- ASSUMED SINGLE CYCLE OPERATION SO FAR

- DRAWBACK: MUST SET CLOCK FREQUENCY FOR SLOWEST OPERATION!

---

E.G. IF MEM ACCESS (LD) IS 200 ps AND
REG ACCESS IS 30 ps, ALU EXECUTION IS
100 ps, IFETCH IS 200 ps

<table>
<thead>
<tr>
<th>BRANCH</th>
<th>IFETCH</th>
<th>REG</th>
<th>ALU</th>
<th>DATA</th>
<th>RETS</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>300 ps</td>
<td>100 ps</td>
<td>0</td>
<td>0</td>
<td>530 ps</td>
</tr>
<tr>
<td>LOAD</td>
<td>200 ps</td>
<td>30 ps</td>
<td>100 ps</td>
<td>200 ps</td>
<td>30 ps</td>
<td>360 ps</td>
</tr>
</tbody>
</table>

WORST CASE:

MUST CLOCK AT 8.68 MHz

- BUT OUR LONGEST PHASE IS ONLY 200 ps!

- IN THEORY, THIS SHOULD ALLOW US TO
  CLOCK AT 5.5 GHz

- ALSO, SINCE EVERYTHING OCCURS DURING ONE
  CLOCK CYCLE, WE MUST REPLICATE FUNCTIONAL
  UNITS! (E.G. ADDERS FOR ALU & PC CALCULATION)
- Solution? PIPELINING (classic 5-stage example)

1. ADD R0, R1, 0  | IF | ID | EX | MEM | WB

2. LD R2, X  | IF | ID | EX | MEM | WB

3. LEA R3, 6  | IF | ID | EX | MEM | WB

- This increases throughput of the processor. Latency for instruction may actually go up!

- # INSTRUCTIONS/sec ↑
- Can we reuse functional units?
- What happens when we have dependencies?

\[
\begin{align*}
\{ & \text{ADD R0, R0, 1} \\
& \text{ADD R3, R0, 2} \\
\text{ADD R0, R0, 1} & | \text{IF} | \text{ID} | \text{EX} | \text{MEM} | \text{WB} \\
\text{IF} & | \text{ID} | \text{EX} |
\end{align*}
\]

Pipeline stalls! (aka "bubbles")
What we just saw is a **data hazard** (later instruct depends on earlier one so we must wait)

- 1) Schedule instructions differently (throw a non-dependent instruction in-between)
  * Compiler does it: **static scheduling**
  * CPU does it: **dynamic scheduling**

If instructions are executed out of order, we call it an **out-of-order** processor.
A lot of special logic needed for this! Expensive

- 2) **Forward results**! Also called **by-passing**

Basic idea: Why wait to write back results when they're in the pipe?

We can also have **structural hazards**.

More than 1 instr. trying to use same resource.

E.g., double-pump register file, or allow concurrent reads/writes ("multi-ported" reg file)
- Control Hazards: Instruction Exec. Depends on Previous Results BRANCHED!

```
ADD $r0, $r0, 1
BRnz foo
ADD $r2, $r0, 2
if LD $r3, bar
JOIN:
BRANCH PREDICTION!
GUESS! EASIEST? ALWAYS GUESS NOT TAKEN
THIS CAN ACTUALLY WORK...
LD $r0, COUNT
Loop ADD $r0, $r0, -1
BRnz End
BR Loop
End HALT
Here, we'll only mispredict \( \frac{1}{\text{COUNT}} \) of the time
```

More sophisticated: Use History
- Index by PC of Branch (Low bits)
  - Last Taken? Take
  - Not? Don't
  - Next Step: Two Bits ...
ST: "Strongly Taken"
WT: "Weakly Taken"
WN: "Weakly Not Taken"
SN: "Strongly Not Taken"

2-BIT SATURATING COUNTER

- Can differentiate between aliased branches (low order bits of PC are same)
  \[ \implies \text{just like caches, add a tag} \]

- Add more history