CMSC 280: Parallel Processing

Jaderick P. Pabico
Institute of Computer Science, College of Arts and Sciences
University of the Philippines Los Baños, College 4031, Laguna

2nd Semester 2020-2021

Today's Discussion

- Implicit parallelism
- Limitations of the performance of memory systems
- Dichotomy of platforms
- Communication model
- Physical organization
- Communication costs
- Messaging cost models and routing
- Mapping techniques

Scope of Parallelism

- Computer Architecture
  - Based on von Neumann model
  - Composed of a processor, memory system, & datapath
- Each component presents significant performance bottlenecks
- Parallelism addresses each of these components in many significant ways.

Scope of Parallelism

- Examples:
  - Data intensive applications utilize high aggregate throughput, like databases, search engines, etc.
  - Server applications utilize high aggregate network bandwidth, like web servers, communication servers
  - Scientific applications typically utilize high processing and memory system performance
- It is important that we understand each of these performance bottlenecks.

Implicit Parallelism

- Microprocessor clock speeds have gained over the past 20 years to about 3 orders of magnitude.
- Higher levels of device integration have made available a large number of transistors
  - From Vacuum Tube, to transistor, to LSI, to VLSI
- The question of how best to use these resources is an important one.

Implicit Parallelism

- Current processors use these resources in multiple functional units and execute multiple instruction in the same cycle.
- The precise manner in which these instructions are selected and executed provides impressive diversity in architectures.

Pipelining and Superscalar Execution

- Pipelining overlaps various stages of instruction execution to achieve performance.
- At a high level of abstraction, an instruction can be executed while the next one is being decoded and the next one is being fecthed.
- This is akin to an assembly line for manufacture of cars.

Pipelining and Superscalar Execution

- Pipelining, however, has several limitations.
- The speed of a pipeline is eventually limited by the slowest stage.
- For this reason, conventional processors rely on very deep pipelines (e.g., 20-stage pipelines in state-of-the-art Pentiums)
- However, in typical program traces, every 5th-6th instruction is a conditional jump.
  - This requires very accurate branch prediction.

Pipelining and Superscalar Execution

- Pipelining, however, has several limitations.
- [illegible] by
- [illegible] ly [illegible] ines
- However, in typical program traces, every 5th-6th instruction is a conditional jump.
  - This requires very accurate branch prediction.

The penalty of a misprediction grows with the depth of the pipeline, since larger number of instructions will have to be flushed.

Pipelining and Superscalar Execution

- One simple way of alleviating these bottlenecks is to use multiple pipelines (in parallel).
- The question then becomes one of selecting these instructions.

BUT, is this practical?

Example of Superscalar Execution

- Adding a list of four numbers on a 2-pipeline processor

```
1. load R1, @1000        1. load R1, @1000        1. load R1, @1000
2. load R2, @1008        2. add R1, @1004         2. add R1, @1004
3. add R1, @1004         3. add R1, @1008         3. load R2, @1008
4. add R2, @100C         4. add R1, @100C         4. add R2, @100C
5. add R1, R2            5. store R1, @2000       5. add R1, R2
6. store R1, @2000                                6. store R1, @2000
```

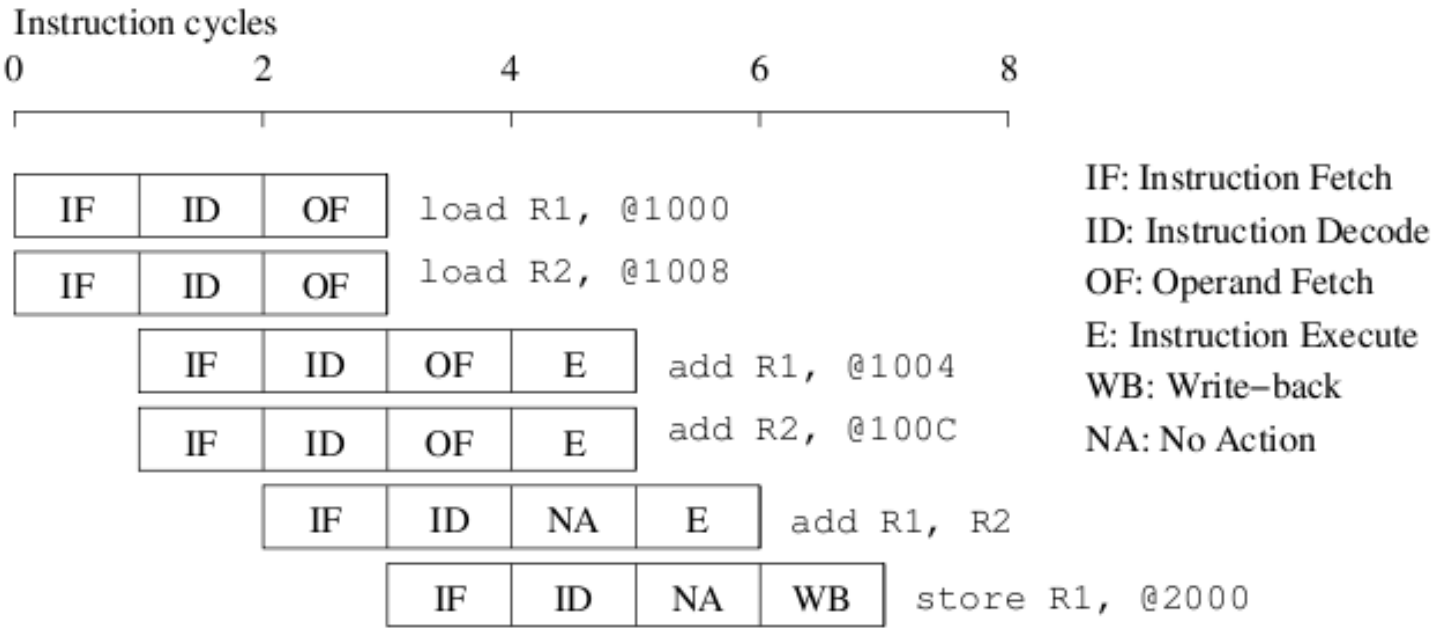
(i) (ii) (iii)

