Binary Sign Extension

A quick post today, explaining the concept of binary sign extension. Imagine the scenario. You have been using a single byte to store values and now want to extend this to store larger numbers? What would you have to do to extend your original 1-byte of data to now store numbers using 2-bytes? This is where binary sign extension comes in.

In this post, we're going to look at two scenarios where sign extension is useful. The first, where your original number is an unsigned binary number and the second where your existing number is a signed binary number.

Extending Unsigned Binary Numbers

The approach to extending an unsigned binary number is pretty simple. It's a single-step process:

> "To extend an unsigned binary number, take the bits from the original number and append as many additional bits of storage as are necessary to the most significant end of the original number. The value of each appended bit is set to zero."

Just to be clear, let's look at an example. Imagine we have a number 01010101_2 and wanted to extend it to use two bytes of storage instead of just one. In line with our rule above, we append an additional 8 bits / 1 byte of storage to the most significant end of the number. Our resulting number is therefore: **00000000** 01010101_2 (The appended bits are in bold). Easy huh? But what about signed binary numbers?

Extending a Signed Binary Number

As with extending unsigned binary numbers, the rules for extending a signed binary number are again simple, though with signed binary numbers there is a slight difference to the rule in order to preserve the sign bit of our original number:

> "When extending a signed binary number represented in Two's Complement, additional bits are appended to the most significant end of the number being extended. When appending these bits, each bit is set to the same state as the the most significant bit (the sign bit) of the number being extended."

Again, it's probably easier to work through an example.

Imagine this time we start with an 8-bit signed number stored in Two's Complement format: **1**010 1100_2 (or -84_{10}) and we want to extend it to use 16-bits / 2-bytes of storage instead of just 8-bits / 1-byte.

As we do when we extend unsigned binary numbers we start by appending an additional 8-bits to the most significant side of our number

(the left hand end).

This time though, instead of setting all the bits to zero, each of these new bits is set to the same value as the sign-bit of the number we're extending (which may be zero for positive numbers or 1 for negative numbers).

In this case the sign bit of the number is 1 so we set all the new bits we append to 1 as well.

Our new result is therefore: **1111 1111** 1010 1100_2 (the bits in bold are the new ones we've added).

As a side note, this new number remains a Two's Complement representation but notice how the sign bit is exactly the same sign as the original we had. In addition, if you walk through the steps you learnt in the previous post (about how to convert a Two's Complement number into decimal), you should also see that this new number still equals -84_{10} (which is bonus!).

Image credit: http://flic.kr/p/8F18eQ

March 14, 2014
Filed Under: **Computer Science**
Tagged With: **Binary**, **Computer Science**