Homework 1:: ACOS

Unsigned:

Task 1:

1. 0 // 2 = 0 → 000000

Answer = 00 0000

2. 13 // 2 = 6 // 2 = 3 // 2 = 1
   1 3 6 13
   1 1 0 1

I wrote a one beneath any odd number and a 0 beneath any even number.
   00 1101
I bit-extended the result

Answer = 00 1101
I will now use the same method for the remaining questions.

3. 24 // 2 = 12 // 2 = 6 // 2 = 3 // 2 = 1
   1 3 6 12 24
   1 1 0 0 0
   01 1000

Answer : 01 1000

4. 63 = 31 = 15 = 7 = 3 = 1
   1 3 7 15 31 63
   1 1 1 1 1 1

Answer: 11 1111

Signed:

1. 16 = 8 = 4 = 2 = 1
   1 2 4 8 16
   1 0 0 0 0

Answer : 01 0000

2. 2 = 1
   1 2
   1 0
   00 0010
   Converting to unsigned (invert all bits and add one)
   11 1101 → 11 1110

Answer : 11 1110
*Basically the most significant bit is -32 and other bits are positive.

Same method will be applied for the remaining questions.

3. 31 = 15 = 7 = 3 = 1
   1 3 7 15 31
   1 1 1 1 1

01 1111

4. 32 = 16 = 8 = 4 = 2 = 1
   1 2 4 8 16 32
   1 0 0 0 0 0 → 01 1111 → 10 0000

Answer : 10 0000

Task 2:
2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0

Unsigned
00 0101
2^0 + 2^2 = 5
Answer: 5

10 1011
2^5 + 2^3 + 2^1 + 2^0 = 32 + 8 + 2 + 1 = 43
Answer : 43

11 1111
2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 63
Answer: 63

10 0000
Answer: 32

Signed
00 0101 = 5
Answer: 5
Algorithm: Take away one and invert bits
10 1011 → 10 1010 → 01 0101 = 2^4 + 2^2 + 1 = 21 → -21
Answer: -21
11 1111 → 11 1110 → 00 0001 = 1 → -1
Answer : -1
10 0000 → 01 1111 → 10 0000 → 32 → -32
Answer: -32

Task 3:
Algorithm: Convert to binary → Convert to Hexadecimal

1.
7 = 3 = 1
1 3 7
0000 0111
Answer : 0x07

2.
240 = 120 = 60 = 30 = 15 = 7 = 3 = 1
1 3 7 15 30 60 120 240

1 1 1 1 0 0 0 0
1111 0000
0xF0

3.
171 = 85 = 42 = 21 = 10 = 5 = 2 = 1
1 2 5 10 21 42 85 171
1 0 1 0 1 0 1 1
1010 1011
0xAB

4.
126 = 63 = 31 = 15 = 7 = 3 = 1
1 3 7 15 31 63 126
1 1 1 1 1 1 0
0111 1110
0x7E

Task 4:
0x3C == 0011 1100
0x7E == 0111 1110
0xFF == 1111 1111
0xA5 == 1010 0101

Task 5:
~0011 1100 == 1100 0011
~0111 1110 == 1000 0001
~1111 1111 == 0000 0000
~1010 0101 == 0101 1010

Task 6:
0xDEADBEEF
Big Endian:
Split 0xDEADBEEF into 0xDE, 0xAD, 0xBE, 0xEF
Let ‘a’ represent a point in the address. Big Endian approach will store the most significant byte (8 bits). In our case it’s 0xDE
a: DE
a+1: AD
a+2: BE
a+3: EF

Conversely, Little Endian will store the least significant byte at ‘a’.
a: EF
a+1: BE
a+2: AD
a+3: DE

Task 7:
7 == 0 0111 → 0000 0111
15 == 0 1111 → 0000 1111

16 = 8 = 4 = 2 = 1 → 1 2 4 8 16 → 1 0000 -1 = 0 1111 rev = 1 0000 = -16
-16 = 1 0000 → 1111 0000
Answer : 1111 0000
5 = 0 0101 - 1 = 0 0100 rev = 1 1011 = -5
1 1011 → 1111 1011
Answer : 1111 1011

Task 8:
7 == 0111
9 == 1001
Using addition algorithm:
7 + 9 == 1 0000
Answer: 1 0000 (or 0001 if we want it to always be 4 bits. Like in C++)

4 == 0100
-5 == 1011
Using addition algorithm:
0100+
1011
=1111 = -1
Answer = -1

Suppose we had an overflow:
E.g: -8 -1
1000+
1111
= 0111
= 10111

BONUS TASK (1)
Let
X = 0101
Y = 1001

1. X = X ^ Y = 1100
2. Y = X ^ Y = 0101 = Y ^ (X ^ Y) = X
3. X = X ^ Y = 1001 = X ^ (Y ^ X) = Y

So I swapped x & y.

XOR property is that it can save information. So in 1 we produce a hybrid and store it in x. Then we remove y from the hybrid by using XOR again and this leaves us with x which we store in y. Then we remove the initial x from the hybrid which leaves us with the initial y which we store in x. So now the values have swapped.
*The x and y are stored in the ALU, and when we XOR we retract them.
https://betterexplained.com/articles/swap-two-variables-using-xor/

BONUS TASK (2)
x & (x - 1) - This is simple, because by subtracting one, I am changing all the zeros that came before the rightmost one to one and the rightmost one to zero.
By applying the & operation I take all bits which are the same, this will be all the bits before the rightmost one.
Example : 0101 1000 - 1 = 0101 0111
0101 1000 &&
0101 0111
= 0101 0000

x | (x + 1) - Adding one means that all trailing ones will be turned to zero and the rightmost zero will be turned to one. Now if an or operation is applied, the trailing zero that has turned to one will return one and the trailing ones will return to one, so the original number will be returned with the rightmost zero turned to one.
1010 0111 + 1 = 1010 1000
1010 0111 ||
1010 1000
= 1010 1111

x | (x - 1) - Also simple, since as previously mentioned -1 changes rightmost 1 to 0 and all trailing zeros to one. So If I take or operation, since the initial rightmost one would be true and now all the trailing ones would be true, the operation turns on all the trailing zeros.
Example:
1010 1000 - 1 = 1010 0111
1010 1000 ||
1010 0111
= 1010 1111