Episode 3.05 – Introduction to Offset or Biased Notation

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. And one more thing. We're going to be doing a bit of figuring in this episode, so it might help to keep a pencil and paper handy.

Over the past few episodes, we've been discussing the binary representation of signed integers by examining additive complements. Twos complement representation allows us to represent signed integers using a fixed number of bits. It fully supports addition for any combination of positive and negative values, and even allows us to subtract values by switching the sign of the subtrahend and sending it to the addition circuitry.

Twos complement has a drawback, however. If the patterns of ones and zeros were sorted from the smallest pattern of all zeros to the highest pattern of all ones, the values of the integers they represent would not be in order. Since the most significant bit of a twos complement value, in other words, the sign bit, is a one for negative values and a zero for positive values, sorting the patterns of ones and zeros like this would place the negative values above the positive ones. For example, in four-bit twos complement representation, positive five is 0101 and negative five is 1011. In the order of four-bit patterns, 1011 comes after 0101.

In computing, this lexicographical order is important. There are times when computers must compare or sort values. The circuitry to do this is simple if the patterns of ones and zeros are ordered from all zeros representing the lowest value to all ones representing the highest. To compare two binary values, we start with the left-most bit, and moving left to right, look for the first pair of bits where one of the bits is a one and the other is a zero. The pattern containing the one is the larger of the two values.
This brings us to biased notation. Biased notation takes the ordered patterns of ones and zeros and shifts them along the integer number line so that the pattern of all zeros is equivalent to the offset or bias. Before doing any conversions, we need to define a bias, sometimes identified as K. This means that the pattern of all zeros in biased notation using an offset of K is negative K. The remaining values increment up from negative K.

The bias, K, is typically half of the total number of unsigned values. For example, a typical bias for 8 bits is half of two to the eighth power, which equals two to the seventh or 128. Let's do a couple of conversions using this bias starting with 00000000. Remember that the pattern of all zeros equals the negative of the bias, which in this case means 00000000 equals -128.

What about some other patterns? Well, let's go to the other end of the range. In unsigned binary, 11111111 equals 255. Using biased notation with our bias of 128, 11111111 equals 255 – 128 or 127. That means the range of integers we can represent with eight-bit biased notation with a bias of 128 is from -128 to 127.

Using this same notation, what does 01001010 equal? In unsigned binary, 01001010 equals 64 + 8 + 2 or 74. That means 01001010 in our biased notation is 74 – 128 or -54.

Going the other way, from a decimal integer to biased notation, means we have to first add the bias, then convert the new value using the method for unsigned binary. For example, let's identify the eight-bit biased notation with an offset of 128 for the decimal value 42. First, adding 42 to 128 gives us 170. Breaking 170 into its powers of two gives us 128 + 32 + 8 + 2. This gives us a binary pattern of 10101010. Now let's talk about the bias, which we've represented with K. As we said earlier, a typical bias is half of the total number of possible patterns of ones and zeros for that number of bits. In other words, for n bits, a typical bias would be 2^{n-1}. Using this bias makes it so that a decimal zero and all of the positive integers above it are represented with a binary pattern where the most significant bit is set to one. All negative numbers have a most significant bit of zero. By using this bias, it is easy to convert to twos complement notation by simply inverting the most significant bit.

For example, we found earlier that 01001010 in eight-bit biased notation with a bias of 128 represents -54. Flip the most significant bit, and we get 11001010, which is -54 in eight-bit twos complement representation. We also determined earlier that 42 in eight-bit biased notation with a bias of 128 was 10101010. Flip the most significant bit, and we get 00101010, which is 42 in twos complement notation. Unfortunately, not all n-bit biased notations use 2^{n-1} as their bias. In a later episode, we will see that IEEE-754, a standard binary format for representing floating point values, uses a bias of one less than 2^{n-1} as its bias.

And one last thing before we go. Way back in Episode 2.5, we discussed how computers represent analog signals using patterns of ones and zeros. It turns out that this is really a form of biased notation. Remember that the analog-to-digital converter maps the range of possible analog values to the range of unsigned binary values with the minimum analog value mapping to the all zeros pattern. This means that the minimum analog value is really a bias.

In due course, we will get to the topic of representing floating point values in binary. This episode has covered one of the critical components used to do that. In our next episode, we're going to cover the other: representing fractions with binary.

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.