmer. The two ALUs A and B are used to respectively add or subtract exponents and significands. The result of the operation in the exponent part is put into Ae. In the significand part, the result of the operation is put into Be. In part B a multiplexer allows selection of Ba or the output of the ALU as the result of the operation. The multiplexer is controlled by a relay Bt (if Bt=0 then Be is set equal to Ba).

The small boxes labeled Ea, Eb, Ec, Ed, Ef, Fa, Fb, Fc, Fd, Ff are switches that open or close the data bus. If the contents of register Af are to be transferred to Aa, for example, the box of relays Ea is set to 1 and the result is Aa:=Af. As can be seen from the diagram, the contents of Af can be transferred to Aa or Ab, whereas the contents of Ae can be transferred to any of Aa, Ab or Af, according to the states of the switches. The structure of part B of the arithmetical unit is very similar, but in addition to the multiplexer controlled by the relay Bt, there is also a shifter between Bf and Ba, and a shifter between Bf and Bb. The first shifter can displace the significand up to two positions to the right and one position to the left. This amounts to a division of Bf by 4 or a multiplication with the constant 2. The second shifter can displace the significand in Af from 1 to 16 positions to the left and from 1 to 15 positions to the right. These shifts are needed for addition and subtraction

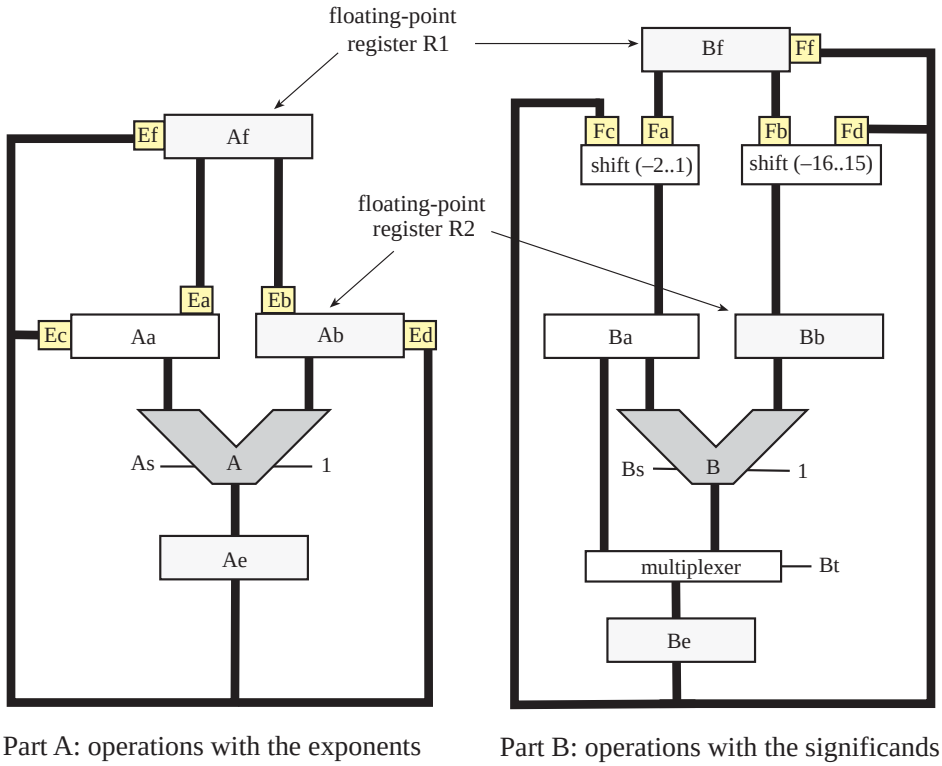


Figure 3: The registers and datapath

of floating-point numbers. Multiplication and division with powers of 2 can therefore be performed when the operands for the next arithmetical operation are fetched and, in this sense, do not consume time.

The number of bits used in the registers are the following:

Af	7 bits	Bf	17 bits
Aa	7	Ba	19
Ab	7	Bb	18
Ae	8	Be	18

As can be seen from this list, Ae uses one extra bit to handle the addition of the exponents of the arguments. Part B of the processor uses two extra bits for the significands (b_{-15}, b_{-16}) and makes explicit b_0, which is not stored in memory. The extra bits at positions -15 and -16 are included to increase the precision of the computations. The total number of bits needed to store the result of an arithmetical operation in Bf is therefore 17 bits. Registers Ba and Bb require more extra bits (ba_2, ba_1, and bb_1) to handle intermediate results of some of the numerical algorithms. In particular the square root algorithm can lead to partial computations in Ba requiring three bits to the left of the decimal point.

The basic primitive operation of the datapath is the addition or subtraction of exponents or significands. When the relays As or Bs are set, the negation of the

second argument (Ab or Bb) is fed into the ALU. Therefore if the relay As is set to 1, the ALU in part A subtracts its arguments, otherwise they are added. The same is true for part B and the relay Bs. The constant 1 is needed to build the two's complement of a number.

Assume that two numbers with the same exponent are to be added. The first exponent is stored in Af, the second in Ab. Since they are equal, no operation has to be performed on this side of the machine. In part B, the significand of the first number is stored in Bf and the significand of the second in Bb. The first step consists of loading Ba with Bf by setting the relay box Fa to 1. The addition is performed next, the relay Bt is set to zero and so the result Ba+Bb is assigned to Be. The relay box Ff is now set to 1 and the result is stored in Bf. As we see the information can move between registers and so flow through the datapath. The computer architect has to provide the correct sequence of activations of the relay boxes in order to get the desired operation. This is done in the Z3 using a technique very similar to microprogramming.

