How the ENIAC took a Square Root

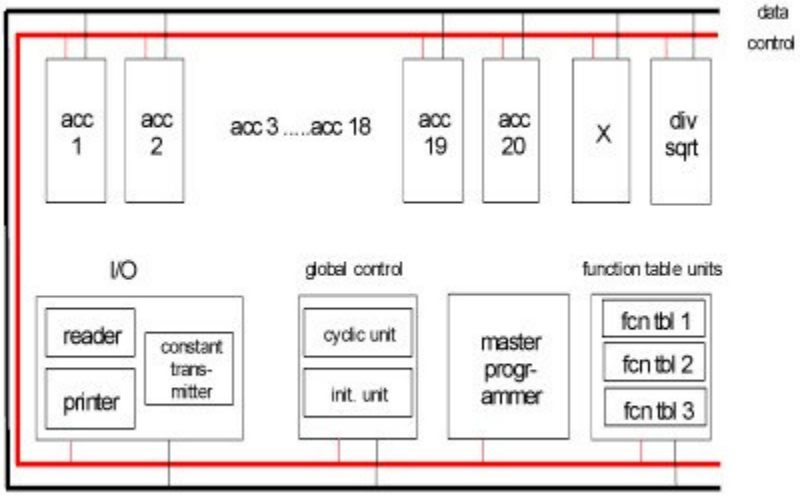
revised 01/19/2009

Abstract: The ENIAC (Electronic Numerical Integrator and Computer) is the world's first electronic computer. However it could only store twenty 10-digit decimal numbers and was programmed by wiring the computational units together. These limitations made it very unlike today's stored-program computers. The ENIAC had hardware to add, subtract, multiply, divide and take a square root. This last operation is interesting since computers normally don't do square roots in hardware. So given the limited capabilities of the ENIAC, how did it take a square root?

History: The ENIAC was a war time effort by the University of Pennsylvania's Moore School of Electrical Engineering for the Army's Ballistics Research Lab at Aberdeen Maryland. Its purpose was to compute "firing tables" for artillery, information that gunners would use to properly aim and fire their guns. During World War II such computational work for firing tables was being done using the Moore School's Differential Analyzer, an analog device that could solve differential equations. 1942 John Mauchly, a physics professor working at the Moore School who had a long time interest in scientific computing, submitted a proposal for using vacuum tube devices for high speed computing. Discussions with J. Presper Eckert, graduate student at the Moore School, convinced him that such a devices was possible. In 1943 when the need for more firing tables became more acute, Mauchly's proposal was brought up with the result that the army awarded a contract to the Moore School to build what we know today as the ENIAC. Mauchy was the principal consultant and Eckert the chief engineer. Work on ENIAC began in the summer of 1943 but the ENIAC was not completed until the after the war ended; the ENIAC was officially unveiled in February 1946.

Overview of the ENIAC: The ENIAC was "build" around twenty 10 decimal digit **Accumulators** which could add and subtract at electronic speeds. To add or subtract two numbers, the contents of one accumulator was sent to a second. Accumulators could "receive" a number, transmit its contents "additively" (for addition) or transmit "subtractively" (for subtraction). The ENIAC was capable of performing 5000 additions/subtractions per second!

An accumulator contained ten decade counters. Each *decade counter* (designed by Eckert) was a ten state circuit that could store a single decimal digit much like a ten position "counter wheel" from a mechanical calculator. An electronic "pulse" (a "square wave") would advance the decade counter one position. Digits were sent as a "train of pulses" so if a decade counter was in the "4" state, upon receiving a train of 3 pulses it would advance to the "7" state. If it received a train of 8 pulses it would advance with "wrap-around" to the "2" state and while generating a "carry-pulse" to the next decade. Subtraction was done by using 9 complement digits (i.e. -7 was sent as 9 - 7 = 2 pulses) with an extra pulse added to the units digit (essentially tens-complement notation). The ten digit "pulse trains" plus a sign "pulse" were sent over eleven parallel wires. An eleventh two-state plus-minus counter was used for the sign.



The ENIAC also had a high-speed **Multiplier** unit for multiplication. The Multiplier contained the logic to multiply a ten digit number by a single digit to obtain a partial product. The partial products were then added together (using accumulators) to obtain the final product. The Multiplier made use of four accumulators to multiply (six for a 20 digit product).

