CS 367 - Recitation #2 **Integer Representation**

Name Adilet Kuroda **G#** 12

Group Member Name: Sulav Acharya

Group Member Name: ____________________

Today's Goals: We want to practice interpreting binary numbers as signed and unsigned, consider how endianness affects storage in memory, and play with some integer operations.

Work in groups of 2-3 students. Everyone will turn in what they've got at the end of recitation on paper and on Blackboard, writing everyone's name down so the GTAs don't have to re-grade work.

Get as much done as you can. At the end of the session submit your file here on blackboard - each member of the group needs to do this. Be sure to put the names of the members of your group into a comment at the top of the file so that the GTAs don't have to look at your group's code more than once. This assignment is not for a grade but you will be given feedback in the form of a 'score' (1-3) and possibly some comments. A score of 3 means everything looks great. A score of two indicates some minor problems. And a score of one indicates that there were some major issues. If you get a 1, don't panic - go see your prof or a GTA to get more extensive feedback.

---

1. Signed and unsigned interpretations

Signed and unsigned are two different ways to interpret an existing bit-pattern. Remember, the only difference is the *sign* of the leftmost column; the magnitude is still $2^{\text{bitwidth-1}}$. Fill in this chart.

bits(6 bits wide)	decimal value (if bits are unsigned)	decimal value (if bits are signed)
00 0101	5	5
10 1010	42	-22
10 0111	39	- 25
11 1100	60	-4
11 0100	52	−12
01 0111	23	23

---

2. Binary addition

Add these 8-bit binary numbers. Each time, check (unsigned) and/or (signed) if overflow occurs when interpreted that way.

Did overflow occur?

(#1)
```
  1001 0010
+ 0111 0111
-----------
  0000 1001
```
(unsigned) ☑ (signed) ☐

(#2)
```
  0110 0000
+ 0111 1110
-----------
  1101 1110
```
(unsigned) ☐ (signed) ☑

(#3)
```
  0110 0011
+ 0011 1101
-----------
  1010 0000
```
(unsigned) ☐ (signed) ☑

Did overflow occur?

(#4)
```
  0100 0000
+ 0010 0100
-----------
  0110 0100
```
(unsigned) ☐ (signed) ☐

(#5)
```
  1100 1100
+ 0011 1100
-----------
  0000 1000
```
(unsigned) ☑ (signed) ☐

(#6)
```
  1111 1111
+ 0111 1111
-----------
  0111 1110
```
(unsigned) ☑ (signed) ☐

---

3. Power of 2 Multiply with Shift

Use shifts and add/subtracts to represent below multiplications. Use **three** shifts or less.

	Shift and add/subtract		Shift and add/subtract
X*9	(x << 3) + x	**X*15**	((x << 3)<<1) -x == (x<<4)-x
X*21	(x << 3) + (x << 3) + (x<<2) +x	**X*33**	((x<<3)<<2)+x ==(x << 5) + x
X*127	((x<<3)<<3)<<1) -x == (x<<7)-x	**X*51**	((x << 3)<<2) + ((x << 3)<<1) + (x <<1) + x

---

4. Unsigned power of 2 Divide with Shift

Fill in the table below bit using 8bit <u>un</u>signed and shifts.
(0b is a special mathematical notation we're using here to indicate binary values)

	In Bits	In Hex		In Bits	In Hex
0b01011100/2^2	0001 0111	0x17	**0b10011111/2^3**	0001 0011	0x13
0x3C/2^5	0000 0001	0x01	**0xEF/2^4**	0000 1110	0x0E
55/2^3	0000 0110	0x06	**99/2^2**	0001 1000	0x18

5.Endianness

An address always refers to a single byte. When a value needs more than one byte to be represented, we always use the following (increasing-address) bytes, e.g. a 4-byte int at address 0x100 actually takes up bytes at addresses 0x100, 0x101, 0x102, and 0x103. There's a choice to be made: with these four spots, what order do we put those multiple bytes? (biggest/leftmost byte, or smallest/rightmost byte at the starting address?).

Here are four definitions and their addresses. Fill in memory for a big- and little-endian system.

Definition	Starting Address	Size (bytes)	Hex Value
`int x = 0xBEEFCAFE;`	0x100	4	BE EF CA FE
`short y = 0x1337;`	0x104	2	13 37
`char z = 0xd9; // 'y'`	0x106	1	D9
`char[] s = "LOL";`	0x108	4	4C 4F 4C 00

Big-Endian Memory:

	...
	0x10C
	0x10B
00	0x10A
4C	0x109
4F	0x108
4C	0x107
D9	0x106
37	0x105
13	0x104
FE	0x103
CA	0x102
EF	0x101
BE	0x100

Little-Endian Memory:

	...
	0x10C
	0x10B
4C	0x10A
4F	0x109
4C	0x108
00	0x107
D9	0x106
13	0x105
37	0x104
BE	0x103
EF	0x102
CA	0x101
FE	0x100

Now we can do this in reverse. Given the following memory, fill in the hex values in this chart.

VALUE	ADDRESS
0x56	0x112
0x34	0x111
0x12	0x110
0x00	0x10F
0x00	0x10E
0xFD	0x10D
0xBA	0x10C
0x59	0x10B
0x21	0x10A
0x13	0x109
0x58	0x108
0x23	0x107
0xA1	0x106
0x65	0x105
0x7C	0x104
0x98	0x103
0xF6	0x102
0x13	0x101
0x45	0x100

address	size	value (little-endian)	value (big-endian)
0x110	2	0x34 12	0x12 34
0x108	8	0x00 00 FD BA 59	0x58 13 21 59 BA F
0x104	4	0x23 A1 65 7C	0x7C 65 A1 23
0x100	4	0x98 F6 13 45	0x45 13 F6 98