The control unit

Figure 4 shows a more detailed diagram of the control unit and of the I/O panels. The circuit Pb decodes the opcode of the instruction read from the punched tape. If it is a memory instruction, circuit Pb sets the address bus to the value of the lower six bits of the opcode. The control unit determines the correct microsequencing of the instructions. There are special circuits for each of the operations in the instruction set.

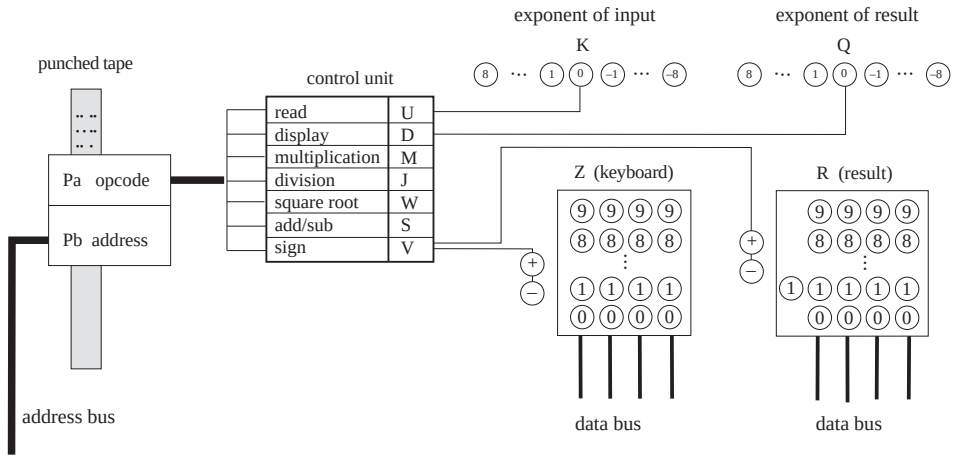


Figure 4: The control unit and I/O panels

Circuit Z represents the panel of buttons used to enter a decimal number in the machine. Only one button in each of the four columns can be activated. The exponent is set by pressing one of the buttons labeled -8 to 8 in circuit K. The output display is very similar to the input panel, but here lamps illuminate the appropriate decimal digits, the exponent of the number (circuit Q), as well as its sign. Note that there is a fifth digit for the output (which can only be 1 or 0).

Once a decimal number has been set, a data bus transmits the digits to register Ba and a complex series of operations is started. The decimal input must be transformed into a binary number. This requires a chain of multiplications, which is longer according to the absolute magnitude of the exponent. If the exponent is zero, the whole

transformation requires 9 cycles, but if it is 8, the operation requires $9 + 4 \times 8 = 41$ cycles.

Microcontrol of the Z3

The heart of the control unit are its microsequencers. Before we describe the way they work it is necessary to take a closer look at the chaining of arithmetical instructions in the Z3. Figure 5 shows the main idea. Each cycle of the Z3 is divided into five stages. Stages IV and V are used to move information around in the machine. During stages I, II and III an addition/subtraction is computed in part A and another in part B of the Z3. We call this the "execute" phase of an instruction. A typical instruction fetches its arguments, executes and writes back the result. Zuse took great care to save execution time by overlapping the fetch stage of the next instruction with the write-back stage of the current one. We can think of an execution cycle as consisting of just two stages, as shown in Figure 5 where the first two cycles of a series of instructions have been labeled. We have adopted this convention in the tabular diagrams of the numerical algorithms discussed later on.

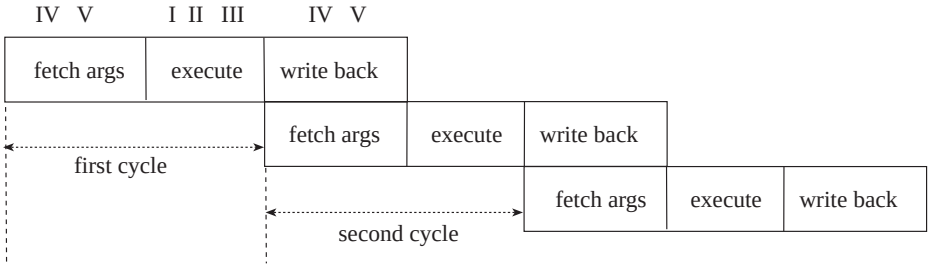


Figure 5: The execution pipeline of the Z3

The microsequencing is done by special control wheels. There is one for the multiplication algorithm, another to control division and another for the square root instruction. The moving arm shown in Figure 6 starts moving clockwise as soon as the control unit decodes the corresponding instruction. In each cycle the arm moves from one position to the next. The arm conduces electricity and activates the circuits with which it comes into contact. In the example shown in the figure, the moving arm sets the relay box Ea to 1 in the first cycle. This leads to the transfer of the contents of register Af into Aa. In the next cycle the relay boxes Ec and Fc are activated. In this way the results of the operations in parts A and B are written back into the registers Aa and Ba respectively. As one can see, such control wheels provide a comfortable platform for modifying the exact sequence of events during an operation. They correspond to the microsequencers used today in modern microprocessors. I stop short of calling them a form of microprogramming, because in this case the microsequence has been hardwired, but it is obvious that microsequencing and microprogramming are closely related.

Extensive use of microsequencing allowed Zuse to simplify the Z3. Once the basic circuits had been laid out, it was just a matter of refining the control until optimal sequences of events could be found. There are a lot of details that must be kept in mind by the engineer designing the "microprogram", otherwise short circuits can destroy the hardware. The Z1 with its mechanical design was still more sensitive in this respect than the Z3. Even after it was completed, there were sequences of instructions that the programmer had to avoid in order not to damage the hardware.

