Bit Manipulation

Bit-masking is a technique to selectively modify individual bits without affecting other bits.

Bit SET

To set a bit, we need to use the OR operator. This is just like an OR logical gate you should've learned in your Digital Design course.

```
 // We want to set Bit #7 of a variable called: REG
REG = REG | 0x80;
// Let's set bit #31:
REG = REG | 0x80000000;
// Here is an easier way to write these:
// (1 << 31) means 1 gets shifted left 31 times to produce 0x80000000
REG = REG | (1 << 31);
// Simplify further:
REG |= (1 << 31);
// Set Bit #21 and Bit #23 at the same timeREG |= (1 << 21) | (1 << 23);
```

Bit CLEAR

To set a bit to 0, in other words reset or clear a bit, the logic is similar, but instead of **ORing** a bit, we will an **AND** function to clear. **Note: that ANDing something with 0 clears it and ANDing something with a 1 does not change it. The tilde (~) operator can help us invert the bits of a value in the following examples:**

```
// Assume we want to reset Bit#7 of a register called: REG
REG = REG &   0x7F;
REG = REG & ~(0x80); // Same thing as above, but using ~ is easier
// Let's reset bit#31:
REG = REG & ~(0x80000000);
// Let's show you the easier way:
REG = REG & ~(1 << 31);
// Simplify further:
```

```
REG &= ~(1 << 31);
// Reset Bit#21 and Bit# 23:REG &= ~( (1 << 21) | (1 << 23) );
```

Bit TOGGLE

```
 // Using XOR operator to toggle 5th bitREG ^= (1 << 5);
```

Bit CHECK

Suppose you want to check bit 7 of a register is set:

```
bool check_bit = REG & (1 << 7);
if(check_bit)
{
 DoAThing()}
```

Now let's work through another example in which we want to wait until bit#9 is 0:

```
// One way:
while(REG & (1 << 9) != 0)
{
  continue;
}
// Another way:
while(REG & (1 << 9))
{
  continue;}
```

Multi-Bit Insertion

```
// Insert a set of continguous bits into a target value.
// Value within target is unknown. This is shown using X's
//
// target   =        0xXXXX'XXXX
//                        ^
//                       /
```

```
//                      /
// value   = 0xABCD --+
// position = 16
// width    = 16
//
// return   =        0xABCD'XXXX
// First you must clear the bits in that location
target &= ~(0xFFFF << 16);
// Now that there are only 0s from position 16 to 31, ew
// can OR those bits with our own set of 1s.target |=  (0xABCD << 16);
```

Multi-Bit Extraction

```
/// Extract a set of contiguous bits from a target value.
///
/// target   =        0x00FE'DCBA
///                            ^
///                           /
///                          /
/// value  = 4 -----------+
/// width    = 8
///
/// return   = 0xCB
// Shift target to the left by 4 to make the 0th bit the start of the bits you want to extract.
// Store the result in to a local variable
uint32_t result = target >> 4;
// Since we only want 8 bits from the result, we need to clear away the rest of the bits from
// the original target.
// AND the result with 0xFF, to clear everything except for the first 8 bits.
result = result & 0xFF;
```

Revision #2
Created 5 years ago by Khalil Estell
Updated 5 years ago by Khalil Estell