CSCI 2500 — Computer Organization
Lab 07 (document version 1.1)

- This lab is due by the end of your lab session on Wednesday, October 16, 2019.
- This lab is to be completed **individually**. Do not share your code with anyone else.
- You **must** show your code and your solutions to a TA or mentor to receive credit for each checkpoint.
- Labs are available on Mondays before your lab sessions. Plan to start each lab early and ask questions during office hours, in the discussion forum on Submitty, and during your lab session.

1. **Checkpoint 1:** For the first checkpoint, you will make use of integer division in MIPS to help a cashier make change using only coins, i.e., only quarters, dimes, nickels, and pennies. More specifically, write a procedure called `change` that accepts as input in register `$a0` an unsigned positive integer. This value is the number of cents to make change for. As an example, if this value is `137`, then change must be made for $1.37, which would be five quarters, one dime, and two pennies.

   To figure this out using MIPS, in your `change` procedure, use the `divu` instruction to first determine how many quarters you need. For the `137` example, dividing `137` by `25` cents gives you a quotient of `5` and a remainder of `12`. The quotient here indicates that you need five quarters, whereas the remainder is then used to determine the number of dimes, nickels, and pennies.

   Note that the `divu` instruction only has two operands. And it produces both a quotient and a remainder. More specifically, it stores the quotient in a special `Lo` register and the remainder in a `Hi` register. You therefore need to use the *Move From Hi* (`mfhi`) and *Move From Lo* (`mflo`) instructions accordingly.

   For example, the code below divides 5 by 3, yielding a 1 in `$t2` with a remainder of 2 in `$t3`:

   ```
   li $t0,5
   li $t1,3
   divu $t0,$t1
   mflo $t2
   mfhi $t3
   ```

   Output your results using the format shown below. Note that when you output `$1.37`, you can either use integer manipulation to display exactly `$1.37` or you can use floating point numbers, which will instead display `$1.37000000`. Either is fine.

   ```
   Enter positive integer:
   137
   For $1.37, you need:     --or--    For $1.37000000, you need:
   5 quarters
   1 dime
   2 pennies
   ```

   Feel free to use a loop to ask the user for more inputs. This will make testing quicker. And carefully match the output format above.

2. **Checkpoint 2:** For the second checkpoint, download the `adder.c` code, which partially implements a one-bit adder. After all the MIPS coding we've been doing, this might be the first time in your life that you're happy to see C code!

   For this checkpoint, fill in C code for the `add_two_bits()` function.

   Also, to receive full credit for this checkpoint, describe what the following line of code does (and why it "cleans" the input):

   ```
   d0 = !!d0;
   ```

   Note that in C, there are a variety of bitwise operations that we can use for this checkpoint. Remember that we can implement a logical bitwise `AND` via the `&` operator. How is this different from the better-known `&&` operator?

   Further, we can implement a logical bitwise `OR` via the `|` operator. Again, how is this different from the better-known `||` operator?

   We can also implement a logical bitwise `XOR` via the `^` operator. And to implement a logical bitwise `NOT`, we use the familiar `!` operator (the only unary operator).

3. **Checkpoint 3:** For the third checkpoint, begin by sketching a two-bit adder, showing the proper carry in and carry out lines connected up correctly to handle two-bit addition. Once you understand how the one-bit adders combine together, implement a four-bit adder in C by only making use of your `add_two_bits()` code from the second checkpoint.

   More specifically, in `main()`, fill in the following code:

   ```
   int main()
   {
     int i;
     int d0[4];
     int d1[4];
     int ci[4];
     int sum[4];

     /* Call add_two_bits() multiple times to implement a four-bit adder: */




     return EXIT_SUCCESS;
   }
   ```