Solutions to Homework 1 (Fall 2014)

COMP 3350-002

1a. $\overset{}{\underset{7}{0}}\ \underset{6}{0}\ \underset{5}{1}\ \boxed{\underset{4}{1}\ \underset{3}{0}\ \underset{2}{0}}\ \underset{1}{0}\ \underset{0}{1}$

1b. The MSB is 0

1c. $\underbrace{0011}_{=3h}\ \underbrace{0001}_{=1h}$ $\quad \boxed{31h}$

2.
$$\begin{array}{r}
\text{Carries}\rightarrow \quad\ \ 1\ 1\ 1\quad \\
00000101 \\
+\,11100111 \\
\hline
11101100
\end{array}
\qquad
\begin{array}{r}
5 \\
+\,231 \\
\hline
236
\end{array}$$

3a. $\underline{1}0000100$ Sign bit is 1, so find 2's complement

$$\begin{array}{lr}
\text{Flip Bits:} & 01111011 \\
\text{Add One:} & +\qquad\quad 1 \\
\hline
 & 01111100 \Rightarrow
\end{array}$$

$2^6+2^5+2^4+2^3+2^2$

$=64+32+16+8+4$

$=124$

So this represents $\boxed{-124}$

3b. $\underline{0}1000000$ Sign bit is 0, so value is positive

$2^6=64$

So this represents $\boxed{64}$

3c. $\underline{1}1111111$ Sign bit is 1, so take 2's complement:

$$\begin{array}{lr}
\text{Flip Bits:} & 00000000 \\
\text{Add One:} & +\qquad\quad 1 \\
\hline
 & 00000001
\end{array}$$

So this represents $\boxed{-1}$

SOLUTIONS TO HW1 (Fa14), CTD

4a.

```
        0 R 1
  2 ) 1   R 0
  2 ) 2   R 1
  2 ) 5   R 0
  2 ) 10  R 1
  2 ) 21  R 0
  2 ) 42
```

42 is 101010b

Take 8-bit 2's complement to find the representation of -42:

```
              00101010
Flip Bits:    11010101
Add One:   +         1
           -----------
              11010110
```

So -42 is represented as $\boxed{11010110b}$

4b. 42 is 00101010b — see #4a

4c. 128 is $2^7 \Rightarrow 10000000_2$

To find -128's representation, flip the bits & add one:

```
   1111111
   01111111
 +        1
 ----------
   10000000
```

-128 is represented as $\boxed{1000\ 0000b}$

5. 3.2 GHz = 3.2×10^9 Hz, so one clock cycle is $\frac{1}{3.2 \times 10^9}$ seconds.

1 ns = 10^{-9} s.

So: $\frac{1}{3.2 \times 10^9} = x \cdot 10^{-9} \Rightarrow x = \frac{1}{3.2} = 0.3125$ ns

SOLUTIONS TO HW1 (Fa14), CTD

6a. 9Ch ⇒ Sign bit is 1, so take 2's complement

Flip the bits (subtract each digit from 15): 63h
Add one: $\boxed{64h}$

$64h = 6 \cdot 16 + 4 = 100$

So 9Ch represents $\boxed{-100}$

6b. 31h ⇒ Sign bit is 0

$3 \times 16 + 1 = 49$, so 31h represents $\boxed{49}$

7a. −47 Convert 47 to hex: 47=2Fh

$$\begin{array}{rl} & 0 \text{ R } 2 = 2_{16} \\ 16 \,\big) & 2 \text{ R } 15 = F_{16} \ \downarrow \\ 16 \,\big) & 47 \end{array}$$

Flip bits: D0h
Add one: D1h

So −47 is represented as $\boxed{D1h}$

7b. 47 is 2Fh — see solution to #7a

8a. No! (trailing NUL indicates end-of-string)

8b. Yes. UTF-8 is backwards-compatible with ASCII: Every ASCII string is also a valid UTF-8 representation of the same string.

SOLUTIONS TO HW1 (Fa14), CTD

9a. Minimum: 0 Maximum: $\underline{2^{73}-1} \approx 9.4447 \times 10^{21}$

9b. Minimum: -2^{72} Maximum: $2^{72}-1$

10. No. The range of 4-bit signed integers is $[-2^3, 2^3-1] = [-8, 7]$.
" " " 5-bit " " " $[-2^4, 2^4-1] = [-16, 15]$,
so it can be represented with 5 bits.

$10 = 01010_2$ Flip the bits: 10101
Add one: $+ \quad 1$
10110

So ~~As~~ the 5-bit two's complement representation of -10 is $\boxed{10110b}$

11, 12. See next page

11. Consider the 4-bit case first. The place values are

$$\underline{}\ \underline{}\ \underline{}\ \underline{}$$
$$-8 \quad 4 \quad -2 \quad 1$$

so the even-numbered bits (0 and 2) add to the value, while the odd-numbered bits subtract from it. So, the smallest value is $1010_{-2} = -10$, and the largest value is $0101_{-2} = 5$. Extrapolating to the 16-bit case, the smallest representable value is $1010101010101010_{-2} = (-2)^{15} + (-2)^{13} + (-2)^{11} + \ldots + (-2)^1 = -43{,}690$, and the largest is $0101010101010101 = (-2)^{14} + (-2)^{12} + (-2)^{10} + \ldots + (-2)^0 = 21{,}845$.

12. Joe Bob's design puts the Store Output Operand step at the "wrong" place in the FDX cycle. Consider an operation that adds two numbers, then stores the sum in memory. The ALU will add the numbers during the Execute step. The Store Output Operand step is where the sum would be stored in memory. However, Joe's step moves the Store Output Operand step *before* the Execute step. It is impossible to store the output operand if it hasn't been computed yet!

(In your answer to this problem, we were mainly looking for an indication that you understand the steps of the FDX cycle. In particular, when an input to an instruction comes from memory, it is retrieved during the Fetch Operands step; when an instruction's result is to be stored in memory, the result is computed by the ALU during the Execute step and then is stored in memory during the Store Output Operand step.)

(Don't worry if you don't get this... this is for completeness...) Now, if you're really clever, you might say, "Well, it can store the output operand during the *next* instruction cycle," like this:

1. Fetch instruction
2. Decode
3. (Nothing to do for Joe's step)
4. Execute (add the two numbers in our example)

5. Fetch next instruction
6. Decode
7. Joe's step (for our example, store the sum from the *previous* instruction)
8. Execute

Unfortunately, this still doesn't quite work, at least not with a naive implementation. A problem occurs if the output operand (the sum resulting from step #4 above) is stored to the same memory location as the next instruction to fetch (#5 above). In that case, delaying the store to step #7 would mean that the fetch (#5) would retrieve the *old* memory value, not the stored sum. So, some additional machinery would be needed to ensure that the fetch in step #5 retrieves the updated value resulting from the execute (step #4) before it is actually stored in step #7.