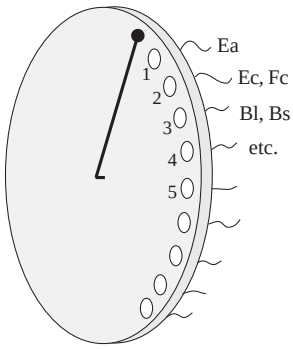


Figure 6: Control wheels for microsequencing

One of those sequences was inadvertently tried at the Berlin Museum of Technology and Transportation which led to slight damaging of the reconstructed Z1 in 1994.

The adders

An important feature of the Z3 is the design of the adders, which compute additions and subtractions using a method called *carry look-ahead*. If binary addition is implemented in a straightforward way, carries have to be passed from one bit position to the next. In the case of the significand we would need 16 cycles just for the transmission of the carry bits.

The adders designed by Zuse are much faster than that – they perform an addition or subtraction in stages I, II, and III of a single cycle. Subtraction is computed by complementing the second argument and adding an extra 1 at the lowest bit position.

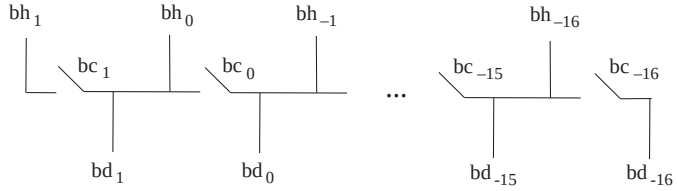


Figure 7: Circuit for carry look-ahead

Consider addition of the registers Ba and Bf. First of all a partial result Bc is computed which is the bitwise XOR of both registers, i.e., $bc_i = ba_i$ XOR bb_i. We will refer to bit i-th of register Bb by bb_i or Bb[i], whatever form seems more convenient. The same notation will be used in the case of other registers. A second partial result is the bitwise AND operation applied to both registers, i.e., $bh_i = ba_i$ AND bb_i. This last operation locates the bit positions at which a carry is needed. The intermediate result Bd is computed by using the circuit shown in Figure 7. The input to the circuit consists of the bits $bh_1, \ldots, bh_{-16}$ computed previously. When a bit is 1, the corresponding line carries a current, otherwise the line is disconnected from the power source (three-state). The resting position of the relays $bc_1, \ldots, bc_{-16}$ is the one shown in the figure. If bit bc_i is equal to 1 the corresponding relay is closed. The final result is $be_i = bd_i$ XOR bc_i. Note that the use of relays makes the propagation of the carries up to the last bit position needed easier. Since all relays are activated simultaneously, the carry is not delayed in going from one bit position to the next.

4 Numerical algorithms

In this section we describe the floating-point algorithms used by the Z3. They are, without exception, the same as those normally used in small sequential floating-point processors [Koren 93].

Floating-point exceptions

The problem with floating-point notation is that special conventions have to be used to deal with the number zero. The Z3 solves this problem and deals with other exceptions (overflow, underflow) by monitoring the value of the exponent after any arithmetical operation or a load from memory. A special circuit looks at the state of the bus Ae and captures exceptions. Any number with exponent -64 is flagged as zero: a relay denoted Nn_1 is set to 1 if the number is stored in the register pair <Af,Bf>. If the number is stored in the register pair <Ab,Bb>, the relay Nn_2 is set to 1. In this way we always know if one or both of the arguments for an arithmetical operation are zero. Something similar is done for any exponent of value 63 (an infinite number, according to the convention). In this case the relays Ni_1 or Ni_2 are set to 1 according to the register pair in which the number is stored.

Operations involving "exceptional" numbers (zero or infinity) are performed as usual, but the result is overridden by the snooping circuit. Assume for example that a multiplication is computed and that the first argument is zero (Nn_1 is set to 1). The computation proceeds as usual, but in each cycle the snooping circuit produces the result -64 at the output of the adder of part A. It does not matter what operations are performed with the significands because the exponent of the result is set to -64 and therefore the final result is zero. Division by an infinite number can be handled in a similar manner. The Z3 can detect undefined operations like $0/0$, $\infty - \infty$, ∞/∞ and $0 \times \infty$. In all these cases the corresponding exception lamp lights on the output panel and the machine is stopped. The Z3 always produces the correct result when one of the arguments is zero or ∞ and the other a number within bounds[4].

An additional circuit looks at the exponent of the result at the output of the exponent's adder. If the exponent is greater than or equal to 63, overflow has occurred and the result must be set to ∞. If the exponent is lower than -64, underflow has occurred and the result must be set to 0. To do this, the appropriate relay (Nn_1 or Ni_1) is set to 1.

Zuse managed to implement exception handling using just a few relays. This feature of the Z3 is one of the most elegant in the whole design. Many of the early microprocessors of the 1970s did not include exception handling and left it to the software. Zuse's approach is sounder, since it frees the programmer from the tedium of checking the bounds of his numbers before each operation.

[4]This was not the case for the Z1. Zuse thought of, but did not implement, exception handling in the Z1. The machine could not correctly perform computations involving zero [Zuse, personal communication].

Addition and subtraction

In order to add or subtract two floating-point numbers x and y, their representation must be reduced to the same exponent. After this has been done only the significands have to be added or subtracted. If the exponents are different, the significand of the smaller number is shifted to the right as many places as necessary (and its exponent is incremented correspondingly to keep the number unchanged) until both exponents are equal. It can of course happen that the smaller number becomes zero after 17 shifts to the right.

The signs of the two numbers are compared before deciding on the type of operation to be executed. If an addition has been requested and the signs are the same, the addition is performed. If the signs are different a subtraction is executed. If a subtraction has been requested and the signs are different, an addition is executed. If the signs are the same, the subtraction is executed. A special circuit sets the sign of the result according to the signs of the arguments and the sign of the result.

