**Chapter 1a HW Solution**

**Problem 1-1.** I usually convert the decimal to hex, then to binary, then negate (if necessary), then add. For part (h) I mentally cancelled both minus signs and just added the positive.

(a) $43 = 2B = 00101011$, $-100 = -64 = 10000100 = 10011100$. Summing these yields $%11000111 = -57$, which is CORRECT.

(b) $-22 = -16 = 11010110$, $-13 = -8D = 10001011 = 11110011$. Summing these yields $%11011101 = -35$, which is CORRECT.

(c) $-34 = -22 = 11011101$, $34 = 22 = 00100010$. Summing these yields $%00000000 = 0$, which is CORRECT.

(d) $25 = 19 = 00011001$, $106 = 6A = 01101010$. Summing these yields $%10000011 = -125$, which is INCORRECT (2's complement overflow).

(e) $45 = 2D = 00101101$, $-82 = -52 = 10100100 = 10101100$. Summing these yields $%11011011 = -37$, which is CORRECT.

(f) $127 = 7F = 10111111$, $-127 = 10000001 = -128 + 1$. Summing these yields $%00000000 = 0$, which is CORRECT.

(g) $-1 = FF = 11111111$ (which was always all ones). Adding -1 to itself yields $%11111110 = -2$, which is CORRECT.

(h) $-47 = -2F = 11100011 = 11010111$. $107 = 6B = 01101011$. Summing these yields $%00111100 = 60$, which is CORRECT.

(i) $-66 = -42 = 10110010 = 10111110$, $-72 = -48 = 10010000 = 10111100$. Summing these yields $%01111010$, which is INCORRECT (result is positive, 2's complement overflow).

(j) $-126 = -82 = 10000001$, $-2 = 8F = 11111111$ (both of these were easy to get). Summing these yields $%10000000 = -128$, which is CORRECT.

**Problem 1-2.** These are much quicker, since binary and hex both use a base that is a power of 2.

(a) $%00111101 = 3D$

(b) $%0100 0100$

(c) $%1110 0001$

(d) $%0000 0000$ (negating zero returns zero; the complement/increment still works)

(e) $%1000 0000$ OVERFLOW! (the original number was -128; its negative is outside the range of 8-bit signed numbers)

(f) $%1000 0011 1111 1111$ (what a pain; same process with 16-bit numbers)

(g) $%0000 0010$

(h) $%0010 0000$
Problem 1-4. I find these binary-to-decimal conversions a little easier than negation. Sometimes I convert from binary to hex, then to decimal.

(a) %0000 0001 = 1
(b) %1111 1000 = -8
(c) %0000 0000 0011 1111 = $3F = 63
(d) %0000 0000 = 0 (zero is zero!)
(e) %1000 0000 0000 0000 = -32768 (all we have is the sign bit)
(f) %0111 1111 = 127 (the largest positive 8-bit number in 2’s complement)
(g) %1000 0000 = -128 (again, only the sign bit)
(h) %1000 0010 = -126 (sign bit of -128 plus 2)
(i) %0100 0000 = $40 = 64
(j) %1111 1111 1111 1111 = -1 (all ones in 2’s complement is always -1, no matter the length).

Problem 1-5. These binary-to-hex conversions are easy; just convert each 4-bit pattern to its hexadecimal equivalent.

(a) %1111 0000 = $F0
(b) %1011 0101 = $B5
(c) %0011 1101 = $3D
(d) %0110 1111 0000 0000 = $6F00
(e) %1110 1011 0110 1100 = $EB6C
(f) %1110 0111 = $E7
(g) %1001 1110 = $9E
(h) %0110 0111 = $67
(i) %10010000 = $90
(j) %0110 1110 1110 1111 = $6EEF

Problem 1-6. When we add %01011101 and %01100101 we get %11000010. Since both the added and the augend are positive, and the result is negative, there is TWO’S COMPLEMENT OVERFLOW (the 6811 V bit would be set).

Problem 1-11. Since $D - C$ causes a borrow from “outside”, this means that $C > D$.

Problem 1-12. The largest possible 16-bit 2’s complement number is a zero “sign bit” followed by all ones, thus it is %0111 1111 1111 1111 = $7FFF = 32767.

Problem 1-15. With 8-bit 2’s complement, positive numbers are in the range $00..$7F (bit 7 zero) while negative numbers are in the range $FF..$80 (bit 7 one).

Problem 1-17. I performed this subtraction by “negating then adding” and I found the result to be:
%10010001 - %01101010 = %10010001 + %10010110 = %00100111. Since both addend and augend are negative, yet
result is positive, there is TWO’S COMPLEMENT OVERFLOW!

**Problem 1-19.** We end with an easy one. As I’ve stated, -1 in any length 2’s complement is always all ones, so...
8-bit: $FF$
16-bit: $FFFF$
32-bit: $FFFFFFFF$ (8 hex characters = 32 bits)

These were pretty tedious, but things *will* get better!