Topics

“I'm all in favor of it.”

– Coach John McKay

In response to a question about his team’s execution

- Data forwarding
- Scoreboarding with in-order execution
- Scoreboarding with out-of-order execution
- Precise interrupts
- Speculative execution

In-Order Execution

- Issue instructions in program order
- Retire (complete) instructions in program order
- May not give optimal performance because of instruction dependencies
  - Created whenever there is a dependence between instructions where pipelining overlap changes the order of access
    - RAW
    - WAW
    - WAR

Data Hazards

- Read After Write (RAW)
  - True data dependence (most common type)
  - Must preserve program order to ensure correct execution
- Write After Read (WAR)
  - Output dependence
  - Only present in pipelines that
    - write in more than one pipe stage
    - allow instructions to proceed after previous instructions stalls
- Write After Write (WAW)
  - Anti-dependence
  - Can occur when instructions are reordered
### Data Hazards

**Time (clock cycles)**

- `add r1, r2, r3`
- `sub r4, r1, r3`
- `and r6, r1, r7`
- `or r8, r1, r9`
- `xor r10, r1, r11`

**OPCODE Dest, Source1, Source2**

- `add r1, r2, r3`
- `sub r4, r1, r3`
- `and r6, r1, r7`
- `or r8, r1, r9`
- `xor r10, r1, r11`

**Reg ALU DMem Ifetch**

Adapted from David Culler’s CS 252 lecture notes. Copyright © 2002 UCB.

### Forwarding to Avoid Data Hazards

**Time (clock cycles)**

- `add r1, r2, r3`
- `sub r4, r1, r3`
- `and r6, r1, r7`
- `or r8, r1, r9`
- `xor r10, r1, r11`

**Reg ALU DMem Ifetch**

Adapted from John Kubiatowicz’s CS 252 lecture notes. Copyright © 2003 UCB.

### HW Change for Forwarding

**NextPC Registers EX/ID DMEM ALU Data Memory**

- **EX**
- **ID**
- **DMEM**
- **ALU**
- **Data Memory**

Adapted from David Culler’s CS 252 lecture notes. Copyright © 2002 UCB.

### Problems?

- How do we prevent **WAR** and **WAW** hazards?
- How do we deal with variable latency?
  - Forwarding for **RAW** hazards harder

**Clock Cycle Number**

<table>
<thead>
<tr>
<th>Instruction</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>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F5,34(R2)</td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F2,40(R3)</td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULTD F0,F2,F4</td>
<td>IF</td>
<td>ID</td>
<td>stall</td>
<td>M1</td>
<td>M2</td>
<td>M3</td>
<td>M4</td>
<td>M5</td>
<td>M6</td>
<td>M7</td>
<td>M8</td>
<td>M9</td>
<td>M10</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBD F8,F6,F2</td>
<td>IF</td>
<td>ID</td>
<td>AI</td>
<td>A2</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVD F10,F0,F6</td>
<td>IF</td>
<td>ID</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>D1</td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD F6,F8,F2</td>
<td>IF</td>
<td>ID</td>
<td>AI</td>
<td>A2</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adapted from John Kubiatowicz’s CS 252 lecture notes. Copyright © 2003 UCB.

RAW

WAR
Example: Pipelined, Superscalar CPU

• **2-way superscalar**
  – Can issue up to 2 instructions per cycle

• **For instructions decoded in cycle n**
  – Execution starts in cycle \( n + 1 \)
  – ADD/SUB completes in cycle \( n + 2 \)
  – MUL completes in cycle \( n + 3 \)

Rules for Issuing Instructions

• **True data dependence (RAW)**
  – Don’t issue if any operand is being written

• **Anti-dependence (WAR)**
  – Don’t issue if result register is being read

• **Output dependence (WAW)**
  – Don’t issue if result register is being written

Scoreboarding: In-Order Issue/Completion

- Instruction 4 has a true dependence (RAW) from Instruction 2
  – Stall until R4 is available
Scoreboarding: In-Order Issue/Completion

