CMPT 295 Assignment 7 Solutions (2%)

1. [3 marks] *Pipeline Analysis*

   - Each stage will cost $600/n$ ps, meaning a min clock cycle of $25 + 600/n$ ps.
   - The instruction will take n cycles, so the latency will be $n(25 + 600/n) = 25n + 600$ ps.
   - The max throughput will be $1000/(25 + 600/n) = 1000n/(25n + 600) = 40n/(n + 24)$ GIPS.

   Thus for part (a), i.e., $n = 1$, min cycle time = 625 ps, latency = 625 ps, and max throughput = 1.6 GIPS.

   For part (b), i.e., $n = 8$, min cycle time = 100 ps, latency = 800 ps, and max throughput = 10 GIPS.

2. [2 marks] *Serial vs Pipelined Instructions*

   The throughputs of both machines are the same: 1/450 ps = 2.2 GIPS.

   The serial machine has just 1 stage of 450 ps, but the 5-stage machine has a latency of $5 \times 450 = 2250$ ps.

3. [4 marks] *Oddly-lengthed Stages*

   (a) [1 mark] Trying all possible placements gets a breakage of $(40, 560)$, $(120, 480)$, $(220, 380)$, $(370, 230)$, $(530, 70)$. The best of these is $(370, 230)$, i.e., between stages D and E, and yields a min clock cycle time of $370 + 25 = 395$ ps and a max throughput of 2.53 GIPS.
   (Latency = 790 ps)

   (b) [1 mark] The best you can do is to place them between stages C, D and D, E to give stages of length $(220, 150, 230)$. The min clock cycle time is now $230 + 25 = 255$ ps and the max throughput is 3.92 GIPS.
   (Latency = 765 ps)

   (c) [1 mark] The min clock cycle time would be $160 + 25 = 185$ ps and the max throughput is 5.41 GIPS.
   (Latency = 1110 ps)

   (d) [1 mark] You should divide stage E. If they can manage anywhere from a $(10, 150)$ split to a $(70, 90)$ split, you can reduce the clock cycle time to $220 + 25 = 245$ ps and increase the max throughput to 4.08 GIPS.
   (Latency = 735 ps)

4. [3 marks] *Persistent Dynamic Memory*

   Local variables are declared on the stack, but they do not persist, i.e., when the function call returns, all local variables are recycled along with the rest of the stack frame.

   Therefore, returning a pointer to the this local variable will refer to its location on the stack. It is likely that any subsequent usage of the stack, e.g., for the local variables used by `printf()`, would overwrite that location and change the value of `x` without an explicit assignment statement.

5. [8 marks] *Linked Lists and Pipeline Stalls*

   (b) [2 marks] The *critical path* of dereference and test, dereference and test, dereference and test, ... can be very long for a linked list. In other words, to access the k^{th} element requires you to access and dereference elements $1, \ldots, k-1$. The dependence of `movq 8(%rdx), %rdx` with the `movq 8(%rdx), %rdx` *of the next loop* is what causes the delay: `rdx` must be loaded before it can be used (dereferenced) on the next loop.

   This isn't so for the array version. There are the same number of memory references, but they don't depend on each other. Thus the processor can run them in parallel.