

# Zuse's Z3 Square Root Algorithm

Talk given at Fall meeting of the Ohio Section of the MAA  
October 1999 - College of Wooster

## Abstract

Brian J. Shelburne  
Dept of Math and Comp Sci  
Wittenberg University

In 1941 Konrad Zuse completed the Z3 sequence controlled calculator. The Z3 was a binary machine (numbers were stored internally in floating point binary) with a 64 by 22 bit memory which could perform the four standard arithmetic operations as well as take square roots. This talk examines the algorithm Zuse used for his square root instruction and how it was implemented on the Z3.

## Zuse's Z3 Square Root Algorithm- The Talk

### Konrad Zuse (1910 - 1995)

born 1910 in Berlin, Germany

studied civil engineering at Technische Hochschule Berlin-Charlottenburg

worked as design engineer in aircraft industry

needed calculations for design and analysis of structures

his first attempt at automating calculation process was to design a form to track  
intermediate results of calculations

not familiar with design of mechanical calculators

### His Early Machines: The Z1 - Z3

**Z1:** begun in 1936

floating-point binary representation

24 bit word length - 1 sign bit, 7 bit exponent, 16 bit mantissa (fraction)

4 decimal digit precision

completely mechanical (memory, arithmetic unit and control)

16 word prototype memory completed 1937

controlled by tape (used discarded 35 mm film) - hand punched!

four functions : + -  $\times$   $\div$

mechanical arithmetic unit unreliable

**Z2:** 1938 - 1940 (smaller hybrid machine)

relay based ALU and control; mechanical memory

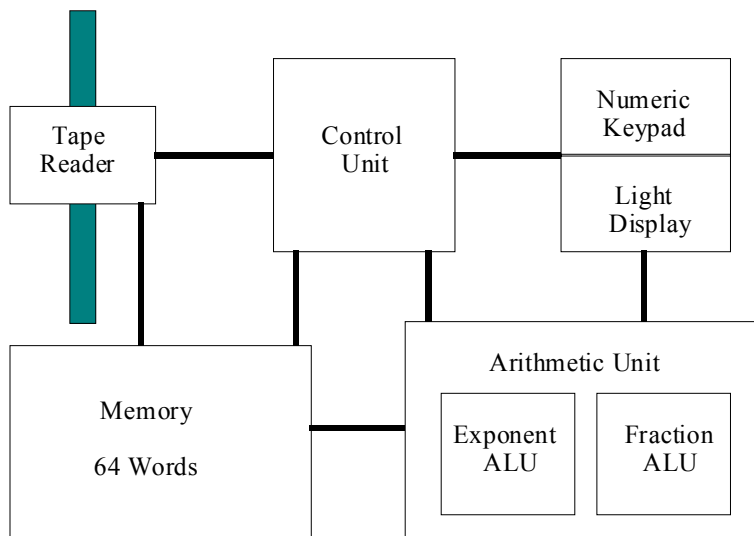
"proof of concept" for relay based calculator

demonstration led to funding for Z3 by Deutsche Versuchsanstalt fur Luftfahrt

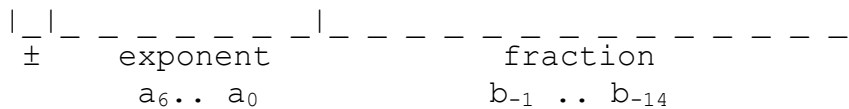
**Z3:** 1939 - 1941 (relay based machine)

same design as Z1 except *included* square root operation

## An Overview of the Z3



**Memory** : 64 by 22 bits - normalized floating point binary



**Numeric Keyboard** input : 4 columns of 10 keys for digits  
 1 row of exponent keys labeled : -8, -7, ... 7, 8

**Display Output** - row of 5 lights for fraction plus exponent lights labeled -8, -7, .. 7, 8

**Tape Control**: nine 8 bit instructions

- |  |                         |
|--|-------------------------|
| 01 110000 - read keyboard                    | 01 100000 - add         |
| 01 111000 - display result                   | 01 101000 - subtract    |
| 11 $z_1 z_2 z_3 z_4 z_5 z_6$ - load address  | 01 001000 - multiply    |
| 10 $z_1 z_2 z_3 z_4 z_5 z_6$ - store address | 01 010000 - divide      |
|  | 01 011000 - square root |

**Arithmetic Unit**: separate ALU's for exponents and fractions

**Control Unit**: instructions were implemented as a sequence of micro-operations which controlled movement of data through machine.