Addition and subtraction are controlled by a chain of relays (not by a control wheel) since the maximum number of cycles needed is low. Figure 8 shows the synchronization required for the addition of two numbers. Initially, the arguments for the addition are stored in the register pairs <Af,Bf> and <Ab,Bb>. In the first cycle the exponents are subtracted. In cycle 2, the significand with the larger exponent is loaded into register Ba and the significand with the smaller exponent into register Bb. The significand in register Bb is shifted as many places to the right as the absolute value of the difference of the exponents (exception handling takes care of the case when the smaller number becomes zero after the shift). In stages I, II and III of cycle 2 the significands are added and finally the processor tests if the result is greater than 2. If this is the case, the significand is shifted one position to the right and the exponent is incremented by 1. Note that the test "if (Be$\geq$2)" in part A of the arithmetical unit is done *after* Be has already been computed in part B during stages I, II, and III of cycle 2.

In the case of a subtraction four or five cycles are needed. Figure 9 shows the synchronization required for a subtraction. The first two cycles are almost identical to the first two cycles of the addition algorithm, but now the significands are subtracted. Cycle 3 is executed only when the difference of the significands is negative. The effect of cycle 3 is just to make the significand of the result positive. Cycle 4 is very important: the difference of two normalized significands can have many zeros in the first bit positions to the left. The result is normalized by shifting Be to the left as many places as necessary (this is done with the shifter between the relay box Fd and register Bb). The number of one-bit shifts is subtracted from the exponent in part A of the processor. In cycle 5 the result is stored in the register pair <Af,Bf>.

cycle	stage	exponent	significand
0	I,II,III		
1	IV,V	Aa:=Af	
	I,II,III	Ae:=Aa−Ab	Be:=0+Bb
2	IV,V	if (Ae≥0) then Ab:=0, Aa:=Af else Aa:=0	if (Ae≥0) then Ba:=Bf, Bb:=Be (shifted) else Ba:=Be, Bb:=Bf (shifted) (Be or Bf are shifted |Ae| places to the right)
	I,II,III	if (Be≥2) then Ae:=Aa+Ab+1 else Ae:=Aa+Ab	Be:=Ba+Bb
3	IV,V	Af:=Ae	if (Be≥2) then Bf:=Be/2 else Bf:=Be

Figure 8: The 3 cycles needed for the addition algorithm. The arguments for the addition are stored in the register pairs <Af,Bf> and <Ab,Bb> before the operation is started.

cycle	stage	exponent	significand
0	I,II,III		
1	IV,V	Aa:=Af	
	I,II,III	Ae:=Aa−Ab	Be:=0+Bb
2	IV,V	if (Ae≥0) then Ab:=0, Aa:=Af else Aa:=0	if (Ae≥0) then Ba:=Af, Bb:=Be (shifted) else Ba:=Be, Bb:=Bf (shifted) (Be or Bf are shifted |Ae| places to the right)
	I,II,III	Ae:=Aa+Ab	Be:=Ba−Bb
3	IV,V	Aa:=Ae, Ab:=0	Ba:=0, Bb:=Be
	I,II,III	Ae:=Aa+Ab	Be:=Ba−Bb
4	IV,V	Aa:=Ae Ab:= number of shift positions	Bb:=Be (shifted) (Be is normalized by shifting to the left)
	I,II,III	Ae:=Aa−Ab	Be:=0+Bb
5	IV,V	Af:=Ae	Bf:=Be

Figure 9: The 4–5 cycles needed for the subtraction algorithm. The first argument is stored in the register pair <Af,Bf> and the second in <Ab,Bb> before the operation is started.

Multiplication

The multiplication algorithm of the Z3 is like the one used for decimal multiplication by hand, that is, it is based on repeated additions of the multiplicator according to the individual digits of the multiplicand. At the begining of the algorithm the first argument is stored in the register pair <Af,Bf>. The second argument is stored in the register pair <Ab,Bb>. The temporal register pair <Aa,Ba> is set to zeroes. Figure 10 shows the microsequencing produced by the multiplication wheel of the control unit. The algorithm takes 16 cycles to run. Note that only the bits of the multiplicand from position -14 to position 0 are used. The exponents are added in the first cycle and the result just loops afterwards in part A of the arithmetical unit. The significands are handled in part B of the unit. Register Ba contains the partial result of the computation. The basic multiplication loop has the following form:

Ba:=Be/2
Be:=Ba + Bb×(i-th bit of Bf)

for $i = -14, \ldots, 0$. The partial result Be is shifted one position to the right to produce Ba:=Be/2. This is done with the shifter connected to the relay box Fc.

The result of the multiplication is a number $1 \leq r < 4$ (for arguments within bounds). In the last cycle thre is a check to see if $r \geq 2$. If this is the case the result is shifted one position to the left and a 1 as added to the exponent of the result.

Division

The division algorithm is similar to the multiplication algorithm, but subtraction is used repetitively instead of addition. At the begining of the algorithm the dividend is stored in the register pair <Af,Bf>. The divisor is stored in the register pair <Ab,Bb>. The temporal register pair <Aa,Ba> is set to zeroes. Figure 11 shows the microsequencing produced by the division wheel of the control unit. The algorithm takes 18 cycles to run.

The main idea of the algorithm is very simple. The exponent of the result is obtained by subtracting the exponents of dividend and divisor. Now for the significand: assume that we want to compute x/y for the significands x and y. Since we are dealing with normalized numbers, the first digit of the result is 1 if $x \geq y$ and zero if $x < y$. In the first case, we set the first digit of the result to 1 and compute the remainder, which is $x - y$. The remainder is divided recursively by y. To do this, it is shifted one position to the left and the new result bit is stored at position $[-1]$ of register Bf (in this way nullifying the effect of the shift). If the result bit is zero, the remainder is just x and the recursive division is continued as in the first case.

