CSCI 322
Principles of Concurrent Programming

Filip Jagodzinski
Announcements

- Homework #1 has been posted to the course website and Canvas
  - 4 “book” questions
  - One programming task

- First lab tomorrow, Thursday
  - Linux refresher
  - Threads
Q: In the context of instruction pipelines, why are stalls necessary?
From last time ...

Because a stage of a pipeline can be processing a single task at any one time, stalls must be introduced when multiple instructions are being carried out to completion.

2ns delay
From last time ...

Because a stage of a pipeline can be processing a single task at any one time, stalls must be introduced when multiple instructions are being carried out to completion.

Q: The above stall is caused by what type of hazard?
From last time ...

Because a stage of a pipeline can be processing a single task at any one time, stalls must be introduced when multiple instructions are being carried out to completion.

Latency: ______________________________________________________________
Throughput: ___________________________________________________________
From last time …

Because a stage of a pipeline can be processing a single task at any one time, stalls must be introduced when multiple instructions are being carried out to completion.

Latency: The time or number of clock cycles needed to complete one instruction*

Throughput: The number of instructions that are completed per some unit of time

* Latency can refer to the pipeline latency, or a stage latency
From last time …

Instruction 3 is delayed by instruction 2, which was delayed by instruction 1

Q: Is the maximum number of stalls that an instruction experiences always 1?
Q: How do we calculate the speedup for a pipelined versus a non-pipelined machine architecture
From last time ...

Q: How do we calculate the speedup for a pipelined versus a non-pipelined machine architecture

For 20,000 instructions ...

Execution time for pipelined machine : $20,000 \times 8\text{ns} = 160,000\text{ns}$
Execution time for non-pipelined machine : $20,000 \times 22\text{ns} = 440,000\text{ns}$
Speedup : $\frac{440,000}{160,000} = 2.75$

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: \( R3 <- R1 + R2 \)

Instruction 2: \( R4 <- R3 + R5 \)

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.
Assume that each of the IF, ID, EX and WB steps of the above 4-step pipeline require 1 clock cycle. For the below 3 instructions (register), how many clock cycles are needed to fully complete them?

Instruction 1 : R3 ← R1 + R2  
Instruction 2 : R4 ← R3 + R5  
Instruction 3 : R3 ← R4 + R2

(in class exercise)
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?
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?

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 3ns 5ns 8ns</td>
<td>6ns 3ns 5ns 4ns 4ns</td>
</tr>
<tr>
<td>IF ID EX WB</td>
<td>IF ID EX WBa WBB</td>
</tr>
</tbody>
</table>

Pipelined : 20,000 * 8ns = 160,000ns
Non-pipelined : 20,000 * 22ns = 440,000ns
Speedup : 440,000 / 160,000 = 2.75

Pipelined : 20,000 * 6ns = 120,000ns
Non-pipelined : 20,000 * 22ns = 440,000ns
Speedup : 440,000 / 120,000 = 3.66
Today

- Stalls
- Pipeline Balancing
- Instruction dependencies
- Instruction Scheduling
- Concurrent Writes
- Concurrent Updates
- Mutual Exclusion
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: 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:

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

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

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 a single instruction, how long would it take this pipeline to complete it?
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

- 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
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: 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

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: 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

1ns  2ns  1ns
\[ \text{A} \quad \text{B} \quad \text{C} \]

- 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: What is the pipeline’s throughput?
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

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:

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

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?

Assume the 2ns stage cannot be broken into to smaller ones
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

This is called balancing a pipeline

```
2ns  2ns  2ns
A    B    C
```

**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

2ns 2ns 2ns
A B C

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

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?
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 = 3

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

*(in class exercise)*
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)

\[
d = \text{?} \\
n = \text{?} \\
\text{# of cycles needed: ?}
\]
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)

$d = 3$
$n = 4$

# 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 : \_\_\_\_\_\_\_

$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)

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

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

\( d = 3 \)
\( n = 4 \)
\( \text{# of cycles needed} : 6 \)

\( d = 4 \)
\( n = 5 \)
\( \text{# of cycles needed} : 8 \)
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)

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

Do you see a pattern?

What is a possible formula?

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

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 sometimes is called a superscalar or multiscalar pipeline. It 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
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)

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**: ?
Synchronization – the Lunch example

Scenario: You and Bob live in different cities
        You want to know who ate lunch first

Q: How do you find out if you or Bob ate lunch first?
Synchronization – the Lunch example

Scenario: You and Bob live in different cities
       You want to know who ate lunch first

Q: How do you find out if you or Bob ate lunch first?