The final computational unit was a **Divider/Square Rooter**. Division and taking a square root was orchestrated as a series of subtractions/additions and shifts which like the Multiplier made use of a number of accumulators but unlike the Multiplier contained no special computational hardware to do either; in other words it used accumulators to do the needed addition and subtraction. All work was done in decimal. Division was done by repeated subtractions followed by repeated additions etc. using a technique called "non-restoring division". As we shall see taking a square root used a similar technique which is probably why the two operations were combined in one unit.

Input Output was provided by a mechanical IBM card Reader and card Punch which were connected to an electronic **Constant Transmitter** used to stored constants. The Constant Transmitter provided the interface between the slow-speed mechanical I/O devices and the rest of the high-speed electronic ENIAC. There were also three **Function Table** units, essentially 100 by 10 digit ROM memories which were set by switches.

The units of the ENIAC were connected by two buses: a data bus used to transmit the "ten digits plus sign" over parallel wires and a control bus. Program control for the ENIAC was distributed, not centralized. Each accumulator contained control logic that would allow it to "work" with other accumulators to perform a sequence of calculations. Programming was accomplished by setting switches on the various units and wiring the connections between them using the control bus for control signals and the data bus for data. A **Master Control** unit was used to "loop" the various sequence of calculations set up between accumulators. A **Cycling Unit** synchronized the various units over a third cycle bus (not shown above). There was also an **Initializing Unit**.

The ENIAC did not use a "fetch-decode-execute" cycle to execute its program since there was no memory to store instructions. And the ENIAC was not "programmed" using paper tape unlike Zuse's Z3 (a device unknown to Mauchly and Eckert) or Aiken's Automatic Sequence Controlled Calculator (Harvard Mark I) completed in the summer of 1944, both of which read their instructions from paper tape readers. The reason for not using paper tape readers was the slow speed of such mechanical devices. If the ENIAC was to be *truly fast*, both instructions and calculations had to be executed at electronic speeds. The only way to effectively do the former

was to "wire" the program into the machine. The idea was not completely new; IBM punch card equipment could be programming in a limited way using plug boards.

All of this was packaged into 40 panels each 2 feet wide by 8 feet high arranged in a U shape in a 30 by 60 foot area. A diagram of the ENIAC from the "ENIAC Progress Report" of June 30, 1945 can be seen <click here>.

A Method for Taking a Square Root

The method used by the ENIAC Square Rooter to take a square root required only addition and subtraction. It was based on the formula that the sum of the first n odd integers is n^2 squared.

$$1+3+5+...+(2n-1)=n^2$$

To calculate the square root of m, find the *smallest integer* n such the sum of the first n odd integers *exceeds* m. This can be done by *subtracting the consecutive odd integers* from m until a negative result is obtained. If n is the smallest integer such that $m-\left(1+3+5+...+\left(2n-1\right)\right)<0$ then $\left(n-1\right)^2 \le m < n^2$ or $n-1 \le \sqrt{m} < n$. If a is the n^{th} odd integer $2n-1$, then $n=\frac{a+1}{2}$ so

$$\frac{a-1}{2} \le \sqrt{m} < \frac{a+1}{2}$$

Example: To estimate the square root of 7251, subtract the consecutive odd integers until a negative result is obtained. Calculate $7251-\left(1+3+5+...+169\right)=26$ and $7251-\left(1+3+5+...+171\right)=-145$. So $a=171$. Thus

$$85=\frac{171-1}{2} \le \sqrt{7251} < \frac{171+1}{2}=86$$

Aside: In comparing the magnitudes of 26 and -145 observe that 85 is the closer approximation. Using linear interpolation, 26/171 = 0.1520 a better approximation, 85.152 can be obtained. Note $\sqrt{7251} \approx 85.15280$.

End of Example

Additional precision is possible if m is first multiplied by 100^k for some integer k. Calculate the square root of $m \times 100^k$ then *divide* by 10^k. For example multiply 7251 by 100^2 to obtain 72,510,000 and take the square root using the above technique.

$$72,510,000-(1+3+5+...+17031)=-12256<0$$

So $\frac{17031-1}{2}=8515\le\sqrt{72,150,000}<\frac{17031}{2}=8516$ which after dividing by 10^2 yields $85.15\le\sqrt{7251}<86.16$.