The basic division loop has the following form:

Ba:=2×Be
if (Ba−Bb ≥ 0) then Be:=Ba−Bb, Bf[i]:=1
else Be:=Ba Bf[i]:=0

for $i = 0, \ldots, -14$. The partial result Be is shifted one position to the left to produce Ba:=2×Be. This is done with the shifter connected to the relay box Fc.

The result of the division of significands is a number $1/2 < r < 2$. This condition is tested in cycles 17 and 18. If $r < 1$, a 1 is subtracted from the exponent and the result is shifted one position to the left in order to get a normalized number.

Square root extraction

The square root algorithm is the jewel in the crown of the Z3. Figure 12 shows the microsequencing required during the 20 cycles needed to compute the square root of a number. The argument for the operation is stored in the register pair <Af,Bf>. The register pair <Aa,Ba> is initialized to zeroes. The algorithm computes the square root of numbers with an even exponent. If the exponent is an odd number, the significand is shifted one place to the left and the exponent is decremented by one. The final exponent (computed in cycle 19) is half this initial exponent.

The main idea of the algorithm is to reduce the square root operation to a division. If we want to compute the square root of x, we need a number Q such that $x/Q = Q$. The result Q is built sequentially by setting the i-th bit to 1 and then testing whether the condition $x > Q^2$ still holds. If this is not the case the i-th bit must be set to 0.

Assume that we have already computed from bit 0 to bit $-i+1$ of the final result. Denote by Q_{-i+1} the significand

$$Q_{-i+1} = \mathrm{Bf}[0] \times 2^0 + \mathrm{Bf}[-1] \times 2^{-1} + \cdots + \mathrm{Bf}[-i+1]2^{-i+1}.$$

Bit $-i$ is then set to to q_{-i} and it must hold that

$$x \geq Q_{-i}^2 = (Q_{-i+1} + q_{-i}2^{-i})^2$$

This is true if

$$(x - Q_{-i+1}^2) - 2^{-i}q_{-i}(2Q_{-i+1} + 2^{-i}q_{-i}) \geq 0$$

Define t_{-i} using the expression

$$2^{-i}t_{-i} = (x - Q_{-i+1}^2)2^{-1}2^1 - 2^{-i}q_{-i}(2Q_{-i+1} + 2^{-i}q_{-i})$$

This can be written as

$$2^{-i}t_{-i} = t_{-i+1}2^{-i+1}2^{-1}2^1 - 2^{-i}q_{-i}(2Q_{-i+1} + 2^{-i}q_{-i})$$

where we have used the recursive definition $2^{-i+1}t_{-i+1} = (x - Q_{-i+1})^2$. Simplifying the last expression we finally get:

$$t_{-i} = 2t_{-i+1} - q_{-i}(2Q_{-i+1} + 2^{-i}q_{-i})$$

If t_{-i} is positive for $q_{-i} = 1$, we set bit $-i$ of the final result to 1, i.e., Bf[$-i$]:=1. If t_{-i} is negative, we set Bf[$-i$]:=0. The recursive computation is started with $t_0 = x$. Q_{-i+1} represents at each step the partial result contained in register Bf. Bit $-i$ is tentatively set and the sign of t_{-i} is tested.

The basic loop of the square root algorithm for bit $-i$ has the following form:

```
Ba:=2×Be
Bb:=2×Bf
Bb[−i]:=1
if (Ba−Bb ≥ 0) then Be:=Ba−Bb, Bf[−i]:=1
                else Be:=Ba,       Bf[−i]:=0
```

All bits of register Bf are used for the computation of the square root. If the original number lies within bounds, the result is also within bounds.

Read and display instructions

The two most complex instructions of the Z3 are those related to the input and output of decimal numbers. A decimal number of four digits entered through the keyboard is first converted into a binary integer. This is done by reading each digit sequentially, transforming it into a binary number and storing it in the bits Ba[−10], Ba[−11], Ba[−12], and Ba[−13] of register Ba. The number in register Ba is multiplied by 10 and the procedure is repeated for the other digits. After 4 iterations the decimal input has been transformed to a binary number (the exponent is adjusted to the correct value). The difficult part is handling the exponent. If the exponent e is positive, the significand has to be multiplied e times with 10. If it is negative, it must be multiplied $|e|$ times with 0.1. Multiplying with 10 is relatively easy: the significand in Be can be shifted one bit to the left and then stored in Ba (that is Ba:=2×Be). At the same time Be can be shifted 3 places to the left and can be stored in Bb (that is Bb:=8×Be). The addition of Ba and Bb then provides the desired result: the multiplication of the original number in Be with the constant 10. The process takes 4 cycles for multiplication, that is 32 cycles for the decimal exponent +8. Since a read operation needs a minimum of 9 cycles, this means that a decimal number with exponent +8 is read in 41 cycles.

In the case of negative exponents, multiplication with the constant 0.1 is performed using the shifters and the adders as well. This multiplication is somewhat more complex, because 0.1 is a periodic number in the binary system. The description of the microsequencing used would take us too far away from the main topics, so we omit it here.

The display instruction works by multiplying or dividing iteratively by 10. If the binary exponent of the number in register R1 is positive, the number is multiplied with 0.1 as many times as needed to make the binary exponent equal to 2 and until the first left four bits of register Bf contain a number between 0 and 9 (0000 and 1001). This is the decimal digit that can be displayed in the next column of the output panel. The number is subtracted from the significand in Bf and the process continues for the following digits. If the binary exponent of the number in register R1 is negative, the process is similar, but multiplications with the constant 10 are used.

