

Required Readings

- **This week**


  - H&H Chapters 7.8 and 7.9

Agenda for Today & Next Few Lectures

- Single-cycle Microarchitectures
- Multi-cycle and Microprogrammed Microarchitectures
- Pipelining
- Issues in Pipelining: Control & Data Dependence Handling, State Maintenance and Recovery, ...
- Out-of-Order Execution
- Other Execution Paradigms
Recall: OOO Execution: Restricted Dataflow

- An out-of-order engine dynamically builds the dataflow graph of a piece of the program

- The dataflow graph is limited to the **instruction window**
  - Instruction window: all decoded but not yet retired instructions

- Can we do it for the whole program?
  - In other words, how can we have a large instruction window?
- Can we do it efficiently with Tomasulo’s algorithm?
Recall: State of RAT and RS in Cycle 7

<table>
<thead>
<tr>
<th>Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL R1, R2 → R3</td>
<td>F</td>
<td>D</td>
<td>E₁</td>
<td>E₂</td>
<td>E₃</td>
<td>E₄</td>
<td>E₅</td>
</tr>
<tr>
<td>ADD R3, R4 → R5</td>
<td>F</td>
<td>D</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADD R2, R6 → R7</td>
<td>F</td>
<td>D</td>
<td>E₁</td>
<td>E₂</td>
<td>E₃</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADD R8, R9 → R10</td>
<td>F</td>
<td>D</td>
<td>E₁</td>
<td>E₂</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MUL R7, R10 → R11</td>
<td>F</td>
<td>D</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADD R5, R11 → R5</td>
<td>F</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Valid</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>0</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>0</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>0</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>0</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>0</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

Source 1                      | Source 2
--------------------------------|--------------------------------|
| V  | Tag | Value | V  | Tag | Value | V  | Tag | Value |
| a  | 0   | x     | 1  | ~   | 4     | x  | 1   | ~   | 1  | ~   | 2     |
| b  | 1   | ~     | 2  |     |       | y  | 0   | b   | 0   | c   |       |
| c  | 1   | ~     | 8  |     |       | z  |      |     |      |     |       |
| d  | 0   | a     | 0  | y   |       | t  |      |     |      |     |       |

+ +

*
Recall: Dataflow Graph

Nodes: operations performed by the instruction
Arcs: tags in Tomasulo’s algorithm

MUL \[ R1, R2 \rightarrow R3 \] (x)
ADD \[ R3, R4 \rightarrow R5 \] (a)
ADD \[ R2, R6 \rightarrow R7 \] (b)
ADD \[ R8, R9 \rightarrow R10 \] (c)
MUL \[ R7, R10 \rightarrow R11 \] (y)
ADD \[ R5, R11 \rightarrow R5 \] (d)
Other Approaches to Concurrency
(or Instruction Level Parallelism)
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Out-of-order execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Fine-Grained Multithreading
- SIMD Processing (Vector and array processors, GPUs)
- Decoupled Access Execute
- Systolic Arrays
Review: Data Flow: Exploiting Irregular Parallelism
Data Flow Summary

- Availability of data determines order of execution
- A data flow node fires when its sources are ready
- Programs represented as data flow graphs (of nodes)

- Data Flow at the ISA level has not been (as) successful

- Data Flow implementations at the microarchitecture level (while preserving Von Neumann semantics) have been very successful
  - Out of order execution is the prime example
Pure Data Flow Advantages/Disadvantages

- **Advantages**
  - Very good at exploiting irregular parallelism
  - Only real dependencies constrain processing
  - More parallelism can be exposed than Von Neumann model

- **Disadvantages**
  - No precise state semantics
    - Debugging very difficult
    - Interrupt/exception handling is difficult (what is precise state semantics?)
  - Too much parallelism? (Parallelism control needed)
  - High bookkeeping overhead (tag matching, data storage)
  - ...
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Out-of-order execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Fine-Grained Multithreading
- SIMD Processing (Vector and array processors, GPUs)
- Decoupled Access Execute
- Systolic Arrays
Superscalar Execution
Superscalar Execution

- **Idea:** Fetch, decode, execute, retire multiple instructions per cycle
  - N-wide superscalar $\rightarrow$ N instructions per cycle

- Need to add the hardware resources for doing so

- Hardware performs the dependence checking between concurrently-fetched instructions

- Superscalar execution and out-of-order execution are orthogonal concepts
  - Can have all four combinations of processors:
    - [in-order, out-of-order] x [scalar, superscalar]
In-Order Superscalar Processor Example

- Multiple copies of datapath: Can fetch/decode/execute multiple instructions per cycle
- Dependencies make it tricky to issue multiple instructions at once

Here: Ideal IPC = 2
In-Order Superscalar Performance Example

lw $t0, 40($s0)
add $t1, $s1, $s2
sub $t2, $s1, $s3
and $t3, $s3, $s4
or $t4, $s1, $s5
sw $s5, 80($s0)

Ideal IPC = 2

Actual IPC = 2 (6 instructions issued in 3 cycles)
Superscalar Performance with Dependencies

Ideal IPC = 2

Actual IPC = 1.2 (6 instructions issued in 5 cycles)
Superscalar Execution Tradeoffs

- Advantages
  - Higher **IPC** (instructions per cycle)

- Disadvantages
  - **Higher complexity for dependency checking**
    - Require checking within a pipeline stage
    - Renaming becomes more complex in an OoO processor
  - More hardware resources needed