Question 1 (6 points):

Labelling the definitions of A and C as A2 and C2 in this basic block,

\[
\begin{align*}
A2 &= \ldots \\
C2 &= \ldots \\
A &= A2 + C2 \\
\ldots &= C2 \\
C &= \ldots 
\end{align*}
\]

we compute the live ranges (and merged live ranges). Then, we have the following interference graph which can be filled using three colours:

![Interference Graph](image)

2 for A, B, C with correct colours
2 + 2 for A2, C2

Question 2 (9 points):

(a) The best throughput is \( \frac{1}{7} \) (7 clock cycles per iteration), with an example schedule as: A; nop; B; C; E; D; F (1 point)

(b) The initiation interval lower bound is 3. The resources constraints induce a lower bound of 3 on the initiation interval because each resource is used by 3 instructions in the loop. Both cycles in the precedence constraint graph have a length of 3 cycles so there is another lower bound of 3. (\( \frac{1}{2} \) point each for including resource constraint, precedence constraint, final answer, having both resource and precedence, 2 total)
(c) No, the pipelining algorithm described in class will schedule A and B such that it is impossible to schedule both SCCs with an initiation interval of 3 cycles. It schedules greedily and then is unable to backtrack. (4 points)

(i) N/A (0 points)

(ii) Yes, there is a schedule for an initiation interval of 3 cycles. The modulo resource table is as follows. An example schedule is: A; nop; nop; nop; B; C; E; D; F (2 points)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
</tr>
</tbody>
</table>

**Question 3: (13 points)**

(a) Label L3 is the only block which can benefit from global instruction scheduling. However, because it is so rarely executed most changes will have a net increase on the overall expected runtime of the block. For p = 0.01, we can effectively move code upwards from L3 to L1 (since L3 will be executed 99% of the time if L1 is executed), but not to L0 or L2. The best schedule is as follows:
We can calculate the expected execution time of the optimized program by adding up the number of instructions on each path multiplied by the probability of executing that path.

Path 1: L0 -> L1 -> L4
2 + 3 + 1 = 6 instructions
.01 * .01 = .0001 probability

Path 2: L0 -> L1 -> L3 -> L4
2 + 3 + 7 + 1 = 13 instructions
.01 * .99 = .0099 probability

Path 3: L0 -> L2 -> L2' -> L3 -> L4
2 + 2 + 3 + 7 + 1 = 15 instructions
.99 * .01 = .0099 probability

Path 4: L0 -> L2 -> L4
2 + 2 + 1 = 5 instructions
.99 * .99 = .9801 probability
The expected instruction count is 5.1783 (a speedup of .2% over the original program).

2 pts for identifying the hot path and moving instructions to L1
1 pt for balancing instructions on the other side
1 pt for identifying the probability of paths
1 pt for expected instruction count

(b) Similarly for \( p = .99 \) we can only move code upwards from block L3 to L2 without incurring a net increase in the program’s execution time. The best schedule is as follows:

The expected execution time of the new program is as follows:

Path 1: L0 -> L1 -> L4
2 + 1 + 1 = 4 instructions
.99 * .99 = .9801 probability

Path 2: L0 -> L1 -> L1' -> L3 -> L4
2 + 1 + 4 + 6 + 1 = 14 instructions
.99 * .01 = .0099 probability

Path 3: L0 -> L2 -> L3 -> L4
2 + 4 + 6 + 1 = 13 instructions
.01 * .99 = .0099 probability

Path 4: L0 -> L2 -> L4
2 + 4 + 1 = 7 instructions
.01 * .01 = .0001 probability

The expected instruction count is 4.1884 (a speedup of .5% over the original program).

2 pts for identifying the hot path and moving instructions to L2
1 pt for balancing instructions on the other side
1 pt for identifying the probability of paths
1 pt for expected instruction count

(c) The probability of paths being executed are 0.16, 0.24, 0.24, 0.36. The expected instruction count for part (a) is 9.48 and for part (b) is 9.64. The expected instruction count for original program is 9.4. The original program is marginally better than the extreme optimizations.

1 pt for identifying probability of paths
3 x 0.5 pts for computing the expected time for the 3 programs
0.5 pt for a comparison

Question 4: (12 points)

(a) The hP tuples inferred in context-insensitive analysis are: (0.5 pts each 14 x 0.5 = 7pts)

hP(h1, a, h2), hP(h1, b, h1)
hP(h1, a, h3), hP(h1, b, h2)
hP(h3, a, h1), hP(h1, b, h3)
hP(h3, a, h2), hP(h2, b, h1)
hP(h3, a, h3), hP(h2, b, h2)
  hP(h2, b, h3),
  hP(h3, b, h1)
  hP(h3, b, h2),
  hP(h3, b, h3)

(b) The hP tuples inferred in context-sensitive analysis are: (0.5 pts each 6 x 0.5 = 3pts)

hP(h1, a, h2), hP(h1, b, h1)
hP(h3,a,h3), hP(h3,b,h3)
hP(h3,a,h1), hP(h2,b,h2)

(c) 2 (1 pt)

(d) 4 (1 pt)