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.
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
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?
Today

Throughput / Latency
Data hazards
Pipeline Calculations, Speedup, Latency

Q: An instruction is completed every ____ ns
Q: A single instruction takes ______ ns to complete
**Pipeline Calculations, Speedup, Latency**

Q: An instruction is completed every _____ ns
Q: A single instruction takes ______ ns to complete

These are “it depends” questions. Because, the answer depends on how many instructions are being processed.

For a single instructions, the answers to these questions are straight-forward.

Q: If there are 20,000 instructions to be completed, then what is the throughput and what is the latency of our 4-stage pipeline. Assume that as soon as a stage is “free”, then the next instruction “in line” is immediately piped into that stage. Also assume that all instructions are independent of the others.
Pipeline Calculations, Speedup, Latency

Q: An instruction is completed every _____ ns
Q: A single instruction takes ______ ns to complete

**Throughput**: The pipeline is governed by the slowest step, which in this case is the WB, so the throughput is 1 instruction / 8ns.
Pipeline Calculations, Speedup, Latency

Q: An instruction is completed every _____ ns

Q: A single instruction takes ______ ns to complete

**Latency**: 6+3+5+8=22ns. Is that correct for 20,000 instructions? Why or why not?
Pipeline Calculations, Speedup, Latency

Q: An instruction is completed every _____ ns
Q: A single instruction takes ______ ns to complete

Latency: 6+3+5+8=22ns. Is that correct for 20,000 instructions? Why or why not?

NO! We’ve already seen that stalls needs to be introduced. So what is the latency?
Pipeline Calculations, Speedup, Latency

Q: An instruction is completed every _____ ns

Q: A single instruction takes ______ ns to complete

Latency: 6+3+5+8=22ns more than 22ns

After a large number of instructions have been processed, the stalls will have back-logged, and delays will begin to accumulate.

Thus .... throughput AND latency are governed by the slowest pipeline stage

We’ll answer today : How should/can the pipeline be re-engineered to eliminate or reduce stalls?
Q: How much faster is the pipelined machine architecture versus a non-pipelined machine architecture (assume a large number of instructions)?

Remember that a pipelined machine architecture means that instructions are processed and interwoven, while a pipeline refers to the stages that are involved.
Q: How much faster is the pipelined machine architecture versus a non-pipelined machine architecture (assume a large number of instructions)?

Remember that a pipelined machine architecture means that instructions are processed and interwoven, while a pipeline refers to the stages that are involved.

For the first homework be able to compute:

Execution time for pipeline architecture:
Execution time for non-pipelined architecture:
Speedup:

On the board discussion
Q: How much faster is the pipelined machine architecture versus a non-pipelined machine architecture (assume a large number of instructions)?

Remember that a pipelined machine architecture means that instructions are processed and interwoven, while a pipeline refers to the stages that are involved.

For the first homework be able to compute:

<table>
<thead>
<tr>
<th></th>
<th>Pipeline Architecture</th>
<th>Non-Pipelined Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time</td>
<td>20,000 * 8ns = 160,000ns</td>
<td>20,000 * 22ns = 440,000ns</td>
</tr>
<tr>
<td>Speedup</td>
<td>440,000 / 160,000 = 2.75</td>
<td></td>
</tr>
</tbody>
</table>

This is an approximation, because at time zero, when the pipeline is not yet fully loaded, output is not 1 instruction every 8 ns. For large numbers of instructions, this approximation approaches very closely the “real” execution time.
Now that we know that the 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?
Pipeline Calculations, Speedup, Latency

Now that we know that the 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: What effect does this have on pipeline throughput? And latency?

On-the-board discussion
Pipeline Calculations, Speedup, Latency

Compare and contrast these two pipelines by answering:

Q: For an initially empty pipeline, what is the pipeline latency and throughput of a single instruction?

Q: For an initially empty pipeline, what is the pipeline latency and throughput for the 50,000\textsuperscript{th} instruction (assume instructions are independent and are queued at time 0)
At the end of last lecture I mentioned two complications that arise in pipelines, but I’ve only discussed one of them. What’s the other one?

