Binary Arithmetic

Computers Arithmetic

Computer do not perform arithmetic in quite the same way we might using paper and pencil. For computers, the size of the number, the number of digits the computer can use is fixed by the machine's architecture. In other words, computers work with fixed-size numbers. In essence, computers work with integers. More complex arithmetic either requires special hardware or software. Arithmetic that works with fixed-size numbers is called ***finite-precision*** mathematics. The size of the fixed-size numbers in a computer is usually 8, 16, 32 or 64 bits.

Using decimal lets examine why finite-precision can be a problem. Imagine if you could only work with numbers limited to four(4) digits. In decimal this would mean that the largest number you could use would be 9999. Anything bigger simply doesn't fit into the four spaces we have.

Even with simple addition, problems arise almost immediately.

Try adding these numbers 5000 + 2500 = 7500. This seems to work just fine.

Now try adding 5000 + 2500 + 3000. All three numbers are perfectly valid numbers in our four digit system. However:

5000 + 2500 + 3000 = **10500** This is a five digit number. It will not fit in four digits. The excess digit, the leftmost digit (in this case a 1) is called ***overflow*** and does not get stored in our limited four digits. This causes a lot of mathematical problems.

This situation is exactly what occurs in all computers. Every computer, including super-computers are finite-precision machines. And remember, since computers think in binary (base 2) while 32 or 64 bits may look and sound like a large number, it still is very limited. The largest 8-bit number for example $(1111\ 1111)_2$ is the equivalent of only 255.

It is impossible to predict in advance every possible situation where overflow might occur, because it is impossible to predict how a given computer system will be used. Modern computer systems check for overflow so programmers can know when or if it occurs and design their programs accordingly.

Things get even more complex when we realize that not all integers are positive. Computer designers and programmers have to worry about negative integers as well. First for simple addition.

Decimal Addition:

Let's review Decimal addition first.

Add the following two numbers:

```
   2 0 5 3
+    2 3 4
-------------
   2 2 8 7
```

No problem. Now look at this example:

```
     1 1
   1 3 7 4
+    1 5 8
--------------
   1 5 3 2
```

This required a far more complex set of mathematical skills.

Binary Arithmetic

Binary Addition

Because binary numbers use only two different digits 1 and 0, the carry in addition occurs very often.

```
  1 0
+  0 1
--------
  1 1       (no carry)
```

```
   1
  0 1
+ 0 1
--------
  1 0      (1+1 = 0 carry 1 to the next column, so 1+1=10)
```

But,

```
   1
  1 1
+ 0 1
--------
1 0 0       (1 + 1 = 0 carry 1 to next column, then again 1 + 1 is 0
                 carry 1 to next column)
```

There are only two ***carry*** combinations
 1 + 1 = 10
 1 + 1 + 1 = 11

```
     1
  1 0 1
+ 0 0 1
----------
  1 1 0     (1 + 1 = 0 carry 1 to next column
```

Now try this:

```
  1   1
  1 0 1
+ 0 1 1
----------
1 0 0 0     (all columns involve carry)
```

And this,

```
     1  1
    1 1 1
 +  0 1 1
-------------
  1 0 1 0   (all columns involve carry, remember 1+1+1 = 1 carry 1 to next column)
```

Binary Arithmetic

Try these examples:

```
   1 0 1 1 0 1 0          1 1 0 0 0 1 1          1 1 0 1 0 1 1
 + 0 1 1 0 0 0 0        + 0 1 1 1 1 0 0        +   1 1 1 0 1 1
 ---------------        ---------------        ---------------
```

```
    1 1                   1                      1 1 1   1 1
   1 0 1 1 0 1 0          1 1 0 0 0 1 1          1 1 0 1 0 1 1
 + 0 1 1 0 0 0 0        + 0 1 1 1 1 0 0        +   1 1 1 0 1 1
 ---------------        ---------------        ---------------
 1 0 0 0 1 0 1 0        1 0 0 1 1 1 1 1        1 0 1 0 0 1 1 0
```

It doesn't matter how many digits (bits) there are in the binary number the rules are the same.
Remember:
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10
1 + 1 + 1 = 11

Another, longer example: It doesn't matter how many bits are involved the process is the same.
If we have a 16 bit number the operations are the same.

```
         1 1 1   1 1               1 1
       1 0 0 1  1 0 1 0  0 0 1 1  0 0 0 1
 +          1 0  0 1 1 1  1 0 0 0  0 0 1 1
 ------------------------------------------
       1 1 0 0  0 0 0 1  1 0 1 1  0 1 0 0
```

Now try these examples: (results on the next page)

a)
```
   1011 0001
 +  111 1100
 -----------
```

b)
```
   1010 1100 1111
 +      1 0000 0101
 -----------------
```

c)
```
   1100 1100 1100
 +   11 1010  0100
 -----------------
```

d)
```
   1111 0000 1111
 +   10 1010  0101
 -----------------
```

Binary Arithmetic

a) 1011 0001 + 111 1100 = 1 0010 1101

b) 1010 1100 1111 + 1 0000 0101 = 1011 1101 0100

c) 1100 1100 1100 + 11 1010 0100 = 1 0000 0111 0000

d) 1111 0000 1111 + 10 1010 0101 = 1 0001 1011 0100