Calculate Square Root without a Calculator

Thumrongsak Kosiyatrakul

CS 0447 — Computer Organization & Assembly Language

Finding the Square Root of a Real Number

The square root of a real number can be calculated using an algorithm that resembles a long division algorithm. The following is an algorithm to find the square root of 10.5625. Note that this algorithm can be used to find square root of any real numbers.

The first step is to separate a real number into groups of two digits starting at the decimal point in both directions. In case of 10.5625, we get three groups 10, 56, and 25. If a group does not have 2 digits, simply add 0 to the left if the group is on the left of the decimal point or add 0 to the right if the group is on the right of the decimal point. For example, if th real number is 123.456, in this case, get four groups, 01, 23, 45, and 60. For this algorithm, we have to maintain two values, the current result and the current remainder. These two values are initialized to 0.

To find the square root of a real number, perform the following steps until the remainder becomes 0 again or until you satisfy with the precision of your result:

1. Pull the left-most unused group (of two digits) and put it to the right of the current remainder. The result is a new current remainder. For example, if the current remainder is 6 and the left-most unused group is 39, the new current remainder is 639. Note that mathematically, this step is equivalent to

$$\texttt{current\_remainder} = (\texttt{current\_remainder} * 100) + \texttt{the\_left-most\_unused\_group}$$

2. Multiply the current result by 2 and again by 10 and put it aside. Let's call this value `temp`.

3. Find the largest integer x $(0 \leq x \leq 9)$ such that

$$(\texttt{temp} + x) * x \leq \texttt{current\_remainder}$$

4. Subtract the current remainder by $(\texttt{temp} + x) * x$ and the result is a new current remainder. In other words,

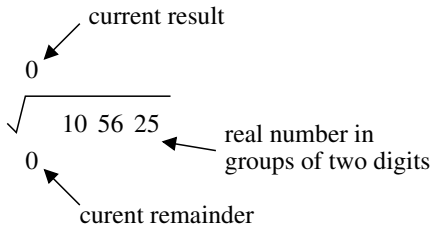
$$\texttt{current\_remainder} = \texttt{current\_remainder} - ((\texttt{temp} + x) * x)$$

5. Put the value x from step 3 to the right of the current result and this becomes a new current result. Mathematically, this step is equivalent to

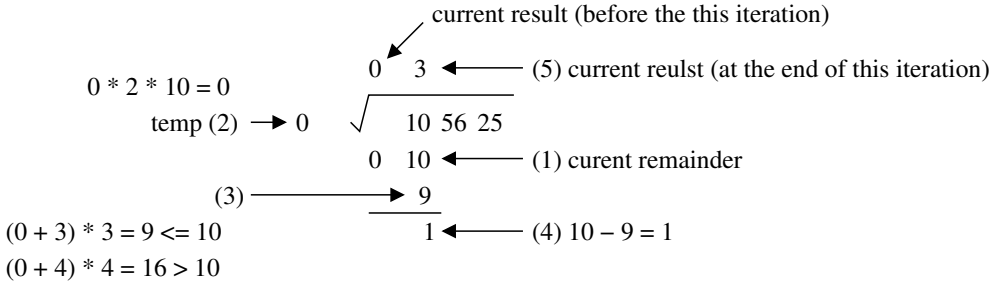
$$\texttt{current\_result} = (\texttt{current\_result} * 10) + x$$

6. Go back to step 1.

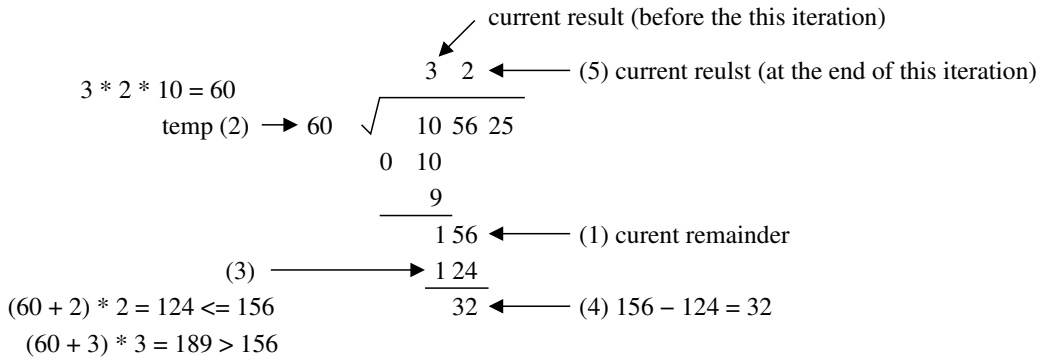
Let's try to find the square root of 10.5626 from the beginning for better understanding of the algorithm. First, separate 10.5625 into groups of two digits (starting from the decimal point in both direction), and initialize the current result and the current remainder to 0 as shown below:



