1. [7 points]
Consider two different datapaths—a 5-cycle unpipelined datapath and a 5-stage pipelined datapath—that implement the same instruction set architecture (ISA). In both datapaths, the 5 hardware stages perform the following functions and have the specified latencies (including the latch latencies).

<table>
<thead>
<tr>
<th>Hardware Stage No.</th>
<th>Functions</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Instruction fetch</td>
<td>305 ps</td>
</tr>
<tr>
<td>2</td>
<td>Instruction decode and register file read</td>
<td>275 ps</td>
</tr>
<tr>
<td>3</td>
<td>ALU operation and data memory address computation</td>
<td>285 ps</td>
</tr>
<tr>
<td>4</td>
<td>Data memory access</td>
<td>305 ps</td>
</tr>
<tr>
<td>5</td>
<td>Register file write</td>
<td>240 ps</td>
</tr>
</tbody>
</table>

(i) (1 point) What is the minimum clock period for each datapath?

The minimum clock period for both datapaths is 305 ps, as the clock period is defined by the stage with the longest latency.

(ii) (1 point) How much time will it take to execute a single instruction in each datapath?

The execution time for a single instruction in both datapaths is $305 \times 5 = 1525$ ps, as the instruction would require 5 clock cycles.

(iii) (3 points) How many clock cycles will it take to execute a program (with $N$ dynamic instructions) in each datapath, if no pipeline hazards occurred? What is the speedup of the pipelined datapath over the unipipelined datapath?

For the 5-cycle unpipelined datapath, each instruction requires 5 clock cycles or 1525 ps; so the total execution time is $1525N$ ps.

For the pipelined datapath, after the first 4 clock cycles are over, one instruction completes every cycle; so the total execution time is $(305N + 1220)$ ps.

Speedup of the pipelined datapath $= \frac{1525N}{305N + 1220}$

(iv) (2 points) How many clock cycles will the pipelined datapath take to execute the same program (with $N$ dynamic instructions), if 10% of the instructions were branch instructions causing 2-cycle stalls each, and 2% of the instructions resulted in cache misses causing 50-cycle stalls each? Assume that there was no overlap between the stalls caused by different instructions.

For the pipelined datapath, total number of stall cycles $= 0.1 N \times 2 + 0.02 N \times 50 = 1.2 N$; so the total stall time $= 305 \times 1.2N$ ps $= 366N$ ps.

Total execution time $= (305N + 366N + 1220)$ ps $= (671N + 1220)$ ps.
2. [7 points]
Consider an existing instruction set architecture (ISA). A new instruction is added to this ISA.
(i) (4.5 points) Describe the changes (if any) that should be made to the following 3 software/hardware parts:
(a) the compiler (high-level language → machine language translator)

The compiler back-end COULD be modified to generate machine language code that uses the new instruction. In this case, executables generated by the compiler are likely to make use of the new instruction.

(b) the CPU data path

For the datapath, there are 2 possibilities:

- Make minimal or no changes to the datapath. In this case, the new instruction is implemented primarily by a sequence of microinstructions in the control unit. For instance, if the new instruction is a MULTIPLY instruction, the control unit can interpret it using a sequence of ADD operations in the existing datapath.

- Enhance the datapath to implement the functionality of the new instruction. For instance, if the new instruction is a MULTIPLY instruction, include a MULTIPLY unit in the datapath. In this case, the control unit tells the datapath’s MULTIPLY unit to perform the multiplication.

(c) the CPU control unit

The CPU control unit SHOULD be modified to interpret the new instruction. The nature of the modification depends on the modification done to the datapath. As mentioned in the answer to part (b), if the datapath undertakes extensive changes, the control unit requires minimal changes.

(ii) (2.5 points) Explain the conditions under which the addition of the new instruction to the ISA helps improve performance.

The addition of the new instruction impacts performance in multiple ways. Additions to the hardware, especially the datapath, may adversely impact the clock speed. Therefore, in order to get performance benefits, one or more of the following conditions need to be satisfied:

- Implementation of the new instruction must not significantly decrease the clock speed.
- The new instruction must not have a high CPI (cycles per instruction).
- The compiler must be able to use the new instruction frequently.
3. [6 points]
With the help of an example assembly language code, explain register dependencies and memory dependencies. In what ways are they similar? In what ways are they different?

Register dependencies are data dependencies that are specified through register names. Memory dependencies are data dependencies that are specified through memory addresses. Consider the following assembly language code snippet:

I1: R1 <-- Mem[R2]
I2: R3 <-- R1 + R3
I3: Mem[R2] <-- R3
I4: R3 <-- R2 - R4
I5: Mem[R5] <-- R3

In this code, there is a register dependency between instruction I1 and instruction I2 through register R1, as I1 writes a value into R1 and I2 reads R1. Similarly, there is a register dependency between I2 and I3 through register R3.

There is a memory dependency between I1 and I3 through the memory address contained in register R2. An optimizing compiler that performs instruction rescheduling must take both register dependencies and memory dependencies into consideration.

Register dependencies are statically known, because register addresses are explicitly stated in the instructions. By inspecting the above code, we can determine all register dependencies. Thus, an optimizing compiler can easily consider register dependencies while performing various optimizations. Memory dependencies, on the other hand, are dynamically determined because memory addresses are computed only at run-time. For instance, in the above code, we are not able to determine statically if there is a memory dependency between I3 and I5. Therefore, memory dependencies pose a hurdle to optimizing compilers.