## How Do You Take A Square Root?

1. Use a calculator
2. Use a slide rule
3. Use a table of logarithms
4. Use Newton's Iteration Formula to solve  $f(x) = x^2 - a$  by using the iteration formula  $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = \frac{1}{2} \times (x_n + \frac{a}{x_n})$  where  $x_0$  equals some initial "guess" ( $x_0 = a/2$  works well for  $a > 1$ ). Note this requires one division per iteration (expensive!).
5. Use the binomial expansion of  $\sqrt{1-x}$

$$\begin{aligned} \sqrt{1-x} &= \\ 1 - \frac{1}{2}x^1 - \sum_{k=2}^{\infty} \frac{1 \times 3 \times 5 \times \dots \times (2k-3)}{2^k \times k!} x^k &= \\ 1 - \frac{1}{2}x^1 - \frac{1}{2^2 \times 2!}x^2 - \frac{1 \times 3}{2^3 \times 3!}x^3 - \frac{1 \times 3 \times 5}{2^4 \times 4!}x^4 - \dots \end{aligned}$$

We can calculate the  $\sqrt{a}$  by finding some value  $b^2$  close to  $a$  and using the above series expansion for  $x = (1 - a/b^2)$  since

$$\sqrt{a} = \sqrt{b^2 + a - b^2} = b \sqrt{1 - (1 - \frac{a}{b^2})}$$

6. Complete the Square: Without loss of generality assume  $1 \leq a < 100$  and suppose  $Q_{k-1}$  is an approximation to  $\sqrt{a}$  accurate to  $10^{-k+1}$  ( $k-1$ st digit below the decimal point). That is

$$Q_{k-1} = d_0 + d_1 \times 10^{-1} + d_2 \times 10^{-2} + d_3 \times 10^{-3} + \dots + d_{k-1} \times 10^{-k+1} \text{ where } d_i \text{ is a digit from 0 to 9.}$$

Find the largest digit  $d_k$  such that

$$a - (Q_{k-1} + d_k \times 10^{-k})^2 = (a - Q_{k-1}^2) - 2 \times (d_k \times 10^{-k}) \times Q_{k-1} - (d_k \times 10^{-k})^2 \geq 0$$

### Completing the Square (cont.)

To find the largest digit  $d_k$  such that

$$a - (Q_{k-1} + d_k \times 10^{-k})^2 = (a - Q_{k-1}^2) - 2 \times (d_k \times 10^{-k}) \times Q_{k-1} - (d_k \times 10^{-k})^2 \leq 0$$

factor out  $(d_k \times 10^{-k})$

$$(a - Q_{k-1}^2) - (d_k \times 10^{-k}) \times (2 \times Q_{k-1} + (d_k \times 10^{-k})) \leq 0$$

and scale (multiply) by  $10^{2k}$ . The sign of the expression is not changed while all values become integers.

$$10^{2k} \times (a - Q_{k-1}^2) - d_k \times (2 \times Q_{k-1} \times 10^k + d_k) \leq 0$$

Thus find the largest digit  $d_k$  such that the above expression is positive. Furthermore the difference

$$10^{2k} \times (a - Q_{k-1}^2) - d_k \times (2 \times Q_{k-1} \times 10^k + d_k) = 10^{2k} \times (a - Q_k^2)$$

can be used to simplify the calculations at the next step.

Finding the largest  $d_k$  which maintain this inequality is the theory behind the following well-known algorithm (well-known before the advent of cheap calculators) used to compute a square root by hand. To demonstrate we take the square root of 20.375.

	4 . 5 1 3			
	/ 20 . 37 50 00			
	16			
	--			
85	4 37			$\leftarrow 10^2 \times (a - Q_0^2)$
	4 25			$\leftarrow d_1 \times (2 \times Q_0 \times 10 + d_1)$
	-----			
901	12 50			$\leftarrow 10^4 \times (a - Q_1^2)$
	9 01			$\leftarrow d_2 \times (2 \times Q_1 \times 10^2 + d_2)$
	-----			
	3 49 00			$\leftarrow 10^6 \times (a - Q_2^2)$
9023	2 70 69			$\leftarrow d_3 \times (2 \times Q_2 \times 10^3 + d_3)$
	-----			
	78 31 00			

First - find the largest integer  $d_0 = Q_0$  whose square is less than or equal to the integer part of the number  $a$ . Square  $Q_0$  and subtract it from the integer part. Bring down the next two digits.

Second. - Find the largest integer  $d_k$  such that  $(20 \times Q_{k-1} + d_k)$  times  $d_k$  is less than the previous remainder. Repeat until ...