Reality Check: The ENIAC could do 5000 additions/subtractions per second. Therefore to subtract 8516 odd integers would take 1.7 seconds! This is not a very efficient way to find the square root.

A More Efficient Approach

The above algorithm can be made more efficient. For example, $\sqrt{7251}$ can be found to four digits of accuracy using no more than 40 additions/subtractions instead of over 8000 if we calculate its square root one digit at a time.

First of all since 7251 is on the order of 100^2, we'll calculate $\sqrt{7251\times100^2}=\sqrt{72,510,000}$ since $72,510,000$ is an integer on the order of 100^4. (The general rule of thumb to calculate $\sqrt{m}$ to k digits of accuracy is to scale m to an integer on the order of 100^k).

Begin by finding the smallest odd integer n such that $m-(1+3+5+...+(2n-1))\times100^{k-1}<0$. Observe that $(2n-1)\le19$ since $(1+3+5+...+19)=100$ and m is on the order of (actually bounded by) 100^k. Therefore $(n-1)^2\times100^{k-1}\le m<n^2\times100^{k-1}$ and so $(n-1)\times10^{k-1}\le\sqrt{m}<n\times10^{k-1}$ which is not a very good estimate.

Example: $72,510,000-(1+3+5+...+17)\times100^3=-849\times100^2<0$ Therefore

$$8\times10^3\le\sqrt{72,510,000}<9\times10^3$$

Observe that the last value subtracted was $17\times100^3=(1700)\times100^2$

End of Example

The last odd multiple of 100^{k-1} subtracted from m above was $\left[(2n-1)\times 100\right]\times 100^{k-2}$.

$(2n-1)\times 100$, an odd multiple of 100, can be rewritten as a sum of ten consecutive odd integers[1].

$$(2n-1)\times 100 = 200\times(n-1)+100 = 10\left[20\times(n-1)\right]+(1+3+5+\ldots+19)$$
$$\left[20(n-1)+1\right]+\left[20(n-1)+3\right]+\left[20(n-1)+5\right]+\ldots+\left[20(n-1)+19\right]$$

Letting $N=(2n-1)\times 100$, we can express this as follows.

$$N=(2n-1)\times 100=\left(\frac{N}{10}-9\right)+\left(\frac{N}{10}-7\right)+\ldots+\left(\frac{N}{10}+7\right)+\left(\frac{N}{10}+9\right)$$

Example continued: $N=1700=161+163+\ldots+177+179$ End

Starting with the last term $\left(\frac{N}{10}+9\right)$ subtract consecutive terms until the expression becomes positive. Subtract out the last term added in, $\left(\frac{N}{10}+j\right)$ for j odd and $-9\le j\le 9$ making j the smallest odd integer such that

$$m-\left(\sum_{i=1}^{n-1}(2i-1)\times 100^{k-1}+\left(\frac{N}{10}-9\right)\times 100^{k-2}+\left(\frac{N}{10}-7\right)\times 100^{k-2}+\ldots+\left(\frac{N}{10}+j\right)\times 100^{k-2}\right)<0$$

The expression being subtracted out is the sum the odd integers from 1 to $\left(\frac{N}{10}+j\right)$ times 100^{k-2}. Hence it is a square.

Example continued: $-849\times 100^2-(179+177+175+173+171)\times 100^2=26\times 100^2>0$ so if we subtract out the last term added back in, $(171)\times 100^2$, we now have

$$7251\times 100^2-(1+3+5+\ldots+171)\times 100^2<0$$

Therefore $(1+3+5+\ldots+169)\times 100^2\le 72,510,000<(1+3+5+\ldots+169+171)\times 100^2$.

Consequently $\frac{171-1}{2}\times 10^2=85\times 10^2\le\sqrt{72,510,000}<86\times 10^2=\frac{171+1}{2}\times 10^2$

End of Example

[1] In particular 100 is the sum of the odd integers $1+3+5+\ldots+19$

If $\left(\frac{N}{10}+j\right)$ for j odd and $-9\le j\le 9$ then $\left(\frac{N}{10}+j\right)$ is the $\left(\frac{N}{20}+\frac{j+1}{2}\right)$ th odd integer.