A. Call Bob at 3pm, and ask, “Heya Bob, when did you eat lunch?”
B. Call Bob at 8am, and ask him, “Heya Bob, when will you eat lunch?”
C. Both of the above will guarantee you find out for certain if you and Bob lunch at the same time
D. Neither Choices A nor B can guarantee that you and Bob eat lunch at the same time
Scenario: You and Bob live in different cities
You want to know who ate lunch first

Q: How do you find out if you or Bob ate lunch first?

A. Call Bob at 3pm, and ask, “Heya Bob, when did you eat lunch?”
B. Call Bob at 8am, and ask him, “Heya Bob, when will you eat lunch?”
C. Both of the above will guarantee you find out for certain if you and Bob lunch at the same time
D. Neither Choices A nor B can guarantee that you and Bob eat lunch at the same time

Q: Why is A not the answer?
Synchronization – the Lunch example

Scenario: You and Bob live in different cities
You want to know who ate lunch first

Q: How do you find out if you or Bob ate lunch first?

A. Call Bob at 3pm, and ask, “Heya Bob, when did you eat lunch?”
B. Call Bob at 8am, and ask him, “Heya Bob, when will you eat lunch?”
C. Both of the above will guarantee you find out for certain if you and Bob lunch at the same time
D. Neither Choices A nor B can guarantee that you and Bob eat lunch at the same time

Q: Why is A not the answer?

If you called and ask at what time he ate, you cannot know if yours and Bob’s clocks are synchronized, so even if you ate at 12:16pm, and he said, “12:16pm,” you are still unsure
Synchronization – the Lunch example

Scenario: You and Bob live in different cities
You want to know who ate lunch first

Q: Assuming Bob is willing to follow instructions, how can you guarantee that you and Bob eat lunch at the same time?
Synchronization – the Lunch example

Scenario: You and Bob live in different cities
You want to know who ate lunch first

Q: Assuming Bob is willing to follow instructions, how can you guarantee that you and Bob eat lunch at the same time?

You instruct Bob to NOT starting eating lunch until you call. Hence, you send a message.

Message passing is the foundation of many solutions to synchronizing multiple instructions streams.
Synchronization – the Lunch example

You
- Eat breakfast
- Work
- Eat Lunch
- Call Bob

Bob
- Eat breakfast
- Wait for a call
- Eat Lunch

Your thread of execution
Bob’s thread of execution
Synchronization – the Lunch example

You
- Eat breakfast
- Work
- Eat Lunch
- Call Bob

Bob
- Eat breakfast
- Wait for a call
- Eat Lunch

Your thread of execution
Bob’s thread of execution

Because programs are executed sequentially (starting with the first instruction and proceeding through to the last), we know the ORDER of events for both your and Bob’s threads.

You and Bob ate lunch **sequentially** (in the same order of events)
Synchronization – the Lunch example

You
• Eat breakfast
• Work
• Eat Lunch
• Call Bob

Bob
• Eat breakfast
• Wait for a call
• Eat Lunch

Your thread of execution

Bob’s thread of execution

The issue is that there is no way to compare events from different threads ... who ate breakfast first?
Synchronization – the Lunch example

You
- Eat breakfast
- Work
- Eat Lunch
- Call Bob

Bob
- Eat breakfast
- Wait for a call
- Eat Lunch

Your thread of execution

Bob’s thread of execution

The issue is that there is no way to compare events from different threads ... who ate breakfast first?

You and Bob ate lunch **Concurrently** (order undetermined)
Synchronization – the Lunch example

You
- Eat breakfast
- Work
- Eat Lunch
- Call Bob

Bob
- Eat breakfast
- Wait for a call
- Eat Lunch

Your thread of execution

Bob’s thread of execution

The issue is that there is no way to compare events from different threads ... who ate breakfast first?

Two events are **concurrent** IF you cannot tell by looking at them which happens (or will happen) first
Synchronization – the Lunch example

You
• Eat breakfast
• Work
• Eat Lunch
• Call Bob

Bob
• Eat breakfast
• Wait for a call
• Eat Lunch

Your thread of execution

Bob’s thread of execution

The issue is that there is no way to compare events from different threads ... who ate breakfast first?

Q: How might you and Bob coordinate eating lunch?
Synchronization – the Lunch example

You
• Eat breakfast
• Work
• Eat Lunch
• Call Bob

Bob
• Eat breakfast
• Wait for a call
• Eat Lunch

Your thread of execution
Bob’s thread of execution

The issue is that there is no way to compare events from different threads ... who ate breakfast first?

