UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EECE 259: Introduction to Microcomputers

Solutions 1: Logic Gates, Binary Numbers and a Computational Datapath

Solutions handed out Jan. 19, 2011

PART 1: Study Question SOLUTIONS

4. Fill in the following table by writing the required values in both binary and decimal. Why is the minimum unsigned value not included in the table?

	Maximum *unsigned*	**Maximum *signed***	**Minimum *signed*** (negative numbers)
4 bits	(binary) % 1111 (decimal) 15	(binary) % 0111 (decimal) 7	(binary) % 1000 (decimal) -8
8 bits	% 1111 1111 255	% 0111 1111 127	% 1000 0000 –128
16 bits	% 1111 1111 1111 1111 65,535	% 0111 1111 1111 1111 32,767	% 1000 0000 0000 0000 –32,768
32 bits	% 1111 1111 1111 1111 1111 1111 1111 1111 4,294,967,295	% 0111 1111 1111 1111 1111 1111 1111 1111 2,147,483,647	% 1000 0000 0000 0000 0000 0000 0000 0000 –2,147,483,648

5. Fill in the rest of the table. Use as many bits / digits as you need. **Assuming unsigned.**

Decimal	**Binary (%)**	**Hexadecimal ($)**
2241_{10}	% 1000 1100 0001	$ 8 C 1
186_{10}	% 1011 1010	$B A
41872_{10}	% 1010 0011 1001 0000	$ A 3 9 0
1976_{10}	% 0111 1011 1000	$ 7 B 8
33825_{10}	% 1000 0100 0010 0001	$ 8 4 2 1
43981_{10}	% 1010 1011 1100 1101	$ A B C D
3832_{10}	% 1110 1111 1000	$ E F 8

6. Convert from hexadecimal to decimal, and back: $1111, $BEEF, $CAFE, $F00D.

Assuming unsigned.

$1111 = 1*4096+1*256+1*16+1 = 4369

4319/16 = 269 rem 1 → $1,

269/16 = 16 rem 1 → $1,

16/16 = 1 rem 1 → $1,

1/16 = 0 rem 1 → $1

Writing backwards ➔ $1111

$BEEF = 11*4096 + 14*256 + 14*16 + 15*1 = 48879

48879/16 = 3054 rem 15 → $F,

3054/16 = 190 rem 14 → $E,

190/16 = 11 rem 14, → $E,

11/16 = 0 rem 11 → $B

Writing backwards ➔ $BEEF

$CAFE = 12 * 4096 + 10*256 + 15*16 + 14*1 = 51966

51966/16 = 3247 rem 14 → $E,

3247/16 = 202 rem 15 → $F,

202/16 = 12 rem 10 → $A,

12/16 = 0 rem 12 → $C

Writing backwards ➔ $CAFE

$F00D = 15 * 4096 + 13*1 = 61453

61453/16 = 3840 rem 3 → $D,

3840/16 = 240 rem 0 → $0,

240/16 = 15 rem 0 → $0,

15/16 = 0 rem 15 → $F

Writing backwards ➔ $F00D

7. Determine the decimal number that results if:
   a. –13 is stored in an 8-bit signed number format and then interpreted (mistakenly) as an 8-bit unsigned number.

      First, determine that 13 in 8-bit binary is % 0000 1101.

      Next, take the two's complement to determine that –13 = % 1111 0011.

      When interpreted as an unsigned number, the value is 128+64+32+16+2+1 = 243.

      The answer is 243.
   b. 253 is stored in memory as an 8-bit unsigned number and then interpreted (mistakenly) as an 8-bit signed number.

      First, determine that 253 in 8-bit binary → % 1111 1101.

      Since the MSB bit is a '1', we know this is negative as a signed number.

      Determine the magnitude of the negative number by finding the two's complement: % 0000 0011 = 3.

      The answer is –3
   c. 13 is stored in an 8-bit signed number format and just the lowest 4 bits are interpreted as a 4-bit signed value.

The value 13 in 8-bit signed binary is % 0000 1101.

The lowest 4 bits are % 1101. This appears to be negative. The magnitude is 3.

The answer is –3.

d. –13 is stored in an 8-bit signed number format and just the lowest 4 bits are interpreted as a 4-bit signed value.

The value of –13 in binary is the two's complement of 13, or % 1111 0011.

The lowest 4 bits are % 0011.

The answer is 3.

8. Perform *sign extension* for the following examples.

   a. Write the value of 4 in binary using 4 bits. Extend this to an 8-bit value.

   The value 4 is %0100 in 4 bits. After extending this to 8 bits, 4 is %0000 0100.

   b. Write the value of –4 in binary using 4 bits. Extend this to an 8-bit value.

   The value –4 is %1100 in 4 bits. After extending this to 8 bits, –4 is %1111 1100.

