a

Problem 1: Comparing Optimized and Unoptimized Versions

Definitions and Assumptions:

- Let T_u be the total execution time of the unoptimized version.
- Let T_o be the total execution time of the optimized version.
- Let C_u be the clock rate of the unoptimized version.
- Let C_o be the clock rate of the optimized version.
- Let I_u be the total number of instructions in the unoptimized version.
- Let I_o be the total number of instructions in the optimized version.
- Let LS_u be the number of load/store instructions in the unoptimized version.
- Let LS_o be the number of load/store instructions in the optimized version.

Given Information:

1. The clock rate of the unoptimized version is 5% higher than the optimized version:
$$C_u = 1.05 \times C_o$$
2. 30% of the instructions in the unoptimized version are loads or stores:
$$LS_u = 0.30 \times I_u$$
3. The optimized version executes 2/3 as many loads and stores as the unoptimized version:
$$LS_o = \frac{2}{3} \times LS_u = \frac{2}{3} \times 0.30 \times I_u = 0.20 \times I_u$$
4. All instructions take one clock cycle.

Calculations:

Unoptimized Version:

Total execution time T_u is the number of instructions divided by the clock rate:

$$T_u = \frac{I_u}{C_u}$$

Optimized Version:

The total number of instructions I_o is the sum of the reduced loads/stores and the unchanged other instructions:

$$I_o = LS_o + (I_u - LS_u) = 0.20 \times I_u + (I_u - 0.30 \times I_u) = 0.20 \times I_u + 0.70 \times I_u = 0.90 \times I_u$$

Total execution time T_o is the number of instructions divided by the clock rate:

$$T_o = \frac{I_o}{C_o} = \frac{0.90 \times I_u}{C_o}$$

Comparison:

To compare the execution times, we express both in terms of the same clock rate. Let's use C_o as the reference:

For the unoptimized version:

$$T_u = \frac{I_u}{1.05 \times C_o}$$

For the optimized version:

$$T_o = \frac{0.90 \times I_u}{C_o}$$

Now, let's compare the two:

$$\frac{T_u}{T_o} = \frac{\frac{I_u}{1.05 \times C_o}}{\frac{0.90 \times I_u}{C_o}} = \frac{1}{1.05 \times 0.90} = \frac{1}{0.945} \approx 1.058$$

Since $\frac{T_u}{T_o} > 1$, the optimized version is faster.

Problem 2: Adding a Register-Memory Addressing Mode

Several researchers have suggested that adding a register-memory addressing mode to a load-store machine might be useful.

Part 1: Percentage of Loads to Eliminate

Definitions and Assumptions:

- Let C be the original clock rate.
- Let C' be the new clock rate after adding the new instruction, which is 5% slower:
$$C' = 0.95 \times C$$

- Let P be the original performance.
- Let P' be the new performance with the new instruction.
- Let L be the total number of load instructions.
- Let x be the percentage of loads that must be eliminated.

Performance Calculation:

Performance is inversely proportional to the product of clock cycles and clock period. To maintain the same performance, the reduction in the number of instructions must compensate for the increased clock period.

The performance equation can be written as:

$$P = \frac{1}{CPI \times T}$$

where T is the clock period.

For the new instruction to have at least the same performance:

$$P = P'$$

Since the new instruction affects only the clock cycle and not the CPI, we have:

$$CPI \times T = CPI' \times T'$$

Given $T' = \frac{T}{0.95}$, we need to find x such that:

$$CPI \times T = CPI' \times \frac{T}{0.95}$$

Since CPI' is affected by the reduction in loads:

$$CPI' = CPI \times (1 - x)$$

Substituting CPI' into the equation:

$$CPI \times T = CPI \times (1 - x) \times \frac{T}{0.95}$$

Simplifying:

$$1 = (1 - x) \times \frac{1}{0.95}$$

Solving for x:

$$1 - x = 0.95$$

$$x = 0.05 \text{ or } 5\%$$

Thus, at least 5% of the loads must be eliminated to maintain the same performance with the new instruction.

Part 2: Situation Where Load Cannot Be Replaced

Consider a situation where a load of a register *Rx* is followed immediately by a use of the same register in an ADD instruction. One scenario where this sequence cannot be replaced by a single ADD instruction is when the value loaded into *Rx* is used multiple times or in different contexts.

Example Scenario:

```
LOAD Rx, 0(Rb)     ; Load value from memory into Rx
ADD Ry, Ry, Rx     ; Use Rx in an ADD operation
MUL Rz, Rz, Rx     ; Use Rx in a MUL operation
```

In this sequence, *Rx* is used in both an ADD and a MUL operation. Replacing the LOAD and ADD with a single ADD instruction would not work because the value in *Rx* is needed for the subsequent MUL operation. The new ADD instruction cannot simultaneously serve both operations, making the replacement invalid in this context.