Name Solution **G#** N/A

Group Member Name: N/A

Group Member Name: N/A

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 1011	11	11
11 0101	53	-11
01 1010	26	26
10 0111	39	-25
10 1100	44	-20
01 0011	19	19

---

2. Binary addition

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

```
                    Did overflow occur?
(#1)
  0001 0101         (unsigned) (signed)
+ 0110 1111             ☐          ☑
  1000 0100

(#2)
  1110 0111         (unsigned) (signed)
+ 1101 1010             ☑          ☐
  1100 0001

(#3)
  1101 0001         (unsigned) (signed)
+ 0011 0111             ☑          ☐
  0000 1000
```

```
                    Did overflow occur?
(#4)
  0101 0101         (unsigned) (signed)
+ 0010 0100             ☐          ☐
  0111 1001

(#5)
  1010 1100         (unsigned) (signed)
+ 0011 0011             ☐          ☐
  1101 1111

(#6)
  1011 1100         (unsigned) (signed)
+ 1000 0101             ☑          ☑
  0100 0001
```

---

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*33	x << 5 + x	**X*60**	x << 6 - x << 2
X*24	x << 4 + x << 3	**X*38**	x << 5 + x << 2 + x << 1
X*77	x << 6 + x << 3 + x << 2 + x	**X*142**	x << 7 + x << 4 - x << 1

---

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 notation we're using here to indicate binary values)*

	In Bits	**In Hex**		**In Bits**	**In Hex**
$53/2^2$	0b00001101	0x0D	**$144/2^5$**	0b00000100	0x04
$0x4B/2^5$	0b00000010	0x02	**$0xCF/2^4$**	0b00001100	0x0C
$0b01010001/2^2$	0b00010100	0x14	**$0b10110101/2^4$**	0b00001011	0x0B

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 0x200 actually takes up bytes at addresses 0x200, 0x201, 0x202, and 0x203. 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 = 0x17011337;`	0x200	4	0x17011337
`char y = 0x21; // '!'`	0x204	1	0x21
`short z = 0xcafe;`	0x206	2	0xCAFE
`char[] s = "edu";`	0x208	4	0x65,0x64,0x75,0x00

Big-Endian Memory:

Value	Address
	...
	0x20C
0x00	0x20B
0x75	0x20A
0x64	0x209
0x65	0x208
0xFE	0x207
0xCA	0x206
	0x205
0x21	0x204
0x37	0x203
0x13	0x202
0x01	0x201
0x17	0x200

Little-Endian Memory:

Value	Address
	...
	0x20C
0x00	0x20B
0x75	0x20A
0x64	0x209
0x65	0x208
0xCA	0x207
0xFE	0x206
	0x205
0x21	0x204
0x17	0x203
0x01	0x202
0x13	0x201
0x37	0x200

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

VALUE	ADDRESS
0x58	0x112
0x43	0x111
0x11	0x110
0x01	0x10F
0xFA	0x10E
0x00	0x10D
0xAF	0x10C
0x1C	0x10B
0x22	0x10A
0x00	0x109
0x18	0x108
0x25	0x107
0xA2	0x106
0x62	0x105
0x5A	0x104
0x99	0x103
0xFF	0x102
0x10	0x101
0x40	0x100

address	size	value (little-endian)	value (big-endian)
0x110	2	0x 4311	0x 1143
0x10C	4	0x 01FA00AF	0x AF00FA01
0x104	8	0x 1C22001825A2625A	0x 5A62A2251800221C
0x100	4	0x 99FF1040	0x 4010FF99