## The Binary Version of the "Completing the Square" Square Root Algorithm

The same algorithm can be done in binary. We find the square root of the binary value 10100.011 which is 20.375 decimal. The floating point representation is  $1.0100011 \times 2^{10}$  (note the even exponent). Therefore the square root of 10100.011 is the square root of 1.0100011 times  $2^{10}$  (or 4).

Remember that multiplication by 2 in binary is equivalent to a left shift. Furthermore scaling by powers of 10 (binary) is scaling by powers of 2.

```

      1 . 0 0 1 0 0 0 0 0 1 1
    /1 . 01 00 01 10 00 00 00 00 00
      1
      -
      0 01          <- 22 × (a - Q02)
100  0 00          <- b1 × (2 × Q0 × 21 + b1)
      -----
           1 00      <- 24 × (a - Q12)
1000  0 00          <- b2 × (2 × Q1 × 22 + b2)
           ----
           1 00 01  <- etc...
10001 1 00 01
           -----
                   0 10
100100  0 00
                   ----
                   10 00
1001000  00 00
                   -----
                   10 00 00
10010000 00 00 00
                   -----
                   10 00 00 00
100100000 00 00 00 00
                   -----
                   10 00 00 00 00
1001000000 00 00 00 00 00
                   -----
                   10 00 00 00 00 00
10010000001 1 00 10 00 00 01
                   -----
                   11 01 11 11 11 00
100100000011 10 01 00 00 00 11   whew !!!

```

So therefore the square root of 10100.011 is approximately  $1.0010000011 \times 2^{10} = 100.10000011 = 4.51171875$  which is "close" to 4.513867521

## The Z3 Square Root Implementation - Completing the Square in Binary

The Z3 uses a variant of the above algorithm to compute a square root. Given  $1 \neq a < 100$  (all values are in binary) if  $Q_{k-1}$  is an approximation to  $\sqrt{a}$  accurate to  $2^{-k+1}$ , that is

$$Q_{k-1} = b_0 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + b_3 \times 2^{-3} + \dots + b_{k-1} \times 2^{-k+1} \text{ where each } b_i \text{ is either 0 or 1}$$

find the largest binary digit  $b_k$  (0 or 1) such that

$$a - (Q_{k-1} + b_k \times 2^{-k})^2 = (a - Q_{k-1}^2) - 2 \times (b_k \times 2^{-k}) \times Q_{k-1} - (b_k \times 2^{-k})^2 \geq 0$$

The beauty of this system is you have two choices for  $b_k$  : 0 or 1. If

$$(a - Q_{k-1}^2) - (2 \times 1 \times 2^{-k} \times Q_{k-1} + (1 \times 2^{-k})^2) \geq 0$$

or after scaling by  $2^k$  if

$$2^k \times (a - Q_{k-1}^2) - (2 \times Q_{k-1} + 2^{-k}) \geq 0$$

then  $b_k$  is 1; otherwise  $b_k = 0$ . For reasons given below the Z3 scaled the above inequality by  $2^k$  instead of  $2^{2k}$  which is analogous to the  $10^{2k}$  used in the decimal version.

Observe that

$$2^k \times (a - Q_{k-1}^2) - (2 \times b_k \times Q_{k-1} + b_k \times 2^{-k}) = 2^k \times (a - Q_k^2)$$

which can be used to simplify calculations at the next step. In binary, multiplication by 2 is done by a left shift. Define  $r_k$  equal to  $2^k \times (a - Q_k^2)$  noting that  $r_0 = (a - 1)$  The algorithm is given below.

