15-740/18-740 Computer Architecture Lecture 7: Pipelining

Prof. Onur Mutlu
Carnegie Mellon University
Fall 2011, 9/26/2011

Review of Last Lecture

- More ISA Tradeoffs
  - Programmer vs. microarchitect
  - Transactional memory
  - Dataflow
- Von Neumann Model
- Performance metrics
  - Performance equation
  - Ways of improving performance
  - IPS, FLOPS, Perf/Frequency

Today

- More performance metrics
- Issues in pipelining
- Precise exceptions

Papers to Read and Review

- Review Set 5 – due October 3 Monday
  - Smith and Plezskun, "Implementing Precise Interrupts in Pipelined Processors," IEEE Transactions on Computers 1988 (earlier version: ISCA 1985).
  - Sprangle and Carmean, "Increasing Processor Performance by Implementing Deeper Pipelines," ISCA 2002.

- Review Set 6 – due October 7 Friday
  - Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," IBM Journal of R&D, Jan. 1967.
  - Smith and Sohi, "The Microarchitecture of Superscalar Processors," Proc. IEEE, Dec. 1995.

Review: The Performance Equation

$$\text{Execution time} = \frac{\text{time}}{\text{program}}$$

$$= \frac{\text{\# instructions}}{\text{program}} \times \frac{\text{\# cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}$$

Algorithm	ISA	Microarchitecture
Program	Microarchitecture	Logic design
ISA		Circuit implementation
Compiler		Technology

Review: Other Performance Metrics: Perf/Frequency

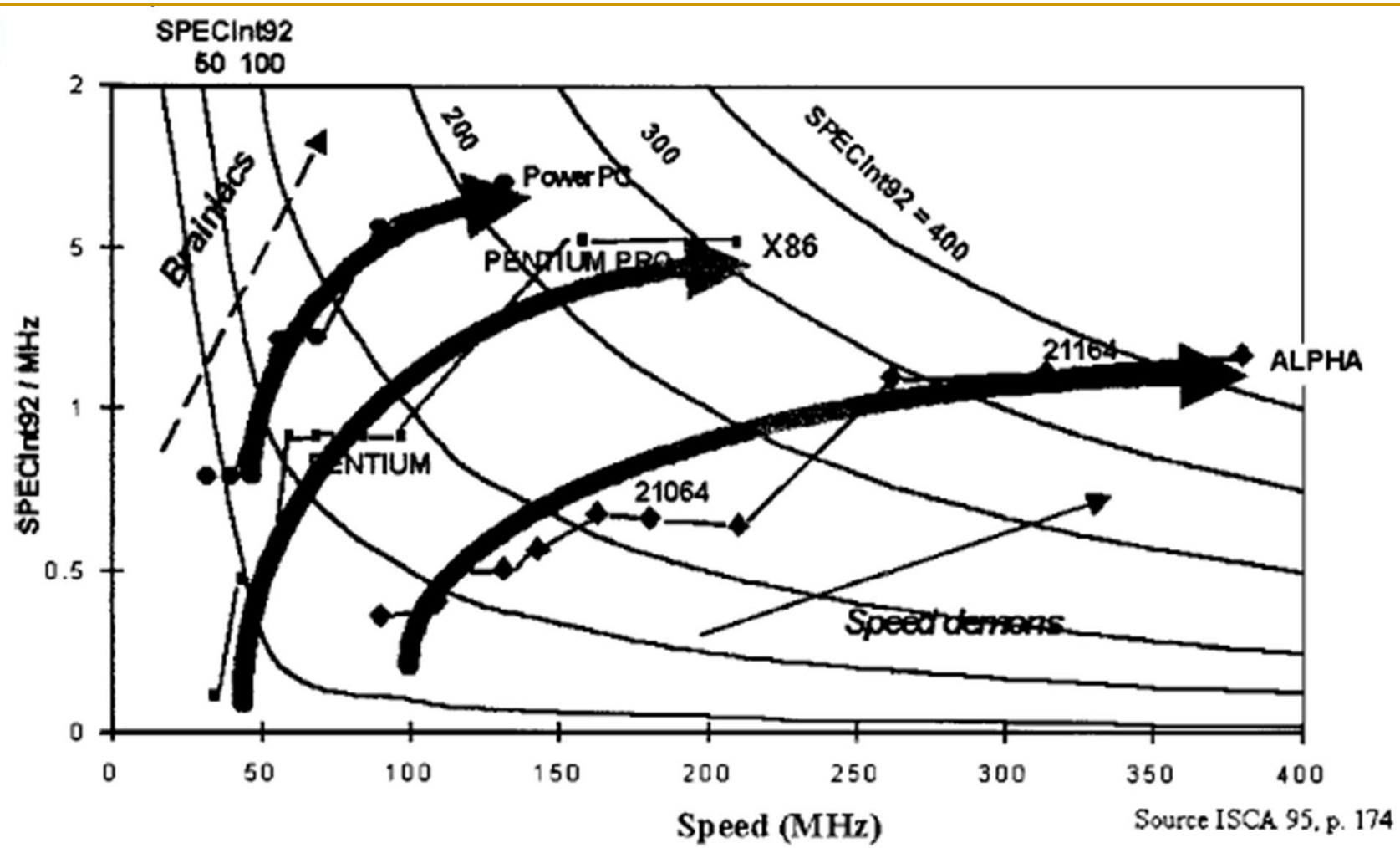
- SPEC/MHz
- Remember $\text{Execution time} = \frac{\text{time}}{\text{program}} = \frac{1}{\text{Performance}}$
- Performance/Frequency

