SCT212-0056/2020 FAVOUR PAUL MUTURI
LAB 1

Part 1: Which version is faster – optimized or unoptimized?

Given:

- Clock rate of unoptimized version = 5% higher than optimized.
    - So if optimized has rate RR, unoptimized has rate 1.05R1.05R
- 30% of instructions in unoptimized are loads/stores.
- Optimized executes 2/3 as many loads/stores (i.e., 66.7% of 30% = 20%).
- Other instructions: same dynamic count.
- All instructions take 1 clock cycle.

Let's assume:

- Total instruction count in unoptimized version = II
- Then: Loads/stores = 0.30 * I = 0.30I
- Other instructions = 0.70I

In the optimized version:

- Loads/stores = ⅔ × 0.30I = 0.20I
- Other instructions = 0.70I
- Total instructions = 0.90I

Let's now compute execution time = Instructions × CPI ÷ Clock Rate
(All CPI = 1)

Unoptimized Execution Time

$T_u = \frac{I}{1.05R}$

Optimized Execution Time

$T_o = \frac{0.90I}{R}$

Compare:

We want to see which is smaller: $\frac{I}{1.05R}$ vs $\frac{0.90I}{R}$

Multiply both sides by R to simplify:

- Unoptimized: $\frac{I}{1.05} \approx 0.952I$
- Optimized: $0.90I$

Since $0.90I < 0.952I$, the optimized version is faster.

Part 2: What % of loads must be eliminated to offset 5% slower clock due to added instruction?

We want performance of the new version (with added register-memory ADD) to be ≥ original.

Let:

- Loads originally = 22.8%
- Store = 14.3%
- Clock slowdown = 5% → clock becomes 1.05× slower

Let xx% of loads be removed.

New total instruction count becomes:

New total=100%−x% of loads=100−(x×0.228)\text{New total} = 100\% - x\% \text{ of loads} = 100 - (x \times 0.228)

Performance improves if:

100−0.228x1.05≥100\frac{100 - 0.228x}{1.05} \geq 100

Multiply both sides by 1.05:

100−0.228x≥105⇒−0.228x≥5⇒x≤−50.228≈−21.93100 - 0.228x \geq 105 \Rightarrow -0.228x \geq 5 \Rightarrow x \leq \frac{-5}{0.228} \approx -21.93

This implies we cannot match performance unless we eliminate more than 100% of loads, which is not possible — therefore, just reducing loads alone isn't sufficient. Other changes would be needed.

Part 3: When can you NOT replace LOAD + ADD with a single ADD?

Example:

LOAD R1, 0(R2)
ADD R3, R3, R1

Cannot be replaced by:

ADD R3, 0(R2)

When?

- If R1 is used again later (e.g., `MUL R4, R1, R5`), eliminating the LOAD removes the required value.
- Also if address computation is complex or indirect and can't be handled in a single instruction.

Discussion D1: Is RISC still RISC?

Modern RISC processors have:

- Large instruction sets
- Complex instructions (SIMD, etc.)

But still RISC in spirit because:

- Load/store architecture
- Fixed instruction size
- Simple addressing modes
- Register-based operations

Conclusion: It's about design philosophy—modern RISC is RISC *at heart*, despite growing complexity.

Discussion D2: Are Intel processors RISC or CISC?

- ISA interface = CISC
- Microarchitecture = RISC-like

So:

- For compiler writers → It’s CISC
- For hardware designers → Behaves like RISC

Answer: Measure complexity at ISA level, since that defines how software interacts with the hardware.