Pipelining

Computer Architecture: Design and Simulation
CMU 15-346, Spring 2021

Outline

- Pipelines
- Hazards (data, control, structural)
- Superscalar

Instruction Execution

- Five steps
  - Next slide w/o latches (drawn at start)
  - How long do instructions take?
  - Is the whole circuit in use during this time?

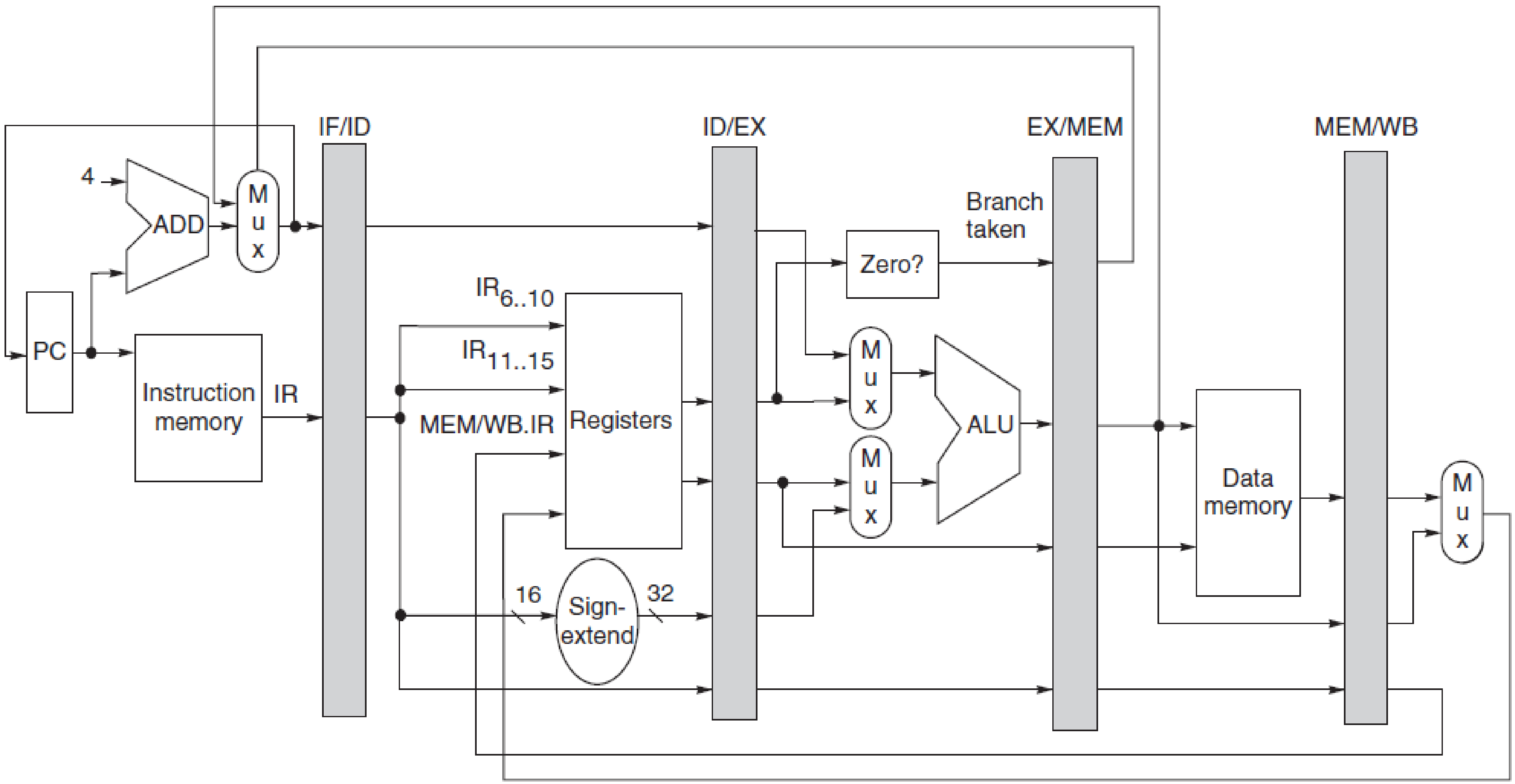

Figure C.22 The data path is pipelined by adding a set of registers, one between each pair of pipe stages. The

