The remaining problems on this assignment require analysis of the two functions, adapted from the HW4 solutions. Except for \texttt{sll} all of the instructions are supported by the pipelined processor from the lecture slides. The \texttt{sll} instruction will flow through the pipeline in the same manner as an \texttt{addi}.

The first is \texttt{manhattan(dist)}:

\begin{verbatim}
i0: slt $t0, $a2, $a0  
i1: beq $t0, $0, i4  
i2: sub $t0, $a0, $a2  
i3: beq $0, $0, i5  
i4: sub $t0, $a2, $a0  
i5: slt $t0, $a3, $a1  
i6: beq $t0, $0, i9  
i7: sub $t1, $a1, $a3  
i8: beq $0, $0, i10  
i9: sub $t1, $a3, $a1  
i10: add $v0, $t0, $t1
\end{verbatim}

The second is \texttt{edge_count_iter}:

\begin{verbatim}
0_start: addi $t0, $0, 0  
i1: addi $t1, $0, -1  
i2: addi $v0, $0, -1  
i3: beq $t0, $a1, i17  
i4: sll $t2, $t0, 2  
i5: add $t2, $a0, $t2  
i6: lw $t2, 0($t2)  
i7: beq $t2, $t1, i15  
i8: sll $t2, $t2, 2  
i9: add $t2, $a0, $t2  
i10: lw $t3, 0($t2)  
i11: beq $t3, $t1, i15  
i12: addi $v0, $v0, 1  
i13: addi $t2, $t2, 4  
i14: beq $0, $0, i10  
i15: addi $t0, $t0, 1  
i16: beq $0, $0, i3  
i17: ndone:
\end{verbatim}
1. Execution Trace of \texttt{manhattan\_dist}, when called on (0,0) and (5,5) on a five-stage pipeline with no forwarding. Assumption: \texttt{beq} resolves on D.

\begin{verbatim}
i0: slt $t0, $a2, $a0  F D E M W
i1: beq $t0, $0, i4  F D D D E M W
i2: sub $t0, $a0, $a2  F F F D E M W  Flush
i4: sub $t0, $a2, $a0  F D E M W
i5: slt $t0, $a3, $a1  F D E M W
i6: beq $t0, $0, i9  F D D D E M W
i7: sub $t1, $a1, $a3  F F F D E M W  Flush
i9: sub $t1, $a3, $a1  F D E M W
i10: add $v0, $t0, $t1  F D D D E M W
\end{verbatim}

2. Execution Trace of \texttt{manhattan\_dist}, when called on (0,0) and (5,5) on a fully-bypassed pipeline with forwarding. Assumption: \texttt{beq} resolves on D.

\begin{verbatim}
i0: slt $t0, $a2, $a0  F D E M W
i1: beq $t0, $0, i4  F D D D E M W
i2: sub $t0, $a0, $a2  F F F D E M W  Flush
i4: sub $t0, $a2, $a0  F D E M W
i5: slt $t0, $a3, $a1  F D E M W
i6: beq $t0, $0, i9  F D D D E M W
i7: sub $t1, $a1, $a3  F F F D E M W  Flush
i9: sub $t1, $a3, $a1  F D E M W
i10: add $v0, $t0, $t1  F D D D E M W
\end{verbatim}
3. (20 pts.) Imagine how edge_count_iter would execute on the fully bypassed pipeline. List all pairs of instructions between which one or more bubbles would occur. If a bubble occurs between i3 and i4, then you should write $\rightarrow$. (HINT: Think systematically through all scenarios that result in an empty slot in the pipeline.)

Bubbles occur when:

(a) A \texttt{lw} is immediately followed by its consumer
- $i6 \rightarrow i7$
- $i10\_etop \rightarrow i11$

(b) A \texttt{beq} is taken
- $i3\_ntop \rightarrow i17\_ndone$
- $i7 \rightarrow i15\_edone$
- $i11 \rightarrow i15\_edone$
- $i14 \rightarrow i10\_etop$
- $i16 \rightarrow i3\_ntop$

(c) A \texttt{beq} immediately follows its producer (2nd bubble if that producer is a \texttt{lw})
- $i3\_ntop$ as branch target, no bubbles
- $i3\_ntop$ as fall-through, no bubbles
- $i6 \rightarrow i7$
- $i10\_etop \rightarrow i11$
- $i14$ only uses $0$, no bubbles
- $i16$ only uses $0$, no bubbles
4. (20 pts.) Now list where in the program (still edge_count_iter on the fully bypassed 5-stage pipe) data operands would be forwarded, and which forwarding path would be used. If i3 forwards the future value of $a0 to i4 using the M-E forwarding path, write $a0, i3 → i4, M-E, per the example below. (HINT: Think through the values consumed by each instruction systematically.)

