Lecture: Pipelining Basics

• Topics: Performance equations wrap-up, Basic pipelining implementation

  ▪ Video 1: What is pipelining?
  ▪ Video 2: Clocks and latches
  ▪ Video 3: An example 5-stage pipeline
  ▪ Video 4: Loads/Stores and RISC/CISC

• Turn in HW1
• Guest teacher, Manju Shevgoor, on Monday
An Alternative Perspective - I

• Each program is assumed to run for an equal number of cycles, so we’re fair to each program

• The number of instructions executed per cycle is a measure of how well a program is doing on a system

• The appropriate summary measure is sum of IPCs or AM of IPCs = \frac{1.2 \text{ instr}}{\text{cyc}} + \frac{1.8 \text{ instr}}{\text{cyc}} + \frac{0.5 \text{ instr}}{\text{cyc}}

• This measure implicitly assumes that 1 instr in prog-A has the same importance as 1 instr in prog-B
An Alternative Perspective - II

• Each program is assumed to run for an equal number of instructions, so we’re fair to each program

• The number of cycles required per instruction is a measure of how well a program is doing on a system

• The appropriate summary measure is sum of CPIs or AM of CPIs = \( \frac{0.8 \text{ cyc}}{\text{instr}} + \frac{0.6 \text{ cyc}}{\text{instr}} + \frac{2.0 \text{ cyc}}{\text{instr}} \)

• This measure implicitly assumes that 1 instr in prog-A has the same importance as 1 instr in prog-B
AM vs. GM

• GM of IPCs = 1 / GM of CPIs

• AM of IPCs represents throughput for a workload where each program runs sequentially for 1 cycle each; but high-IPC programs contribute more to the AM

• GM of IPCs does not represent runtime for any real workload (what does it mean to multiply instructions?); but every program’s IPC contributes equally to the final measure
Problem 6

• My new laptop has a clock speed that is 30% higher than the old laptop. I’m running the same binaries on both machines. Their IPCs are listed below. I run the binaries such that each binary gets an equal share of CPU time. What speedup is my new laptop providing?

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>AM</th>
<th>GM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old-IPC</td>
<td>1.2</td>
<td>1.6</td>
<td>2.0</td>
<td>1.6</td>
<td>1.57</td>
</tr>
<tr>
<td>New-IPC</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
</tr>
</tbody>
</table>

AM of IPCs is the right measure. Could have also used GM. Speedup with AM would be 1.3.
Speedup Vs. Percentage

• “Speedup” is a ratio = old exec time / new exec time

• “Improvement”, “Increase”, “Decrease” usually refer to percentage relative to the baseline
  = (new perf – old perf) / old perf

• A program ran in 100 seconds on my old laptop and in 70 seconds on my new laptop
  ▪ What is the speedup?  \( \frac{1}{70} / \frac{1}{100} = 1.42 \)
  ▪ What is the percentage increase in performance?
    \( \frac{1}{70} - \frac{1}{100} \) / \( \frac{1}{100} \) = 42%
  ▪ What is the reduction in execution time?  30%
Building a Car

Unpipelined

Start and finish a job before moving to the next

Jobs

Time
The Assembly Line

Pipelined

Break the job into smaller stages

Jobs

Time
Clocks and Latches

Stage 1 \hspace{5cm} \rightarrow \hspace{5cm} \text{Stage 2}
Clocks and Latches

Stage 1 → L → Stage 2

Clk
Some Equations

- Unpipelined: time to execute one instruction = $T + T_{ovh}$
- For an N-stage pipeline, time per stage = $T/N + T_{ovh}$
- Total time per instruction = $N (T/N + T_{ovh}) = T + N T_{ovh}$
- Clock cycle time = $T/N + T_{ovh}$
- Clock speed = $1 / (T/N + T_{ovh})$
- Ideal speedup = $(T + T_{ovh}) / (T/N + T_{ovh})$
- Cycles to complete one instruction = $N$
- Average CPI (cycles per instr) = 1
Problem 1

• An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 equal sequential pipeline stages. Answer the following, assuming that there are no stalls in the pipeline.

