Machine Internals / Pipelining

CS230 Tutorial 05
Fun facts about The Stack

● The stack and an array are not the same
  ○ they are both part of memory, but you use them differently

● The stack is the memory the program uses to run
  ○ return addresses
  ○ saved/unsaved temporaries
  ○ whatever you want to add to it

● To access anything on the stack you need an address
  ○ this will come from $30, this is what lw and sw are for
  ○ you also need lw, sw for arrays but arrays are not the stack!
    ■ to add one spot to the stack you move $30 by -4
    ■ to move to the next element in an array you move the address of the element by +4
  ○ don’t use lw, sw to copy from one register to another, they’re meant to copy to/from memory
Machine Internals / Performance: quick recap

- **CPI = cycles per instruction**
  - determined by CPU hardware (CPU = central processing unit)
  - different instructions might have different CPI

- **Instruction count**
  - determined by program, instruction set, compiler

- **Clock cycles = instruction count * CPI**

- **CPU time = instruction count * CPI * cycle time**

\[
\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}
\]
Machine Internals / Performance: example

- Using the formulas on the previous slide, let’s try an example:

Consider two different implementations of the same instruction set architecture. There are four classes of instructions, A, B, C, and D. The clock rate and CPI of each implementation are given in the following table.

<table>
<thead>
<tr>
<th></th>
<th>Clock Rate</th>
<th>CPI Class A</th>
<th>CPI Class B</th>
<th>CPI Class C</th>
<th>CPI Class D</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1.6 GHz</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>P2</td>
<td>2 GHz</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

a. Given a program with $10^6$ instructions divided into classes as follows: 10% class A, 20% class B, 50% class C and 20% class D, which implementation is faster?
Machine Internals / Performance: solution

- Solution to the previous example

Since the clock rate for P1 is 1.6 GHz, this means:

- $1.6 \times 10^9$ cycles = 1 sec
- 1 cycle = $1/(1.6 \times 10^9)$ sec
- 1 cycle = $0.625 \times 10^{-9}$ sec OR 625 ps

Since the clock rate for P2 is 2 GHz, through a similar calculation, we see that
- 1 cycle = 500 ps

Total CPU time for the program using P1 is:

- $(0.1 \times 625 \text{ ps} + 0.2 \times 2 \times 625 \text{ ps} + 0.5 \times 3 \times 625 \text{ ps} + 0.2 \times 4 \times 625 \text{ ps}) \times 10^6$
- $= 1750000000 \text{ ps}$
- $= 1.75 \text{ milliseconds}$

The total CPU time for the program using P2 is:

- $(0.1 \times 2 \times 500 \text{ ps} + 0.2 \times 2 \times 500 \text{ ps} + 0.5 \times 2 \times 500 \text{ ps} + 0.2 \times 2 \times 500 \text{ ps}) \times 10^6$
- $= 100000000 \text{ ps}$
- $= 1 \text{ millisecond}$

This means that the P2 implementation is faster.
Machine Internals / Performance: ex. cont.

- Building off the previous example (same details):

  Consider two different implementations of the same instruction set architecture. There are four classes of instructions, A, B, C, and D. The clock rate and CPI of each implementation are given in the following table.

<table>
<thead>
<tr>
<th></th>
<th>Clock Rate</th>
<th>CPI Class A</th>
<th>CPI Class B</th>
<th>CPI Class C</th>
<th>CPI Class D</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1.6 GHz</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>P2</td>
<td>2 GHz</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

b. What is the global CPI for each implementation?
Machine Internals / Performance: solution

- Solution to the previous example

*Soln:*
The program clock cycles are divided as 10% A, 20% B, 50% C, and 20% D.
For P1 we know A = 1CPI, B = 2CPI, C = 3CPI, and D = 4CPI so we can multiply.
\[(0.1 \times 1 + 0.2 \times 2 + 0.5 \times 3 + 0.2 \times 4) = 2.8 \text{ CPI}\]

We do the same for P2, knowing that A, B, C, D all have 2CPI on P2.
\[(0.1 \times 2 + 0.2 \times 2 + 0.5 \times 2 + 0.2 \times 2) = 2 \text{ CPI}\]

OR
Since all of the individual instructions in P2's set have a CPI of 2, then P2's global CPI is also 2.
Pipelining

- So there was a laundry example..... here's a nice picture of it.
Pipelining Performance

- Consider the following table of pipelining performance

  - assume time for stages is
    - 100ps for register read/write
    - 200ps for other stages
  - compare pipelined data path with single-cycle

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>
Pipelining Performance: example

Use the table to solve the following:

Assume that the period of the clock cycles in a CPU is 200ps. How many clock cycles are required for the instruction (no pipelining): add $1, $2, $3?

