EEE4120F 2026
System Benchmarking Metrics Summary

1. Application Metrics

- **Operations per Second (ops):** Used as a primary metric for applications.
- **Definition of an 'Operation':** In this context, an operation is viewed as a function or a sequence of instructions.
- **Exclusions:** Benchmarking of operations specifically excludes input and output (I/O) time.
- **Clarification:** An operation is distinct from a single machine instruction, a thread, or a process, as these may be worked on intermittently.

2. Programming Language Metrics

The effectiveness of a language is often measured by two primary metrics:

Metric	Description	Example
Expressiveness	The power of the language to write diverse computer programs.	**C** has high expressiveness; **SQL** is significantly less expressive for general tasks.
Code Density	The ability to represent complex processing needs in a small number of lines.	**SQL** and Domain Specific Languages (DSLs) are very dense; **C** is less dense, requiring more code for similar operations.

3. Processing and Hardware Metrics

There is an abundance of metrics used to benchmark processing and data paths:

- **Popular Processing Metrics:** MIPS, GIPS, and FLOPS.
- **Data Path:** Typically measured in **megabits per second**.
- **Function Units:** Accelerators that complete an operation in a single cycle may be rated in **cycles per second**.

Average Cycles Per Instruction (ACPI)

Average Cycles Per Instruction, or the **ACPI** (sometimes referred to as **CPI**) of a **CPU** is a term that has been used in many papers and data sheets. It is perhaps not used as extensively in scientific articles as it was in the past. That is because there is such a diversity of computing solutions out there for which this expression is not so relevant.

The Role of Modern Hardware

For example, an **FPGA** might grab some data, update a running average, and turn on some **LEDs** all in one clock cycle. **ACPI** is more useful for comparing processors that run similar types of instructions. For example, you might be comparing two **CPUs** to run a certain application:

- **Processor 1:** No pipelining, with an **ACPI of 1**.
- **Processor 2:** A classic **RISC** with a five-stage pipeline and an **ACPI** of, say, **4**.

Pipelining and Clock Speed

A lower **ACPI** may seem better at face value, but is it?

- The first processor might be clocked at **10 MHz**, a slower clock to account for the maximum propagation delays in the **CPU** circuit.
- The second processor, having its circuit split into pipeline sections that run concurrently, might support **100 MHz**.

Now it's not so clear which one would win a processing race. Probably the second processor, because **100 divided by the ACPI of 4** is **25 million instructions per second**, which is **2.5 times better** than the first. However, if you optimize your code, the second pipelined processor might run closer to **100 million instructions per second** if the pipeline is kept full and there are no stalls—meaning in each clock, one instruction is coming in and one instruction is completing. That is a speedup closer to **10**.

Calculating ACPI

The **ACPI** depends on a processor architecture, usually in relation to a program to benchmark. This typically concerns performance measurements for a pipelined architecture. It is pretty obvious that a non-pipelined program is going to have a **CPI of 1** regardless of the program it runs; its **ACPI** would be **1** as well.
To work out the **CPI** for a processor given a particular program, the essential formulas are used. The overall **ACPI** is the sum of the cycles per instruction multiplied by the frequency of that instruction for each instruction that appears in the program concerned. This gives out a quantity of the **ACPI** for the given program.

Key Formula for ACPI

Based on the description provided in the audio, the formula for calculating the overall Average Cycles Per Instruction for a program is:

$$ACPI = \sum_{i=1}^{n} CPI_i * F_i$$

Where:

- CPI_i is the cycles per instruction for a specific instruction type *i*.
- F_i is the frequency (or percentage) of that instruction appearing in the program.

Here's a practical method to see how instruction mixes and clock speeds impact overall performance. This script is aimed to be modular so that it can plug in to different architectures (e.g. used on a basic RISC vs. use on a more complex SoC).

ACPI and MIPS Performance Calculator

This script calculates the **Average Cycles Per Instruction (ACPI)** based on the frequency of different instruction types and then determines the **Millions of Instructions Per Second (MIPS)** for a given clock frequency.

Python

```python
def calculate_performance(clock_mhz, instruction_mix):
    """
    Calculates ACPI and MIPS for a given processor and program mix.

    Args:
        clock_mhz (float): Clock frequency in MHz.
        instruction_mix (list of dicts): List containing 'cpi' and
                                         'frequency' (0.0 to 1.0)
    """
    # Calculate ACPI: Sum of (CPI_i * Frequency_i)
    acpi = sum(item['cpi'] * item['frequency'] for item in instruction_mix)

    # Calculate MIPS: Clock Rate (MHz) / ACPI
    mips = clock_mhz / acpi

    return acpi, mips

# --- Example Scenario: 5-Stage RISC Pipeline ---
# 100 MHz Clock Speed
f_clk = 100

# Typical instruction distribution for a simple program
# Note: Frequencies must sum to 1.0
prog_mix = [
    {'type': 'ALU Operations', 'cpi': 1, 'frequency': 0.50},
    {'type': 'Loads/Stores',   'cpi': 2, 'frequency': 0.30},
    {'type': 'Branches',       'cpi': 3, 'frequency': 0.20}
]

acpi_val, mips_val = calculate_performance(f_clk, prog_mix)

print(f"--- Processor Performance Results ---")
print(f"Clock Frequency: {f_clk} MHz")
print(f"Calculated ACPI: {acpi_val:.2f}")
print(f"Calculated MIPS: {mips_val:.2f}")
```

How this reinforces the lecture:

1. **Instruction Weighting:** Students can see that even if most instructions take 1 cycle, a few "heavy" instructions (like branches or memory access) can quickly pull the ACPI above 1.0.
2. **The Pipeline Paradox:** consider comparing a **10 MHz** processor with an ACPI of **1.0** vs. a **100 MHz** processor with an ACPI of **4.0**. This script will prove that this higher-clocked, higher-ACPI chip still yields better MIPS (25 vs 10).
3. **Optimization:** Consider simulating "code optimization" by reducing the frequency of high-CPI instructions (like Branches) to see the MIPS value climb.