Three different code fragments for adding a list of four numbers

Example of Superscalar Execution

- Adding a list of four numbers (2 pipelines)

```
1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

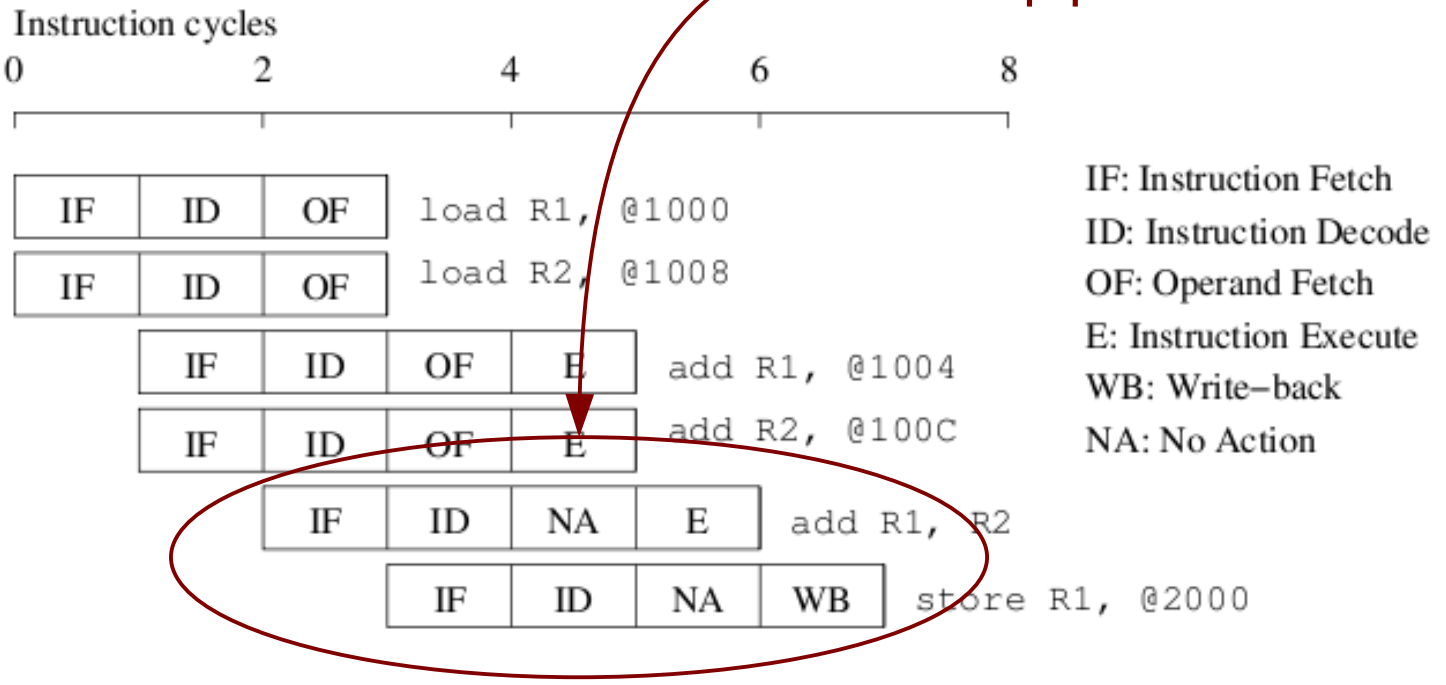


Example of Superscalar Execution

- Adding a list of four numbers (2 pipelines)

```
1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

Because of data dependency, one pipeline is not utilized



Example of Superscalar Execution

- Adding a list of four numbers (2 pipelines)

Can you perform the same for the other two examples?

```
1. load R1, @1000
2. add R1, @1004
3. add R1, @1008
4. add R1, @100C
5. store R1, @2000
```

(ii)

```
1. load R1, @1000
2. add R1, @1004
3. load R2, @1008
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

(iii)

Example of Superscalar Execution

- Adding a list of four numbers (2 pipelines)

Can you
the sa
the
two ex

The example, as well as the one that you have done, illustrate that different instruction mixed with identical semantics can take significantly different execution time.