9. To convert a positive number to negative one in two's complement, you invert all the bits and add 1. Try doing this for 8-bit values 0 and –128. What happens? Why?

0 is % 0000 0000. Inverting gives % 1111 1111 and adding 1 gives % 0000 0000.

So, the two's complement of 0 is still 0. Some signed number systems can be confusing because they use two different binary values to represent +0 and –0.

–128 = % 1000 0000. Inverting gives %0111 1111 and adding 1 gives %1000 0000.

So, the two's complement of –128 = –128. The expected value would be the magnitude or +128, but instead we get the "wrong" answer. This occurs because there is an overflow. All positive numbers can be expressed as a negative number in two's complement form. However, the minimum / smallest / "most negative" number in two's complement form does not have a matching positive counterpart. This can sometimes lead to unexpected results!

10. Given a 4-bit adder that computes A+B, find a way of re-using that circuit by adding some additional logic so it will compute A–B.

Observe that $A–B = A+(–B) = A+\overline{B}+1$. Hence, you can compute $A–B$ by using the 4-bit adder: A appears on one input, and $\overline{B}$ appears on the other input (use 4 NOT gates after B). You can get the +1 term by setting the ***initial carry-in*** of the adder to 1 instead of 0.

11. The logic gates to produce carryout c_1 of a *full adder* were shown in class. The logic for the carryout C flag was also given. Verify that these two equations are the same. *Hint*: you can create a truth table and verify that both **C** and c_1 are the same, or you can manipulate the algebraic equations to show they are equivalent. Try it both ways!

In class, we determined that C = $a_7 \cdot b_7 + a_7 \cdot \overline{r_7} + b_7 \cdot \overline{r_7}$ for 8-bit addition. We can replace r with s since they both represent the sum. If the numbers are only 1-bit wide, we would find C = c_1 = $a_0 \cdot b_0 + a_0 \cdot \overline{s_0} + b_0 \cdot \overline{s_0}$. Also, remember the sum bit is $s_0 = a_0 \oplus b_0 \oplus c_0$.

By inspecting the Full Adder and Half Adder schematics, you can see the carry-out for the full adder is $c_1' = a_0 \cdot b_0 + c_0 \cdot (a_0 \oplus b_0)$.

The goal now is to show that $c_1 = c_1'$ using either a *truth table* or *algebraic manipulation*.

For the *truth table* approach, create a truth table with a_0, b_0 and c_0 as inputs. Compute s_0 and c_1 according to $s_0 = (a_0 \oplus b_0 \oplus c_0)$ and $c_1 = a_0 \cdot b_0 + a_0 \cdot \overline{s_0} + b_0 \cdot \overline{s_0}$. Also, compute $c_1' = a_0 \cdot b_0 + c_0 \cdot (a_0 \oplus b_0)$. Note the two columns ($c_1$ and c_1') are identical, so you are done!

a_0	b_0	c_0	s_0	c_1	c_1'
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	1

Algebraic manipulation is trickier. Start with $c_1 = a_0 \cdot b_0 + a_0 \cdot \overline{s_0} + b_0 \cdot \overline{s_0}$ and plug in the value for $s_0 = a_0 \oplus b_0 \oplus c_0$ to get an ugly mess. Then show that $c_1 = c_1'$. You won't have to do something this hard for the quiz or exam, so I'll put the solution in a separate page.

12. In class, you were given the logic to detect overflows while *adding* two signed numbers ($V_{add} = \overline{a_7} \cdot \overline{b_7} \cdot r_7 + a_7 \cdot b_7 \cdot \overline{r_7}$). Using a similar approach, design logic for *subtracting* two signed numbers, that is compute V_{sub} for the operation A–B.

    $V_{sub} = a_7 \cdot \overline{b_7} \cdot \overline{r_7} + \overline{a_7} \cdot b_7 \cdot r_7$

13. Summarize the logic expression for each flag. Keep in mind that N and Z only look at the result bits, r_7 to r_0. Expressions for V_{add} and N are given as examples.

    a. $C = a_7 \cdot b_7 + a_7 \cdot \overline{r_7} + b_7 \cdot \overline{r_7}$

    b. $B = \overline{a_7} \cdot b_7 + b_7 \cdot r_7 + \overline{a_7} \cdot r_7$

    c. $V_{add} = \overline{a_7} \cdot \overline{b_7} \cdot r_7 + a_7 \cdot b_7 \cdot \overline{r_7}$

    d. $V_{sub} = a_7 \cdot \overline{b_7} \cdot \overline{r_7} + \overline{a_7} \cdot b_7 \cdot r_7$

    e. $N = r_7$

    f. $Z = \overline{r_7} \cdot \overline{r_6} \cdot \overline{r_5} \cdot \overline{r_4} \cdot \overline{r_3} \cdot \overline{r_2} \cdot \overline{r_1} \cdot \overline{r_0}$

14. (**Long, but you need practice!**). Perform each of the operations in the table below. Express the answer in binary, decimal, and hexadecimal, and give ALL flags: Z, N, V_{sub}, V_{add}, B and C using the logic equations from problem 14. The first row is done for you.