- What are the cycle times in the two processors?
- What are the clock speeds?
- What are the IPCs?
- How long does it take to finish one instr?
- What is the speedup from pipelining?
Problem 1

- An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 equal sequential pipeline stages. Answer the following, assuming that there are no stalls in the pipeline.

- What are the cycle times in the two processors? 5.2ns and 1.2ns
- What are the clock speeds? 192 MHz and 833 MHz
- What are the IPCs? 1 and 1
- How long does it take to finish one instr? 5.2ns and 6ns
- What is the speedup from pipelining? 833/192 = 4.34
Problem 2

- An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 sequential pipeline stages. The stages have the following lengths: 1ns; 0.6ns; 1.2ns; 1.4ns; 0.8ns. Answer the following, assuming that there are no stalls in the pipeline.

  - What is the cycle time in the new processor?
  - What is the clock speed?
  - What is the IPC?
  - How long does it take to finish one instr?
  - What is the speedup from pipelining?
  - What is the max speedup from pipelining?
Problem 2

- An unpipelined processor takes 5 ns to work on one instruction. It then takes 0.2 ns to latch its results into latches. I was able to convert the circuits into 5 sequential pipeline stages. The stages have the following lengths: 1ns; 0.6ns; 1.2ns; 1.4ns; 0.8ns. Answer the following, assuming that there are no stalls in the pipeline.

- What is the cycle time in the new processor? 1.6ns
- What is the clock speed? 625 MHz
- What is the IPC? 1
- How long does it take to finish one instr? 8ns
- What is the speedup from pipelining? 625/192 = 3.26
- What is the max speedup from pipelining? 5.2/0.2 = 26
A 5-Stage Pipeline

Source: H&P textbook
A 5-Stage Pipeline

Use the PC to access the I-cache and increment PC by 4
A 5-Stage Pipeline

Read registers, compare registers, compute branch target; for now, assume branches take 2 cyc (there is enough work that branches can easily take more)
A 5-Stage Pipeline

ALU computation, effective address computation for load/store
A 5-Stage Pipeline

Memory access to/from data cache, stores finish in 4 cycles
A 5-Stage Pipeline

Write result of ALU computation or load into register file
RISC/CISC Loads/Stores
Problem 3

- For the following code sequence, show how the instrs flow through the pipeline:
  - ADD R1, R2, → R3
  - BEZ R4, [R5]
  - LD [R6] → R7
  - ST [R8] ← R9
## Pipeline Summary

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RR</th>
<th>ALU</th>
<th>DM</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, → R3</td>
<td>Rd R1,R2</td>
<td>R1+R2</td>
<td>--</td>
<td>Wr R3</td>
</tr>
<tr>
<td>BEZ R1, [R5]</td>
<td>Rd R1, R5</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Compare, Set PC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 8[R3] → R6</td>
<td>Rd R3</td>
<td>R3+8</td>
<td>Get data</td>
<td>Wr R6</td>
</tr>
<tr>
<td>ST 8[R3] ← R6</td>
<td>Rd R3,R6</td>
<td>R3+8</td>
<td>Wr data</td>
<td>--</td>
</tr>
</tbody>
</table>
Problem 4

• Convert this C code into equivalent RISC assembly instructions

\[ a[i] = b[i] + c[i]; \]
Problem 4

- Convert this C code into equivalent RISC assembly instructions

\[ a[i] = b[i] + c[i]; \]

```
LD [R1], R2   # R1 has the address for variable i
MUL R2, 8, R3   # the offset from the start of the array
ADD R4, R3, R7   # R4 has the address of a[0]
ADD R5, R3, R8   # R5 has the address of b[0]
ADD R6, R3, R9   # R6 has the address of c[0]
LD [R8], R10   # Bringing b[i]
LD [R9], R11   # Bringing c[i]
ADD R10, R11, R12  # Sum is in R12
ST [R7], R12   # Putting result in a[i]
```
Problem 5

• Design your own hypothetical 8-stage pipeline.
Title

• Bullet