Assignment 1
Rounding 10nd base: binary and hex number systems
ENGN/PHYS 208—Winter 2021

Background

Everything in Electronics land is 0's and 1's. ***Binary*** and ***hexadecimal*** ("***hex***") numbers are thus natural representations in electrnoics land. Here we'll explore these number systems and then play a bit an Arduino or other similar MCU to see how this all works in the real world. Let's get this electronics party started!

1 In Theory

1. **Hex, Binary, Decimal Representation:** We've discussed in class that many sensor devices utilize unique ***hex addresses*** to communicate with the host microcontroller unit (MCU; e.g., Arduino on Adafruit Feather board). This allows a single MCU to properly route digital traffic to and from multiple sensors. For example, imagine you connect 2 inertial measurement units (IMUs) to a robot. Sensor 1 has an address of 0x68 and Sensor 2 has an address of 0x9C.

   (a) Convert hex addresses for both sensors into binary form. We'll play a bit with Arduino later to have it print out the hex, binary, and decimal versions.

   (b) Convert both addresses into decimal form. Show your method of calculation.

   Take home message: decimal, binary, and hex, are just three different representations of the same number; always good to get some a quick bit of practice converting back and forth; make sure you feel comfortable going between all 3 number systems.

2. **Binary Addition and Representation:** Imagine you have two binary numbers loaded in memory as 2 individual ***bytes*** (8-bit numbers): A = 0b10001100 and B = 0b00110110. These numbers could result from 2 successive Arduino analog reads from a vibrational sensor. Perhaps we'd like to average the readings, which involves computing a sum.

   (a) Compute the sum $C = A + B$, expressing your result as an 8-bit binary number (byte). Show details of each addition operation—i.e. indicate the sum and carry bits as you move from the ***LSB*** to ***MSB***. Note the approximate amount of time it takes you, the human, to compute the result and record your answer.

   (b) Convert A, B, and C in decimal (base 10) form. Do this ***by hand first***. Of course, you can check your answer using a binary to decimal converter available online, using the Arduino, or other software package.

   (c) What is the maximum decimal value that can be represented by one byte?

   (d) Are the binary and decimal representations of the result C in agreement? (Hint: they should be but you should verify).

(e) Approximately how long should it take the Arduino to compute the sum $A + B$? We know that the Arduino Uno has a clock speed of 16 MHz. Assume that on each clock pulse, the ATMega328 microcontroller unit can execute the summing operation on 2 bits (e.g., $A_0 + B_0$). A bit later on, we'll find out just how much time it actually takes the Arduino to compute, so we can compare our theory estimate to reality!

3. **Addition/Numeric Overflow:** Now imagine you have two new bytes (perhaps the next 2 analog reads from a force sensor) $D =$ 0b10001101 and $E =$ 0b11101100.

   (a) Convert D and E to decimal form, and compute the sum $F = D + E$.

   (b) Compute the sum $F = D+E$, expressing your result in as a *single byte*. Show the details of your arithmetic operations.

   (c) Convert your binary answer back to decimal form. Do this by hand first; you can check your answer with a binary to decimal converter online after that to verify.

   (d) Are the decimal and single byte representations in agreement? (Hint: they shouldn't be). If not, why not?

   (e) What would be an acceptable Arduino data type to properly represent F that requires the least amount of memory?

2 In Practice: Microcontroller Playground

The purpose of this section is 2-fold: To see how theory actually plays out, and to get you familiar/comfortable with looking up Arduino reference material to incorporate into your code ("sketches" in Arduino parlance)

1. **Counting and Numeric Overflow:**

   (a) Copy this example code for integer counting/increment.

   (b) Modify the delay time to be 10 ms instead of 1000 ms so that you don't have to wait a relatively eternity for numbers to print to the serial monitor. See the Arduino help on the delay() function, as necessary.

   (c) Compile and upload the code. View the result in the serial monitor. Notice that variable `countUp` is specified as the `int` data type. What is the maximum value that this variable can take on? What happens when Arduino tries to increment past this max value? Exceeding the maximum value is known as ***overflow***. See the "Notes and Warnings" section under the example code you previously copied.

   (d) Now change`countUp` to be a byte variable type. What is the max value that a byte can take on? What happens when Arduino tries to increment past this max value?

   (e) Repeat one last time, but this time declare `countUp` to be an unsigned int type.

   (f) Let's say have an Arduino Uno and a sensor whose maximum reading results in a decimal value of 300,000. What is the appropriate variable type to use in this case? Of course, you could declare everything as double type, but that will waste precious finite memory resources. The moral of the story is that you should carefully choose the variable type when coding!

2. **Numeric Representation and Human-Readable Presentation:** Examine this example code which demos various data representations. Copy and paste the example code. Modify it to include to show the the contents printed in the serial monitor for variable x using the Serial.write() function.

   For your assignment submission include:

   (a) Screen shot of serial monitor output (or copy and paste contents into excel or word)

   (b) Brief explanation of the results obtained.

3. **Rockin' Round the Clock—Arduino Computation Time:** How much time is required to sum 2 numbers using Arduino Uno? Does Arduino always take the same amount of time to sum 2 numbers? Let's do the experiment...Write code to find out!

   Helpful hints: Firstly, you'll need to measure the amount of time that elapses. See help for the functions millis() and micros(). Note that you can use `micros()` repeatedly anywhere you want. For example, you could use these timestamp functions immediately before and immediately after computing a sum, the compute the difference to find the elapsed time.

   To answer the second question ("Does this operation always require the same amount of time?") you will need to do the perform N iterations of the sum operation and compute how long it takes at each iteration. Something like $N = 100$ iterations would be sensible/appropriate. So you'll need to set up a for() loop and store the result in an array and/or output the results to the serial monitor. See the reference for Serial.println() function.

To summarize, you need to develop code that:

(a) Uses timestamps to compute the time it takes Arduino to compute the sum of two numbers (declared as byte type).

(b) print the results to the serial monitor so that you can perform further statistical analysis. Namely, report the mean, median, and standard deviation of the computation time.

(c) Optionally, Program the Arduino to do the math for you. There are some wonderful helpful tidbits for computing basic statistics in the example code linked here. In summary, you'll make a new sketch cobbled together from various bits and pieces of existing code. Good coding practice is to borrow from others. Or as the famous composer Igor Stravinsky quipped: "Good composers borrow; great composers steal!"

What you need to submit for this problem:

(a) Your code (preferably the .ino file, be sure to ***put your last initials as file name suffix***)

(b) Summary of statistics/results obtained

(c) Brief interpretation of the results. Specifically, how did the actual computation time compare to the theoretical computation time? (See problem 2e).

(d) What causes computation time in the Arduino to be vary from one iteration to the next? That is, why isn't the computation time always exactly, say, 4 μs? For that matter, what is the smallest discrete chunk of time you can discern with your MCU? What other processes might be going on that the Arduino must attend to. We'll see later in the term the critical difference between ***software-timed*** and ***hardware-timed*** loops.