Hint: the flags are **merely logic equations** that don't care *which* operation is being performed.

Operation	Results								
	Binary	**Decimal**	**Hex**	**C**	**B**	V_{add}	V_{sub}	**Z**	**N**
a. 114 + 24	% 1000 1010	138_{10}	$ 8 A	0	1	1	0	0	1
b. $37 + $34	% 0 1 1 0 1 0 1 1	107	$6 B	0	0	0	0	0	0
c. $37 + $44	% 0 1 1 1 1 0 1 1	123	$7 B	0	0	0	0	0	0
d. $13 + $EC	% 1 1 1 1 1 1 1 1	255 (-1)	$F F	0	1	0	1	0	1
e. $13 + $ED	% 0 0 0 0 0 0 0 0	0	$0 0	1	1	0	0	1	0
f. $13 + $EE	% 0 0 0 0 0 0 0 1	1	$0 1	1	1	0	0	0	0
g. $83 + $96	% 0 0 0 1 1 0 0 1	25	$1 9	1	0	1	0	0	0
h. $F0 + $02	% 1 1 1 1 0 0 1 0	242 (-14)	$F 2	0	0	0	0	0	1
i. $24 – $3B	% 1 1 1 0 1 0 0 1	233 (-23)	$E 9	0	1	1	0	0	1
j. $FD – $07	% 1 1 1 1 0 1 1 0	246 (-10)	$F 6	0	0	0	0	0	1

11. b) Algebraic solution

i) In class, we determined that C = $a_7 \cdot b_7 + a_7 \cdot \overline{r_7} + b_7 \cdot \overline{r_7}$ for 8-bit addition. We can replace r with s since they both represent the sum. If the numbers are only 1-bit wide, we would find C = c_1 = $a_0 \cdot b_0 + a_0 \cdot \overline{s_0} + b_0 \cdot \overline{s_0}$. Also, remember the sum bit is $s_0 = a_0 \oplus b_0 \oplus c_0$.

ii) By inspecting the Full Adder and Half Adder schematics, you can see the carry-out for the full adder is $c_1' = a_0 \cdot b_0 + c_0 \cdot (a_0 \oplus b_0)$.

iii) The goal below is to show that $c_1 = c_1'$ using *algebraic manipulation*.

Start with $c_1 = a_0 \cdot b_0 + a_0 \cdot \overline{s_0} + b_0 \cdot \overline{s_0}$, and manipulate it to show that $c_1 = c_1'$.

Notice $s_0 = a_0 \oplus b_0 \oplus c_0 \rightarrow \overline{s_0} = s_0 \oplus 1 \rightarrow a_0 \overline{s_0} = a_0(a_0 \oplus b_0 \oplus c_0 \oplus 1)$.

If $a_0 = 0$ then $a_0 \overline{s_0} = 0$. In that case, the $a_0 \overline{s_0}$ term is unable to set $c_1 = 1$.

If $a_0 = 1$, then $a_0 \overline{s_0} = a_0(a_0 \oplus b_0 \oplus c_0 \oplus 1) = 1(1 \oplus b_0 \oplus c_0 \oplus 1) = b_0 \oplus c_0$.

Hence, we can simplify $a_0 \overline{s_0} = a_0(b_0 \oplus c_0)$.

Similarly, the $b_0 \overline{s_0}$ term in c_1 can be simplified to $b_0 \overline{s_0} = b_0(a_0 \oplus c_0)$.

Rewriting c_1 becomes:

$$
\begin{aligned}
c_1 &= a_0 b_0 + a_0(b_0 \oplus c_0) + b_0(a_0 \oplus c_0) \\
&= a_0 b_0 + a_0(b_0 \overline{c_0} + \overline{b_0} c_0) + b_0(a_0 \overline{c_0} + \overline{a_0} c_0) \\
&= a_0 b_0 + a_0 b_0 \overline{c_0} + a_0 \overline{b_0} c_0 + a_0 b_0 \overline{c_0} + \overline{a_0} b_0 c_0 \\
&= a_0 b_0 + a_0 b_0 \overline{c_0} + a_0 b_0 \overline{c_0} + c_0(a_0 \overline{b_0} + \overline{a_0} b_0) \\
&= a_0 b_0 (1 + \overline{c_0} + \overline{c_0}) + c_0(a_0 \oplus b_0) \\
&= c_1'
\end{aligned}
$$

Since $c_1 = c_1'$, the notes are correct.

You won't have to do something this hard for the quiz or exam.