Pipelined Control Overview

This design shows the correct logic for synchronizing control signals and instructions, but lacks forwarding logic and hazard detection.
Data Hazards in ALU Instructions

Consider this sequence:

```
sub $2, $1, $3  # value for $2 known end of EX stage;
                # stored in $2 in WB stage

and $12, $2, $5 # enters ID stage when sub enters EX;
                # and needs $s2 when enters EX stage;
                # sub is in MEM stage by then;
                # $2 has not been written yet
```

Data hazard?
Data Hazards in ALU Instructions

So this sequence leads to a data hazard involving $2$:

\[
\text{sub } 2, 1, 3 \\
\text{and } 12, 2, 5
\]

Can we resolve the hazard simply by forwarding?

Yes!

But we must deliver the computed value at the right time; the next tick. And, that value will be sitting in the EX/MEM interstage buffer.
Detecting the Hazard

On the one hand, this is obvious. The first instruction writes a value into a register that is subsequently used as input by the second instruction:

```
sub  $2, $1, $3
and $12, $2, $5
```

We must know the register numbers for both instructions in order to detect the hazard.

More precisely, we must know \( rd \) for the first instruction and both \( rs \) and \( rt \) for the second instruction.

So, we must save those register numbers, via the interstage buffers.

Some notation will help us speak precisely about what's going on:

\[
\text{B/RegisterRX} = \text{register number for RX sitting in interstage pipeline buffer B}
\]
A Glance Ahead

Passing the register numbers:
- rs (left operand) register number
- rt (right operand) register number
- rd (destination) register number

Logic “box” that manages forwarding of operands
A Glance Ahead

Passing the register numbers:

1: rd for instruction currently in EX stage

2: rd for instruction currently in MEM stage

3: rd for instruction currently in WB stage

They may all be different!
Detecting the Hazard

Now, for this sequence of instructions:

\[
\text{sub} \quad \$2, \$1, \$3 \\
\text{and} \quad \$12, \$2, \$5
\]

So, we detect the hazard because we see that:

\[
\text{EX/MEM}.\text{RegisterRd} == \text{ID/EX}.\text{RegisterRs}
\]

Hence, we must forward the ALU output value from the \text{EX/MEM} interstage buffer to the \text{rs} input to the ALU.

Apparently, we'll need to:

- pass (at least some) register numbers forward via the interstage buffers
- add a logic unit to compare those register numbers to detect hazards
- add data connections to support transferring data values being forwarded
- add some more selection logic (multiplexors)
Data Hazards in ALU Instructions

Now, consider this sequence:

```
sub $2, $1, $3   # value for $2 known in EX stage
and $12, $2, $5   # enters ID stage when sub enters EX
or $13, $6, $2   # enters ID stage when sub enters MEM;
                 # $2 has not been written yet
```

Tick 0:  sub
Tick 1:  and sub
Tick 2:  or and sub
Tick 3:  or and sub
Tick 4:  or and sub

Data hazard?
Again, we have a data hazard:

\[
\begin{align*}
\text{sub} & \quad $2, \; $1, \; $3 \quad \# \text{ value for } $2 \text{ known in EX stage} \\
\text{and} & \quad $12, \; $2, \; $5 \quad \# \text{ enters ID stage when sub enters EX} \\
\text{or} & \quad $13, \; $6, \; $2 \quad \# \text{ enters ID stage when sub enters MEM;}
\end{align*}
\]

Tick 0: \quad \text{sub}

Tick 1: \quad \text{and} \quad \text{sub}

Tick 2: \quad \text{or} \quad \text{and} \quad \text{sub}

Tick 3: \quad \text{or} \quad \text{and} \quad \text{sub}

Tick 4: \quad \text{or} \quad \text{and} \quad \text{sub}

\text{Yes!}

\text{Now, we must deliver the computed value after a delay of one tick, from MEM/WB.}
Detecting the Hazard

Again, we have a data hazard:

\[
\text{sub } \$2, \$1, \$3 \\
\text{and } \$12, \$2, \$5 \\
\text{or } \$13, \$6, \$2
\]

So, we detect the hazard because we see that:

\[
\text{MEM/WB.RegisterRd} == \text{ID/EX.RegisterRt}
\]

Hence, we must forward the ALU output value from the MEM/WB interstage buffer to the \text{rt*} input to the ALU.