Example: For $N=1700$, and $j=1$ $\left(\frac{1700}{10}+1\right)=171$ is the $\left(\frac{1700}{20}+\frac{1+1}{2}\right)=86$ th odd integer

Resetting n to equal $\left(\frac{N}{10}+j\right)$ we have

$$\left(1+3+5+...+(2n-3)\right)\times 100^{k-2}\le m<\left(1+3+5+...+(2n-3)+(2n-1)\right)\times 100^{k-2}$$

Note that the last value subtracted was $\left((2n-1)\times 100\right)\times 100^{k-3}$

At this point we repeat the process.

Example: $72,510,000-(1+3+5+...+171)\times 100^2=-145\times 100^2<0$

$$(171\times 100)\times 100=\left(\left(\frac{17100}{10}\right)-9\right)\times 100+\left(\left(\frac{17100}{10}\right)-7\right)\times 100+...+\left(\left(\frac{17100}{10}\right)+9\right)\times 100=$$
$$1701\times 100+1703\times 100+...+1717+1719\times 100$$

Adding back the odd integers in reverse order starting from 1719 we observe

$$72,510,000-\left((1+3+5+...+169)\times 100^2+(1719+1717+1715+...+1701)\times 100\right)=899\times 100>0$$

and

$$72,510,000-\left((1+3+5+...+169)\times 100^2+(1719+1717+1715+...+1703)\times 100\right)=-804\times 100<0$$

Thus $\frac{1703-1}{2}\times 10=8510\le\sqrt{72,510,000}<8520=\frac{1703+1}{2}\times 10$

where 1703 is the 852^{nd} odd integer

and $72,510,000-(1+3+5+...+1703)\times 100=-80400<0$

So

$$1703\times 100=\left(\frac{170300}{10}-9\right)+\left(\frac{170300}{10}-7\right)+\ldots+\left(\frac{170300}{10}+7\right)+\left(\frac{170300}{10}+9\right)=$$
$$17021+17023+\ldots+17037+17039$$

Adding back the odd integers in reverse order starting from 17039 we observe

$$72,510,000-\left((1+3+5+\ldots+1703)\times 100+\left(17021+17023+17025+\ldots+17029\right)\right)=4775>0$$

And

$$72,510,000-\left((1+3+5+\ldots+1703)\times 100+\left(17021+17023+17025+\ldots+17031\right)\right)=-12256<0$$

Thus $$\frac{17031-1}{2}=8515\le\sqrt{72,510,000}<8516=\frac{17031+1}{2}$$

All this was done in less than 40 additions/subtractions

End of Example

Reality Check: With a reduction of the number of additions/subtractions to around 40, the ENIAC could now calculate the square root in approximately 0.008 seconds.

How this Algorithm was Implemented on the ENIAC

On the ENIAC the hardware to take a square root was combined with the divider since the sequence of operations used is similar to those used to divide (a non-restoring division technique was used). Taking a square root was done using a couple of accumulators to execute the series of subtractions and additions discussed. The divider/square rooter essentially orchestrated the sequence of steps needed. Unlike the high-speed multiplier unit the divider/square rooter contained no special circuits to perform arithmetic.

The square rooter actually performed calculations to obtain *twice* the square root. To obtain $2\sqrt{m}$, the value m was deposited to an accumulator which we shall call the *numerator*. Then a second accumulator, the *denominator*, was initialized to 10^8 (100^4) by setting the 9th digit to 1.

For example, to calculate twice the square root of 72,510,000 two accumulators were initialized as follows

```
Numerator:   0,072,510,000          Denominator:   0,100,000,000
```

Step 1: Subtract increasing odd multiples of 10^8 (100^4) from the numerator until a negative result is obtained.

The denominator was subtracted from the numerator and the denominator was incremented by 2 in the 9th digit. This was repeated until there was a sign change in the numerator. Note that the denominator is incremented after the subtraction but before the sign of the numerator is tested so that the contents of the denominator is not the last value subtracted!

```
Numerator: -0,027,490,000          Denominator:   0,300,000,000
```

At this point we know that $m-1\times100^4<0$. If N was the last *denominator* value subtracted[2] then

$$N=\left(\frac{N}{10}-9\times100^{k-1}\right)+\left(\frac{N}{10}-7\times100^{k-1}\right)+\ldots+\left(\frac{N}{10}+7\times100^{k-1}\right)+\left(\frac{N}{10}+9\times100^{k-1}\right)$$

where $k=4$. So $\left(\frac{N}{10}+9\times100^3\right)$ was the last *odd multiple* of 100^3 subtracted out. Therefore starting with this value *add back* the decreasing sequence of odd multiples of 100^3 until a sign change is obtained. *If* the denominator contained $N\times100^3$,

```
Numerator: -0,027,490,000          Denominator(?):   0,100,000,000
```

the value to be added back *could* be obtained by right shifting the denominator and setting the 7th position to 9.

```
Numerator: -0,027,490,000          Denominator(?):   0,019,000,000
```

