CSCI 322
Principles of Concurrent Programming

Filip Jagodzinski
Announcements

• Homework #1 will be posted
  • “book” questions
  • Due via “canvas” in 1 week
  • Architecture review

• Second lab this Friday
  • Threads
  • More “Linux refresher”

• Textbook has been ordered and will be on reserve shortly; a “reserve” link will be available via Canvas
From last time ...

Calculating the speedup for a pipelined versus a non-pipelined machine architecture

For 20,000 instructions ...

Execution time for pipeline machine: 20,000 * 8ns = 160,000ns
Execution time for non-pipelined machine: 20,000 * 22ns = 440,000ns
Speedup: 440,000 / 160,000 = 2.75

Q: Why are these execution times for 20,000 instructions approximations?
From last time ...

Q: what is the difference between a data and a structural hazard?
From last time ...

Instruction 1: \( R3 \leftarrow R1 + R2 \)

Instruction 2: \( R4 \leftarrow R3 + R5 \)

Data hazards arise when instructions that have dependencies are executed one after another.

The ID stage of Instruction 2 cannot begin until AFTER WB of Instruction 1 completes.
From last time ...

Latency and throughput are both governed by the slowest pipeline step. How might a computer architect modify the pipeline stages to improve execution time when large numbers of instructions are being processed?
Latency and throughput are both governed by the slowest pipeline step. How might a computer architect modify the pipeline stages to improve execution time when large numbers of instructions are being processed?

<table>
<thead>
<tr>
<th></th>
<th>Pipeline A</th>
<th></th>
<th>Pipeline B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6ns</td>
<td>3ns</td>
<td>5ns</td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>ID</td>
<td>EX</td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>ID</td>
<td>EX</td>
</tr>
<tr>
<td>IF</td>
<td></td>
<td>ID</td>
<td>EX</td>
</tr>
</tbody>
</table>
From last time ...

### Pipeline A

<table>
<thead>
<tr>
<th>6ns</th>
<th>3ns</th>
<th>5ns</th>
<th>8ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WB</td>
</tr>
</tbody>
</table>

### Pipeline B

<table>
<thead>
<tr>
<th>6ns</th>
<th>3ns</th>
<th>5ns</th>
<th>4ns</th>
<th>4ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>WBa</td>
<td>WBB</td>
</tr>
</tbody>
</table>

Latency and throughput are both governed by the slowest pipeline step. How might a computer architect modify the pipeline stages to improve execution time when large numbers of instructions are being processed?

Q: For 20,000 instructions, what is the approximate speedup of using a pipelined versus a non-pipelined architecture for pipelines A and B?
Latency and throughput are both governed by the slowest pipeline step. How might a computer architect modify the pipeline stages to improve execution time when large numbers of instructions are being processed?

Q: For 20,000 instructions, what is the approximate speedup of using a pipelined versus a non-pipelined architecture for pipelines A and B?

<table>
<thead>
<tr>
<th>Pipeline A</th>
<th>Pipeline B</th>
</tr>
</thead>
<tbody>
<tr>
<td>6ns IF</td>
<td>6ns IF</td>
</tr>
<tr>
<td>3ns ID</td>
<td>3ns ID</td>
</tr>
<tr>
<td>5ns EX</td>
<td>5ns EX</td>
</tr>
<tr>
<td>8ns WB</td>
<td>4ns WBa</td>
</tr>
<tr>
<td>3ns WB</td>
<td>4ns WBB</td>
</tr>
</tbody>
</table>

Pipelined : \(20,000 \times 8\text{ns} = 160,000\text{ns}\)

Non-pipelined : \(20,000 \times 22\text{ns} = 440,000\text{ns}\)

Speedup : \(440,000 / 160,000 = 2.75\)

Pipelined : \(20,000 \times 6\text{ns} = 120,000\text{ns}\)

Non-pipelined : \(20,000 \times 22\text{ns} = 440,000\text{ns}\)

Speedup : \(440,000 / 120,000 = 3.66\)
Assume the above 4-stage pipeline, and that the computer on which this pipeline is used is a pipelined architecture.