- Instruction 2 actually completes during cycle 3
  – Must retire instructions in order

Scoreboarding: In-Order Issue/Completion

- Instruction 6 has an anti-dependence (WAR) from Instructions 4 & 5
  – Stall until R1 is available

Scoreboarding: In-Order Issue/Completion

- Instruction 7 has a true dependence (RAW) from Instruction 6
  – Stall until R1 is available

Scoreboarding: In-Order Issue/Completion

- Instruction 8 has an anti-dependence (WAR) from Instruction 7
  – Stall until R1 is available
Out-of-Order (O-O-O) Execution

- What if we skipped the dependent instructions?
  – Potentially find independent instructions to execute

- What about the program’s result?
  – Must still guarantee the same result as in-order execution

Scoreboarding: O-O-O Issue/Completion

- Skip Instruction 4 to issue Instruction 5
- Allow Instruction 2 to retire to issue Instruction 4

Scoreboarding: O-O-O Issue/Completion

- Register renaming in Instruction 6 to eliminate WAR hazards with Instructions 4 and 5
  – Must maintain the data dependence with Instruction 7

Scoreboarding: O-O-O Issue/Completion

- Register renaming in Instruction 8 to eliminate WAR hazard with Instruction 5
Precise Interrupt

- Need the capability to store the CPU state
- With out-of-order completion
  - If an interrupt occurred, it would be difficult to save current state
  - Not possible to say all instructions up to some address had been executed and all instructions beyond it had not
- In-order completion ensures precise interrupts

Precise Interrupts/Exceptions

- An interrupt or exception is considered precise if there is a single instruction (or interrupt point) for which all instructions before that instruction have committed their state and no following instructions including the interrupting instruction have modified any state.
  - This means, effectively, that you can restart execution at the interrupt point and "get the right answer"
  - Implicit in the example below of a device interrupt:
    - Interrupt point is at first lw instruction

Exception/Interrupt Classifications

- Exceptions: relevant to the current process
  - Faults, arithmetic traps, and synchronous traps
  - Invoke software on behalf of the currently executing process
- Interrupts: caused by asynchronous, outside events
  - I/O devices requiring service (DISK, network)
  - Clock interrupts (real time scheduling)
- Machine Checks: caused by serious hardware failure
  - Not always restartable
  - Indicate that bad things have happened
    - Non-recoverable ECC error
    - Machine room fire
    - Power outage

A Related Classification: Synchronous vs. Asynchronous

- Synchronous: means related to the instruction stream, i.e. during the execution of an instruction
  - Must stop an instruction that is currently executing
  - Page fault on load or store instruction
  - Arithmetic exception
  - Software Trap Instructions
- Asynchronous: means unrelated to the instruction stream, i.e. caused by an outside event.
  - Does not have to disrupt instructions that are already executing
  - Interrupts are asynchronous
  - Machine checks are asynchronous
Speculative Execution

- **Basic block**
  - Linear sequence of code with one entry and one exit
  - No control (branch) instructions

- **Insufficient parallelism in basic blocks**
  - Allow reordering across basic blocks
  - **Hoisting** - moving instructions upward over a branch
    - Effective for potentially slow instructions like LOAD

- **By hoisting, you do not know if that code would have been executed**
  - Typically a compiler does the scheduling

Problem with Speculative Execution

- **Speculative execution could cause exceptions**
  - Ex: LOAD could cause a cache miss
    - If the data was needed, then the exception is OK
    - If the data was not needed, then the exception is bad
  - Some architectures include SPECULATIVE-LOAD
    - Only tries to retrieve data from cache

- **Speculative execution could cause correct programs to fail**
  - Ex: if (x ≠ 0) z = y / x
    - If the DIV is hoisted, the program could cause a divide by zero error despite the condition check
    - Use a poison bit for speculative instructions that fail

Summary

- Data forwarding hardware can eliminate stalls due to true data dependencies

- Scoreboarding is a technique to monitor the dynamic scheduling of instructions

- A typical machine will have in-order issue, out-of-order execution, and in-order completion