By passing a message from one thread to the next (the phone call), both threads are synchronized.
Synchronization – the Lunch example

You
• y1: Eat breakfast
• y2: Work
• y3: Eat Lunch
• y4: Call Bob

Bob
• b1: Eat breakfast
• b2: Wait for a call
• b3: Eat Lunch

“label” each event in both execution threads

Q: Assuming y4 is “matched” to b2, does that change whether you and Bob eat breakfast concurrently?
Synchronization – the Lunch example

You
• Eat breakfast
• 8am: Work
• 9am: Eat Lunch
• 10am: Call Bob

Bob
• 7am: Eat breakfast
• 10am: Wait for a call
• 11am: Eat Lunch

“label” each event in both execution threads

Q: Assuming y4 is “matched” to b2, does that change whether you and Bob eat breakfast concurrently? No

Just because the threads are synchronized at “10am,” you could have eaten breakfast at 6am or 5am, so eating breakfast still happens concurrently.
Q: In the above example, what is the motivation for sending a “message” from one thread (you) to another (Bob)?

Abstractly ... two “threads” of execution (you and Bob) are synchronized so that “you” eat lunch before Bob

Q: On a single core computer, how can the above two threads complete/execute differently from one invocation to another?
In the above scenario, we ONLY care about you eating lunch before Bob eats his. Either you or Bob can eat breakfast first.

Q: What are the possible orderings of i1 through i7 such that i4 happens BEFORE i7?
Instruction Scheduling

You
• i1: Eat breakfast
• i2: Work
• i3: Eat Lunch
• i4: Call Bob

Bob
• i5: Eat breakfast
• i6: Wait for a call
• i7: Eat Lunch

In the above scenario, we ONLY care about you eating lunch before Bob eats his. Either you or Bob can eat breakfast first.

Q: What are the possible orderings of i1 through i7 such that i4 happens BEFORE i7?

Order A : i1, i5, i2, i6, i7, i3, i4
Order B : i1, i2, i5, i6, i7, i3, i4
Order C : i1, i2, i5, i3, i4, i6, i7
Order D : i1, i2, i3, i4, i5, i6, i7
...

Order of completion?
In the above scenario, we ONLY care about you eating lunch before Bob eats his. Either you or Bob can eat breakfast first.

**Q:** What are the possible orderings of i1 through i7 such that i4 happens BEFORE i7?

- **Order A:** i1, i5, i2, i6, i7, i3, i4
- **Order B:** i1, i2, i5, i6, i7, i3, i4
- **Order C:** i1, i2, i5, i3, i4, i6, i7
- **Order D:** i1, i2, i3, i4, i5, i6, i7

**msg**

Incorrect order
Incorrect order
Correct order
Correct order
A pipeline allows “more” instructions to be executed “concurrently”....
A pipeline allows “more” instructions to be executed “concurrently” ....

A multiscalar architecture duplicates the pipeline units
A pipeline allows “more” instructions to be executed “concurrently” ....

A multiscalar architecture duplicates the pipeline units

Duplicating the register file and/or controls units allows a computer to thread (multithreading)

(much more on this later on in the course)
public class SillyMath{
    public static void main (String args[]){
        int i=3, j=46, k=4, l=76, prod, sum;
        double sRoot;
        prod = i * j;
        sum = k + l;
        sRoot = Math.pow((double)76,0.3);
    }
}

Q: Which portions of this code can be threaded?
Instruction Scheduling

public class SillyMath{
    public static void main (String args[]){
        int i=3, j=46, k=4, l=76, prod, sum;
        double sRoot;
        prod = i * j;
        sum = k + l;
        sRoot = Math.pow((double)76,0.3);
    }
}

The byte code equivalent of u might actually be multiple instructions, u1, u2, etc. Likewise for v.
public class SillyMath{
    public static void main (String args[]){
        int i=3, j=46, k=4, l=76, prod, sum;
        double sRoot;
        prod = i * j;
        sum = k + l;
        sRoot = Math.pow((double)76,0.3);
    }
}
public class SillyMath{
    public static void main (String args[]){
        int i=3, j=46, k=4, l=76, prod, sum;
        double sRoot;
        prod = i * j;
        sum = k + l;
        sRoot = Math.pow((double)76,0.3);
    }
}
Synchronization ...

You: eat breakfast, work, eat lunch, call Bob

Bob: eat breakfast, wait for call, eat lunch

Lunch synchronization by message passing ...

This ensured the order of who eats lunch first/second
Synchronization ...

Main Memory

L2 Cache

L1 data

Register file

Control units

pipeline

You: eat breakfast, work, eat lunch, call Bob

Bob: eat breakfast, wait for call, eat lunch

Scenario: You and Bob are bitter enemies (no sending of messages), AND you do NOT want to be in the lunch room at the same time ... how can this be achieved?
Synchronization ...

You: eat breakfast, work, eat lunch, call Bob

Bob: eat breakfast, wait for call, eat lunch

Keep a conch shell in the office ... ONLY if you have the conch shell can you go into the lunch room
Synchronization...