```
1, @1000
, @1004
2, @1008
, @100C
, R2
R1, @2000
```

Lessons learned

- Scheduling of instruction is determined by a number of factors:
  - **True data dependency**: The result of one operation is an input to the next
  - **Resource dependency**: Two operations require the same resource
  - **Branch dependency**: Scheduling instructions across conditional branch statements cannot be done deterministically *a-priori*.

Lessons learned

- The scheduler, a piece of hardware looks at a large number of instructions in an instruction queue and selects appropriate number of instructions to execute concurrently based on these factors.
- The complexity of this hardware is an important constraint on superscalar processors.

Issues

- In the simpler model, the instructions can be issued only in the order in which they are encountered.
  - i.e., if the 2nd instruction cannot be issued because it has a data dependency with the 1st, only one instruction is issued in the cycle
  - This is called **in-order** issue

Issues

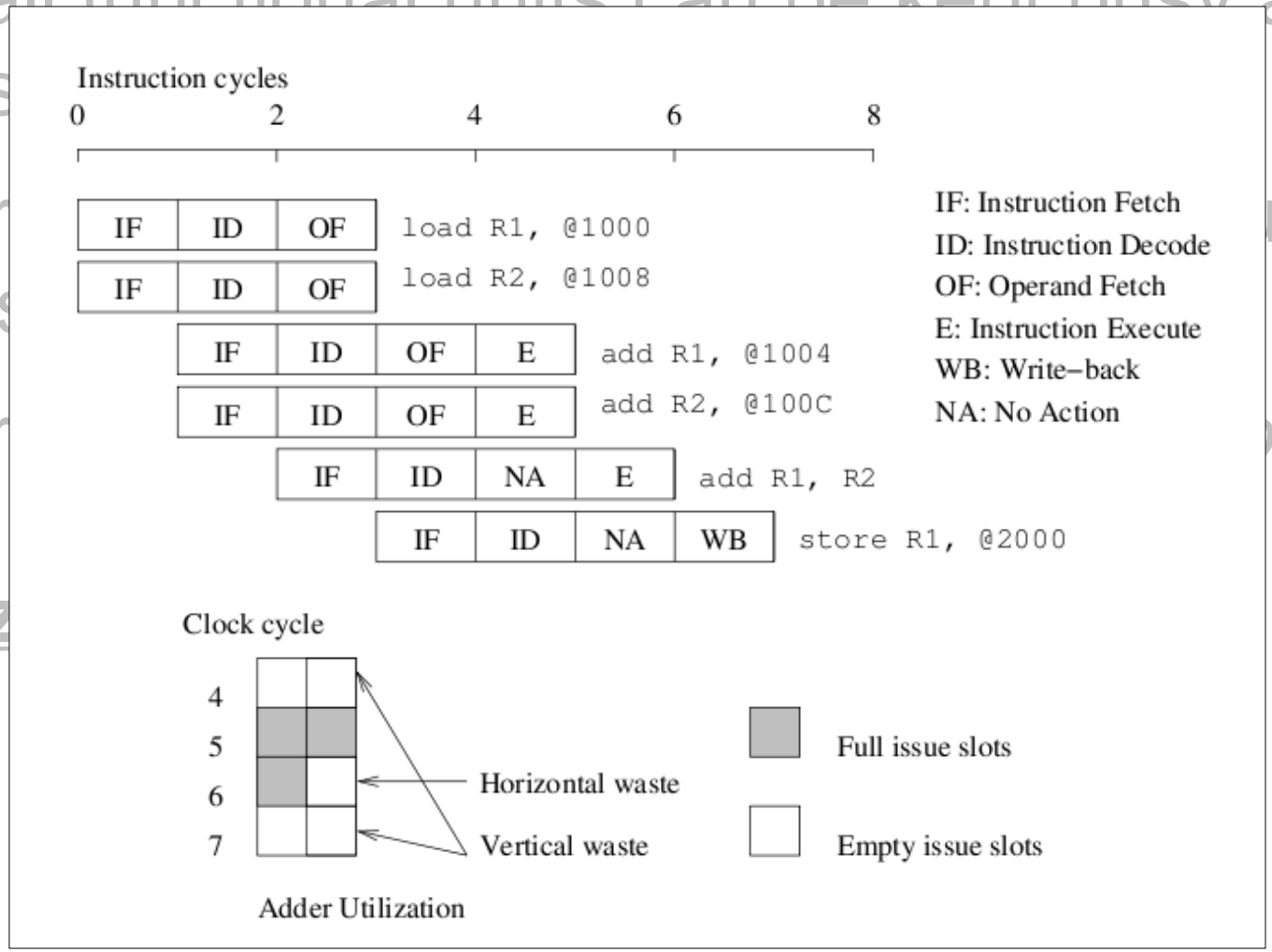
- Under a non-simple model, instructions may be issued out of order
  - i.e., if the 2nd instruction has data dependencies with the 1st, but the 3rd does not, the 1st and 3rd instructions can be co-scheduled.
  - This is called **dynamic** issue
- In-order vs. Dynamic Issues
  - Performance of in-order issue is generally limited

Efficiency Considerations

- Not all functional units can be kept busy at all times
- If during a cycle, *no functional units* are utilized, this is referred to as **vertical waste**.
- If during a cycle, only *some of the functional units* are utilized, this is referred to as **horizontal waste**.

Efficiency Considerations

- Not all functional units can be kept busy at all times
- If dur… utilized, this is
- If dur… *onal units* **horiz…**



Efficiency Considerations

- Due to limited parallelism in typical instruction traces, dependencies, or the inability of the scheduler to extract parallelism, the performance of superscalar processors is eventually limited.
- Conventional microprocessors typically support 4-way superscalar execution.

VLI Word Processors

- The hardware cost and complexity of the superscalar scheduler is a major consideration in processor design.
- To address this issues, VLIW processors rely on compile time analysis to identify and bundle together instructions that can be executed concurrently.
- These instructions are packed and dispatched together, and thus the name very long instruction word.

VLI Word Processors

- The hardware cost and complexity of the su s a major consideration in
- To VLIW processors rely on s to identify and bundle tog t can be executed concurrently.
- These instructions are packed and dispatched together, and thus the name very long instruction word.

This concept was used with some commercial success in the Multiflow Trace Machine (1984)

VLI Word Processors