$$= \frac{\frac{\text{time}}{\text{cycle}}}{\frac{\text{\# instructions}}{\text{program}} \times \frac{\text{\# cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}}$$

$$= 1 / \{\frac{\text{\# cycles}}{\text{program}}\}$$

- What is wrong with comparing only "cycle count"?
  - Unfairly penalizes machines with high frequency
- For machines of equal frequency, fairly reflects performance assuming equal amount of "work" is done
  - Fair if used to compare two different same-ISA processors on the same binaries

Review: An Example Use of Perf/Frequency



- Ronen et al, IEEE Proceedings 2001

Other Potential Performance Metrics

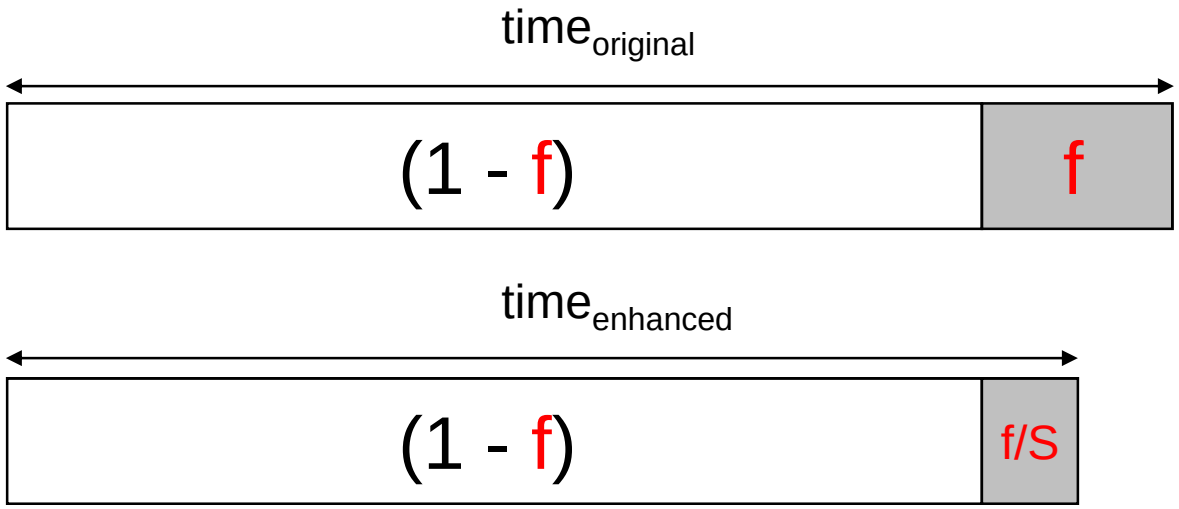
- Memory bandwidth
  - Sustained vs. peak
- Compute/bandwidth balance
  - Arithmetic intensity
- Energy consumption metrics

Amdahl's Law: Bottleneck Analysis

- Speedup= $time_{without\ enhancement}$ / $time_{with\ enhancement}$
- Suppose an enhancement speeds up a fraction f of a task by a factor of S

$$time_{enhanced} = time_{original} \cdot (1-f) + time_{original} \cdot (f/S)$$

$$Speedup_{overall} = 1 / ((1-f) + f/S)$$



Focus on bottlenecks with large f (and large S)

Microarchitecture Design Principles

- Bread and butter design
  - Spend time and resources on where it matters (i.e. improving what the machine is designed to do)
  - Common case vs. uncommon case
- Balanced design
  - Balance instruction/data flow through uarch components
  - Design to eliminate bottlenecks
- Critical path design
  - Find the maximum speed path and decrease it
    - Break a path into multiple cycles?