However, this was not done by the ENIAC square rooter! The ENIAC *could* accomplish the same purpose by *left shifting* the numerator instead of *right shifting* the denominator and setting the *8th* digit to 9 instead of the *7th* digit. This scaling trick would also eventually add 4 digits of accuracy to the final answer.

```
Numerator: -0,274,900,000          Denominator(?):   0,190,000,000
```

---

[2] In the previous section we set N equal to $(2n-1)\times100$ where $(2n-1)$ was the n^{th} odd integer and showed

$$N\times100^{k-1}=\left(\left(\frac{N}{10}-9\right)+\left(\frac{N}{10}-7\right)+\ldots+\left(\frac{N}{10}+7\right)+\left(\frac{N}{10}+9\right)\right)\times100^{k-1}$$

However, in this section it's better to define N to be $(2n-1)\times100^k$. The calculations are very similar.

But there was one further complication! The denominator contains the wrong value since it was incremented by 2 before the sign of the numerator was tested. So instead of setting the 8th digit to 9, it subtracts 11 from the 9th and 8th digits

```
Numerator: -0,274,900,000          Denominator:  0,300,000,000
                                   Denominator:  0,190,000,000
```

after subtracting 11

Step 2: Add back!

Add the denominator back into the numerator and decrement the denominator by 2 in the 8th digit until a sign change is obtained (numerator is positive).

```
Numerator: -0,274,900,000          Denominator:  0,190,000,000
           -0,084,900,000                        0,170,000,000
            0,085,100,000                        0,150,000,000
```

Note that `0,170,000,000` not `0,150,000,000` was the last value added back. If $N=(2n-1)\times 100^k$ was the last odd multiple of 100^k added back, since N can be expressed as the sum of ten odd multiples of 100^{k-1}, then $\left(\frac{N}{10}-9\times 100^{k-1}\right)$ was the last (smallest) odd multiple of 100^{k-1} added back. Beginning with $\left(\frac{N}{10}-9\times 100^{k-1}\right)$ subtract the increasing sequence of odd multiples of 100^{k-1} until a sign change is obtained (to negative). The ENIAC accomplished this by *left shifting* the numerator, not *right shifting* the denominator, and *adding* 11 to the 8th and 7th digits instead of *subtracting* 9 from the 7th digit since the numerator was smaller by 2 in the 8th digit (smaller by 20 in the 9th and 8th digits) .

```
Numerator:  0,851,000,000          Denominator:  0,161,000,000
```

At this point repeat alternately *subtracting/adding* the *denominato*r from/to the *numerator* and *incrementing/decrementing* the denominator by 2 in the p^{th} position until a sign change in the numerator is obtained. When the repeated *subtraction/addition* sequence terminates with a sign change, *left shift* the numerator, *subtract/add* 11 to the p and p-1st positions in the denominator, and decrement p.

Step 3: Subtract

```
                                                  P=7
                                   Denominator:  0,150,000,000
                                          add       11
Numerator:  0,851,000,000          Denominator:  0,161,000,000
              ...                                  ...
           -0,145,000,000                        0,173,000,000
```

Step 4: Add Back

```
                                          P=6
                          Denominator:  0,173,000,000
                              subtract      1,1
Numerator: -1,450,000,000 Denominator:  0,171,900,000
             ...                           ...
           0,089,900,000                 0,170,100,000
```

Step 5: Subtract

```
                                          P=5
                          Denominator:  0,170,100,000
                                 add        11
Numerator:  0,899,000,000 Denominator:  0,170,210,000
             ...                           ...
           -0,122,560,000                0,170,330,000
```

Step 6: Add Back

```
                                          P=4
                          Denominator:  0,170,330,000
                              subtract       11
Numerator: -1,225,600,000 Denominator:  0,170,319,000
             ...                           ...
           0,136,896,000                 0,170,303,000
```

