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
Other Approaches to Concurrency
(or Instruction Level Parallelism)
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Fine-Grained Multithreading
- Out-of-order Execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- SIMD Processing (Vector and array processors, GPUs)
- Decoupled Access Execute
- Systolic Arrays
Review: Data Flow: Exploiting Irregular Parallelism
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</td>
<td>R1, R2 → R3</td>
<td>F</td>
<td>D</td>
<td>E₁</td>
<td>E₂</td>
<td>E₃</td>
<td>E₄</td>
</tr>
<tr>
<td>ADD</td>
<td>R3, R4 → R5</td>
<td>F</td>
<td>D</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADD</td>
<td>R2, R6 → R7</td>
<td>F</td>
<td>D</td>
<td>E₁</td>
<td>E₂</td>
<td>E₃</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>R8, R9 → R10</td>
<td>F</td>
<td>D</td>
<td>E₁</td>
<td>E₂</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td>R7, R10 → R11</td>
<td>F</td>
<td>D</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>R5, R11 → R5</td>
<td>F</td>
<td>D</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
---|---
<table>
<thead>
<tr>
<th>V</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>~</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>~</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>a</td>
</tr>
</tbody>
</table>

Source 1 | Source 2
---|---
<table>
<thead>
<tr>
<th>V</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
<td>~</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>b</td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ + \]

\[ * \]
Recall: Dataflow Graph (Reverse-Engineered)

- **Nodes:** operations performed by the instruction
- **Arrows:** tags in Tmasulo’s algorithm

### Dataflow graph

- **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** $R5, R11 \rightarrow R5 (d)$
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 model semantics) have been very successful
  - Out of order execution is the prime example
Recall: ISA-level Tradeoffs: Program Counter

- Do we need a Program Counter (PC or IP) in the ISA?
  - Yes: Control-driven, sequential execution
    - An instruction is executed when the PC points to it
    - PC automatically changes sequentially (except for control flow instructions)
  - No: Data-driven, parallel execution
    - An instruction is executed when all its operand values are available (dataflow)

- Tradeoffs: MANY high-level ones
  - Ease of programming (for average programmers)?
  - Ease of compilation?
  - Performance: Extraction of parallelism?
  - Hardware complexity?
Pure Data Flow Advantages/Disadvantages

- **Advantages**
  - Very good at exploiting irregular parallelism
    - Only real dependences 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)
  - How to enable mutable data structures
  - ...
Recall: ISA vs. Microarchitecture Level Tradeoff

- A similar tradeoff (control vs. data-driven execution) can be made at the microarchitecture level.

  ISA: Specifies how the **programmer sees** the instructions to be executed:
  - Programmer sees a sequential, control-flow execution order vs.
  - Programmer sees a dataflow execution order.

- Microarchitecture: **How the underlying implementation actually executes** instructions:
  - Microarchitecture can execute instructions in any order as long as it obeys the semantics specified by the ISA when making the instruction results visible to software.
  - Programmer should see the order specified by the ISA.
Readings & Lectures on Data Flow Model


More detailed Lecture Video & Slides on DataFlow:
- http://www.youtube.com/watch?v=D2uue7izU2c
Lecture Video on Dataflow Model

http://www.youtube.com/watch?v=D2uue7izU2c
Approaches to (Instruction-Level) Concurrency

- Pipelining
- Fine-Grained Multithreading
- Out-of-order Execution
- Dataflow (at the ISA level)
- **Superscalar Execution**
- VLIW
- 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 → 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
- Dependences make it tricky to dispatch multiple instructions in the same cycle
  - Need dependence detection between concurrently-fetched instructions

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 Dependences

Ideal IPC = 2

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

lw $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or $t3, $s5, $s6
sw $s7, 80($t3)
Review: How to Handle Data Dependences

- Anti and output dependences are easier to handle
  - write to the destination only in last stage and in program order

- Flow dependences are more interesting

- Six fundamental ways of handling flow dependences
  - Detect and wait until value is available in register file
  - Detect and forward/bypass data to dependent instruction
  - Detect and eliminate the dependence at the software level
    - No need for the hardware to detect dependence
  - Detect and move it out of the way for independent instructions
  - Predict the needed value(s), execute “speculatively”, and verify
  - Do something else (fine-grained multithreading)
    - No need to detect
Superscalar Execution Tradeoffs

- **Advantages**
  - Higher instruction throughput
    - Higher IPC: instructions per cycle (i.e., lower CPI)

- **Disadvantages**
  - Higher complexity for dependence checking
    - Require checking within a pipeline stage
    - Register renaming becomes more complex in an OoO processor
    - Potentially lengthens critical path delay $\rightarrow$ clock cycle time
  - More hardware resources needed

- **Recall:** Execution time of an entire program
  - $\{\# \text{ of instructions}\} \times \{\text{Average CPI}\} \times \{\text{clock cycle time}\}$
Lecture 17a: Dataflow & Superscalar Execution

Prof. Onur Mutlu

ETH Zürich
Spring 2021
30 April 2021