Episode 3.01 – Adding and Subtracting Ones and Zeros

Welcome to the Geek Author series on Computer Organization and Design Fundamentals. I'm David Tarnoff, and in this series we are working our way through the topics of Computer Organization, Computer Architecture, Digital Design, and Embedded System Design. If you're interested in the inner workings of a computer, then you're in the right place. The only background you'll need for this series is an understanding of integer math, and if possible, a little experience with a programming language such as Java.

Representing numbers with bits is one thing. Doing something with them is an entirely different matter. So let's discuss some of the basic mathematical operations that computers perform with binary numbers. It may seem like a waste of time to dedicate an episode to something so trivial, but a clear understanding of binary arithmetic will lay the foundation for new binary representations that support negative and floating-point values. It will also provide background for a later episode when we design the circuitry to perform addition. And for those of us who program, an understanding of these operations will help us understand the consequences of the variable types we select and how they affect performance.

Let's start with addition. Regardless of the numbering system used, the addition of two multi-digit numbers is performed using a column-by-column addition of individual digits from right to left. For example, in order to add decimal 283 to decimal 545, we begin by adding the ones column digits 3 and 5 to get 8 for the ones column of the result. Since the result of 8 is a single decimal digit, this operation can be contained in a single column.

The same cannot be said of the tens column when adding 283 to 545. Eight plus 4 is 12, which requires two decimal digits for the result. Since the result won't fit in a single digit, we have to do something with the overflow. The next highest column represents magnitudes that are ten times that of the current column. Therefore, we can take ten from the addition result of 12 and add it as carry of one to the column immediately to the left. Another way of saying this is that 8 plus 4 generates a 2 in the tens column of the result and a carry of one into the hundreds column.

Binary addition works the same way except that we're limited to two digits. Three of the addition operations, zero plus zero, zero plus one, and one plus zero, result in a single digit of zero or one and will not generate a carry. The fourth case, one plus one, results in a decimal 2, a digit which does not exist in binary. In this case, we need to create a carry or overflow that will go to the next column.

In binary, the column to the left equals two times the current column. Just as we did with decimal, we subtract one instance of the next highest bit position from our result. In the case of one plus one equals two, we translate a magnitude of two into a carry to the next column. Therefore, one plus one in binary equals zero with a carry of one. Another way of looking at it is that one plus one equals a binary 10_2. Going back to our decimal addition of 283 to 545, the carry generated from the tens column makes our addition in the hundreds column 1 plus 2 plus 5, which equals 8. This means that in some columns, it might be necessary to add three digits together: the two original digits and a carry from the column immediately to the right. This means that the highest possible result from the addition of two decimal digits along with a potential carry from the column to the right is nineteen. Since nineteen is the largest possible sum, then the largest carry to the next highest position will always be one.

In binary, accounting for a potential carry from the right adds four new cases to the original four presented earlier: 1+0+0, 1+0+1, 1+1+0, or 1+1+1.

The first case results in a one with no carry to the next column. The second and third cases are similar to the case where two ones are added together to get a result of zero with a carry. The last case, however, has three ones added together which equals a decimal 3. Subtracting 2 from this result places a new result of one in the current column and sends a carry to the next column. And just as in decimal addition, the carry in binary is never greater than one.

Let's try a quick example with four bits. In decimal, we know that 6 plus 7 equals 13. In four bit binary, 6 is equivalent to 0110_2 and 7 is equivalent to 0111_2. As with decimal addition, we start with the right most or least significant bit position, in other words, the ones place. Six has a zero in the ones position while 7 has a one in the ones position. We know that zero plus one equals one with no carry to the next column. Therefore, our result has a one in the ones position.

Both 6 and 7 have ones in the twos position. Since there is no carry from the ones position, the twos position of the result is found by adding one plus one. This gives us a result of zero with a carry of one to the fours column.

In the fours position, 6 has a one and 7 has a one. One plus one plus the carry of one from the twos column gives us a result of one in the fours column and a carry of one to the eights column.
Lastly, both 6 and 7 have zeros in the eights position. Adding these two zeros to the carry of one from the fours column gives us one for the eights position of the result. If we put the bits of the result together, we see that a binary 0110_2 (6) added to a 0111_2 (7) equals 1101_2. Converting 1101_2 to decimal gives us 13, which fortunately is the result of adding 6 to 7.

Now let's move to subtraction. Just as with addition, the process of subtraction works the same in the binary numbering system as it does in the decimal. Starting at the ones column, the magnitude of the subtrahend digit is subtracted from the corresponding digit of the minuend to generate a difference for that column. If the minuend's digit is smaller than the subtrahend's digit, then a borrow equivalent to the base is taken from the next highest digit of the minuend. This is repeated column-by-column from right to left until the subtraction is complete. Wow, hard to believe that a process so simple could be made to sound so complicated.

When it comes to binary, there are four possible scenarios when subtracting one binary digit from another: zero minus zero, zero minus one, one minus zero, and one minus one. As long as the minuend is greater than or equal to the subtrahend, the process is confined to a single column. Zero minus zero is zero, one minus zero is one, and one minus one is zero. The only time a borrow is required in binary is when we need to subtract one from zero. In that case, a one is borrowed from the column immediately to the left, which changes the zero of the minuend to a binary 10_2 or a decimal 2. And just like decimal, if the digit of the minuend immediately to the left is zero, we move left to the next digit position until we find a one to borrow, then propagate that borrow back down to the minuend that needed it.

Now let's see how this works with another four-bit example. Twelve minus 9 equals 3, right? In four-bit binary, 12 is equivalent to 1100_2 and 9 is equivalent to 1001_2. We begin with the least significant bit. In the ones place, 12 contains a 0 and 9 contains a 1. We need to borrow from the next highest column for this case. It turns out that the twos column of the binary 12 also has a zero in it. That means we need to go to the fours column to check for a borrow. Fortunately, 12 has a 1 in the fours column. We borrow that one to make the twos column 10_2 or a decimal two. One is borrowed from the twos column to make the ones column 10_2 or a decimal two. Now we can compute the least significant bit of our result: two minus one is one.

In the twos column, the minuend digit is now a one after all of the borrowing, and the subtrahend of 9 has a zero in that column. That means that the twos position of our result contains one minus zero, which equals one. In the fours column, the minuend digit is now a zero after the borrowing for the ones column, and the subtrahend's digit in the fours column is zero. Zero minus zero gives us zero for the fours digit for the result. Lastly, both the minuend and the subtrahend have ones in the eights column meaning that the eights digit of our result is one minus one or zero. If we put the bits of the result together, we see that a binary 1100_2 minus 1001_2 equals 0011_2 or a decimal three.

Okay, so this episode was rudimentary. It was presented here, however, in order to lay the foundation for new binary representations where some of the patterns of ones and zeros will be reserved for negative numbers and to present a clear-cut definition of addition that we will be using later to replicate in hardware.

For transcripts, links, or other podcast notes, please check us out at intermation.com where you will also find links to our Instagram, Twitter, Facebook, and Pinterest pages. Until then remember that while the scope of what makes a computer is immense, it's all just ones and zeros.