Cycle Time (Frequency) vs. CPI (IPC)

- Usually at odds with each other

- Why?
  - Memory access latency: Increased frequency increases the number of cycles it takes to access main memory

  - Pipelining: A deeper pipeline increases frequency, but also increases the "stall" cycles:
    - Data dependency stalls
    - Control dependency stalls
    - Resource contention stalls

- Average CPI (IPC) affected by exploitation of instruction-level parallelism. Many techniques devised for this.

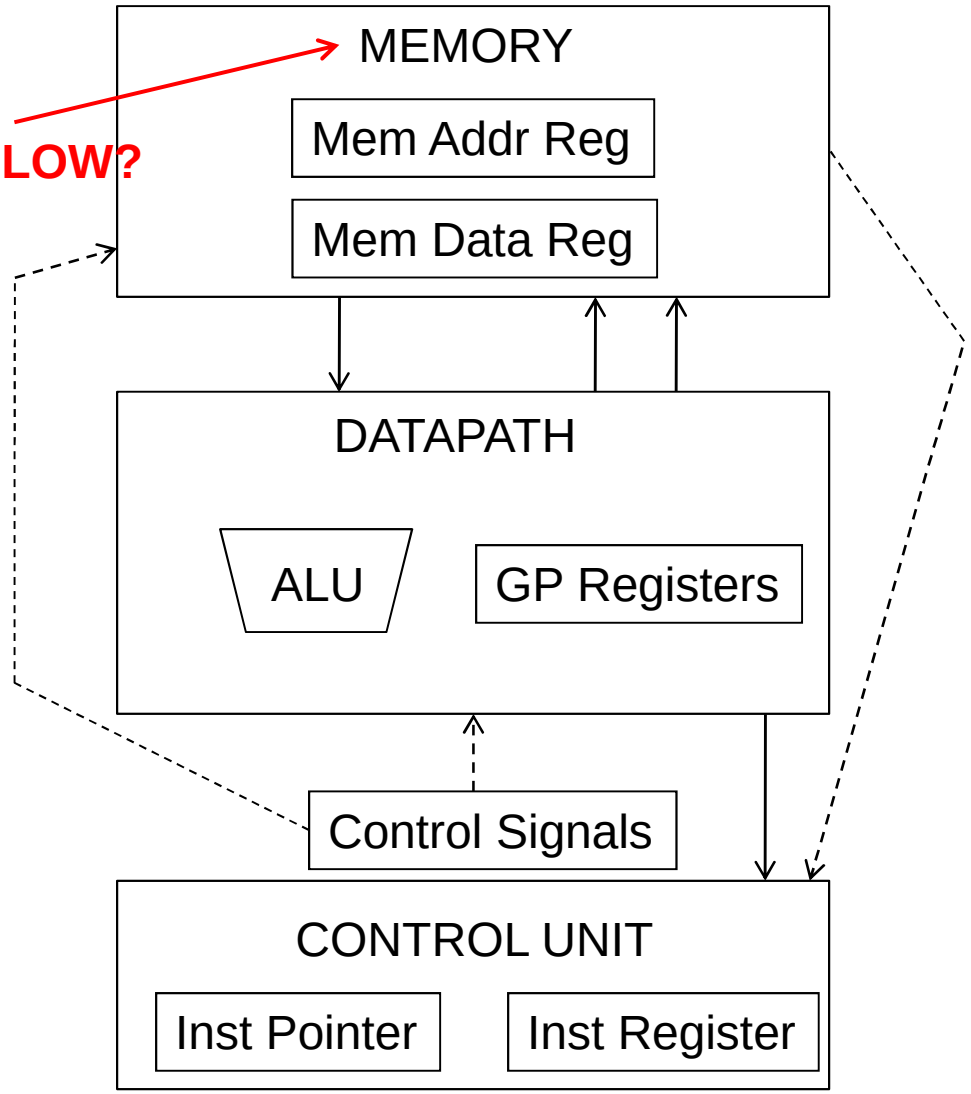
Intro to Pipelining (I)