Complication 1: Structural hazards (governed by slowest stage)
Complication 2: ________________________________
Data Hazards ...

From the “code” perspective

Instruction 1: \( a = b + c \)

Instruction 2: \( d = a + e \)

Q: How quickly can these two instructions be processed by a pipeline

This seems like innocent-enough code. Q: But what could go wrong?
Remember to “think” like a computer processes instructions
Data Hazards ...

From the “code” perspective:

Instruction 1: $a = b + c$

Instruction 2: $d = a + e$

When we see such code in a program, we assume that instruction 2 is “executed” after instructions 1 is “executed,” which is a requirement, because the $a$ operand in instruction 2 is the output of instruction 1.

Because CPUs employ pipelines, it’s not that straight-forward.

To see how/why data hazards come up, let’s think about this from a register perspective.
From the “code” perspective
Instruction 1: \( a = b + c \)
Instruction 2: \( d = a + e \)

From the “register” perspective
Instruction 1: \( R3 \leftarrow R1 + R2 \)
Instruction 2: \( R4 \leftarrow R3 + R5 \)

Q: What does the above register terminology specify?
Data Hazards ...

From the “code” perspective

Instruction 1: \[a = b + c\]
Instruction 2: \[d = a + e\]

From the “register” perspective

Instruction 1: \[R3 \leftarrow R1 + R2\]
Instruction 2: \[R4 \leftarrow R3 + R5\]

Q: What does the above register terminology specify?

Q: Assuming each stage of the 4-stage pipeline consumes a single clock cycle, how many clock cycles are needed to complete the above 2 instructions?
Data Hazards ...

Clock cycle = 0

Instruction 1 : R3 <- R1 + R2

Instruction 2 : R4 <- R3 + R5

At time 0, which stages of the pipeline are being utilized?
Data Hazards ...

Clock cycle = 1

Instruction 1 : R3 <- R1 + R2

Instruction 2 : R4 <- R3 + R5

At clock cycle 1, which stage(s) of the pipeline are being utilized?
Data Hazards ...

Clock cycle = 1

Instruction 1 : R3 <- R1 + R2

Instruction 2 : R4 <- R3 + R5

At clock cycle 1, which stage(s) of the pipeline are being utilized?
Data Hazards ...

Clock cycle = 2

Instruction 1 : R3 <- R1 + R2

Instruction 2 : R4 <- R3 + R5

At clock cycle 2, which stage(s) of the pipeline are being utilized?
At clock cycle 2, which stage(s) of the pipeline are being utilized?

Are there any issues (yet)?
Clock cycle = 3

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

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

At clock cycle 3, which stage(s) of the pipeline are being utilized?
At clock cycle 3, which stage(s) of the pipeline are being utilized?

Are there any issues (yet)?
At clock cycle 3, which stage(s) of the pipeline are being utilized?

Are there any issues (yet)?

During clock cycle 3, instruction 2 is being decoded, which states, “get values from registers 3 and 5” ... but instructions 1 has not yet completed, because its WB stage has not yet been performed!

This is called a Read After Write (RAW) Data hazard

Notice that this is NOT a structural hazard

How do we fix this?
Data Hazards ...

Clock cycle = 3

Instruction 1 : R3 <- R1 + R2

Instruction 2 : R4 <- R3 + R5

We do NOT perform ID of Instruction 2 until AFTER WB of instruction 1 completes
Data Hazards ...

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

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

We do NOT perform ID of Instruction 2 until AFTER WB of instruction 1 completes

Clock cycle = 4
Data Hazards ...

Clock cycle = 5

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

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

We do NOT perform ID of Instruction 2 until AFTER WB of instruction 1 completes
Data Hazards ...

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

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

We do NOT perform ID of Instruction 2 until AFTER WB of instruction 1 completes
Data Hazards ...

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

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

Clock cycle = 7

We do NOT perform ID of Instruction 2 until AFTER WB of instruction 1 completes.
Stalls
Instruction dependencies