You (y):
- Eat breakfast
- Retrieve conch shell
- Eat Lunch
- Put conch shell back

Bob (b):
- Eat breakfast
- Take conch shell
- Eat Lunch
- Put conch shell back
Synchronization ...

You (y)
- Eat breakfast
- Retrieve conch shell
- Eat Lunch
- Put conch shell back

Bob (b)
- Eat breakfast
- Take conch shell
- Eat Lunch
- Put conch shell back

Goal: enforce mutual exclusion so that only one person at any time can be eating lunch

Q: How might you enforce the “conch” shell approach?
You (y)
- Eat breakfast
- Retrieve conch shell
- Eat Lunch
- Put conch shell back

Bob (b)
- Eat breakfast
- Take conch shell
- Eat Lunch
- Put conch shell back

Conch = \{0,1,2,3,4,...\}

Q: How might you enforce the “conch” shell approach?

Goal: enforce mutual exclusion so that only one person at any time can be eating lunch.
Shared Variables – concurrent write

Thread A

i1: x=5
i2: print x

Thread B

i3: x=7
Q: What path (scheduling sequence) prints a 5 and sets the final value to 5?
Q: What path (scheduling sequence) prints a 7 and sets the final value to 7?
Q: What path (scheduling sequence) prints a 5 and sets the final value to 5?
Q: What path (scheduling sequence) prints a 7 and sets the final value to 7?
Q: What is the difference between an update and a write?
Shared Variables – concurrent updates

Thread A

i1: x=5
i2: print x

These are examples of concurrent **writes**

Thread B

i3: x=7

Q: What is the difference between an update and a write?

i1: count = count + 1
i2: count = count + 1
Shared Variables – concurrent updates

**Q:** What is the difference between an update and a write?

- **Thread A**
  - i1: `x=5`
  - i2: `print x`

- **Thread B**
  - These are examples of concurrent writes
  - i3: `x=7`

**An update** reads a value in a variable, computes a new value based on the “old” value, and writes the new value to the variable.

**Q:** What is a synchronization error that might arise among updates for multiple threads?

- i1: `count = count + 1`
- i2: `count = count + 1`
**Shared Variables – concurrent updates**

Remember that “executing” an instruction involves multiple pipeline steps, including loading registers, loading ALUs, executing ALUs, fetching results from ALU, etc.
**Shared Variables – concurrent updates**

<table>
<thead>
<tr>
<th>Thread A</th>
<th>Thread B</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1: count = count + 1</td>
<td>i2: count = count + 1</td>
</tr>
<tr>
<td>a1: load count</td>
<td>b1: load count</td>
</tr>
<tr>
<td>a2: add 1</td>
<td>b2: add 1</td>
</tr>
<tr>
<td>a3: store count</td>
<td>b3: store count</td>
</tr>
</tbody>
</table>

Register/ALU “view”

Remember that “executing” an instruction involves multiple pipeline steps, including loading registers, loading ALUs, executing ALUs, fetching results from ALU, etc.

Assume initial value of count = 4

**Q: What is final value of count if execution order is a1 < b1 < a2 < b2 < a3 < b3?**

(in class exercise)
Shared Variables – concurrent updates

Assume initial value of count = 4

Q: What execution order do we need to impose so thread A’s “update” does not interfere with thread B’s update?

Remember that “executing” an instruction involves multiple pipeline steps, including loading registers, loading ALUs, executing ALUs, fetching results from ALU, etc.

Assume initial value of count = 4

Q: What execution order do we need to impose so thread A’s “update” does not interfere with thread B’s update?
Shared Variables – concurrent updates

Assume initial value of count = 4

Q: What execution order do we need to impose so thread A’s “update” does not interfere with thread B’s update?

Sample “solution” : a1 < a2 < a3 < b1 < b2 < b3
Sample “solution” : b1 < b2 < b3 < a1 < a2 < a3

Q: What similarities do these two “solutions” share?

Remember that “executing” an instruction involves multiple pipeline steps, including loading registers, loading ALUs, executing ALUs, fetching results from ALU, etc.
Shared Variables – concurrent updates

Remember that “executing” an instruction involves multiple pipeline steps, including loading registers, loading ALUs, executing ALUs, fetching results from ALU, etc.

An operation or instruction stream that CANNOT be interrupted is called **atomic**.

Sample “solution”:

- $a_1 < a_2 < a_3 < b_1 < b_2 < b_3$
- $b_1 < b_2 < b_3 < a_1 < a_2 < a_3$
take home exercise: Assuming a starting value of count = 4, what are all possible final values of count if Threads A and B are run concurrently, to completion?
Up Next

- Cache Friendly code
- Concurrency
- `await`