This homework relates to the following program.

```
start:  j B
A:   lw $8, 0($4)
     addi $8, $8, 1
     sw $8, 0($4)
     addi $4, $4, 4
     addi $5, $5, -1
B:  slt $2, $5, $0
    beq $2, $0, A
```

memory is initialized to 0, except addresses 0x10010000 - 0x10010100 which have data values in them.

registers are initialized to 0, except the following:

$4 has the value 0x10010000
$5 has the value 10 (decimal)

1. (2 points) what does this code do?

This code increments the values in an 11-element array starting at 0x10010000.

2. (18 points, 3 each) Complete the following table for the execution of this program. In the case of pipelined processors, assume all possible forwarding, speculative execution, and predictors initialized to not-taken.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock (MHz)</th>
<th>Cycles</th>
<th>CPI</th>
<th>Runtime (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Cycle</td>
<td>10</td>
<td>80</td>
<td>1</td>
<td>8000</td>
</tr>
<tr>
<td>Multi-Cycle</td>
<td>40</td>
<td>318</td>
<td>3.98</td>
<td>7950</td>
</tr>
<tr>
<td>No Prediction (1-cycle stall)</td>
<td>40</td>
<td>118</td>
<td>1.48</td>
<td>2950</td>
</tr>
<tr>
<td>predict not taken</td>
<td>40</td>
<td>118</td>
<td>1.48</td>
<td>2950</td>
</tr>
<tr>
<td>Pipelined</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-bit predictor</td>
<td>38</td>
<td>98</td>
<td>1.23</td>
<td>2579</td>
</tr>
<tr>
<td>2-bit predictor</td>
<td>38</td>
<td>100</td>
<td>1.25</td>
<td>2632</td>
</tr>
</tbody>
</table>

Work on next page
Single cycle:
Cycles = 3+7*11 = 80
CPI = Cycles/Instructions = 1
Runtime = 80 * 1/(10MHz) = 80*100ns = 8000ns

Multi-Cycle:
Cycles = 3+4+3+(5+4+4+4+4+4+4+3)*11 = 318
CPI = 318/80 = 3.98
Runtime = 318 * 1/(40MHz) = 311*25ns = 7950ns

Pipelined:
Without bubbles, this code would take 1 cycle per instruction, plus the cycles for the last instruction to complete. The only instructions in this program that could create bubbles are lw and beq. The lw is executed 11 times and beq 12 times (once on entry to the loop).

The instruction following lw is dependent on the loaded result, so it will consistently create a 1-cycle bubble, adding 11 cycles to the runtime in all cases. Branches are resolved in ID, so normally create a 1-cycle bubble without prediction or when predicted incorrectly.

The branch in this program, however, is dependent on the EX stage of the preceding instruction. This increases its mispredict bubble to 2 cycles. While accounted for in these solutions, this complexity was not intended. Homework assuming a 1-cycle stall from mispredict will receive full credit.

Because the last instruction in the program is a branch, we need to consider how many cycles that branch will take to complete. In the correctly predicted case, it would normally take 4 cycles, like any other instruction. However, because of the data hazard in ID it will take 5. In the incorrectly predicted case, it will only take 3 cycles since the next would begin execution after the ID phase completes (leaving EX, MEM, and WB stages) and the data hazard has already been accounted in the bubble. Similar to the branch bubble, this was an unintended complexity and a simple 4 cycle completion time will receive full credit.

no prediction:
Cycles = 80+11+12·2+3 = 118
CPI = 118/80 = 1.48
Runtime = 118 * 1/(40MHz) = 118*25ns = 2950ns

predict not taken:
Prediction is incorrect all but the last time.
Cycles = 80+11+11·2+5 = 118
same as no prediction.

1-bit prediction:
Prediction is incorrect the first and last time.
Cycles = 80+11+2·2+3 = 98
CPI = 98/80 = 1.23
Runtime = 98 * 1/(38MHz) = 98*26.3ns = 2579ns

2-bit prediction:
Assuming starting in strong-not-taken, prediction is incorrect first two and last time.
Cycles = 80+11+3·2+3 = 100
CPI = 100/80 = 1.25
Runtime = 100 * 1/(38MHz) = 100*26.3ns = 2632ns