- Single-cycle machines
  - Each instruction executed in one cycle
  - The slowest instruction determines cycle time

- Multi-cycle machines
  - Instruction execution divided into multiple cycles
    - Fetch, decode, eval addr, fetch operands, execute, store result
    - Advantage: the slowest "stage" determines cycle time
  - Microcoded machines
    - Microinstruction: Control signals for the current cycle
    - Microcode: Set of all microinstructions needed to implement instructions → Translates each instruction into a set of microinstructions
    - Wilkes, "The Best Way to Design an Automatic Calculating Machine," Manchester Univ. Computer Inaugural Conf., 1951.

Microcoded Execution of an ADD

- ADD DR ← SR1, SR2
- Fetch:
  - MAR ← IP
  - MDR ← MEM[MAR]
  - IR ← MDR
- Decode:
  - Control Signals ← DecodeLogic(IR)
- Execute:
  - TEMP ← SR1 + SR2
- Store result (Writeback):
  - DR ← TEMP
  - IP ← IP + 4

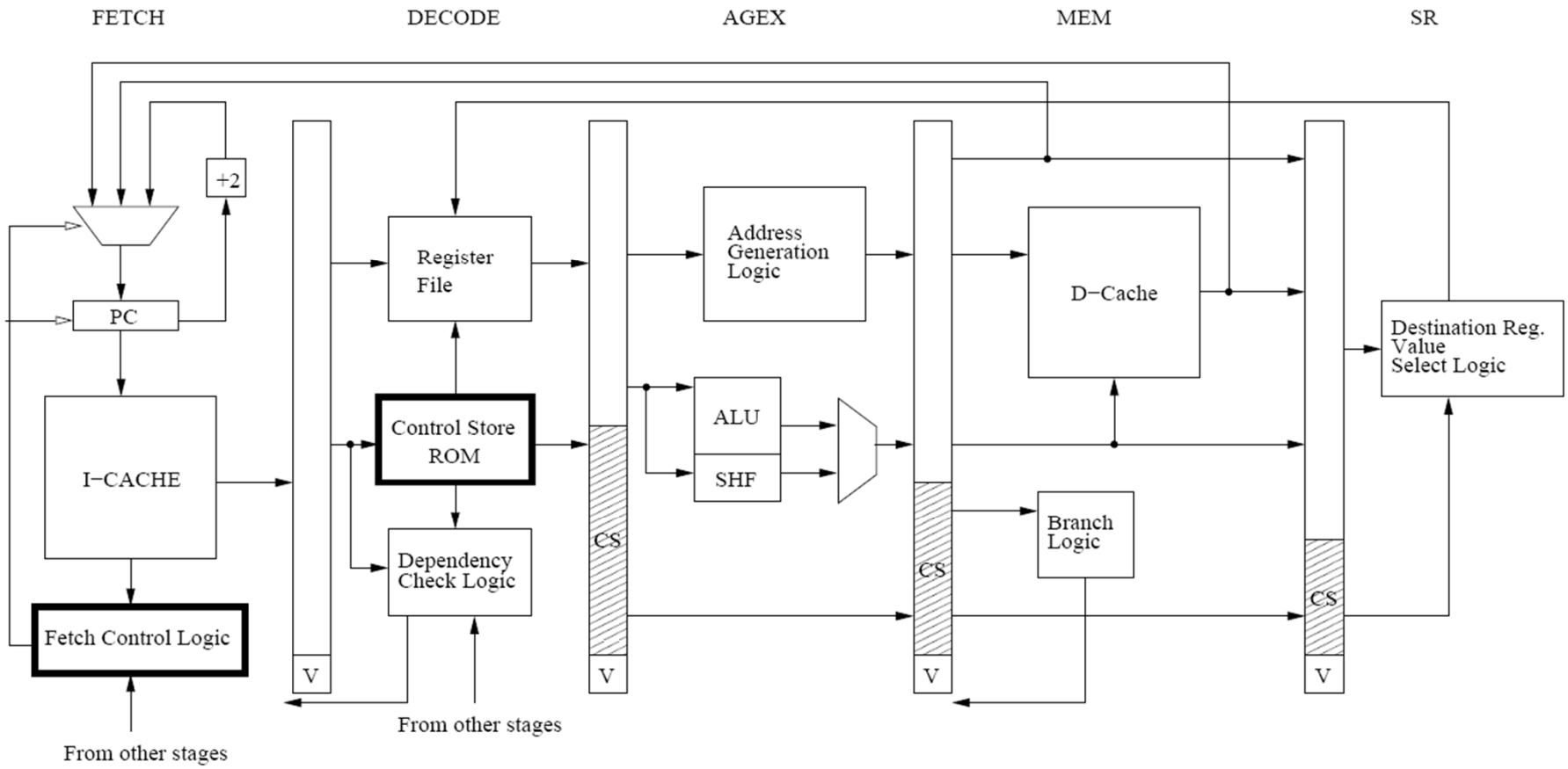


Intro to Pipelining (II)

- In the microcoded machine, some resources are idle in different stages of instruction processing
  - Fetch logic is idle when ADD is being decoded or executed

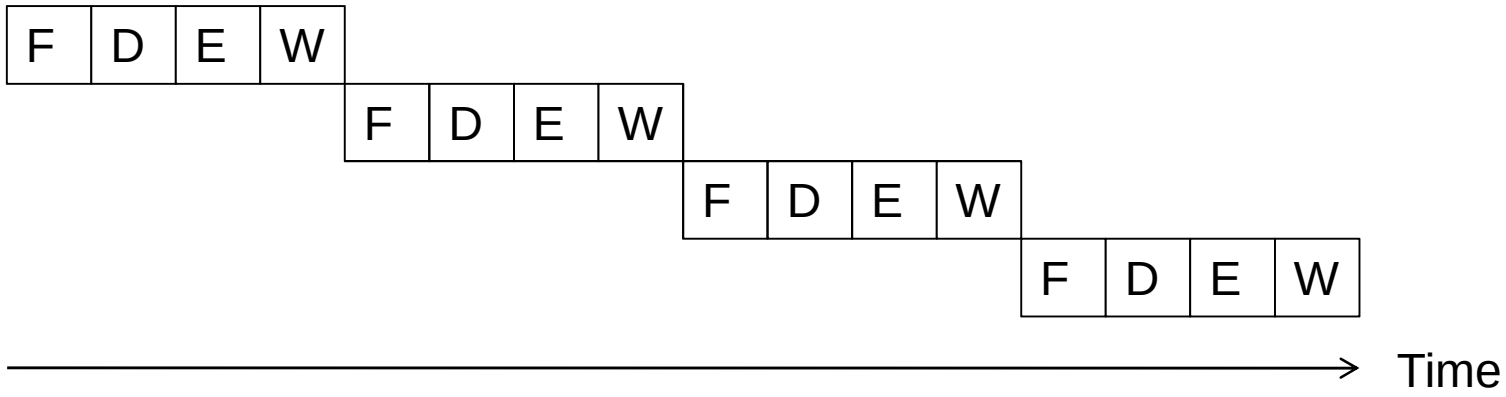
- Pipelined machines
  - Use idle resources to process other instructions
  - Each stage processes a different instruction
  - When decoding the ADD, fetch the next instruction
  - Think “assembly line”
- Pipelined vs. multi-cycle machines
  - Advantage: Improves instruction throughput (reduces CPI)
  - Disadvantage: Requires more logic, higher power consumption