So… detecting data hazards is a multi-stage affair.

* QTP: why does this one go to the rt input?
Data Hazards in ALU Instructions

Now, consider this sequence:

```
sub $2, $1, $3   # value for $2 known in EX stage;
and $12, $2, $5   # enters ID stage when sub enters EX;
or $13, $6, $2   # enters ID stage when sub enters MEM;
add $14, $2, $2   # enters ID stage when sub enters WB;
# $2 has not been written yet, but...
```

Tick 0: sub
Tick 1: and sub
Tick 2: or and sub
Tick 3: add or and sub
Tick 4: add or and sub

Data hazard?
Data Hazards in ALU Instructions

Now, there's almost a hazard… but not quite…

```
sub   $2, $1, $3   # value for $2 known in EX stage;
and  $12, $2, $5   # enters ID stage when sub enters EX;
or   $13, $6, $2   # enters ID stage when sub enters MEM;
add  $14, $2, $2   # enters ID stage when sub enters WB;
```

Tick 0: sub
Tick 1: and sub
Tick 2: or and sub
Tick 3: add or and sub
Tick 4: add or and sub
Tick 5: add or and sub

Now, we deliver the computed value to the register file in the first half of tick 4, and it's not read until the second half of that tick!
Data Hazards in ALU Instructions

Now, consider this sequence:

\[
\begin{align*}
\text{sub} & \quad $2, \quad $1, \quad $3 \quad \# \text{value for } $2 \text{ known in EX stage;} \\
\text{and} & \quad $12, \quad $2, \quad $5 \quad \# \text{enters ID stage when sub enters EX;} \\
\text{or} & \quad $13, \quad $6, \quad $2 \quad \# \text{enters ID stage when sub enters MEM;} \\
\text{add} & \quad $14, \quad $2, \quad $2 \quad \# \text{enters ID stage when sub enters WB} \\
\text{sw} & \quad $15, \quad 100($2) \quad \# \text{enters ID stage after sub is done}
\end{align*}
\]

Tick 0: \quad sub

Tick 1: \quad and \quad \quad \quad \quad \quad sub

Tick 2: \quad or \quad \quad \quad \quad \quad \quad \quad and \quad \quad \quad \quad sub

Tick 3: \quad add \quad \quad \quad \quad or \quad \quad \quad \quad \quad \quad \quad and \quad \quad \quad \quad \quad \quad \quad sub

Tick 4: \quad sw \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad add \quad \quad \quad \quad or \quad \quad \quad \quad \quad \quad \quad and \quad \quad \quad \quad \quad \quad \quad \quad sub

Tick 5: \quad sw \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad sw \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad add \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad or \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad and

**Data hazard?**
Detecting the Need to Forward

Here's what we (seem to) know so far:

ALU-related data hazards occur when

\[
\begin{align*}
\text{EX/MEM.} \text{RegisterRd} &= \text{ID/EX.} \text{RegisterRs} \\
\text{EX/MEM.} \text{RegisterRd} &= \text{ID/EX.} \text{RegisterRt} \\
\text{MEM/WB.} \text{RegisterRd} &= \text{ID/EX.} \text{RegisterRs} \\
\text{MEM/WB.} \text{RegisterRd} &= \text{ID/EX.} \text{RegisterRt}
\end{align*}
\]

However, we have overlooked (at least) one thing…
Detecting the Need to Forward

We don't need to forward unless the forwarding (earlier) instruction does actually write a value to a register:

\[
\text{EX/MEM.RegWrite} \equiv 1 \\
\text{MEM/WB.RegWrite} \equiv 1
\]

And we only forward if \( \text{Rd} \) for that instruction is not \( \$\text{zero} \):

\[
\text{EX/MEM.RegisterRd} \neq 0 \\
\text{MEM/WB.RegisterRd} \neq 0
\]
Datapath Change: ALU Operand Selection

Forwarding unit selects among three candidates for the register operands.