pre-condition  $100 > a \geq 1$

1. set  $Q_0 = 1$
2.  $r_0 = (a - 1)$
3. for  $k = 1, 2, 3, \dots$

Compute  $2 \times r_{k-1} - (2 \times Q_{k-1} + 2^{-k})$

If positive or zero

$$\quad \quad \quad Q_k = Q_{k-1} + 2^{-k}$$

$$\quad \quad \quad r_k = 2 \times r_{k-1} - (2 \times Q_{k-1} + 2^{-k})$$

Otherwise

$$\quad \quad \quad Q_k = Q_{k-1}$$

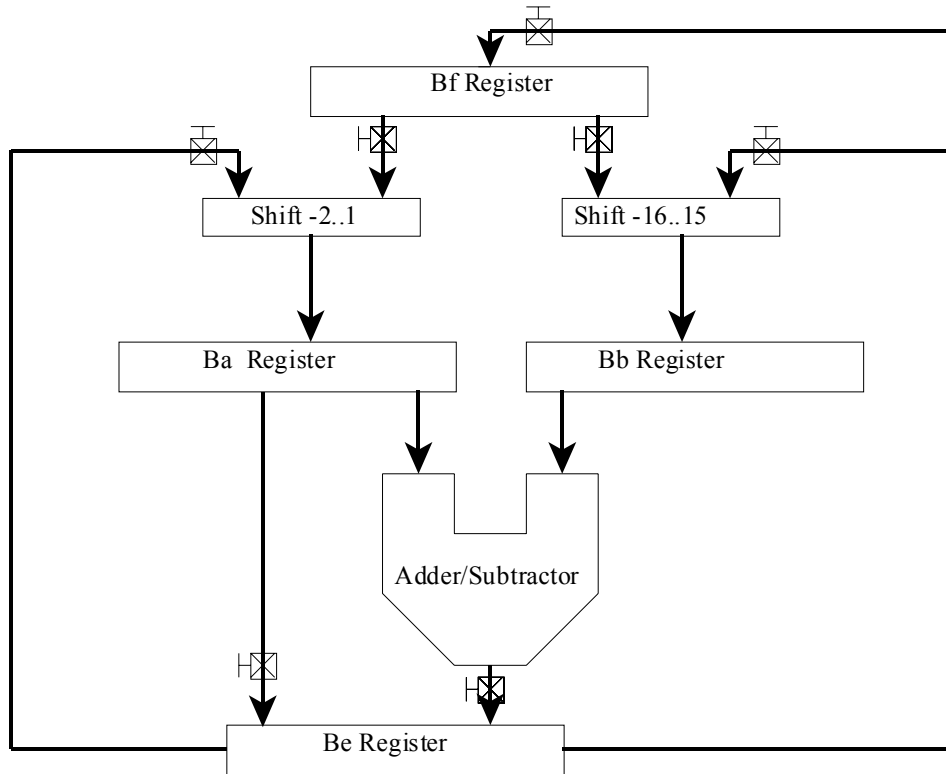
$$\quad \quad \quad r_k = 2 \times r_{k-1}$$

**Example - Compute the square root of 1.0100011**

<b>k</b>	<b><math>2 \times r_{k-1} - (2 \times Q_{k-1} + 2^{-k})</math></b>		<b><math>r_k = a - Q_k^2</math></b>	<b><math>Q_k</math></b>
0			0.0100011	1.0
1	0.100011 - 10.1	negative	0.100011	1.0
2	1.00011 - 10.01	negative	1.00011	1.00
3	10.0011 - 10.001	positive	0.0001	1.001
4	0.001 - 10.0101	negative	0.001	1.0010
5	0.01 - 10.01001	negative	0.01	1.00100
6	0.1 - 10.010000	negative	0.1	1.001000
7	1.0 - 10.0100001	negative	1.0	1.0010000
8	10.0 - 10.01000001	negative	10.0	1.00100000
9	100.0 - 10.010000001	positive	1.101111111	1.001000001
10	11.011111111 - 10.010000011	positive	1.001111011	1.0010000011

This agrees with the example given above.

## A Closer Look at the Architecture of the Z3 Arithmetic Unit



Z3 had separate ALU's to handle the exponent and fractional parts of a number

Basic data path of fraction ALU consisted of four registers (Bf, Ba, Bb, and Be) an adder/subtractor unit and two shifter units connected by data lines. A number of "valves" controlled the follow of data around the data path

Hardware operations included **addition** and **subtraction**, **pre-shifting** input to the Ba and Bb registers, **detecting a positive or zero result** and **setting individual bits** in a register.

A *micro-operation* consisted of a timed sequence of "valves" openings that allowed the contents of registers to flow through the pre-shifters and adder/subtractor. Detecting a positive or zero result could conditionally alter the flow of data.

Instructions were implemented as *sequences* of micro-operations.

## Micro-operation Implementation of Z3 Square Root

Note Bf[-k] refers to bit -k in register Bf

```

0.   Bf := a                // Load Bf with a
     Ba := Bf (or 2 × Bf if exponent is odd) // Ba = a
     Bb[0] := 1             // Q0 = 1

1.   if Ba - Bb $ 0 then    // test
     Be := Ba - Bb
     Bf[0] := 1            // set bit
   else
     Be := Ba

                                     // set up next comparison
     Ba = 2 × Be           // 2 × r0
     Bb = 2 × Bf; Bb[-1] := 1 // 2 × Q0 + 2-1

2.   if Ba - Bb $ 0 then    // test
     Be := Ba - Bb
     Bf[-1] := 1           // set bit
   else
     Be := Ba

                                     // set up next comparison
     Ba := 2 × Be          // 2 × r1
     Bb := 2 × Bf; Bb[-2] := 1 // 2 × Q1 + 2-2