Step 7: Subtract

```
                                            P=3
                          Denominator:  0,170,303,000
                                 add        1,1
Numerator:  1,368,960,000 Denominator:  0,170,304,100
             ...                           ...
           -0,163,784,100                0,170,305,900
```

Step 8: Add Back

```
                                              P=2
                          Denominator:  0,170,305,900
                              subtract         11
Numerator: -1,637,841,000 Denominator:  0,170,305,790
             ...                           ...
           0,065,216,000                 0,170,305,590
```

Step 9: Subtract

```
                                                        P=1
                                    Denominator:  0,170,305,590
                                            add             11
Numerator:  0,652,160,000           Denominator:  0,170,305,601
            0,481,854,399                         0,170,305,603
            0,311,548,796                         0,170,305,605
            0,141,243,191                         0,170,305,607
           -0,029,062,416                         0,170,305,609
```

The last odd integer subtracted was a = 170,305,607 which made the numerator negative, Thus after scaling we have

$$17,030.5606 \le 2\times\sqrt{72,510,000} < 17,030.5608$$

However, since the denominator was decremented by two *after* the final subtraction was performed, the last value in the denominator, 170,305,609 must have been used as the approximation to twice the square root. So after scaling

$$2\times\sqrt{7,2510,000} \approx 17,030.5609$$

If you compare the magnitudes of the last two numerator values, 141,243,192 and -29,062,416, the better approximation would be obtained from 170,305,607 (or 170,305,609) instead of 170,305,605. The ENIAC square rooter had an optional round off mechanism. It left shifted the numerator one more time and then added/subtracted the denominator 5 times from it depending on whether the denominator was previously decremented/incremented. If there is *no sign change* the last position in the doubled root was incremented or decremented by 2. Thus

```
Numerator: -0,290,624,160          Denominator:  0,170,305,609

Numerator:  0,560,903,885  <- sign change!
```

Since there is a sign change, 17,030.5609 is the best approximation for the doubled square root. The decimal point is between the 4th and 5th positions so the doubled root is approximately 17,030.5609.

This method also works for small numbers. Here we demonstrate how the ENIAC would take the square root of 2 an accuracy of 4 digits below the decimal point.

Step 1: Subtract increasing odd multiplies of 10^8 from the numerator until a negative result is obtained. Remember that the ENIAC always increments the denominator by 2 in the p = 9th digit.

```
                                                    P=9
Numerator:  0,000,000,002          Denominator:  0,100,000,000
           -0,099,999,998                        0,300,000,000
```

Step 2: Left shift the numerator and subtract $11*10^7$ from the denominator. Add back decreasing odd multiples of 10^7 to the numerator until a positive result is obtained.

```
                                                    P=8
                                   Denominator:  0,300,000,000
                                       subtract     11
Numerator: -0,999,999,980          Denominator:  0,190,000,000
             ...                                     ...
           0,000,000,020                        -0,010,000,000
```

Step 3: Left shift the numerator and add $11*10^6$ to the denominator. Subtract increasing odd multiples of 10^6 from the numerator until a negative result is obtained.

```
                                                    P=7
                                   Denominator: -0,010,000,000
                                           add      11
Numerator:  0,000,000,200          Denominator:  0,001,000,000
           -0,000,999,800                        0,003,000,000
```

Step 4: Left shift the numerator and subtract $11*10^5$ from the denominator. Add back decreasing odd multiples of 10^5 to the numerator until a positive result is obtained.

```
                                                    P=6
                                   Denominator:  0,003,000,000
                                       subtract      1,1
Numerator: -0,009,998,000          Denominator:  0,001,900,000
             ...                                     ...
           0,000,002,000                        -0,000,100,000
```

Step 5: Left shift the numerator and add $11*10^4$ to the denominator. Subtract increasing odd multiples of 10^4 from the numerator until a negative result is obtained.

```
                                                    P=5
                                   Denominator: -0,000,100,000
                                           add         11
Numerator:  0,000,020,000          Denominator:  0,000,010,000
             ...                                     ...
           -0,000,020,000                        0,000,050,000
```

Step 6: Left shift the numerator and subtract $11*10^3$ from the denominator. Add back decreasing odd multiples of 10^3 to the numerator until a positive result is obtained.

```
                                                    P=4
                                  Denominator:  0,000,050,000
                                      subtract          11
Numerator: -0,000,200,000         Denominator:  0,000,039,000
             ...                                    ...
           0,000,004,000                         0,000,027,000
```