Value from register fetch in ID stage

Value from WB stage

Value from ALU execution
Datapath Change: ALU Operand Selection

Pipeline Forwarding 17

Computer Organization II

©2005-2013 McQuain
Forwarding Paths

Select source for left operand rs

Select source for right operand rt

Possible rd numbers

rs and rt for the instruction in the EX stage,
ID/EX.RegisterRs, ID/EX.RegisterRt

Select correct rd number

rd # from MEM stage, EX/MEM.RegisterRd

rd # from WB stage, MEM/WB.RegisterRd
Conditions for EX Hazard

If \( (EX/MEM.\text{RegWrite} \quad \text{and} \quad EX/MEM.\text{RegisterRd} \neq 0 \quad \text{and} \quad EX/MEM.\text{RegisterRd} == ID/EX.\text{RegisterRs}) \)
then
ForwardA = 10

If \( (EX/MEM.\text{RegWrite} \quad \text{and} \quad EX/MEM.\text{RegisterRd} \neq 0 \quad \text{and} \quad EX/MEM.\text{RegisterRd} == ID/EX.\text{RegisterRt}) \)
then
ForwardB = 10

QTP: could BOTH occur with respect to the same instruction?
Conditions for MEM Hazard

If (MEM/WB.RegWrite and MEM/WB.RegisterRd != 0 and MEM/WB.RegisterRd == ID/EX.RegisterRs)
then ForwardA = 01

If (MEM/WB.RegWrite and MEM/WB.RegisterRd != 0 and MEM/WB.RegisterRd == ID/EX.RegisterRt)
then ForwardB = 01

QTP: could BOTH an EX hazard and a MEM hazard occur with respect to the same instruction?
Double Data Hazard

Consider the sequence:

- add $1, $1, $2
- sub $1, $1, $3
- or $1, $1, $4

Both hazards occur… which value do we want to forward?

Tick 2:
- or
- sub

Tick 3:
- ... 
- or
- sub
- add
Double Data Hazard

Consider the sequence:

```
add $1, $1, $2
add $1, $1, $3
add $1, $1, $4
```

Revise MEM hazard condition:
- Only forward if EX hazard condition is not true
Revised Conditions for MEM Hazard

If (MEM/WB.RegWrite and
    MEM/WB.RegisterRd != 0 and
    not (EX/MEM.RegWrite and
        EX/MEM.RegisterRd != 0 and
        EX/MEM.RegisterRd == ID/EX.RegisterRs ) and
    MEM/WB.RegisterRd == ID/EX.RegisterRs )
then
    ForwardA = 01

If (MEM/WB.RegWrite and
    MEM/WB.RegisterRd != 0 and
    not (EX/MEM.RegWrite and
        EX/MEM.RegisterRd != 0 and
        EX/MEM.RegisterRd == ID/EX.RegisterRt ) and
    MEM/WB.RegisterRd == ID/EX.RegisterRt )
then
    ForwardB = 01
MEM Hazard Breakdown

If ( ( MEM/WB.RegWrite and MEM/WB.RegisterRd != 0 )

and

not ( EX/MEM.RegWrite and EX/MEM.RegisterRd != 0 and EX/MEM.RegisterRd == ID/EX.RegisterRs )

and

MEM/WB.RegisterRd == ID/EX.RegisterRs )

then

ForwardA = 01

Instruction leaving MEM stage DOES write a value

Instruction leaving EX stage DOES NOT write a value

OR

it doesn’t write to Rs register of instruction leaving ID stage

Instruction leaving MEM stage DOES write a value to the Rs register of instruction leaving ID stage
Simplified Datapath with Forwarding
This design shows the correct logic for synchronizing control signals and instructions, and forwarding logic, but lacks hazard detection.
A Question to Ponder

Consider the following scenario:

\begin{align*}
\text{add} & \quad \texttt{$t1, t2, t3$} \\
\text{lw} & \quad \texttt{$t1, (t2)$}
\end{align*}

The **Forwarding unit** design sees $t1$ as an input register for the second lw...

So it would forward... needlessly.

Is that a problem?

How could we fix this?