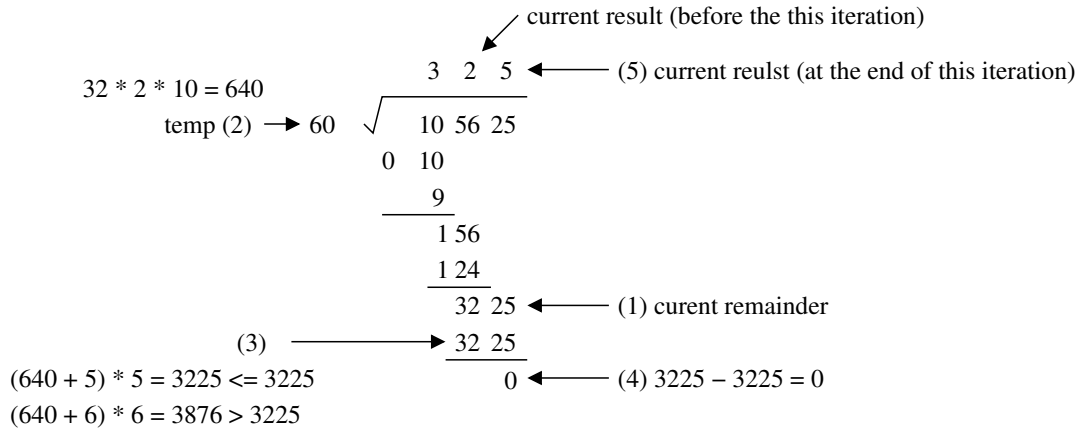
The following show the the first iteration of this algorithm:



After the second iteration:

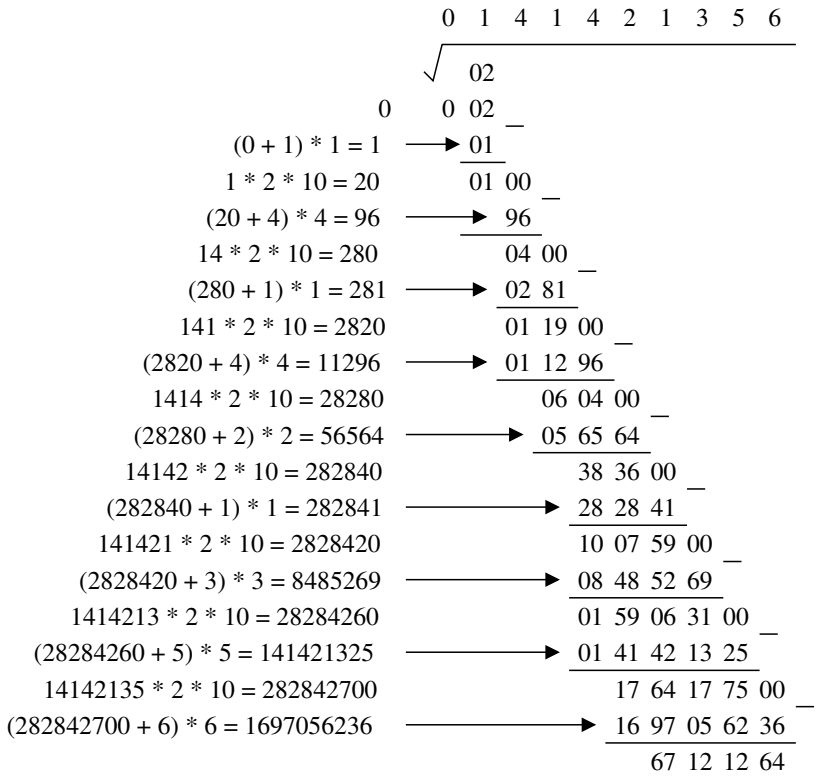


After the third iteration is shown below:



Note that the third iteration is the last iteration since the current remainder becomes 0. Since the decimal point is in between 10 and 56, the square root of 10.5626 is 3.25.

The following shows the process of finding the square root of 2 (after 9 iterations) using the algorithm discussed in class:



The above algorithm shows that the square root of two is approximately 1.41421356. Note that since the square root of 2 is an irrational number, the algorithm will not stop. Therefore, you have to stop when you satisfy with the precision of your result.