Step 7: Left shift the numerator and add $11*10^2$ to the denominator. Subtract increasing odd multiples of 10^2 from the numerator until a negative result is obtained.

```
                                                    P=3
                                  Denominator: -0,000,027,000
                                          add           1,1
Numerator:  0,000,040,000         Denominator:  0,000,028,100
              ...                                    ...
           -0,000,016,400                        0,000,028,500
```

Step 8: Left shift the numerator and subtract $11*10^1$ from the denominator. Add back decreasing odd multiples of 10^1 to the numerator until a positive result is obtained.

```
                                                    P=2
                                  Denominator:  0,000,028,500
                                      subtract           11
Numerator: -0,000,164,000         Denominator:  0,000,028,390
             ...                                    ...
           0,000,006,040                         0,000,028,270
```

Step 9: Left shift the numerator and add 11 to the denominator. Subtract increasing odd integers from the numerator until a negative result is obtained.

```
                                                    P=1
                                 Denominator:  -0,000,028,270
                                         add              11
Numerator:  0,000,060,400         Denominator:  0,000,028,281
            0,000,032,119                       0,000,028,283
            0,000,003,836                       0,000,028,285
           -0,000,024,449                       0,000,028,287
```

Thus twice the square root of 2 is between 2.8284 and 2.8286 though we use the value 2.8287 for twice $\sqrt{2}$. If we round off by left-shifting the numerator the adding the denominator five times to the numerator we do not get a sign change so subtract two from the denominator and use 2.8285 as our best approximation to $2\times\sqrt{2}$. Note that . 2.8285/2 = 1.41425 agrees favorably with $\sqrt{2} \approx 1.414213562$.

Summary: To my knowledge, only two of the "early" computers ever implemented a square root operation in hardware: Zuse's Z3 and the ENIAC both of which had limited memory and/or programming capabilities. In the "First Draft of the Report on the EDVAC" written in 1945, von Neumann incorporates the design for a square rooter (which von Neumann notes is similar to the "divider network"). Yet in his later 1946 paper "Preliminary discussion of the logical design of an electronic computing instrument" which von Neumann authored with Arthur Burks and Herman Goldstine, he leaves out the design of a square rooter "because such a device would involve more equipment than we feel desirable in first model". Hardware was expensive in the early computers and designs reflected a "keep it simple" philosophy. (I should point out that neither the EDVAC or the EDSAC, computers which were heavily influenced by the "First Draft" paper, had a square root operation. And neither did the IAS machine which was influenced by the "Preliminary discussion" paper or the EDSAC at Cambridge University. Besides, the increased memory capacity and the flexibility of programming found in later stored program computers allowed square roots to be easily done in software. The Z3 and ENIAC were not stored program computers.

So given that square roots are *hard* to compute (how many of us can take one by hand?), how did a relatively primitive computer like the ENIAC take a square root? The ENIAC implemented a well known square root algorithm that could be carried out by anyone with desk calculator with addition, subtraction, and shift capabilities. Since the ENIAC was very good at addition and subtraction (it had 20 accumulators that could do both at electronic speeds), taking a square root turned out just to be a matter of sequencing their operations in the correct way.

Additional Links

1946 Technical Report on The ENIAC: http://ftp.arl.army.mil/~mike/comphist/46eniac-report
This is the first four chapters of the June 1, 1946 Report on the ENIAC. Excellent primary source material from the U.S. Army Research Laboratory at the Aberdeen Proving Grounds, MD.

History of Computing Information: http://ftp.arl.army.mil/~mike/comphist: web page with other links to ENIAC materials also maintained at the ARL at Aberdeen

---

References

ENIAC: The Triumphs and Tragedies of the World's First Computer; Scott McCartney: This is an excellent general history of the ENIAC

From ENIAC to UNIVAC: An Appraisal of the Early Eckert-Mauchly Computers, Nancy Stern: Another excellent general history of the ENIAC.

"The ENIAC: History, Operation, and Reconstruction in VLSI" by Jan Van der Spiegel, James F. Tau, Titiimaea F. Ala'ilima Lin Ping Ang; from *The First Computers: History and Architectures*

edited by R. Rojas and U. Hashagen: This paper contains many of the technical details of the ENIAC. The square root algorithm was obtained from this source.