Figure from Comp Arch: Quantitative Approach

Pipeline Speedup

- Pipelines are an improvement

$$
\begin{aligned}
\text{Speedup from pipelining} &= \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}} \\
&= \frac{\text{CPI unpipelined} \times \text{Clock cycle unpipelined}}{\text{CPI pipelined} \times \text{Clock cycle pipelined}} \\
&= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}
\end{aligned}
$$

- What are the ideal values in this equation?
  - CPI unpipelined is depth (see pipeline figure)

- What practical limitations exist?
  - Stalls
  - Overhead

Stalls

- Program characteristics => Hazards => Stalls
  - But not always

Hazards

- Structural hazard
  - Limitation of hardware resources
  - E.G. Single-ported memory delaying IF / MEM
  - Trade-offs in resources to avoid this hazard

- Data hazard
  - Dependency related

Types of Dependencies

- True dependency

  ```
  ADD R1, R2, R3
  LD R4, 0(R1)
  ```

- Anti-dependency

  ```
  ADD R1, R2, R3
  SUB R3, R2, R4
  ```

- Output dependency

  ```
  ADD R1, R2, R3
  SUB R1, R2, R3
  ```

Data Hazards becoming Stalls

- RAW Hazard
  - Add forwarding / bypass logic

Load R1, 0(R2)   // load delay slot option...

ADD R3, R1, R4

- WAW and WAR Hazard
  - Add more registers
  - Mostly a problem in out-of-order
- RAR Hazard
  - Problem if not enough read ports on reg file

CPI and Registers

- If CPI is proportional to stalls,
  then adding more registers reduces CPI

- Stay tuned for register renaming

Consider the following

http://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-an-unsorted-array

Branch Prediction

- When, where, whether
- Delay slots
- 1-bit
- 2-bit
- Gselect, gshare, Yeh/patt
- Tournament

Control Hazards

- When does a processor know that there is a branch?
- Where might the branch go?
- When does a processor know whether the branch is taken?

- How frequent are branches?

When is there a branch?

- When does the processor fetch the next instruction?
  - Start of IF stage
- Decode a branch
  - End of ID stage

Branch instruction	IF	ID	EX	MEM	WB		
Branch successor		IF	IF	ID	EX	MEM	WB
Branch successor + 1				IF	ID	EX	MEM
Branch successor + 2					IF	ID	EX

Delay Slots

- Rather than predict branches, the stalls are a "feature"

- When is the next instruction to execute the one after the branch?

Whether a branch is taken

- Static
  - Forward not-taken
  - Backward taken
- Dynamic
  - Index a table using low-order instruction bits
- 1-bit
  - Store the last result, predict using it
- 2-bit
  - James E Smith, 1981

Correlating Predictions

- Branch History Register
  - Shift in the most recent branch outcome

- Gshare
  - Use the BHR to select which table to index for the entry

- Gselect
  - XOR the BHR and index bits to select which entry

Yeh-Patt (1991)

- Keep histories for each branch
  - Insight: Similar histories have similar predictions
- Index a history table
  - Use the history entry to select the predictor
- About 96-98% accurate for integer codes (SPEC?)

Tournament Predictors

- Why have one, when you can have more than one
  - Aliasing problem in a predictor

- Meta predictor or majority vote

- What else?
  - Machine learning

Where a branch goes

- Branch target buffer
  - Simple cache for branch targets

- Return address stack

Summary

- CPI = 1 + stalls

- Is ideal CPI 1?