What is the approximate decrease in time for executing 20,000 instructions if step IF is split into IF1 and IF2, with latencies 5ns each, and step WB is split into WB1 and WB2 with 6ns and 4ns, respectively? Assume that all instructions are independent.

(in class exercise)
Today

A detailed latency example
   Pipeline depth
   Synchronization
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1ns</td>
<td>2ns</td>
<td>1ns</td>
</tr>
</tbody>
</table>

• We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
• Eventually there is a back-log of delays

Q: What does “back-log” mean?
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

1ns  2ns  1ns
A     B     C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

Q: What does “back-log” mean?

Q. Assume 20 independent instructions are queued and ready to be processed by the above 3-stage pipeline. Also assume a pipelined architecture. How many nanoseconds is instruction 12 stalled in the pipeline?

(in class exercise)
A detailed example

**Throughput AND latency are governed by the slowest pipeline stage**

Assume a 3-stage pipeline, with stages A, B and C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

The “grid-paper” proof:

Q: If there were a single instruction, how long would it take this pipeline to complete it?

<table>
<thead>
<tr>
<th></th>
<th>1ns</th>
<th>2ns</th>
<th>1ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

The “grid-paper” proof:

Q: If there were a single instruction, how long would it take this pipeline to complete it?

4ns
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

1ns 2ns 1ns

A B C

• We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
• Eventually there is a back-log of delays

The “grid-paper” proof:

Q: If there were 2 instructions, and assuming that the machine architecture is pipelined, how many ns are needed for this pipeline to complete the 2 instructions?
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

The “grid-paper” proof:

Q: If there were 2 instructions, and assuming that the machine architecture is pipelined, how many ns are needed for this pipeline to complete the 2 instructions?

Instruction 2 is stalled 1ns, has a pipeline latency of 5ns, and completes in 6ns from the start of the instruction stream
A detailed example

**Throughput AND latency are governed by the slowest pipeline stage**

Assume a 3-stage pipeline, with stages A, B and C:

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage.
- Eventually there is a back-log of delays.

The “grid-paper” proof:

Q: How many delays (stalls) does a third instruction encounter? And what is the total nanoseconds needed to complete 3 instructions?
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

<table>
<thead>
<tr>
<th></th>
<th>1ns</th>
<th>2ns</th>
<th>1ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

The “grid-paper” proof:

Q: How many delays (stalls) does a third instruction encounter? And what is the total nanoseconds needed to complete 3 instructions?
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

The “grid-paper” proof:
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

The “grid-paper” proof:

The throughput is 1 instruction every 2 ns (equal to the slowest pipeline stage)
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a backlog of delays

The “grid-paper” proof:

Delays (stall times) for each instruction are getting longer and longer
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

• We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
• Eventually there is a back-log of delays

Q: If delays are getting longer and longer, how might the pipeline be re-engineered so that delays do not propagate to infinity?
**A detailed example**

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B, and C

- We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
- Eventually there is a back-log of delays

Q: If delays are getting longer and longer, how might the pipeline be re-engineered so that delays do not propagate to infinity?

Answer: Make EACH pipeline stage as long as the longest stage

Q: How does that affect delay times?
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

1ns  2ns  1ns
A    B    C

• We mentioned that the pipeline’s throughput is governed by the slowest pipeline stage
• Eventually there is a back-log of delays

Q: If delays are getting longer and longer, how might the pipeline be re-engineered so that delays do not propagate to infinity?

Answer: Make EACH pipeline stage as long as the longest stage

Q: How does that affect delay times?
Answer: The pipeline takes longer to complete a single instruction (6ns instead of 4ns), but if many instructions are executed, then delays do NOT backlog
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

The "grid-paper" proof:
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

![Diagram showing a 3-stage pipeline with stages A, B, and C, illustrating throughput and latency.]

The “grid-paper” proof:

The throughput is STILL 1 instruction every 2 ns (equal to the duration of the slowest pipeline stage)
A detailed example

Throughput AND latency are governed by the slowest pipeline stage

Assume a 3-stage pipeline, with stages A, B and C

The “grid-paper” proof:

Overall each instruction takes 6ns to complete, and the first instruction exists at t=6 instead of t=4

Q: What is the advantage of the 2-2-2ns pipeline over the 1-2-1ns pipeline?
A detailed example

A brain teaser question: assume the following two pipelines, P1 and P2

Q: How many instructions must be executed before P2 completes a corresponding instruction earlier than if it were executed on P1?