```

and in general

```

if Ba - Bb $ 0 then           // if  $2r_{k-1} - (2 \times Q_{k-1} + 2^{-k}) \leq 0$  then

     Be := Ba - Bb           //  $r_k = 2 \times r_{k-1} - (2 \times Q_{k-1} + 2^{-k})$ 
     Bf[-k] := 1            //  $Q_k = Q_{k-1} + 2^{-k}$ 
else
     Be := Ba               //  $r_k = 2 \times r_{k-1}$ 
                             //  $Q_k = Q_{k-1}$ 
                             // set up next comparison
     Ba := 2 × Be           //  $2 \times r_k$ 
     Bb := 2 × Bf; Bb[-(k+1)] := 1 //  $2 \times Q_k + 2^{-(k+1)}$ 

```

At the end the exponent which was even was divided by 2 (a right shift) yielding the square root.

The thing to notice was the *economy of operations* needed to compute a square root - shifts, subtractions, comparisons and setting individual bits within a word.

## Summary

1. The square root algorithm used by the Z3 was the binary analog of the "completing the square" algorithm, a widely known (?) manual technique for obtaining a square root one digit at a time.

2. For  $1 \leq k < 100$  and  $Q_{k-1}$  an approximation of  $\sqrt{a}$  accurate to  $2^{-k+1}$  set bit  $b_k$  equal to 1 if

$$a - (Q_{k-1} + b_k \times 2^{-k})^2 \geq 0$$

otherwise setting bit  $b_k$  equal to 0. It was shown that this is the same as checking if

$$2^k \times (a - Q_{k-1}^2) - (2 \times Q_{k-1} + 2^{-k}) \geq 0$$

and leads to the iteration algorithm Repeat until tired ...

Compute  $2 \times r_{k-1} - (2 \times Q_{k-1} + 2^{-k})$

If positive or zero

$$\quad Q_k = Q_{k-1} + 2^{-k}$$

$$\quad r_k = 2 \times r_{k-1} - (2 \times Q_{k-1} + 2^{-k})$$

Otherwise

$$\quad Q_k = Q_{k-1}$$

$$\quad r_k = 2 \times r_{k-1}$$

where  $r_{k-1} = (a - Q_{k-1}^2)$  and initial condition  $Q_0 = 1$  and  $r_0 = (a - 1)$

3. A close examination of this algorithm reveals that it requires only the operations of **subtraction**, **left shift** (multiplication by 2), **set bit**, and **test if positive or zero**. - all of which are easy to implement in binary logic.

4. The Z3 had a data path that could easily implement these operations and a micro-operation sequencer that could iterate each step of the algorithm the requisite number of times. Use of floating point binary notation helped.

5. Binary representation is used in computers because for economic reasons computer hardware is built out of bi-stable components. However, many complicated arithmetic operations like multiplication, division and square root are easy to implement in binary. Even though we humans find binary notation complicated to use (even moderately sized numbers take a lot of bits), because of the ease in implementing complicated algorithms in binary, there is something to be said for the advantages of binary notation in and of itself.

*Any sufficiently advanced civilization uses a number system based on a power of two.*

## **Epilog : A Follow Up on the Z3 - the Z4**

The Z3 was really a "proof of concept" for the more ambitious Z4, a 32 bit machine begun after the Z3 was completed. Unlike the Z3, the Z4 used a mechanical memory of 1000 words (which took up a cubic meter of space!). Before it was completed it was moved to Gottingen to avoid the Berlin air raids which had destroyed the Z3.

Just before Gottingen was "liberated" at the end of WWII, the Z4 was moved to Hinterstein Bavaria. In 1950 the refurbished Z4 with some additional features that gave it a conditional jump was moved to ETH Zurich. In 1955 it was moved to Basel where it gave "acceptable service" until 1960.

After the war Zuse founded Zuse Kommandit Gesellschaft (Zuse KG) which produced for the Leitz Optical Company a relay based, punch-tape controlled "computer" called the Z5 which was based on the Z4 design. This led to the production of a number of relay-based calculators, called the Z11. In 1956 Zuse began work on a vacuum-tube based computer called the Z22 with a transistor version following. A series of mergers and takeover finally lead to Zuse KG becoming part of Siemens.

### **References**

R. Rojas, "Konrad Zuses' Legacy: The Architecture of the Z1 and Z3", *Annals of the History of Computing*, vol. 19, no. 2, pp. 5 - 16, 1997

I. Koran, *Computer Arithmetic Algorithms*, Englewood Cliff, N.J.: Prentice-Hall, 1993

M. Williams, *A History of Computing Technology*, Los Alamitos, CA: IEEE Computer Society Press, 1997