A Simple Pipeline

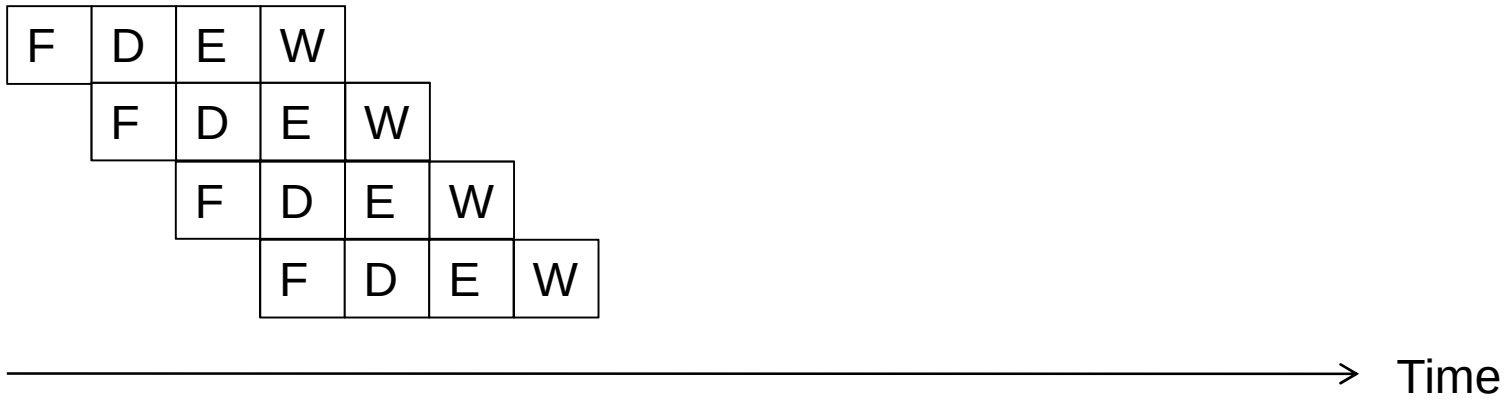


Execution of Four Independent ADDs

- Multi-cycle: 4 cycles per instruction



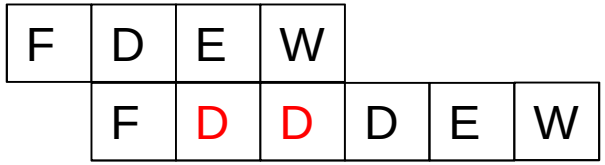
- Pipelined: 4 cycles per 4 instructions (steady state)



Issues in Pipelining: Increased CPI

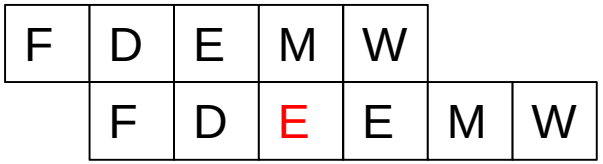
- Data dependency stall: what if the next ADD is dependent

ADD R3 ← R1, R2
ADD R4 ← R3, R7



  - Solution: data forwarding. Can this always work?
    - How about memory operations? Cache misses?
    - If data is not available by the time it is needed: STALL
  - What if the pipeline was like this?

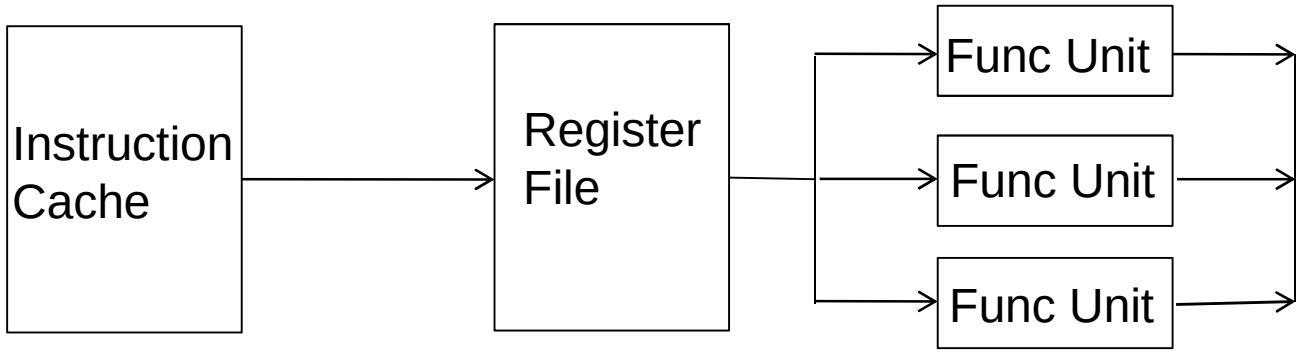
LD R3 ← R2(0)
ADD R4 ← R3, R7



    - R3 cannot be forwarded until read from memory
    - Is there a way to make ADD not stall?

Implementing Stalling

- Hardware based interlocking
  - Common way: scoreboard
  - i.e. valid bit associated with each register in the register file
  - Valid bits also associated with each forwarding/bypass path



Data Dependency Types

- Types of data-related dependencies
  - Flow dependency (true data dependency – read after write)
  - Output dependency (write after write)
  - Anti dependency (write after read)

- Which ones cause stalls in a pipelined machine?
  - Answer: It depends on the pipeline design
  - In our simple strictly-4-stage pipeline, only flow dependencies cause stalls
  - *What if instructions completed out of program order?*