Data Hazards in ALU Instructions

Consider this sequence:

- sub $2, $1, $3
- and $12, $2, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)

We can resolve hazards with forwarding

- How do we detect when to forward?
### Dependencies & Forwarding

**Time (in clock cycles)**

<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>10</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)`
Forwarding Paths
Detecting the Need to Forward

- Pass register numbers along pipeline
  - e.g., ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register

- ALU operand register numbers in EX stage are given by
  - ID/EX.RegisterRs, ID/EX.RegisterRt

- Data hazards when
  1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  2b. MEM/WB.RegisterRd = ID/EX.RegisterRt
Detecting the Need to Forward

- But only if forwarding instruction will write to a register!
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- And only if Rd for that instruction is not $0
  - The value of $0 is always 0 when fetched from Reg
  - Previous instruction could set it to non-zero
    - slt $0, $1, $2
  - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0
Forwarding Conditions

- **EX hazard**
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    \[\text{ForwardA} = 10\]
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    \[\text{ForwardB} = 10\]

- **MEM hazard**
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    \[\text{ForwardA} = 01\]
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    \[\text{ForwardB} = 01\]
Datapath with Forwarding
Load-Use Data Hazard

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

Need to stall for one cycle
Load-Use Hazard Detection

- Check when instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard when
  - ID/EX.MemRead and
    - ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
      (ID/EX.RegisterRt = IF/ID.RegisterRt))
- If detected, stall and insert bubble
Stall/Bubble in the Pipeline

Program execution order (in instructions)
- \texttt{lw} \ $2, 20(\$1)
- and \textbf{becomes} \ nop
- and \$4, \$2, \$5
- or \$8, \$2, \$6
- add \$9, \$4, \$2

Stall inserted here
Pipelined Datapath and Control
How to Stall the Pipeline

- Force control values in ID/EX register to 0
  - EX, MEM and WB do **nop** (no-operation)
- Prevent update of PC and IF/ID register
  - Same instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for \( \text{lw} \)
    - Can subsequently forward to EX stage
Datapath with Hazard Detection
Stalls and Performance

- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure
Pipelined Datapath and Control
Branch Hazards

- If branch outcome determined in MEM

Program execution order (in instructions)

- 40 beq $1, $3, 28
- 44 and $12, $2, $5
- 48 or $13, $6, $2
- 52 add $14, $2, $2
- 72 lw $4, 50($7)

Flush these instructions (Set control values to 0)
Reducing Branch Delay

- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator

- Example: branch taken

  36:    sub $10, $4, $8
  40:    beq $1, $3, 7
  44:    and $12, $2, $5
  48:    or  $13, $2, $6
  52:    add $14, $4, $2
  56:    slt $15, $6, $7
       ...
  72:    lw  $4, 50($7)
Example: Branch Taken

and $12, $2, $5
beq $1, $3, 7
sub $10, $4, $8
before<1>
before<2>

Clock 3
Example: Branch Taken

Iw $4, 50($7)  
Bubble (nop)  
beq $1, $3, 7  
sub $10, ... before<1>

Clock 4
Data Hazards for Branches

- If a comparison register is a destination of 2\textsuperscript{nd} or 3\textsuperscript{rd} preceding ALU instruction

```
add $1, $2, $3
add $4, $5, $6
...
beq $1, $4, target
```

- Can resolve using forwarding
Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2\textsuperscript{nd} preceding load instruction
  - Need 1 stall cycle

lw  \$1, addr
add  \$4, \$5, \$6
beq stalled
beq \$1, \$4, target
Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles

```
lw $1, addr
beq stalled
beq stalled
beq $1, $0, target
```
Final datapath and control