Pipelining

CSC 211 – December 1, 2020
No questions!
What is Pipelining?
throughput = 0.25 loads/hr

throughput = 1 load/hr

Instruction Fetch (IF)
Instruction Decode (ID)
Execute (EX)
Memory (MEM)
Write Back (WB)
Comparing Performance

Compute maximum instruction latency and for single-cycle and pipelined datapaths using these parameters:

IF = 150ps
ID = 100ps
EX = 100ps
MEM = 200ps
WB = 100ps
Pipeline registers = 20ps

Single-Cycle

Latency: 650ps

Throughput:

\[
\frac{1000\text{ps}}{650\text{ps/instr.}} \approx 1.5 \text{ instr/ns}
\]

\[
= 1.5 \text{ billion instr/sec.}
\]

Pipelined

Latency:

5 x 220ps = 1100ps

Throughput:

\[
\frac{1000\text{ps}}{220\text{ps/instr.}} \approx 4.5 \text{ instr/ns}
\]

\[
= 4.5 \text{ billion instr/sec.}
\]
Little's Law

\[ L = \lambda \cdot W \]

- \( L \) = length of queue (or \( N \) of tasks running concurrently)
- \( \lambda \) = completion rate (inst. throughput)
- \( W \) = wait time (inst. latency)

\[
\lambda = \frac{L}{W} \quad \Rightarrow \quad \lambda = \frac{5 \text{ inst.}}{1000 \text{ ps.}} \quad \Rightarrow \quad \frac{1 \text{ inst.}}{220 \text{ ps.}}
\]
Pipeline Practice
Show the pipeline execution for:

addi $t0, $zero, 10
addi $t1, $zero, 11
addi $t2, $zero, 12
addi $t0, $t0, 1
addi $t1, $t1, 2
addi $t2, $t2, 3
Show the pipeline execution for:

addi $t0, $zero, 0
addi $t0, $t0, 1
addi $t0, $t0, 1
addi $t0, $t0, 1
addi $t0, $t0, 1
addi $t0, $t0, 1

$\text{Assume $t0$ is initially 10}$

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

Write happens in 1st half of clock cycle

What is the value of $t0$ at the end? $t0 = 12$

How can we fix this instruction sequence?
Here's the fixed version

```assembly
addi $t0, $zero, 0
nop
nop
addi $t0, $t0, 1
nop
nop
addi $t0, $t0, 1
nop
nop
addi $t0, $t0, 1
```
Show the pipeline execution for:

lw $s0, 0($t0)
addi $t0, $t0, 4
lw $s1, 0($t1)
addi $t1, $t1, 4
add $s2, $s0, $s1

Static Scheduling Continued: reorder instructions

Where will this sequence go wrong? (Where does it differ from a single-cycle datapath?)
The last instruction uses $s1 before it has been written

How can we fix this sequence w/ static scheduling? Swap the order of the 2nd and 3rd insts
Show the pipeline execution for:

A  addi $t0, $zero, 0  -  A
B  add  $t0, $t0, $s0       D  nop
C  add  $t0, $t0, $s1       B  nop
D  addi $t1, $zero, 1  -  E  nop
E  add  $t1, $t1, $s2       C  F
F  add  $t1, $t1, $s3

1. Where does this instruction sequence go wrong on a pipelined datapath?
   Instructions B, C, E and F read values before they are ready.

2. How can we fix it with static scheduling?
   See schedule above.
Show the pipeline execution for:

addi $s0, $zero, 0
j somewhere
addi $s0, $s0, 1
addi $s1, $s1, 1
...
somewhere:
addi $s1, $zero, 1
Controlling a Pipelined Datapath