Overview

- Structural hazards and design solutions
- Data hazards and design solutions.

Structural Hazards and Solutions

- Structural hazards occur when one resource is to be used by more than one instruction in one cycle

- Solutions:
  - Never access a resource more than once per cycle
  - Allocate each resource to a single pipeline stage
    - Duplicate resources if necessary e.g. IMEM/DMEM
  - Every instruction must follow the same leading sequence of cycles/stages
    - Strictly speaking, unnecessary trailing cycles/stages can be dropped e.g. BRA/JMP don’t need to complete WB (or even MEM)
    - Skipping a stage can introduce structural hazards e.g. R-type instructions cannot skip MEM stage since the preceding instruction is in WB stage already \(\rightarrow\) structural hazard on RegFile write port resources

Recall: Pipelined datapath and control
**Exercise: Structural hazards limit performance**

- Assume there is only one memory for instruction and data in the pipelined processor. Assume there are 20% load/store instructions. What is the average CPI due to this type of hazards?

---

**Data Hazards**

- Data hazards are caused by the data dependency
- There are four types of data dependency between instructions
  - **RAR** • Read after Read
  - **WAW** • Write after Write
  - **WAR** • Write after Read
  - **RAW** • Read after Write

---

**Exercise**

Identify all of the data dependencies in the following code.

```
add $2, $5, $4
add $4, $2, $5
sw $5, 100($2)
add $3, $2, $4
```

---

**Data Hazards – Design Solutions**

- Avoid some hazards by properly partitioning/scheduling tasks in the pipeline. For example,
  - eliminating **WAR** by always fetching operands early in pipe in ID stage
  - eliminating **WAW** by doing all WBs in order in the last stage
- Detect and resolve remaining ones
  - stall
    - To be discussed later
  - forward (if possible)
Data Hazards in Our Pipeline
• Happen when there are RAW data dependencies between instructions executing in the pipeline

<table>
<thead>
<tr>
<th>Time (in clock cycles)</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>Value of register $2$:</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>
<tr>
<td>Value of EX/MEM:</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>–20</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Value of MEM/WB:</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>–20</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</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($2$)

Edges sloping back in time Indicate dependencies

Forwarding Implementation
• Detect data hazards
  – Check for data dependencies between each instruction and its preceding instructions in the pipeline
• Forward the proper data values

Solution: Forwarding
• A very efficient approach to reducing data hazards

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

Value of register $2$: 10 10 10 10 10–20 –20 –20 –20
Value of EX/MEM: X X X X X X X X
Value of MEM/WB: X X X X X X X X

Detecting a data hazard (1)
• Based on pipeline register field contents
• Source register dependent on destination register of the first preceding instruction
  – 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  – 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
• Source register dependent on destination register of the second preceding instruction
  – 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  – 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt
Detecting a data hazard (2)

- Based on pipeline register field contents
- Source register dependent on destination register of the first preceding instruction
  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
- Source register dependent on destination register of the second preceding instruction
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Detecting a data hazard – example

sub $2, $1, $3         IF ID EX ME WB
and $12, $2, $5       IF ID EX ME WB
or $13, $6, $2        IF ID EX ME WB
add $14, $2, $2       IF ID EX ME WB
sw $15, 100($2)        IF ID EX ME WB

Dependent instructions: sub-and

Data hazard type: 1a

Detecting a data hazard – example

sub $2, $1, $3         IF ID EX ME WB
and $12, $2, $5       IF ID EX ME WB
or $13, $6, $2        IF ID EX ME WB
add $14, $2, $2       IF ID EX ME WB
sw $15, 100($2)        IF ID EX ME WB

Dependent instructions: sub-and, sub-or

Data hazard type: 1a, 2b
Detecting a data hazard – more considerations

• Previous instructions must
  1. Write back a result
  2. If they write to $0 (or $zero), the result is never stored, and hence need not to be forwarded
     • Recall: $0 is a special register in MIPS that always has value 0

Revised Detect Condition – EX hazard

• Data forwarded from MEM stage:
  Detection:
  
  IF EX/MEM.RegWrite
  AND EX/MEM.RegisterRd != 0
  AND EX/MEM.RegisterRd = ID/EX.RegisterRs
  forward ALU result to first ALU op

  IF EX/MEM.RegWrite
  AND EX/MEM.RegisterRd != 0
  AND EX/MEM.RegisterRd = ID/EX.RegisterRt
  forward ALU result to second ALU op

Revised Detect Condition – EX hazard

• Data forwarded from WB stage:
  Detection:
  
  IF MEM/WB.RegWrite
  AND MEM/WB.RegisterRd != 0
  AND MEM/WB.RegisterRd = ID/EX.RegisterRs
  forward MEM data to first ALU op

  IF MEM/WB.RegWrite
  AND MEM/WB.RegisterRd != 0
  AND MEM/WB.RegisterRd = ID/EX.RegisterRt
  forward MEM data to second ALU op

Multiple data dependencies

• What if a data is dependent on both the MEM and WB data??
  add $1, $1, $2
  add $1, $1, $3
  add $1, $1, $4

• Forward the more recent results – forward the MEM data

Modify WB hazard detection:

  IF MEM/WB.RegWrite
  AND MEM/WB.RegisterRd != 0
  AND MEM/WB.RegisterRd = ID/EX.RegisterRs
  forward WB data to first ALU op

  IF MEM/WB.RegWrite
  AND MEM/WB.RegisterRd != 0
  AND MEM/WB.RegisterRd = ID/EX.RegisterRt
  forward WB data to second ALU op
Modified datapath with forwarding unit

Exercise

Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding?

- add $2, $5, $4
- add $4, $2, $5
- sw $5, 100($2)
- add $3, $2, $4