6 Performance Evaluation [50 points]

A multi-cycle processor $P1$ executes *load instructions* in **6 cycles**, *store instructions* in **6 cycles**, *arithmetic instructions* in **2 cycles**, and *branch instructions* in **2 cycles**. Consider an application A where 40% of all instructions are load instructions, 20% of all instructions are store instructions, 30% of all instructions are arithmetic instructions, and 10% of all instructions are branch instructions.

(a) [5 points] What is the CPI (cycles per instruction) of application A when executing on processor $P1$? Show your work.

$CPI = 0.4 \times 6 + 0.2 \times 6 + 0.3 \times 2 + 0.1 \times 2$
$CPI = 4.4$

(b) [5 points] A new design of the processor doubles the clock frequency of $P1$. However, the latencies of *all* instructions increase by 4 cycles. We call this new processor $P2$. The compiler used to generate instructions for $P2$ is the same as for $P1$. Thus, it produces the same number of instructions for program A. What is the CPI of application A when executing on processor $P2$? Show your work.

$CPI = 0.4 \times 10 + 0.2 \times 10 + 0.3 \times 6 + 0.1 \times 6$
$CPI = 8.4$

(c) [20 points] Which processor is faster ($P1$ or $P2$)? By how much (i.e., what is the speedup)? Show your work.

$P2$ is 1.05× faster than $P1$.

Explanation.
$Execution\_Time\_P1 = instructions \times CPI_{P1} \times clock\_time$
$Execution\_Time\_P2 = instructions \times CPI_{P2} \times \frac{clock\_time}{2}$
$clock\_time = \frac{1}{clock\_frequency}$

Assuming that $Execution\_Time\_P2 < Execution\_Time\_P1 \implies \frac{Execution\_Time\_P1}{Execution\_Time\_P2} > 1$. Thus:
$\implies \frac{instructions \times CPI_{P1} \times clock\_time}{instructions \times CPI_{P2} \times \frac{clock\_time}{2}}$
$\implies \frac{4.4 \times clock\_time}{8.4 \times \frac{clock\_time}{2}}$
$\implies \frac{4.4}{4.2}$
$\implies 1.05$

(d) [20 points] You want to improve the original $P1$ design by including one new optimization *without* changing the clock frequency. You can choose **only one** of the following options:

(1) **ALU**: An optimized *ALU*, which *halves* the latency of both arithmetic and branch instructions.

(2) **LSU**: An *asymmetric load-store unit*, which *halves* the latency of load operations but *doubles* the latency of store operations.

Which optimization do you add to $P1$ for application A? Show your work and justify your choice.

The ALU optimization.

Explanation.
Application A executes 40% load, 20% store, 30% arithmetic, and 10% branch instructions.
By Amdahl's Law, we have:

$$Speedup_{ALU} = \frac{1}{(1-0.3-0.1)+\frac{0.3+0.1}{2}} = 1.25$$
$$Speedup_{LSU} = \frac{1}{(1-0.4-0.2)+\frac{0.4}{2}+0.2\times 2} = 1.0$$

The ALU optimization provides 1.25× speedup, while the LSU provides no speedup at all.

Alternative Solution.
With the ALU, the new CPI of processor $P1$ will be:
$CPI_{ALU} = 0.4 \times 6 + 0.2 \times 6 + 0.3 \times \frac{2}{2} + 0.1 \times \frac{2}{2}$
$CPI_{ALU} = 4.0$

With the LSU, the new CPI of processor $P1$ will be:
$CPI_{LSU} = 0.4 \times \frac{6}{2} + 0.2 \times (6 \times 2) + 0.3 \times 2 + 0.1 \times 2$
$CPI_{LSU} = 4.4$

Since $CPI_{ALU} < CPI_{LSU}$, integrating the ALU will improve the overall cycles-per-instruction.