Finding the Square Root of a Positive Integer in Binary

As we discussed in class, any algorithms that work with decimal numbers, they should work with binary numbers. It is true in the case of finding the square root as well. Technically, it is simpler to find a square root of a number in binary than in decimal. Since the different between decimal and binary is the base (10 vs 2). Any 10^x in the algorithm become 2^x. Note that multiply a number in binary by 2^x is the same as shift the number left x times.

You need separate the number into groups of 2 bits and initialize the current result and the current remainder to 0. **Do not forget that every bit is either 0 or 1**. Thus, the algorithm to find the square root of a positive integer in binary is as follows:

1. Pull the left-most unused group (of two bits) and put it to the right of the current remainder. The result is a new current remainder. For example, if the current remainder is 1 and the left-most unused group is 01, the new current remainder is 101. Note that mathematically, this step is equivalent to

$$\texttt{current\_remainder} = (\texttt{current\_remainder} * 4) + \texttt{the\_left-most\_unused\_group}$$

2. Multiply the current result by 2 and again by 2 and put it aside. Let's call this value `temp`.

3. Find the largest integer x ($0 \leq x \leq 1$) such that

$$(\texttt{temp} + x) * x \leq \texttt{current\_remainder}$$

   Note that this step is much simpler in binary. Since a bit in binary is either 0 or 1. So, if x is 1, $(\texttt{temp} + x) * x$ is simply $\texttt{temp} + x$. So, you only need to compare $\texttt{temp} + x$ and the current remainder.

4. Subtract the current remainder by $(\texttt{temp} + x) * x$ and the result is a new current remainder. In other words,

$$\texttt{current\_remainder} = \texttt{current\_remainder} - ((\texttt{temp} + x) * x)$$

5. Put the value x from step 3 to the right of the current result and this becomes a new current result. Mathematically, this step is equivalent to

$$\texttt{current\_result} = (\texttt{current\_result} * 2) + x$$

6. Go back to step 1.

Recall that Q8.8 format of a floating-point number x is $x \times 256$ and rounded to the closest integer. Thus, you only need to calculate the square root of a positive integer number. The following is an example of finding the square root of 2.0 where 2.0 is represented in Q8.8 format ($2.0 * 256 = 512_{10} = 0000000100000000_2$).

```
                                  0  0  0  0  1  0  1  1  0  1  0  1  0
                                ---------------------------------------
                               | 00 00 00 10 00 00 00 00
0 << 2 = 0                     | 00 00
(0 + 0) * 0 = 0                | 00 00
                               |---------
0 << 2 = 0                     |    00 00
(0 + 0) * 0 = 0                |    00 00
                               |---------
0 << 2 = 0                     |       00 00
(0 + 0) * 0 = 0                |       00 00
                               |------------
0 << 2 = 0                     |          00 10
(0 + 1) * 1 = 1                |          00 01
                               |---------------
1 << 2 = 100                   |             01 00
(100 + 0) * 0 = 0              |             00 00
                               |---------------
10 << 2 = 1000                 |             01 00 00
(1000 + 1) * 1 = 1001          |                10 01
                               |------------------
101 << 2 = 10100               |                01 11 00
(10100 + 1) * 1 = 10101        |                01 01 01
                               |------------------
1011 << 2 = 101100             |                   01 11 00
(101100 + 0) * 0 = 0           |                   00 00 00
                               |---------------------
10110 << 2 = 1011000           |                   01 11 00 00
(1011000 + 1) * 1 = 1011001    |                   01 01 10 01
                               |---------------------
101101 << 2 = 10110100         |                      01 01 11 00
(10110100 + 0) * 0 = 0         |                      00 00 00 00
                               |------------------------
1011010 << 2 = 101101000       |                      01 01 11 00 00
(101101000 + 1) * 1 = 101101000|                      01 01 10 10 00
                               |------------------------
10110101 << 2 = 1011010100     |                      00 00 00 10 00 00
(1011010100 + 0) * 0 = 0       |                      00 00 00 00 00 00
                               |------------------------
                               |                            10 00 00
```

The result is $101101010_2 = 362_{10}$ which is the Q8.8 format of $362/256.0 = 1.4140625 \approx \sqrt{2}$. As we discussed in class, a square root of a Q8.8 format is a Q4.4 format. That is why the above algorithm has to continue for four more iteration to get Q4.8 format.

Note that the algorithm should stop as soon as the current remainder becomes 0. However, for simplicity, we can let the computer computes exactly 12 iterations every time without checking whether the remainder is already 0.