5 Complete architecture of the Z3

We are now in a position to make sense of the detailed diagram of the Z3 shown in Figure 13. We see some of the components which were discussed in the previous sections.

The control unit and the I/O panels were discussed before. Notice that the four decimal digits of the input keyboard are tranferred to register Ba using the relay boxes Za, Zb, Zc, and Zd, which are activated one after the other.

The relay boxes Eg and Ei are used to set some useful constants directly into the exponent registers (+13 and −4). The shifter Ee between register Af and register Aa is used for the square root algorithm. The exponent of the result (Aa) becomes half the exponent (Af) of the original number.

Ah_1 is a relay acting as a flip-flop. When it is set to 0, the register pair <Af,Bf> is accessed by load operations. When it is set to 1, the register pair <Ab,Bb> is accessed. This relay is reset to 0 by the control line a_i. The control lines a_l, a_j, b_l, and b_j are used to clear the registers Af, Ab, Bf, and Bb when needed.

The box labeled “zero, infinite” below Ae represents the circuits for exception handling. They snoop permanently on the data bus (results of operations and data from memory) and raise the corresponding exception flags when needed. The shifter

below Be is used to displace the significand one bit to the right. This provides the normalization needed for the significand whenever Be≥2.

Fp and Fq are the relays which control the number and direction of one-bit shifts in the shifter below the relay boxes Fc and Fa. Fh, Fi, Fk, Fl, and Fm have the same function in relation to the other shifter. Using these five bits the numbers between -16 and 15 can be represented and this is also the range of the second shifter. When such a shift is performed, the number represented by the relays Fh to Fm is transferred through the relay box Bn to register Ab in order to modify the exponent of the result. If the number is shifted 10 positions to the left then +10 is subtracted from the exponent of the result. Such drastic shifts are needed mostly after subtractions.

Look again at the diagram of the Z3. Everything makes sense now and looks as conventional as any modern small floating-point processor. It is indeed amazing how Konrad Zuse was able to find the adequate architecture right from the beginning. The Z3 processor employs just 600 relays; the memory needs three times as much. By having to optimize the design, by having to save hardware everywhere, Zuse was *forced* to think and rethink the logical structure of his machine. He was not allowed the luxury of the almost unlimited funding allocated by the US military for the development of the ENIAC or by IBM for the Mark I. He was all alone and while this may have worked to his advantage from the conceptual side, it may also have worked to his disadvantage when considering the negligible impact that the Z1 and Z3 had on the emerging American computer industry after the war [Stern 81].

6 The invention of the computer

The main defect of the Z3 was the absence of a conditional branch in the instruction set. It would not have been difficult to implement it: although it is rather clumsy to do when the program is stored on punched tape, the necessary mechanism would have required just a few additional circuits.

Sometimes the dividing line between calculating machines and universal computers is drawn by differentiating between machines with externally or internally stored programs. I have argued elsewhere [Rojas 93] that this is not a valid criterion. An external program can work as an interpreter of numerical data. The external program becomes a fixed part of the processor and the data becomes the program, much in the same way as a universal Turing machine works as an interpreter. I have argued that what is needed for universal computation is a minimal instruction set and indirect addressing [Rojas 94]. Indirect addressing can be simulated by writing self-modifying programs, so that the instruction set becomes the defining criterion. A machine with enough memory, an accumulator, and capable of executing the instructions CLR (clear), INC (increment), LOAD, STORE, and BZ (branch if zero) is a universal computer. In this sense, the Z1 *was not* a fully fledged computer, but neither were any of the other early machines. The ABC was a special purpose machine for Gauss elimination, the Harvard Mark I lacked conditional branching although it featured loops. The ENIAC was not even programmable through software: the building blocks had to be hardwired in dataflow fashion. Conditional branching was available in the ENIAC only in a limited way and self-modifying programs were of course out of the question.

Tables 2 and 3 show the most relevant information about the early computing machines mentioned in section 1. As should be clear from the tables none of them fulfills all the necessary requirements for a universal computer. We also include the Mark 1 machine built in Manchester from 1946 to 1948, because as far as we know this was the first machine to fit our definition of a universal computer. The Mark 1

Table 2: Comparison of architectural features

Machine	memory and CPU separated?	conditional branching?	soft or hard programming	self-modifying programs?	indirect addressing?
Zuse's Z1	√	×	soft	×	×
Atanasoff's	√	×	hard	×	×
H-Mark I	×	×	soft	×	×
ENIAC	×	partially	hard	×	×
M-Mark 1	√	√	soft	√	×

Table 3: Some additional architectural features

Machine	internal coding	fixed-point or floating-point?	bit-sequential arithmetic?	architecture	technology
Zuse's Z1	binary	floating	no	sequential	mechanical
Atanasoff's	binary	fixed-point	yes	vectorized	electronic
H-Mark I	decimal	fixed-point	no	parallel	electromechanical
ENIAC	decimal	fixed-point	no	dataflow	electronic
M-Mark 1	binary	fixed-point	yes	sequential	electronic

was built under the direction of F.C. Williams and T. Kilburn. This machine stored its program in random access digital memory implemented with CRT tubes. All necessary instruction primitives were available (in modified form) and although it lacked indirect addressing, self-modifying programs could be written. The first program ran in June 1948 and calculated the highest proper factor of 2^{18} [Lavington 75]. In September Alan Turing was appointed Reader in Mathematics in Manchester and wrote some programs for the first universal computer in the world. His vision of universal computation published in 1936, the same year in which the storage unit of the Z1 was completed, had at last become true. Tables 2 and 3 are emphatic: the invention of the computer was a collective achievement encompassing two continents and twelve years.

