This lab investigates the theoretical limits of Instruction Level Parallelism (ILP).

1. Suppose you have an *ideal* processor which has the following attributes:

   i. There are an unlimited number of Reservation Stations (RS) and unlimited space in the Reorder Buffer (ROB). So, effectively, there is an unlimited number of temporary registers available for renaming.

   ii. Branch prediction is as close to 100% as needed and is done by the end of instruction fetch. So it is possible to fetch as many instructions as needed up to and including a branch (but not beyond). The Reorder Buffer never needs to be flushed of incorrectly fetched instructions.

   iii. All jumps (including function return, i.e. jump register) can be predicted as close to 100% as needed.

   iv. Effective memory addresses are known so that a load can be moved before a store provided the addresses are different.

   v. The hit rate of the L1-caches is as close to 100% as needed, so that all loads and stores can be done in a *fixed* number of clock cycles, and there is no limit on the number of these active at any given time.

   vi. There are sufficient local buses and internal data paths so that any needed transfers (e.g. execution units to/from Reservation Stations) can be done in one clock cycle without conflict.

   vii. There is no upper limit on the number of instructions which can be issued in one clock cycle (to the Reservation Stations and Reorder Buffer). There are as many functional units as needed.

Optimization with this ideal processor would mean that every instruction was executed *as early as possible*, limited only by the data dependencies.

```
addi $t3, $zero, 5
loop:
  l.d $f2, 0($t1)
  mull.d $f3, $f2, $f0
  l.d $f4, 0($t2)
  add.d $f5, $f4, $f3
  s.d $f5, 0($t2)
  addui $t1, $t1, 8
  addui $t2, $t2, 8
  addi $t3, $t3, -1
  bne $t3, $zero, loop
```

**Assignment** Consider the above loop (from Lab 6) with 5 iterations. Can you run it any faster on the ideal processor than you ran it before (on Lab 6)? Keep the same timing assumptions but allow multiple fetch and issue. So, it takes one clock cycle to fetch *n*
consecutive instructions in order, one clock cycle to decode the \( n \) instructions, one clock cycle to issue up to \( n \) instructions, and one clock cycle to commit up to \( n \) results. Note that if a branch is present it must be the last instruction fetched. So you can fetch \( n \) consecutive instructions stopping only at a branch. Assume that all data is in the L1 data cache and that this can be read/written in one clock cycle. Assume that integer addition and subtraction can be done in one clock cycle, that floating point addition takes 4 clock cycles, and that floating point multiplication takes 7 clock cycles.