Objective: In this recitation, you will learn about real chips and improve your understanding of how ware runs on pipelines.

There are three tasks in this recitation. The UTAs will help to manage the time.

1. Pipelining, Dependences, and Hazards
Assume the 5-stage pipeline from class (and the textbook), and assume full bypassing wherever possible. Construct a 4-line MIPS code snippet with two data dependences through registers. One of the dependences should be a hazard (i.e., require a stall) and the other should not (i.e., can be resolved via bypassing).

2. Stalls and Bypasses
Assume the 5-stage pipeline we’ve used in class (F, D, X, M, W). Assume the pipeline forwards/bypasses operands whenever possible and stalls only when needed to satisfy a RAW dependence that can’t be forwarded/bypassed. Assume all loads and stores hit in the 1-cycle data cache. Assume we’re running the code from exercise 1 above.

Complete a pipeline diagram for the first 7 cycles, and assume that the add is in the Fetch (F) stage in cycle 1, as shown below.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $1, $2, $3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>xor $2, $1, $2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw $1, 4($2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub $2, $5, $1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $2, 4($5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: If short on time, it’s okay to stop here.
3. Pipelining in the Real World (*time permitting*)

Time permitting, break up into groups of 3 or 4 people each, and in each group, pick one of these real-world processors. UTAs will help to make sure that every group has a different processor.

1. Intel’s Core i7, code name: Broadwell
2. Intel’s Atom, code name: Cedarville
3. AMD’s Bobcat, code name: Llano
4. ARM’s Cortex-M7
5. IBM’s Power7
6. Qualcomm’s Scorpion CPU
7. Alpha 21364 (also called EV7)
8. Atmel ATMEGA328 microcontroller¹

Now learn as much as you can about your chosen processor:

(a) What is the ISA?
(b) How many registers does it have?
(c) How many stages are in its pipeline?
(d) What are those stages? Don’t be worried if many stage names don’t make sense yet, e.g., “register renaming”, “wakeup”, etc.
(e) What is the clock frequency? (There is likely to be a range of frequencies.)
(f) What are the sizes and associativities of the caches (L1I, L1D, L2, L3)? What are the cache block sizes?
(g) What are the sizes and associativities of the TLBs?

¹ This is a microcontroller rather than a CPU, meaning it has lots of general purpose IO pins for interfacing with custom circuits. You’ll find that it’s different from the CPUs in this exercise in a few other ways, too.
4. Types of Dependences and the Hazards They Create *(time permitting)*

There are multiple kinds of data dependences in programs. For the pipeline we learned in lecture, the one that can lead to hazards is RAW (read-after-write). You can also have WAR (write-after-read) and WAW (write-after-write) dependencies, which would affect designs that do out-of-order execution.

It’s useful to be able to identify RAW, WAR, and WAW dependences through registers and through memory. There’s no need to worry about RAR (read-after-read), since these “dependences” are never problematic.

Some dependences lead to hazards and some don’t. Whether a dependence leads to a hazard is a function of the pipeline design.

(a) Construct a 2-line MIPS code snippet that has a RAW dependence through a register.

(b) Construct a 3-line MIPS code snippet in which the 2\textsuperscript{nd} and 3\textsuperscript{rd} instructions have RAW dependences on the register written by the 1\textsuperscript{st} instruction.

(c) Construct a 2-line MIPS code snippet that has a RAW dependence through memory.

(d) Construct a 2-line MIPS code snippet that \textit{may or may not} have a RAW dependence through memory. (Think about how you might not be able to tell if there’s a RAW dependence!)

(e) Construct a 2-line MIPS code snippet that has a WAR dependence through a register.

(f) Construct a 2-line MIPS code snippet that has a WAW dependence through a register.

(g) Construct a 3 or 4 line MIPS code snippet that has at least one RAW, WAR, and WAW dependence.