Acknowledgements

Deciphering the sketchy documentation available was possible only with the collaboration of several of my students at the Universities of Halle and Berlin. I thank Alexander Thurm and Axel Bauer who implemented a gate-level simulation of the Z3 processor. We became aware of synchronization problems when the simulation refused to run. I also thank Franz Konieczny, Reimund Spitzer, and Roland Schultes who wrote part of a stand-alone simulation of the processor in C. We started working on the Z3 with the help of Konrad Zuse, who gladly answered our questions. It was amazing to see how, after almost sixty years, the whole design of the Z3 was still in his head. Unfortunately, Konrad Zuse died in December 1995 before this description of his work was ready. This paper is dedicated to his memory.

References

[Aiken, Hopper 46] H. Aiken and G. Hopper, "The Automatic Sequence Con-

trolled Calculator", reprinted in [Randell 82], pp. 203-222.

[Burks, Burks 81] A. W. Burks and A. R. Burks, "The ENIAC: First General-Purpose Electronic Computer", *Annals of the History of Computing*, Vol. 3, N. 4, 1981, pp. 310-399.

[Burks, Burks 88] A. W. Burks and A. R. Burks, *The First Electronic Computer – The Atanasoff Story*, The University of Michigan Press, Ann Arbor, 1988.

[Czauderna 79] K.-H. Czauderna, *Konrad Zuse, der Weg zu seinem Computer Z3*, Oldenbourg Verlag, Munich, 1979.

[Knuth 81] D. Knuth, *The Art of Computer Programming – Seminumerical Algorithms*, Vol. 2, 1981.

[Koren 93] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[Lavington 75] S.H. Lavington, *A history of Manchester computers*, NCC Publications, Manchester, 1975.

[Lavington 80] *Early British Computers*, Digital Press, Manchester, 1980.

[Randell 82] B. Randell, *The Origins of Digital Computers*, Springer-Verlag, Berlin, 1982.

[Rojas 93] R. Rojas, "Who invented the computer? The debate from the viewpoint of computer architecture", in W. Gautschi (ed.), *Fifty Years Mathematics of Computation*, Proceedings of Symposia in Applied Mathematics, AMS, 1993, pp. 361–366.

[Rojas 94] R. Rojas, "On Basic Concepts of Early Computers in Relation to Contemporary Computer Architectures", *Proceedings of the 13th World Computer Congress*, Hamburg, pp. 324–331.

[Schweier, Saupe 88] U. Schweier and D. Saupe, "Funktions- und Konstruktionsprinzipien der programmgesteuerten mechanischen Rechenmaschine Z1", Arbeitspapiere der GMD 321, Bonn, 1988.

[Stern 81] N. Stern, *From ENIAC to UNIVAC*, Digital Press, Bedford, 1981.

[Zuse 41] K. Zuse, *Patentanmeldung Z-391*, German Patent Office, Berlin, 1941.

[Zuse 70] K. Zuse, *Der Computer mein Lebenswerk*, Springer-Verlag, Berlin, 1970.

cycle	stage	exponent	significand
0	I,II,III		
1	IV,V	Aa:=Af	
	I,II,III	Ae:=Aa+Ab	if (Bf[−14]=1) then Be:=Ba+Bb else Be:=Ba
2	IV,V	Aa:=Ae, Af:=0, Ab:=0	Ba:=Be/2
	I,II,III	Ae:=Aa+Ab	if (Bf[−13]=1) then Be:=Ba+Bb else Be:=Ba
3	IV,V	Aa:=Ae	Ba:=Be/2
	I,II,III	Ae:=Aa+Ab	if (Bf[−12]=1) then Be:=Ba+Bb else Be:=Ba
4	⋮	⋮	⋮
i	IV,V	Aa:=Ae	Ba:=Be/2
	I,II,III	Ae:=Aa+Ab	if (Bf[$i-15$]=1) then Be:=Ba+Bb else Be:=Ba
14	⋮	⋮	⋮
15	IV,V	Aa:=Ae	Ba:=Be/2
	I,II,III	if (Be $\geq$ 2) then Ae:=Aa+1	If (Bf[0]=1) then Be:=Ba+Bb else Be:=Ba
16	IV,V	Af:=Ae	if (Be $\geq$ 2) then Bf:=Be/2 else Bf:=Be Bb:=0

