CS472 - Computer Architecture Lecture 11

D. Kevin McGrath

May 7, 2012
Pipeline hazards and C++ AMP

- Structural hazards
- Data hazards
- Control hazards
- C++ AMP examples
Structural hazards

1. Structural hazards

2. Data hazards

3. Control hazards

4. C++ AMP
Structural hazards

- A planning instruction cannot execute in the proper clock cycle
- Occurs due to the hardware not supporting the combination of instructions requested
  - Multiple uses of the ALU or FPU
  - Having a combined instruction and data memory
- Careful consideration of ISA and hardware requirements can eliminate most cases of structural hazards
- Simplest hazard type to deal with
Outline

1. Structural hazards
2. Data hazards
3. Control hazards
4. C++ AMP
**Data hazards**

- Planned instruction cannot execute in the proper clock cycle due to data dependencies
- The data it needs to use is not yet available
- Add instruction followed immediately by a subtract:
  
  ```
  add $s0, $t0, $t1  
  sub $t2, $s0, $t3
  ```

- Destination of add is operand of sub
- Would require the pipeline to stall, inserts “bubbles”
• Solution to this problem known as forwarding or bypassing

• Can only move *forward* in the datapath

- Requires additional connection in the datapath
Load-use data hazard

- Can’t always avoid data hazards via forwarding
  - If value not computed when needed
  - Can’t forward back in time

Program execution order (in instructions):

- `lw $s0, 20($t1)`
- `sub $t2, $s0, $t3`
Hazard detection

- Consider the sequence:
  
  \[
  \begin{align*}
  \text{sub} & \quad $2, \ $1, \ $3 \\
  \text{and} & \quad $12, \ $2, \ $5 \\
  \text{or} & \quad $13, \ $6, \ $2 \\
  \text{add} & \quad $14, \ $2, \ $2 \\
  \text{sw} & \quad $15, \ 100($2)
  \end{align*}
  \]

- Hazards can be resolved with forwarding
  - How to detect when to forward?
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($2$)
Dependencies and forwarding

- Pass register numbers along pipeline
- ID/EX.RegisterRs is register number Rs in the ID/EX pipeline register
- ALU operand register numbers in EX stage 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
Dependencies and forwarding

- Only if forwarding instruction will write to a register – only if one of the signals is active:
  - EX/MEM.RegWrite
  - MEM/WB.RegWrite
- Only if Rd for that instruction is not $\text{zero}$
  - EX/MEM.RegisterRd $\neq 0$
  - MEM/WB.RegisterRd $\neq 0$
Forwarding conditions

- **EX hazard**
  - if (EX/MEM.RegWrite and EX/MEM.RegisterRd != 0 and EX/MEM.RegisterRd == ID/EX.RegisterRs): ForwardA = 10
  - if (EX/MEM.RegWrite and EX/MEM.RegisterRd != 0 and EX/MEM.RegisterRd == ID/EX.RegisterRt): ForwardB = 10

- **MEM hazard**
  - if (MEM/WB.RegWrite and MEM/WB.RegisterRd != 0 and MEM/WB.RegisterRd == ID/EX.RegisterRs): ForwardA = 01
  - if (MEM/WB.RegWrite and MEM/WB.RegisterRd != 0 and MEM/WB.RegisterRd == ID/EX.RegisterRt): ForwardB = 01
Double data hazard

- Consider the sequence
  - \texttt{add} $1, 1, 2$
  - \texttt{add} $1, 1, 3$
  - \texttt{add} $1, 1, 4$
- Both EX and MEM hazards occur
  - Want to use most recent value
- Have to revise MEM hazard condition
  - Only forward if EX hazard condition is false
Revised MEM forwarding condition

- 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):
  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):
  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
Load-use hazard detection

- Check when using 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
How to stall a pipeline

- Force control values in ID/EX register to 0
  - EX, MEM, and WB do noop
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for lw instruction
Bubble in the pipeline

Program execution order (in instructions)

- `lw $2, 20($1)`
  - becomes `nop`
- `and $4, $2, $5`
- `or $8, $2, $6`
- `add $9, $4, $2`
Datapath with hazard detection

Structural hazards

Data hazards

Control hazards

C++ AMP
Stalls and performance

- Stalls reduce performance
- Required to get correct results
- Compiler can arrange code to avoid hazards and stalls
- Requires knowledge of pipeline structure
  - Intel has access to this for icc on the IA32/64 architecture
  - gcc does not, generally, have access to this
Outline

1. Structural hazards
2. Data hazards
3. Control hazards
4. C++ AMP
Control hazards

- Proper instruction cannot be executed in the correct clock cycle as the incorrect instruction was fetched.
- Branch outcome not determined until MEM stage
- Two choices for dealing with this:
  - Stall on branches
  - Predict branches – assume the branch is taken
- Additionally, move logic for branch to ID stage
Branch prediction

- Deeper pipelines have larger branch penalties
- More instructions need to be flushed
- Make use of dynamic prediction
  - Branch history table
  - Indexed by branch instruction address
  - Stores outcome
  - Makes decision based on this history
Outline

1. Structural hazards
2. Data hazards
3. Control hazards
4. C++ AMP