- The hardware cost and complexity of the su… a major consideration in …
- To… VLIW processors rely on … to identify and bundle tog… t concurrently.
- These instructions are pa… together, and thus the na… instruction word.

This concept was used with some commercial success in the Multiflow Trace Machine (1984)

Variant of this concept are employed in the Intel IA64 processors

VLI Word Processors

- Issue hardware is simpler.
- Compiler has a bigger context from which to select co-scheduled instructions.
- Compilers,however, do not have runtime information such as cache misses.
  - Scheduling is, thus, inherently conservative.
- Branch and memory prediction is more difficult.

VLI Word Processors

- VLIW performance is highly dependent on the compiler.
  - A number of techniques are critical, such as
    - Loop unrolling
    - Speculative execution
    - Branch prediction
- Typical VLIW processors are limited to 4-way to 8-way parallelism.

Limitations of Memory System

- Memory system, and not processor speed, is often the bottleneck for many applications.
- Memory system performance is largely captures by two parameters:
  - Latency, and
  - Bandwidth

Limitations of Memory System

- **Latency**: The time from the issue of a memory request to the time the data is available at the processor
- **Bandwidth**: The rate at which data can be pumped to the processor by the memory system.

Limitations of Memory System

- **Latency**: The time from the issue of a memory request to the time the data is available at the processor
- **Bandwidth**: The rate at which data can be pumped to the processor by the memory system.

What's the difference?

Limitations of Memory System

- Example: Fire-hose
  - If the water comes out of the hose 2 seconds after the hydrant is turned on, the latency of the system is 2 seconds.
  - Once the water starts flowing, if the hydrant delivers water at the rate of 100 L/s, the bandwidth of the system is 100 L/s.
  - If you want immediate response from the hydrant, it is important to reduce latency.
  - If you want to fight big fires, you want high bandwidth.

Memory Latency

- **Example**: Consider a processor operating at 1 GHz (1ns clock) connected to a DRAM with a latency of 100ns (no caches). Assume that the processor has two multiply-add units and is capable of executing four instructions in each cycle of 1ns.
- Observations:
  - The peak processor rating is 4GFLOPS
  - Since the memory latency is equal to 100 cycles and block size is one word, every time a memory request is made, the processor must wait 100 cycles before it can process the data.

Memory Latency

- NOW, consider the problem of computing a dot-product of two vectors
  - A dot-product computation performs one multiply-add on a single pair of vector elements
  - It follows that the peak speed of this computation is limited to one floating point operation every 100 ns, or just 10 MFLOPS.
  - How is this compared to the peak rate of 4GFLOPS?