DLX Floating Point

- Extend MIPS Pipeline to Floating Point Operations

- Functional units more complex than simple integer ALU
  - Require several clock cycles for a FP arithmetic operation
    - Add: 4 cycles, Multiply: 7 cycles, Divide: 25 cycles; Square root: 112 cycles

- Different functional units (FUs) for different operations

- Separate set of Floating Point Registers (FP registers): F0 .... F31
Floating Point Unit

- **Separate Functional unit (FU)** for each of the FP Arithmetic Instructions:
  - ADD.D, MUL.D, DIV.D

- **Load and Store** use integer ALU (EX) for address calculation:
  - L.D, S.D

- **Integer MUL** and **DIV** use the FP units

- FP instructions take differing amounts of time
  - e.g. + (4 cycles), * (7 cycles), / (25 cycles)

- **Monolithic ALU** (as in integer unit) inappropriate
  - Substantially larger than simple integer ALU operations
FP Pipeline Model

IF → ID → ADD (2 cycle) → MUL (4 cycle) → DIV (5 cycle) → EX → MEM → WB
Design Choices

- Designing for the Worst Case
  - Clock Pipeline at the speed of the slowest Functional Unit (FU)
    - Entire pipeline operates at 1/5 frequency
  - Not an attractive solution!!
    - MIPS rating falls by 80%
Design Choices

• Optimizing the Common Case

  • Assume integer EX instructions are the common case
    • Slow instructions are less frequent

  • Slow the pipeline only when needed (FP ADD, MUL, DIV) instructions
    • Insert appropriate number of stall cycles when a slow instructions is in EX stage

Example to show potential benefit:

ADD 4%, MUL 4%, DIV 1% of instructions
CPI = 1 + 4% x 1 + 4% x 3 + 1% x 4 cycles = 1 + .04 + .12 + .04 = 1.20
• MIPS rating drops by about 16%
FP Pipeline Model

IF → ID → MUL → ADD (2 cycle) → EX → MEM → WB

MUL (4 cycle)

DIV (5 cycle)
Stall due to Multi-Cycle ALU Operation

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>MEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A: ADD R1, R2, R3
B: ADD R4, R5, R6
C: MUL.D F2, F4, F6
D: MUL.D F8, F10, F12
E: MUL.D F14, F16, F18
Structural Hazards

Are there any structural hazards in the design?
Are there instructions that are delayed because of insufficient hardware resource?

1. ID/EX Pipeline register: Contention for datapath

   - Sequence of integer EX goes through pipeline at 1 cycle/instruction
   - A MUL instruction holds the ID/EX register for 4 cycles

     MUL   F0, F2, F4
     ADD   R6, R8, R10 (Stalls 3 cycles)
1. ID/EX Pipeline register: Contention for datapath

   • Sequence of integer EX goes through pipeline at 1 cycle/instruction
   • A MUL instruction holds the ID/EX register for 4 cycles

   \[
   \begin{align*}
   \text{MUL} & \quad F0, F2, F4 \\
   \text{AND} & \quad R6, R8, R10 \quad \text{(Stalls 3 cycles)}
   \end{align*}
   \]

   • Enhance the ID/EX Pipeline Register to hold 2 (or more) instructions simultaneously
FP Pipeline Model

- IF
- ID
- ADD (2 cycle)
- MUL (4 cycle)
- DIV (5 cycle)
- EX
- MEM
- WB
Structural Hazards

2. **EX stage: Contention for FUs**

   - **EX unit:** no contention

   - **Successive** (or close by) **FP instructions contend** for the Functional unit
     
     MUL F0, F2, F4
     MUL F6, F8, F10 (Stalls 4 cycles)
FP Pipeline Model

IF -> ID -> MUL -> EX

EX -> MEM

MEM -> WB

ADD (2 cycle) -> MUL (4 cycle) -> DIV (5 cycle)
2. **EX stage: Contention for FUs**

   - Replicate Functional Units

   - How much replication?

     - 2 Adders implies no structural hazards for any sequence of ADDs
     - 2 Multipliers:
       - Consecutive MULs no structural hazards;
       - 3 consecutive MULs : 2 stall cycles

   - If insufficient stall instruction in ID stage till resource available
Structural Hazards

2. EX stage: Contention for FUs

- Replicate Functional Units
- Pipelined functional units
  - Require pipeline registers between stages of the FU
  - Could be slower than non-pipelined design

- Each FU may be non-pipelined, fully pipelined, or partially pipelined.
- Depends on cost, time, frequency of operation

![4-stage Pipelined Multiplier](image)

Initiation Interval: Time between successive operations

Fully Pipelined FU has initiation interval of 1 cycle: No stalls needed
Pipeline Functional Units

2-stage fully pipelined Adder
4-stage fully pipelined Multiplier
5-cycle non-pipelined Divider
Hybrid Functional Units

1 2-cycle latency Fully Pipelined Adder
2 4-cycle latency 2-stage Partially Pipelined Multipliers
1 5-cycle (monolithic) Divider
Structural Hazards

3. MEM stage: Contention for access to data memory
   - Only LOAD and STORE instructions want to use data memory unit
   - Both follow same path through the pipelined and access MEM in cycle 4
   - No contention

4. WB stage: Contention for
   i. Write ports in register file
   ii. Data paths through MEM stage to WB
Structural Hazard: WB stage

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>IF</td>
<td>ID</td>
<td>+</td>
<td>+</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>B</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
</tbody>
</table>

A: ADD.D F0, F2, F4
B: L.D F18, 100(R4)

Contention for:

- Write ports in Register File in WB stage (cycle 6)
- Data paths through MEM stage (cycle 5)
Structural Hazard: WB stage

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>+</td>
<td>+</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>+</td>
<td>+</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A: DIV.D  F0, F2, F4  
B: MUL.D  F6, F8, F10  
C: ADD.D  F12, F14, F16  
D: ADD.D  F18, F20, F22  
E: L.D  F24, 100(R4)

Contention for:

Write ports in Register File in WB stage (cycle 9)
Data paths through MEM stage (cycle 8)
Solutions for WB Structural Hazards

1. **Multiple write ports** in register file
   - Extra hardware. Slowdown
   - Should we design for the peak vs average number of writes per cycle?

2. **Buffer** requests at WB stage and write one at a time
   - How deep should the buffer queue be?

3. Stall: Allow only 1 write to propagate to the WB stage
   
   **In MEM stage (EX/MEM pipeline register)**
   - Easy (+)
   - Prioritize based on heuristics (longest latency) (+)
   - Need to propagate stall backwards (-)
   - Two sources of resource stalls (-)

   **In ID stage**: Only release instruction that won’t cause hazard in WB stage
   - Centralized handling of stalls (+)
   - Occurs earlier than necessary (-)

We will allow S.D and FP instruction to go through MEM stage at the same time
Stall in MEM stage

DIV (5 cycle non pipelined)
Stall in ID stage

Check if instruction currently in ID will use WB at the same cycle as a previously issued instruction. If so Stall else Issue the instruction

Simple hardware implementation:
- Shift register of length L equal to length of longest path from ID to WB
  - Tracks the usage of WB for the next L cycles
  - Bit j of the Shift Register is True whenever an issued instruction will use WB j cycles from now
  - Every cycle shift the contents by 1 bit (so bit j becomes bit number j-1)

Assume instruction in ID wants to use register file in the WB stage:
1. Determine how many cycles later will instruction in ID use the WB stage (say d)
   (Depends on FU required by the instruction)

2. Check if bit d of register is set or not.
   If set   Stall current instruction for 1 cycle
   else    Set bit d of shift register to 1

3. Shift register one bit position