- assume time for stages is
  - 100ps for register read/write
  - 200ps for other stages
- compare pipelined data path with single-cycle

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td></td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td>100 ps</td>
<td>600ps</td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td></td>
<td></td>
<td>500ps</td>
</tr>
</tbody>
</table>
Pipelining Performance: solution

- Solution to the previous example

Since everything has to be coordinated on the beat of the clock, even the stages that only take 100 ps cannot move forward to the next stage until after 200ps. Since add is an example of a R-format instruction (i.e. register access only, does not require memory access), this instruction would take four stages to complete, so it would take 4 clock cycles.
Pipelining Performance: example

- Complete the following example:

  Consider the following code. Assume that all registers have values previously assigned.

  ```
  add $1, $2, $3
  lw $4, 0($2)
  sw $5, 4($2)
  add $6, $3, $3
  lw $7, 4($8)
  ```

  What would the CPU time be to complete these instructions with no pipelining? What would the CPI be for the program in this case?
Pipelining Performance: solution

- Solution for example with no pipelining

\[
\begin{align*}
\text{line 1} & \quad 4 \text{ cycles} \\
\text{line 2} & \quad 5 \text{ cycles} \\
\text{line 3} & \quad 4 \text{ cycles} \\
\text{line 4} & \quad 4 \text{ cycles} \\
\text{line 5} & \quad 5 \text{ cycles} \\
\end{align*}
\]

Total cycles: 22
Total CPU time = 22 * 200 ps = 4400 ps = 4.4 ns
Total instructions: 5
\[\text{CPI} = \frac{22}{5} = 4.4\]
Pipelining Performance: example

- Complete the same example:

  Consider the following code. Assume that all registers have values previously assigned.

  ```
  add $1, $2, $3  
lw $4, 0($2)     
sw $5, 4($2)     
add $6, $3, $3  
lw $7, 4($8)     
  ```

  What would the CPU time be to complete these instructions with pipelining? What would the CPI be for the program in this case?
Pipelining Performance: solution

- Solution for example with pipelining

Since there are no hazards with this code, line 1 starts with the first cycle, line 2 starts with second cycle, ..., line 5 starts with the fifth cycle, and takes 5 cycles to complete.

Total cycles: 9 cycles
Total CPU time = 9 * 200 ps = 1800 ps = 1.8 ns
Total Instructions: 5
CPI = 9/5 = 1.8
Pipeline Hazards

- Three types of pipeline hazards:
  - Data hazard: need some result that isn’t ready yet
  - Control hazard: don’t know where a branch instruction is going yet
  - Structural hazard: two instructions need the same thing at different stages

- Structural hazard
  - For example one IF stage needs memory while another MEM stage also needs it
  - We assume memory and register accesses are fast enough that structural hazards are not a problem
Pipelining Hazards - Data Hazard

- What happens when one instruction needs something that isn’t ready?
  - Example: \( \text{add } \$3, \$1, \$2 \)
    \( \text{add } \$5, \$3, \$4 \)

- This is **data hazard**, because \( \$3 \) is set in the WB stage of the first instruction, but is needed in the EX stage of the second
  - This is solved by **forwarding** - an extra connection from EX stage to next EX stage

- Sometimes forwarding doesn’t work
  - Example: \( \text{lw } \$3, 0(\$1) \)
    \( \text{add } \$4, \$5, \$3 \)
  - Stall for the \( \text{lw} \) to finish MEM stage before forwarding \( \$3 \) to the EX stage of \( \text{add} \)
Pipelining Hazards - Control Hazard

● **Control hazard**: How do we know what to do after a branch?
  ○ Example: `beq $0, $3, somelabel`  
    `add $5, $4, $2`

● Add extra hardware so we know results of branch instructions after ID stage
  ○ Now we only have to stall once!

● Can we do better?
  ○ Yes, **branch prediction** is where we guess the result of a branch instruction, and then if we are wrong we have to undo our work and go back.
  ○ There are different types:
    ■ **static branch prediction**: guess all backward branches taken, all forward not taken
    ■ **dynamic branch prediction**: use history of taken/not-taken to decide guess
Assignment and midterm reminders

- Always test your code on the university servers before you hand it in
- Hand in the .asm file (not the .mips file)
- Midterm is this week Thursday (Feb 8th)
- (Probably) a good way to study:
  - practice!! practice old assignment questions
  - make friends and study together in groups
    - look at the posts on Piazza about forming study groups
  - one lecture this week is review
  - office hours if you get very stuck (try to come with specific questions)