Directions:

• Submit via Gradescope.

• You may use up to two of your remaining late days for this assignment, for a late deadline of February 28th, 2020 at 4:30 pm.

• Remember to complete the corresponding Gradiance quizzes by the start of class on the due date. **There are no late days for Gradiance.**

• This is an individual assignment. You are allowed to discuss the homework with others, but you must write the solution individually. If you look up any material in the textbook or online, you should cite it appropriately.
**Problem 1.** Software Pipelining

Consider the following loop:

```c
for (i = 1; i < n; i++) {
    X[i+1] = X[i] + Y[i];
    Y[i+3] = X[m] * 2;
}
```

$m$ in the above code is some constant. Assume LD and ST take 1 clock, ADD and MUL take 2 clocks. The machine has two MEM units that can execute one LD and one ST, and two ALU units that can execute ADD and MUL in one clock. The machine can autoincrement address registers.

1. Draw the data dependence graph by showing four types of nodes, LD, ST, ADD, and MUL.

   First we write down the assembly code corresponding to this loop (assuming R0, R1, and Rm are properly initialized pointers):

   ```
i1: LD R2, *(R0++)
i2: LD R3, *(R1++)
i3: ADD R2, R3, R4
i4: ST *(R0), R4
i5: LD R5, *(Rm)
i6: MUL R5, $2, R6
i7: ST *2(R1), R6
   ```
2. Label the edges of the dependency graph according to the type of dependency (true dependency, anti-dependency or output dependency).

Node 5 to node 4 is a WAR dependency (anti). All other edges are RAW dependencies (true).

3. What is the lower bound on the initiation interval?

   Based on precedence constraints, 4, due to the loop of length 4 and iteration difference 1. Based on resource constraint, 3 since there are three loads. In conclusion, 4.
Problem 2. Dependency Analysis and Parallelization

for (i = 1; i < n; i++) {
    for (j = i+1; j < i+m; j++) {
        X[4*i, j] = X[8*i+1, j] + Y[i, j];
        Y[j-2, i] = X[8*i+3, j] + Y[i, j];
    }
}

1. Draw the iteration space for this loop.

Solution:

![Iteration Space Diagram]

2. What are the data dependencies in this loop? Hint: a data dependency is something of the form “read/write A[i], read/write B[j].”

write X[4*i, j], read X[8*i+1, j] (optional due to odd/even indices)
write X[4*i, j], read X[8*i+3, j] (optional due to odd/even indices)
write Y[j-2, i], read Y[i, j]
3. What are the data dependency equations that must not be satisfied for the loop to be parallelizable?

\[
\begin{align*}
1 \leq i &< n \\
1 \leq i' &< n \\
i &< j < i + m \\
i' &< j' < i' + m \\
j - 2 &= i' \\
i &= j' \\
\begin{bmatrix} i \\ j \end{bmatrix} &\neq \begin{bmatrix} i' \\ j' \end{bmatrix}
\end{align*}
\]

Optional equations (same bounds for \(i, i', j, j'\)):

\[
\begin{align*}
4 \times i &= 8 \times i' + 1 \\
j &= j'
\end{align*}
\]

\[
\begin{align*}
4 \times i &= 8 \times i' + 3 \\
j &= j'
\end{align*}
\]

4. Is this loop nest 1 or 2-d parallelizable without loop transformations? You do not need to show the exact solution to the equations, but justify your answer.

The inner loop is parallelizable as \(Y[]\) writes and reads cannot overlap for fixed \(i\) indices as \(j > i\), therefore the second dimension indices will never overlap. \(X[]\) reads and writes never overlap since writes are to even indices while reads are to odd indices. Writes can never conflict with themselves in the innermost loop as they are uniquely indexed by the outer loop index.

The outer loop is not parallelizable as writes and reads to \(Y[]\) may conflict, for example \((i=2, j=3)\) reads \(Y[2, 3]\) while \((i=3, j=4)\) writes to \(Y[2, 3]\).
Problem 3. Global Instruction Scheduling Assume you have a statically scheduled machine that can only issue one operation every clock. All operations have a latency of one clock cycle, with the exception of its memory load operation, which has a latency of three clock cycles. Consider the following locally scheduled program:

Assume that only \( r \) is live at the end of the program. Each branch in the flow graph is labeled with the probability that it is taken dynamically. To answer the following, you may apply any of the code motions discussed in class, but no other optimizations.

1. Is this the best globally scheduled code that can be generated given that \( p = 0.1 \)? If not, provide the improved code along with its expected execution time.
Path1: L0,L1,L4: 9 instructions
Path2: L0,L2,L4: 8 instructions
Path3: L0:L2,L3,L4: 12 instructions.
The expected execution time is 9 * 0.9 + 8 * 0.09 + 12 * 0.01 = 8.94 clock cycles.

2. Repeat part 1 given that $p = 0.5$.

In this case the code cannot be further improved as speculatively executing either $w = *h$ or $g = *f$ will not result in runtime gain.

3. What is the percentage improvement in execution time from the original program for your programs in part 1 and 2, if one was provided?

The original program takes $10 * 0.9 + 6 * 0.09 + 10 * 0.01 = 9.64$ clock cycles. Therefore, the reduction is runtime is roughly 7.2%.
Problem 4. More Software Pipelining

Consider the following dependence graph for a single iteration of a loop, with resource constraints:

1. What is the bound on the initiation interval $T$ according to the precedence and resource constraints for this program?

   5 due to data dependence in cycle. $\text{Max (cycle length/ iteration diff)} = 5/1 = 5$

   5 due to resource constraints as well (left column resource).

2. What is the minimum initiation interval? Show a modulo reservation table for an optimal software pipelined schedule. Also show the code and schedule for an iteration in the source loop.

   The minimum initiation interval is 5. Modulo reservation table:
3. Can the scheduling algorithm described in class produce the optimal schedule for this loop? If not, show the modulo reservation table and code schedule generated by the algorithm.

The scheduling algorithm described in class will backtrack only within strongly connected components, and schedule greedily across them. Therefore it will not produce an optimal schedule: it will pack E immediately after A (in the first available spot), eliminating the ability to schedule the D/F SCC which is a critical cycle (no slack).

The algorithm will be able to schedule instructions with an initiation interval of 6. Modulo reservation table:

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

Single iteration: A, nop, B, nop, C, nop, D, E, nop, F

We also accepted answers that scheduled E before D
Problem 5. Affine Transforms

Apply affine transform to find the largest degree of outermost loop parallelism. Show the transformed code, marking the loops that are parallelizable.

for (i = 1; i <= N; i++) {
    for (j = 1; j <= N; j++) {
        A[i,j] = A[i-1,j] + X[i,j];
    }
}

for (i = 2; i <= N; i++) {
    for (j = i+1; j <= N; j++) {
        for (k = 1; k <= N; k++) {
            B[i,k] = B[i-1,k] + Y[j];
        }
    }
}

for (i = 1; i <= N; i++) {
}

Solution:

for (pj = 1; pj <= N; pj++) { // parallelizable
    for (pi = 1; pi <= N; pi++) {
        if (2 <= pi) {
            for (j = pi+1; j <= N; j++) {
                B[pi,pj] = B[pi-1,pj] + Y[j];
            }
        }
    }
}

There is only one degree of outermost loop parallelism.