ECE 251: Computer Architecture

Week 07 Notes - Advanced Assembly & Final Project Launch

Prof Rob Marano

Spring 2026

1. SPIM Operating System Simulator

Since SPIM is a simulator, it provides native OS-level library calls via `syscall` to handle protected hardware functions like I/O.

- **Convention**: Load the Service Code in `$v0`, arguments in `$a0`-`$a3`, then execute `syscall`.
- **Common Service Codes**:
  - 1: Print Integer (`$a0` = int)
  - 4: Print String (`$a0` = address of `.asciiz`)
  - 5: Read Integer (Return in `$v0`)
  - 10: Exit Cleanly
  - –: File I/O available via 13 (Open), 14 (Read), 16 (Close).

2. Tracing Recursive Procedures

Recursion rigidly tests a programmer's Stack Management abilities.

- A function calling *itself* overwrites the `$ra` (Return Address) tracker and `$a0` arguments on every single execution loop.
- The CPU Stack physically grows **downward** (`addi $sp, $sp, -12`) inside the CPU, placing "plates" of saved variables on top of each other.
- When the base case is breached, recursion collapses **upward**, popping the plates (‘lw‘ to restore `$ra`) layer-by-layer to cleanly unspool the jump sequence back to `main`.

3. Hardware Execution: SPIM vs. Physical Silicon

Why did the intricate *Quake III* Inverse Square Root function (14 instructions) decisively beat the standard IEEE CPU method (2 instructions)?

- **Software Emulators**: Enforce a flat, abstracted CPI of 1.0. (SPIM finishes 2 steps 7x faster than 14 steps).
- **Physical 1999 Silicon**: Floating Point Division (`div.s`) could violently stall the CPU pipeline for **up to 54 clock cycles**.
- **The Result**: The Quake Algorithm relied entirely on raw integer bit-shifting (1 CPI) resulting in 14 total cycles natively, compared to $\sim$54 cycles mechanically wasting time waiting for the FPU division gate logic.

4. The Final Group Project: 100 Points

The Objective: Organically design and implement a fundamental von Neumann computer Architecture from physical scratch.

1. **Team Logistics**: Teams of exactly 2.
2. **The Core Deliverable**: A fully functional **SystemVerilog** processor (minimum 4-bit) capable of running a functional program simulation via a Test Bench.
3. **The Architecture**: You must rigidly define your own explicit **Instruction Set Architecture (ISA)**, documenting the Opcode boundaries, Datapath layout, ALU functions, and Memory access limits natively.
4. **The Baseline**: Replicate a historical 4-bit/8-bit microchip (Intel 4004) or design a functional custom MIPS32 module implementation.