- $0 for 0, start, no forward
- $0 for i1, no forward
- $0 for i2, no forward
- $t0 for i3, ntop, when falling through, no forward due to distant producer
- $a1 for i3, ntop, when falling through, no forward due to distant producer
- $t0 for i3, ntop, when branch target, no forward due to distant producer and branching stall
- $a1 for i3, ntop, when branch target, no forward due to distant producer
- $t0 for i4, no forward due to distant producer
- $a0 for i5, no forward due to distant producer
- $t2 for i5, i4 → i5, M-E
- $t2 for i6, i5 → i6, M-E
- $t1 for i7, no forward due to distant producer
- $t2 for i7, no forward, resolved with stalls
- $t2 for i8, resolved with stalls
- $a0 for i9, no forward due to distant producer
- $t2 for i9, i8 → i9, M-E
- $t2 for i10, etop, when falling through, i9 → i10, etop, M-E
- $t2 for i10, etop, when target branch, no forward due to distant producer
- $t1 for i11, no forward due to distant producer
- $t3 for i11, no forward, resolved with stalls
- $v0 for i12, no forward due to distant producer
- $t2 for i13, no forward due to distant producer
- $t1 for i11, no forward due to distant producer
- $0 for i14, no forward
- $t0 for i15, edone, when target branch of i7, no forward due to distant producer
- $t0 for i15, edone, when target branch of i11, no forward due to distant producer
- $0 for i16, no forward
5. (20 pts.) Assuming a very large graph with one million nodes and an average of 500 edges per node, what is the CPI of the `edge_count_iter` code? Assume that the `beq` in `i7` is taken 2% of the time.

Separate the code into two parts:

(a) **Iterations when branch `i7` is taken (2%)**

- The dynamic instruction sequence has 7 instructions
  \{`i3_ntop`, `i4`, `i5`, `i6`, `i7`, `i15_edone`, `i16`\}
- Instructions with a CPI of 4: \{`i7`\} (beq immediately follows its producer which is `lw` resulting in 2 bubbles or a CPI of 3. In addition, the branch is taken for a total of 4.)
- Instructions with a CPI of 2: \{`i16`\} (A taken branch)
- Instructions with a CPI of 1: \{`i3_ntop`, `i4`, `i5`, `i6`, `i15_edone`\}

\[(\frac{1}{7} \times 4) + (\frac{1}{7} \times 2) + (\frac{5}{7} \times 1) = 1.5714\]

(b) **Iterations when branch `i7` is not taken (98%)**

- The dynamic instruction sequence has 5 instructions
  \{`i10_etop`, `i11`, `i12`, `i13`, `i14`\}
- Instructions with a CPI of 3: \{`i11`\} (beq immediately follows its producer which is `lw` resulting in 2 bubbles or a CPI of 3.)
- Instructions with a CPI of 2: \{`i14`\} (A taken branch)
- Instructions with a CPI of 1: \{`i10_etop`, `i12`, `i13`\}

\[(\frac{1}{5} \times 3) + (\frac{1}{5} \times 2) + (\frac{3}{5} \times 1) = 1.6\]

- The 'modified' or more specific dynamic instruction will also be accepted. The sequence has 2507 instructions:

  Entering loop: \{`i7`, `i8`, `i9`\}
  Loop: \{`i10_etop`, `i11`, `i12`, `i13`, `i14`\} each 500 times.
  Exiting loop: \{`i10_etop`, `i11`, `i15_edone`, `i16`\}
- Instructions with a CPI of 4: \{`i11_exit`\} (beq immediately follows its `lw` producer and branches)
- Instructions with a CPI of 3: \{`i7`, `i11_loop`\} (beq immediately follows its `lw` producer resulting in 2 bubbles or a CPI of 3.)
- Instructions with a CPI of 2: \{`i14`, `i16`\} (Taken branches)
- Instructions with a CPI of 1: \{`i8`, `i9`, `i10_etoploop`, `i10_etopexit`, `i12`, `i13`, `i15_edone`\}

\[
[(\frac{1}{2507} \times 3) + (\frac{2}{2507} \times 1)] + [(\frac{500}{2507} \times 1) + (\frac{500}{2507} \times 3) + (\frac{500}{2507} \times 1) + (\frac{500}{2507} \times 1) + (\frac{500}{2507} \times 2)]
+ [(\frac{1}{2507} \times 1) + (\frac{1}{2507} \times 4) + (\frac{1}{2507} \times 1) + (\frac{1}{2507} \times 2)] = 1.6007
\]

Overall CPI = (.02 \times 1.5714) + (0.98 \times 1.6) = 1.599428 \approx 1.6