Bittle

A little bit-manipulation library that focuses on offering compiler independent behavior for common bit operations.

This library was written using C11 and compiled using gcc.
Tests created using the cmocka test harness and built using Cmake.

Library Architecture

How to include the Library

This library is a single header library and is therefore completely self contained within the header file **“bittle.h”**

Macro Notes

Many functions in this library come in 4 different variants depending on the desired length you want your operations truncated to.

Some compilers choose to bit extend any bit manipulations operations to the largest width primitive type on the target architecture. This can cause operations that would normally result in a 0 when all bits are shifted out, to return values that would not logically result from these operations.

functions with a (n)b suffix will adhere to the width designated by the number of bits proceeding the b.

All macros of a particular bit width are grouped together inside an **#ifdef** block that will only evaluate to true if that data width is compilable on the target platform. see THIS link for further information

Macros

`LS_(n)b (value, num_shifts) | Logical Left Shift`

These macros will shift the value given by num_shifts left while also adhering to the width denoted by n.

`LRS_(n)b (value, num_shifts) | Logical Right Shift`

These macros will shift the value given by num_shifts right while concatenating a 0 to the leftmost bit while
also adhering to the width denoted by n.

`ARS_(n)b(number, num_shifts) | Arithmetic Right Shift`

These macros will shift the bits to the right while also shifting in the proper signed bit depending
on the sign of the original value used

`RR_(n)b(number, num_shifts) | Rotate Right`

These macros will rotate the bits in the given value by concatenating any bits that are shifted out right into
the most significant bit place.

It’s important to know that when the macro for a bit width *smaller* than the input number is called, that only the the least significant *n* will be rotated

`RL_(n)b(number, num_shifts) | Rotate Left`

These macros will rotate the bits in the given value by concatenating any bits that are shifted out left into
the least significant bit place.

It's important to know that when the macro for a bit width *smaller* than the input number is called, that only the the least significant *n* will be rotated

`CountSetBits_(n)b(number, countVariable) | Count Number of Set Bits`

These macros will count the number of bits set for ***n*** least significant bits in the given value.

The count is stored in the variable provided as countVariable in the macro.

countVariable value is not reset to 0 and the bit count will be added to whatever value it had before the macro

Compiling the tests

In order to compile and run the tests you must have several components installed.

- You must have cmake installed in order to compile the tests
- You must have cmocka version 1.1.0 or later installed.

If compiling on windows there are 2 .dll files need to run the tests. These files are included with the source and will be automatically copied into the executable directory if not present there already.

When all of these things are present simply run "**make check**" in the outermost directory and the tests should automatically build and run.