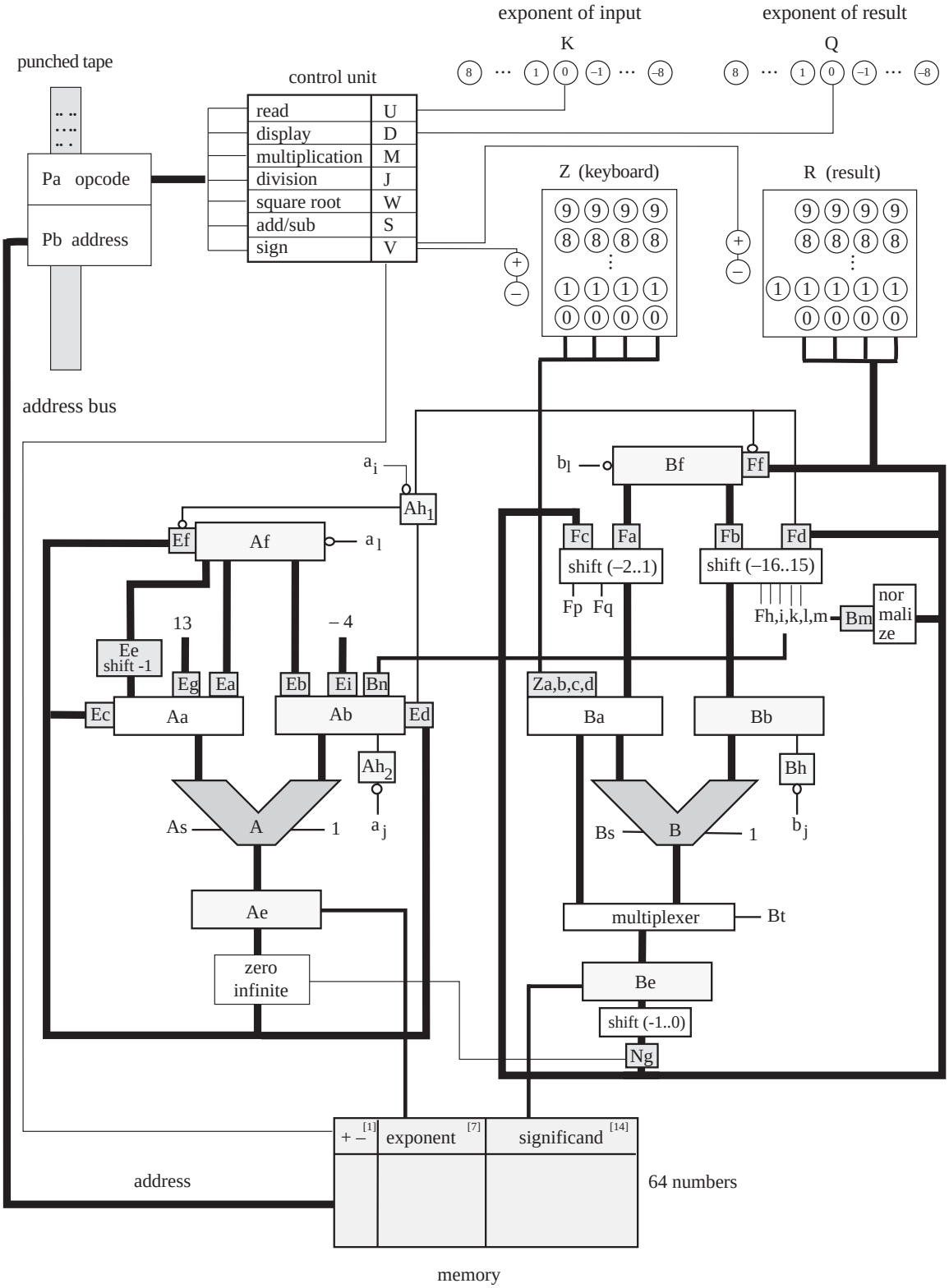
Figure 10: The 16 cycles needed for the multiplication algorithm. The i-th bit of register Bf is denoted by Bf[i]. The first argument is stored in the register pair <Af,Bf> and the second in <Ab,Bb> before the operation is started.

cycle	stage	exponent	significand
0	I,II,III		
1	IV,V	Aa:=Af	Ba:=Bf
	I,II,III	Ae:=Aa−Ab	if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
2	IV,V	Aa:=Ae Ab:=0	Bf:=0 if (bt=1) then Bf[0]:=1 Ba:=2×Be
	I,II,III	Ae:=Aa+Ab	if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
3	IV,V	Aa:=Ae	if (bt=1) then Bf[−1]:=1 Ba:=2×Be
	I,II,III	Ae:=Aa+Ab	if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
4	⋮	⋮	⋮
i	IV,V	Aa:=Ae	if (bt=1) then Bf[2−i]:=1 Ba:=2×Be
	I,II,III	Ae:=Aa+Ab	if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
15	⋮	⋮	⋮
16	IV,V	Aa:=Ae	if (bt=1) then Bf[−14]:=1 Ba:=2×Be
	I,II,III	Ae:=Aa+Ab	if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
17	IV,V	if (Bf[0] = 0) then Ab:=−1	Ba:=Bf, Bb:=0
	I,II,III	Ae:=Aa+Ab	Be:=Ba+Bb
18	IV,V	Af:=Ae	if (Bf[0]=0) then Bf:=2×Be else Bf:=Be

Figure 11: The 18 cycles needed for the division algorithm. The i-th bit of register Bf is denoted by Bf[i]. The dividend is stored in the register pair <Af,Bf> and the divisor in <Ab,Bb> before the operation is started.

cycle	stage	exponent	significand
0	I,II,III		
1	IV,V		If (Af[0]=1) then Ba:=2×Bf else Ba:=Bf Bb[0]:=1
	I,II,III		if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
2	IV,V		Bf:=0 // if (bt=1) then Bf[0]:=1 Ba:=2×Be, Bb:=2×Bf, Bb[−1]:=1
	I,II,III		if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
3	IV,V		if (bt=1) then Bf[−1]:=1 Ba:=2×Be, Bb:=2×Bf, Bb[−2]:=1
	I,II,III		if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
4	⋮	⋮	⋮
i	IV,V		if (bt=1) then Bf[2−i]:=1 Ba:=2×Be, Bb:=2×Bf, Bb[1 − i]:=1
	I,II,III		if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
17	⋮	⋮	⋮
18	IV,V		if (bt=1) then Bf[−16]:=1 Ba:=2×Be, Bb:=2×Bf
	I,II,III		if (Ba−Bb ≥ 0) then Be:=Ba−Bb, bt:=1 else Be:=Ba, bt:=0
19	IV,V	Aa:=Af/2	Ba:=Bf, Bb:=0
	I,II,III	Ae:=Aa+0	Be:=Ba+Bb
20	IV,V	Af:=Ae	Bf:=Be

Figure 12: The 20 cycles needed for the square root algorithm. The i-th bit of registers Bf and Af are denoted by Bf[i] and Af[i] respectively. The argument is stored in the register pair <Af,Bf> before the operation is started.



26

Figure 13: The complete architecture of the Z3