ESS: Exercise Set 2

C Language

Question 1:
Write a routine to count the number of bits that are set in a 32 bit unsigned int.

Solution:

Like the assembler version, we can simply do a shift and determine whether the LSB is set. This has a worst case run time proportional to the word-length.

```
uint8_t loop_bitcount (uint32_t n)
{
    uint8_t count=0;
    // loops while n is greater than zero
    while (n > 0)
    {
        count += (n & 0x01); // value of 1 if lsb is set
        n >>= 1;         // right shift n (equivalent to divide by 2)
    }
    return count;
}
```

There is a divide and conquer approach, which is very elegant and about 5 times faster.

Question 2:
Write a routine to compute the nth Fibonacci number, up to the 30th.

Solution:

Iterative implementation (recursion is banned!).

```
uint32_t fib (uint8_t n)
{
  // check bounds
  assert(n<30);
  // setup initial state
  uint32_t z_2 = 0; // prior prior state
  uint32_t z_1 = 1; // prior state
  uint32_t z_0;         // current state
  uint8_t k;
  if (n == 0)
  {
    return 0;
  }
  if (n == 1)
  {
    return 1;
  }
  for (k = 2; k < n; k++)
  {
    z_0 = z_1 + z_2;
    // delay
    z_2 = z_1;
    z_1 = z_0;
  }
  return z_0;
}
```

Question 3:

Write functions to set and clear the n^{th} bit of a 16 bit value. As input, take the bit to set (0 to 15) and the original value. Return the new value.

⋆ Write two more functions, one to toggle a particular bit (i.e. flip it) and one to return the value of a particular bit position.

Solution:

The solution to this is to first create the bit pattern we need to set/clear the bit in question. To set the bit, we simply shift `0x01` n times to the left. We then perform a logical OR to set the bit and leave the rest of the register unchanged. To clear the bit, we do the same, invert it and then use logical AND to clear that particular bit position. These can also be expressed as macros instead of full functions.

```
// Set bit n of a register
// @param value: input value
// @param bit: bit to set
// @return new value
uint16_t bit_set (uint16_t value, uint8_t bit)
{
  // bounds checking?
  assert(bit < 16);
  value |= (0x01 << bit);
  return value;
}

// Clear bit n of a register
// @param value: input value
// @param bit: bit to clear
// @return new value
uint16_t bit_clear (uint16_t value, uint8_t bit)
```

```
{
  // bounds checking?
  assert(bit < 16);
  value &= ~(0x01 << bit);
  return value;
}
```

⋆ To toggle, use XOR (^), and to retrieve the n-th bit, shift and mask with $0x01$.

```
// Toggle bit n of a register
// @param value: input value
// @param bit: bit to toggle
// @return new value
uint16_t bit_toggle (uint16_t value, uint8_t bit)
{
  // bounds checking?
  assert(bit < 16);
  value ^= (0x01 << bit);
  return value;
}

// Clear bit n of a register
// @param value: input value
// @param bit: bit to clear
// @return 1 if bit is set, 0 if bit is clear
uint16_t bit_get (uint16_t value, uint8_t bit)
{
  // bounds checking?
  assert(bit < 16);
  if (value & (0x01 << bit))
  {
    return 1;
  }
  return 0;
}
```