Pipelining hazards

- Situations that prevent starting the next instruction in the next cycle
  - **Structural hazards**
    - A required resource is busy
  - **Data hazard**
    - Need to wait for previous instruction to complete its data read/write
  - **Control hazard**
    - Deciding on control action depends on previous instruction
2. Data Hazards: compute-use

$add \ s0, \ s2, \ s3$

$and \ t0, \ s0, \ s1$

$or \ t1, \ s4, \ s0$

$sub \ t2, \ s0, \ s5$
Data Hazard: load-use

Program execution order (in instructions)

- `lw $2, 20($1)`
- `and $4, $2, $5`
- `or $8, $2, $6`
- `add $9, $4, $2`
- `slt $1, $6, $7`
Handling data hazards

A. Compile-time techniques
B. Forward data at run time
C. Stall the processor at run time
A. Data hazard elimination using compile-time techniques (code rescheduling)

- Reorder code to avoid use of load result in the next instruction
- C code for $A = B + E; \ C = D + F$;
- Compiler must be aware of pipeline structure

```
lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1, $t2
sw $t3, 12($t0)
lw $t4, 8($t0)
lw $t5, 16($t0)
add $t5, $t4, $t5
sw $t5, 20($t0)
```
A. Data hazard elimination using compile-time techniques (nop)

- Insert enough nops until result is ready (wastes cycles)

```
add $s0, $s2, $s3          IM    add    RF    $s2    $s3    DM    $s0    RF
nop                        IM    nop    RF    DM    RF
nop                        IM    nop    RF    DM    RF
and $t0, $s0, $s1          IM    and    RF    $s0    $s1    DM    $t0    RF
or  $t1, $s4, $s0          IM    or     RF    $s4    $s0    DM    $t1    RF
sub $t2, $s0, $s5          IM    sub    RF    $s0    $s5    DM    $t2    RF
```
B. Data hazard elimination using data forwarding/bypassing during runtime

- Don’t wait for result to be stored in a register ➔ forward the results whenever the results
- Requires extra connections in the datapath
Dependencies and forwarding

<table>
<thead>
<tr>
<th>Value of register $2$:</th>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10/-20</td>
<td>-20</td>
<td>-20</td>
<td>-20</td>
<td>-20</td>
</tr>
</tbody>
</table>

Program execution order (in instructions):
- sub $2, $1, $3
- and $12, $2, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)
Circuitry for forwarding

b. With forwarding
Forward unit design: When to Forward?

• **Three conditions must be met:**

  1. EX/MEM.RegWrite and/or MEM/WB.RegWrite are true

  2. Destination register(s) are equal to the source registers of the next 1 - 2 instructions. That is,
     - EX/MEM.RegisterRd == ID/EX.RegisterRs
     - EX/MEM.RegisterRd == ID/EX.RegisterRt
     - MEM/WB.RegisterRd == ID/EX.RegisterRs
     - MEM/WB.RegisterRd == ID/EX.RegisterRt

  3. Destination register in EX/MEM and/or MEM/WB is not $0$. That is,
     - EX/MEM.RegisterRd ≠ 0
     - MEM/WB.RegisterRd ≠ 0
Double data hazard

- Consider the sequence:
  - add $1, $1, $2
  - add $1, $1, $3
  - add $1, $1, $4

- Both hazards occur
  - Want to use the most recent

- Revise MEM hazard condition
  - Give priority to EX results. That is, only fwd from MEM if EX hazard condition isn’t true
Forwarding illustration

sub $2, $1, $3
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2
3rd cycle

sub $2, $1, $3
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2
4th cycle

\[
\begin{align*}
\text{add} & \quad 9, 4, 3 \\
\text{and} & \quad 4, 4, 2 \\
\text{sub} & \quad 4, 2, 5 \\
\text{sub} & \quad 2, \ldots \\
\end{align*}
\]
5th cycle

After $1$

add $9$, $4$, $2$

or $4$, $4$, $2$

and $4$, ...

sub $2$, ...

sub $2$, $1$, $3$

and $4$, $2$, $5$

or $4$, $4$, $2$

add $9$, $4$, $2$
6th cycle