1.
    a. $1_{dec} = 0000001_{bin}$
    b. $42_{dec} = 00101010_{bin}$
    c. $256_{dec} = 000000010000000_{bin}$ or 100000000_{bin}
    d. $4{,}294{,}967{,}296_{dec} = 100000000000000000000000000000000_{bin}$
2.
    a. $10000000_{bin} = 64_{dec}$
    b. $10101010_{bin} = 170_{dec}$
    c. $11110000_{bin} = 240_{dec}$
    d. $11001100_{bin} = 204_{dec}$
3.
    a. $111_{bin} + 111_{bin} = 011110_{bin} = 30_{dec}$
    b. $1010_{bin} + 1010_{bin} = 010100_{bin} = 20_{dec}$
    c. $11101_{bin} + 1010_{bin} = 0100111_{bin} = 39_{dec}$
    d. $1101_{bin} – 11_{bin} = 01010_{bin} = 10_{dec}$
    e. $10001_{bin} – 100_{bin} = 01101_{bin} = 13_{dec}$
    f. 101_{bin} x 10_{bin} = 01010bin = 10_{dec}
    g. 1011_{bin} x 11_{bin} = 0100001bin = 33_{dec}
    h. $1101_{bin} / 11_{bin} = 4.3_{dec}$

4. Hexadecimal numbers represent 4 bits. One common use for the hexadecimal number system to be used is to show human readable binary conversion. The standard range for hexadecimal is 00 to FF which allows for 8 bits or 1 byte to be stored. Each character holds a single bit. As a decimal value for a byte ranges from 0 – 255, hexadecimal shortens it so that it is only 2 characters long instead of the decimal 3. It is also commonly used to represent computer memory address and error codes within the system.
5.
    a. $10000000_{bin} = -128_{dec}$
    b. $10101010_{bin} = -86_{dec}$
    c. $1111000_{bin} = -16_{dec}$
    d. $1001100_{bin} = -52_{dec}$
    e. $-16_{dec} = 11110000_{bin}$
    f. $128_{dec} = 010000000_{bin}$
    g. $-128_{dec} = 10000000_{bin}$
    h. $-123_{dec} = 110000101_{bin}$
6.
    a. $11111_{bin} \mid 11111_{bin} = 11111$
    b. $11111_{bin} \wedge 11111_{bin} = 0$
    c. 10101_{bin} & 11111_{bin} = 9061
    d. $10101_{bin} \mid 11111_{bin} = 12151$
    e. $00000_{bin} \wedge 11111_{bin} = 11111$
    f. $1 \ll 3 = 8_{dec} = 1000_{bin}$
    g. $100 \gg 2 = 25_{dec} = 11001_{bin}$
    h. ~10101 = -10102

i. $100 << 1 = 200_{dec}$ 1001000_{bin}

j. $1010 >> 1 = 505_{dec}$ 111111001_{bin}

k. ~11111 = -11112

7.

a. Set an single bit to 0
number &= ~(1 << x)

b. Set an single bit to 1
number |= 1 << x

c. Check the value of a single bit
bit = (number >> x) & 1