Unsimplified Datapath with Forwarding

Yes:
- add
- sub
- and
- or
- slt
- sw

No:
- lw
- beq
- j

This design shows the correct logic for synchronizing control signals and instructions, and forwarding logic, but lacks hazard detection.
Load-Use Data Hazard

Consider the following sequence of instructions:

\[
\begin{align*}
\text{lw} & \quad $t2, \ 20($t1) \quad \# \text{writes a value} \\
\text{and} & \quad $t4, \ $t2, \ $t5 \quad \# \text{reads that value}
\end{align*}
\]

This hazard cannot be resolved by simple forwarding… why not?

The value \text{lw} writes into $t2 is not available until \text{lw} completes the MEM stage, but \text{and} needs that value when it enters the EX stage, which is when \text{lw} enters the MEM stage.

QTP: why can this situation not occur if the \text{writing} instruction is R-type?
Handling a Load-Use Hazard

A load-use hazard requires delaying the execution of the using instruction until the result from the loading instruction can be made available to the using instruction.

```
lw  $t2, 20($t1)   # loads $t2
and  $t4, $t2, $t5  # uses $t2
```

If we can stall the execution of the using instruction for one cycle:

- value to be loaded to $t2 will be available in the MEM/WB buffer when the using instruction moves from ID to EX

- that value can be forwarded to the using instruction as the using instruction enters the EX stage
Detection

**When** can we detect the existence of a load-use hazard?

When we are decoding the *using* instruction --- if we remember right information about the preceding instruction.

What do we need to remember?

- whether the preceding instruction reads a value from data memory
- whether the preceding instruction writes a value to the register file
- whether that value is written to a register that current instruction reads from

**Why do we not need to consider this question?**
Load-Use Hazard Detection

The *loading* instruction must be just that… so it writes to register rt.

There is a load-use hazard when

\[ \text{ID/EX.MemRead AND} \]
\[ ( ( \text{ID/EX.RegisterRt} == \text{IF/ID.RegisterRs}) \text{ OR } ( \text{ID/EX.RegisterRt} == \text{IF/ID.RegisterRt} ) ) \]

If detected… do what?

1 iff we're executing a load instruction

ID/EX shows register being written to; IF/ID shows registers being read from
How to Stall the Pipeline

"If it isn't written down, it didn't happen."

Force all control values in ID/EX register to 0
- when *using* reaches ID stage
- EX, MEM and WB do a **nop**

Prevent update of PC and IF/ID registers
- *using* instruction is decoded again
- instruction after the *using* instruction will be fetched again
- 1-cycle stall allows MEM to read data for \( \text{lw} \)
  - can subsequently forward data to *using* instruction in EX stage
Trace

lw  $2, 20($1)     # 1
and $4, $2, $5     # 2
or  $8, $2, $6     # 3
add $9, $4, $2     # 4

When and reaches the ID stage, the hazard involving $2 is detected.

All the control signals from the ID stage are set to 0 and the PC and IF/ID interstage buffer are prevented from updating.
Resetting the control signals and locking PC and IF/ID cause:

Because IF/ID is not updated, the and instruction is processed through ID again.

Because PC is not updated, the or instruction is fetched again in the IF stage.

And:
- EX operates as usual (with all relevant signals 0)
- EX sends only 0 control signals to MEM for the next cycle

lw reaches the MEM stage and reads the value to be written to $2$. That value goes into MEM/WB.
On the next cycle:

The control signals for \texttt{and} (set in \texttt{ID} in the previous cycle) reach \texttt{EX}.
The value for \texttt{$2$} in \texttt{MEM/WB} is forwarded to the ALU in \texttt{EX}.

And:
- \texttt{MEM} operates as usual (with all relevant signals 0)
- \texttt{MEM} sends only 0 control signals to \texttt{WB} for the next cycle

Instructions preceding \texttt{and} proceed normally…
Stall/Bubble in the Pipeline

On the following cycles:

```
IF → ID → EX → MEM → WB
```

- add
- or
- and

- STALL

… and so on…

The execution time has increased by one clock cycle.
Simplified Datapath with Hazard Detection
Stall Details

Stall == 1 iff load-use hazard

Stall Details

InhibitWrite prevents updating of storage

AND gates allow “erasing” of normal control signals

Pipeline Stalls 12
Unsimplified Datapath with Hazard Detection

Yes:
- add
- sub
- and
- or
- slt
- sw
- lw

No:
- beq
- j
A Question to Ponder

Consider the following scenario:

```
lw  $t1, ($t3)  # rd == t1
lw  $t1, ($t2)  # rt == t1 (really the destination reg)
```

The Hazard Detection design sees $t1 as an input register for the second `lw`... So it would stall... needlessly.

How could we fix this?