Solutions to Assignment 3
Number Systems and Logical Operators

Question 1

The answers below show the decimal, binary and hexadecimal values. Spaces have been inserted into the binary values to make them easier to read.

1. `8 = 1000 = 0x8`
2. `7 = 111 = 0x7`
3. `16 = 1 0000 = 0x10`
4. `15 = 1111 = 0xf`
5. `256 = 1 0000 0000 = 0x100`
6. `255 = 1111 1111 = 0xff`
7. `237 = 1110 1101 = 0xed`

Question 2

1. `1011 = 0xb = 11`
2. `1011 1011 = 0xbb = 187`
3. `1000 0000 = 0x80 = 128`
4. `11 1100 = 0x3c = 60`
5. `0011 1100 = 0x3c = 60`

Question 3

1. `0x0e = 1110 = 14`
2. `0xe = 1110 = 14`
3. `0xAA = 1010 1010 = 170`
4. `0xFA = 1111 1010 = 250`
5. `0x40 = 0100 0000 = 64`
6. `0x18 = 0001 1000 = 24`

Question 4

```
/* Print the binary value of an integer
   less than 32768. We start at the
   largest applicable power of 2 and
   work our way down to the smallest
   power of 2.  For each power, if that
   power is "contained" in the number we
   remove it and print a '1', otherwise
   we print a '0'.  When all powers have
   been tested the result is that we
   have printed the binary
   representation of the number. */

#include <stdio.h>

void printbin ( int n )
{
  int p ;
  p = 16384 ;
  while ( p >= 1 ) {
    if ( n >= p ) {
      n = n - p ;
      printf ( "1" ) ;
    } else {
      printf ( "0" ) ;
    }
    p = p / 2 ;
  }
  printf ( "\n" ) ;
}

/* printbin() tests: */

main()
{
  printbin(0) ;
  printbin(1) ;
  printbin(2) ;
  printbin(4) ;
  printbin(237) ;
  printbin(32767) ;
}
```

Question 5

1. A bitwise 'and' operation with a '1' bit retains the value of that bit. A bitwise 'and' operation with a '0' bit always sets that bit to '0'.

   `( 0xaa & 0x0f )`

```
  = 0x0a
```

2.
```
    ( 0x3c & 0xf0 ) | ( 0x3c & 0x0f )
  = ( 0x30 ) | ( 0x0c )
  = 0x3c
```

3. Note that the && is the *logical* and operator.

```
    3 * ( 0xf0 && 0x0f )
  = 3 * ( 0x1 )
  = 0x3
```

4. An exclusive-or with a ’1’ bit inverts that bit.

```
    ( 0x3c ^ 0xff ) + ( 1 < 3 )
  = ( 0xc3 ) + ( 1 )
  = 0xc4
```

5.
```
    ~ ( 128 | ’ ’ )
  = ~ ( 0x80 | 0x20 )
  = ~ ( 0xa0 )
  = 0x5f
```

6. Note that the || is the *logical* ‘or’ operator.

```
    128 || ( ’ ’ == 0x20 )
  = 128 || ( 0x20 == 0x20 )
  = 128 || 1
  = 0x01
```