Instruction 12 completes at t=26

Q: What is the advantage of balancing the pipeline stages, if execution time does not change much between P1 and P2?
Once a pipeline is balanced, ALL stages have the same latency.

Q: How can we assess a pipeline’s performance using only the count of stages?
Pipeline depth

The *depth* of a pipeline is how many distinct stages it has.

Depth is not concerned with time (latency) but only the NUMBER of stages.
Pipeline depth

The *depth* of a pipeline is how many distinct stages it has.

Depth is not concerned with time (latency) but only the NUMBER of stages.

If a pipeline is balanced (each stage takes the same amount of time) ....

Q: For a balanced 4-stage pipeline, how many cycles are needed to process 30 instructions which are independent?

(in class exercise)
Pipeline depth

Q: What is a formula in terms of depth, $d$, and number of instructions, $n$, that specifies the number of cycles needed to execute $n$ instructions? (a “cycle” is the “time” it takes to perform each pipeline stage)

$d = \square$

$n = \square$

# of cycles needed: \square
Q: What is a formula in terms of depth, \( d \), and number of instructions, \( n \), that specifies the number of cycles needed to execute \( n \) instructions? (a “cycle” is the “time” it takes to perform each pipeline stage)

\[ d = 3 \]
\[ n = 4 \]

# of cycles needed : 6
Q: What is a formula in terms of depth, $d$, and number of instructions, $n$, that specifies the number of cycles needed to execute $n$ instructions? (a “cycle” is the “time” it takes to perform each pipeline stage)

$d = 3$

$n = 4$

# of cycles needed: 6

$d = ?$

$n = ?$

# of cycles needed: ?
Q: What is a formula in terms of depth, \( d \), and number of instructions, \( n \), that specifies the number of cycles needed to execute \( n \) instructions? (a “cycle” is the “time” it takes to perform each pipeline stage)

\[
\text{# cycles} = d + n - 1
\]

For \( d = 3 \) and \( n = 4 \), the number of cycles needed is 6.

For \( d = 4 \) and \( n = 5 \), the number of cycles needed is 8.
Q: What is a formula in terms of depth, $d$, and number of instructions, $n$, that specifies the number of cycles needed to execute $n$ instructions? (a “cycle” is the “time” it takes to perform each pipeline stage)

$$d = 3 \quad n = 4 \quad \text{# of cycles needed: 6}$$

Do you see a pattern? What is a possible formula?

$$\text{# cycles} = d + n - 1$$

$$d = 4 \quad n = 5 \quad \text{# of cycles needed: 8}$$
Q: In addition to balancing a pipeline, what other approach might a computer architect employ to improve the efficiency (reduce the number of stalls) in a pipeline?
Q: In addition to balancing a pipeline, what other approach might a computer architect employ to improve the efficiency (reduce the number of stalls) in a pipeline?

Add more (duplicate) functional units in the pipeline

At each time step, two As, Bs, Cs are available. This is called *instruction level parallelism*, and is different from parallelism that involves multiple cores.
Synchronization

At this point hopefully I’ve convinced you that structural and data hazards may induce delays in a pipeline when processing multiple instructions “at the same time”

We’ll refer to architecture again in this course, but keep in mind that even future concurrency examples that do NOT explicitly discuss architecture are motivated by architecture components

... now on to more “abstract” examples
Synchronization

We’ve seen already that at any one moment in time, the computer is processing many “commands”

**Thread**: the smallest sequence of instructions that can be independently scheduled and managed by the operating system (scheduler)

Q: If events are dependent (one must happen after another), how do we enforce (correct) scheduling?
Synchronization

When more than 1 thread is running, synchronization is important (we’ve already seen this in the data hazard example illustrated in a previous lecture)

Some terminology, assuming two events:

**Serialization**: Event A must happen before Event B

**Mutual Exclusion**: Events A and B must NOT happen at the same time

**Concurrent**: ?
Up Next ...

Instruction Scheduling
Concurrent Writes
Concurrent Updates
Mutual Exclusion