# Desmond Oketch- SCT212-0083/2021

# COMPUTER TECHNOLOGY

# Lab 1

# Computer Architecture – tutorial 1

# E1: Performance Comparison

Given:

- Unoptimized clock rate is 5% higher (so optimized is slower).
- 30% of the instructions in the unoptimized version are loads/stores.
- The optimized version executes 2/3 as many loads and stores as the unoptimized one.
- All instructions take 1 clock cycle.

**Step 1: Define Variables**

Let:

- Iu = number of instructions in the unoptimized version
- Io = number of instructions in the optimized version
- Clku = clock cycle time of unoptimized version
- Clko = clock cycle time of optimized version

Since the unoptimized clock is 5% faster, the clock period (cycle time) is 5% shorter:

Clku=0.95×ClkoClku = 0.95 \times Clko

**Step 2: Instruction Count Calculation**

- In the unoptimized version:
    - Loads/stores = 30% of instructions
    - Other instructions = 70% of instructions

- In the optimized version:
    - Loads/stores are reduced to 2/3 of their number.
    - Other instructions stay the same.

Thus, optimized instruction count:

Io=0.7Iu+(23×0.3Iu)Io = 0.7Iu + \left(\frac{2}{3} \times 0.3Iu\right) Io=0.7Iu+0.2IuIo = 0.7Iu + 0.2Iu Io=0.9IuIo = 0.9Iu

The optimized program executes 90% as many instructions as the unoptimized one.

## Step 3: CPU Time Comparison

CPU time formula:

CPU Time=Instruction Count×CPI×Clock Cycle TimeCPU\ Time = Instruction\ Count \times CPI \times Clock\ Cycle\ Time

Since CPI = 1 (all instructions take 1 cycle), the formula simplifies to:

CPU Time=Instruction Count×Clock Cycle TimeCPU\ Time = Instruction\ Count \times Clock\ Cycle\ Time

- Unoptimized CPU Time:

Tu=Iu×ClkuT_{u} = Iu \times Clku

- Optimized CPU Time:

To=Io×ClkoT_{o} = Io \times Clko =0.9Iu×Clko= 0.9Iu \times Clko

Substituting Clku=0.95×ClkoClku = 0.95 \times Clko:

Tu=Iu×0.95ClkoT_{u} = Iu \times 0.95Clko

Now compare ToT_o and TuT_u:

- Optimized time:

To=0.9Iu×ClkoT_{o} = 0.9Iu \times Clko

- Unoptimized time:

$Tu = 0.95Iu \times Clko$

Take the ratio:

$\frac{T_{o}}{T_{u}} = \frac{0.9}{0.95} = 0.947$

Thus:

$To = 0.947 \times T_{u}$

Meaning the optimized version is ~5.3% faster.

Conclusion: The optimized version is faster.

# E2: Register-Memory Addressing

**Part 1: What % of loads must be eliminated?**

Given:

- Clock cycle increases by 5% with the new addressing mode.
- Instruction mix (from Table 1):

| Instruction | Frequency |
|---|---|
| Load | 22.8% |
| Store | 14.3% |
| Add | 14.6% |
| ... | ... |

Assume initially:

- Execution time is proportional to
  Instruction Count×CPI×Clock Cycle Time\text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}.
- CPI is unchanged.
- Only the clock period and instruction count are affected.

Let:

- xx = fraction of loads eliminated.

New instruction count:

New Instructions=(1−x)×Load Instructions+Other Instructions\text{New Instructions} = (1 - x) \times \text{Load Instructions} + \text{Other Instructions}

Relative execution time (normalized):

Execution Time Ratio=(1+0.05)×[(1−x)×0.228+(1−0.228)]\text{Execution Time Ratio} = (1 + 0.05) \times \left[ (1 - x) \times 0.228 + (1 - 0.228) \right]

Set Execution Time Ratio = 1 (same performance):

(1.05)×[(1−x)×0.228+0.772]=1(1.05) \times \left[ (1 - x) \times 0.228 + 0.772 \right] = 1

Expand:

(1.05)×(1−0.228x)=1(1.05) \times (1 - 0.228x) = 1 1.05−0.2394x=11.05 - 0.2394x = 1 0.2394x=0.050.2394x = 0.05 x=0.050.2394≈0.2088x = \frac{0.05}{0.2394} \approx 0.2088

Thus, about 20.9% of loads must be eliminated for the new machine to have at least the same performance.

## Part 2: Example where replacement is NOT possible

Example:

```
LOAD R1, 0(R2)
LOAD R3, 0(R4)
ADD R5, R1, R3
```

Here:

- We load R1 and R3 separately from two different memory addresses.
- Then ADD them.

Problem:
The proposed optimization assumes you replace LOAD and ADD when the ADD uses the loaded value directly.
However, if you need two different loads, you can't merge both loads into one ADD instruction.

Conclusion:
When multiple different memory values are needed before the ADD, you cannot replace with a single register-memory ADD.