Lecture 3: Basic MIPS Architecture
Basic MIPS Architecture

• Now that we understand clocks and storage of states,
• we’ll design a simple CPU that executes:
  • basic math (add, sub, and, or, slt)
  • memory access (lw and sw)
  • branch and jump instructions (beq and j)
Implementation Overview

- We need memory
  - to store instructions
  - to store data
  - for now, let’s make them separate units

- We need registers, ALU, and a whole lot of control logic

- CPU operations common to all instructions:
  - use the program counter (PC) to pull instruction out of instruction memory
  - read register values
View from 30,000 Feet

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

Note: we haven’t bothered showing multiplexors

Source: H&P textbook
Clocking Methodology

- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?
- Keep in mind that the latched value remains there for an entire cycle
Implementing R-type Instructions

- Instructions of the form `add $r1, $r2, $r3`
- Explain the role of each signal

Source: H&P textbook
Implementing Loads/Stores

- Instructions of the form `lw $r1, 8($r2)` and `sw $r1, 8($r2)`
- Explain the role of each signal

Where does this input come from?

Source: H&P textbook
Implementing J-type Instructions

- Instructions of the form $\text{beq}\ r1, r2, \text{offset}$
View from 10,000 Feet
View from 5,000 Feet

Source: H&P textbook
The entire instruction executes in a single cycle
Green blocks are latches
At the rising edge, a new PC is recorded
At the rising edge, the result of the previous cycle is recorded
At the falling edge, the address of LW/SW is recorded so we can access the data memory in the 2nd half of the cycle
Multi-Stage Circuit

• Instead of executing the entire instruction in a single cycle (a single stage), let’s break up the execution into multiple stages, each separated by a latch.
The Assembly Line

Unpipelined

Start and finish a job before moving to the next

Jobs

Time

Pipelined

Break the job into smaller stages
Performance Improvements?

• Does it take longer to finish each individual job?

• Does it take shorter to finish a series of jobs?

• What assumptions were made while answering these questions?

• Is a 10-stage pipeline better than a 5-stage pipeline?
Quantitative Effects

- As a result of pipelining:
  - Time in ns per instruction goes up
  - Each instruction takes more cycles to execute
  - But… average CPI remains roughly the same
  - Clock speed goes up
  - Total execution time goes down, resulting in lower average time per instruction
- Under ideal conditions, speedup
  - = ratio of elapsed times between successive instruction completions
  - = number of pipeline stages = increase in clock speed