CS204 Data Representation/Built-in Types/Overflow - Lab Questions and Solutions

1) Please make the decimal to binary conversion below.

120 : **0111 1000**

2) Please make the binary to decimal conversion below.

0111 0001: **113**

3) (a) Please convert the decimal numbers below to binary using Sign/Magnitude, 1's complement and 2's complement representations. Assume 8-bit storage.

i. Decimal number: 35
Sign/Magnitude Representation: **0010 0011**
1's complement: **0010 0011**
2's complement: **0010 0011**

ii. Decimal number: -60
Sign/Magnitude Representation: **1011 1100**
1's complement: **1100 0011**
2's complement: **1100 0100**

(b) Please convert the binary numbers below to decimal assuming that the binary numbers are represented in Sign/Magnitude, 1's complement and 2's complement.

iii. Binary number: 0110 0001
If this number is in Sign/Magnitude Representation: **97**
If this number is in 1's complement: **97**
If this number is in 2's complement: **97**

iv. Binary number: 1110 1101
If this number is in Sign/Magnitude representation: **-109**
If this number is in 1's complement: **-18**
If this number is in 2's complement: **-19**

4) Please make the subtraction operation below after converting the operands to binary in Sing/Magnitude, 1's complement and 2's complement representations. Please do the math in binary.

110_{10} - 80_{10} = ?

SOLUTION:

Think of this 110_{10} + (-80_{10}) = ?

<u>Sign/Magnitude:</u>

110: 0110 1110

-80: 1101 0000

------Operation----

 0110 1110

 1101 0000

+---------------

1/0011 1110 = 62 (Wrong!)

carry overflow

<u>1's complement:</u>

110: 0110 1110

-80: 1010 1111

------Operation----

  0110 1110

  1010 1111

+---------------

1/0001 1101 = 29 (Wrong!)

carry overflow

<u>2's complement:</u>

110: 0110 1110

-80: 1011 0000

------Operation----

  0110 1110

  1011 0000

+---------------

1/0001 1110 = 30 (Correct!)

carry overflow

5) What is the output?

a)
```
char ch;
ch = -190;

cout << ch << endl;
cout << (int)ch << endl;
```

B
66

b)
```
char ch;
ch = -67;

cout << ch << endl;
cout << (int) ch << endl;
```

¢
-67

c)
```
unsigned char ch;
ch = 200;

cout << ch << endl;
cout << (int) ch << endl;
```

╚
200

d)
```
unsigned char ch;
ch = -67;

cout << ch << endl;
cout << (int) ch << endl;
```

¢
189

e)
```
short k = -61200;
cout << "Short Integer: " << k << endl;
```

Short Integer: 4336

f)
```
short ints = -20000;
unsigned short intus = ints;
cout << "implicit unsigned type-casting: " << intus << endl;
```

implicit unsigned type-casting: 45536

g)
```
short  ints = -20000;
cout << "explicit unsigned type-casting: " << (unsigned short) ints << endl;
```

explicit unsigned type-casting: 45536

h)
```
unsigned short usnum = 25800;
cout << "explicit signed type-casting: " << (short) usnum << endl;
```

explicit signed type-casting: 25800

i)
```
unsigned short usnum2 = 68450;
cout << "explicit signed type-casting: " << (short) usnum2 << endl;
```

explicit signed type-casting: 2914

6) What are the outputs of the following pieces of code?

a) Operations with different type of operands

```
int a = 5;
int b = -10;
unsigned int c = 3;

//In this expression, the order will be left to right.
//In the expression of a+b since both of them are signed numbers
//result will be -5 as usual. Yet, then there is a expression of the
//result of the a+b (signed) and -c (unsigned); thus the result of a+b
//will be type-casted to unsigned and the result will not be as expected.
if (a+b-c < 0)
{
	cout << "The result should definitely be less than zero!" << endl;
	cout << "Expression: " << a+b-c << endl;
}
else
{
	cout << "Hmm, what is going on?" << endl;
	cout << "Expression: " << a+b-c << endl;
}
```

Hmm, what is going on?
Expression: 4294967288

b) Overflow

```
//This loop will not be an infinite loop.
//It will iterate for a while; but when the
//count is 0 then the loop finished and this happens
//after several overflows.
unsigned char count = 13;
while(count > 0)
{
	cout << (int) count << endl;
	count = count-5;
}
```

There will be overflow for *count;* but this loop will not be infinite since at some point count will be zero and while loop will finish.

Output:

13
8
3
254

249
244
239
234
229
224
219
214
209
204
199
194
189
184
179
174
169
164
159
154
149
144
139
134
129
124
119
114
109
104
99
94
89
84
79
74
69
64
59
54
49
44
39
34
29
24
19
14
9
4
255
250
245
240
235
230
225
220
215
210
205
200
195
190

185
180
175
170
165
160
155
150
145
140
135
130
125
120
115
110
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5