ECE 251: Computer Architecture

Week 06 Notes - Single-Cycle Performance & Pipelining Primer

Prof Rob Marano

Spring 2026

1. Assessing Performance: The Single-Cycle Processor

The foundational model of the MIPS architecture executes exactly **one instruction at a time per clock cycle**.

The Classic CPU Performance Equation

$$\text{Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

- **CPI (Cycles Per Instruction)** is strictly 1.
- **The Trade-off**: The entire CPU Clock Cycle Time must be elongated to accommodate the absolute slowest instruction (typically the `lw` Load Word command, which traverses Data Memory).
- *Result*: Faster operations (like `add`) inherently waste hardware time rigidly waiting for the slow clock cycle to tick over.

2. Pseudodirect vs PC-Relative Targets

Both Branches and Jumps manually alter where the internal CPU logic executes *next*, but their 32-bit compilation limits dictate how far they can reach.

- **Branches (PC-Relative, `beq`)**:
  - 16-bit parameter. Small leaps locally within the code block.
  - Calculates offset: Target $=$ PC $+ 4 +$ (offset $\times 4$).

- **Jumps (Pseudodirect, `j`)**:
  - 26-bit parameter. Vast leaps overriding local logic.
  - Concatenates the Jump instruction bits directly against the upper 4 bits of the current PC tracker to define a raw 32-bit coordinate boundary.

3. Pipelining Primer: Where exactly are we jumping?

Understanding where the CPU physically points during jumping mechanics is vastly critical in advanced Pipelining.

- **Question**: Immediately right after a branch or a jump instruction, what is the relative value of the PC?
- **Answer**: Immediately after a branch or jump, the hardware automatically increments the Program Counter to **PC + 4**.
- The instruction situated at `PC + 4` executes regardless of the branch outcome. This is called the **Branch Delay Slot**.

4. The Branch Delay Slot and `jal`

How do we safeguard the Return Address (`$ra`) during a procedure call knowing the Delay Slot executes implicitly?

1. A `jal` (Jump and Link) instruction executes at the current PC.
2. The instruction at **PC + 4** (the Delay Slot) executes immediately inherently.
3. *We do not want the procedure returning to the Delay Slot, it just executed!*
4. The `jal` natively calculates **PC + 8** (the instruction safely past the Delay Slot) and deliberately stores this value into the `$ra` tracking register.

By forcing the compiler/programmer to manually acknowledge the Delay Slot, the CPU hardware extracts massive optimizations maintaining steady instructional flow.