Digital Design & Computer Arch.

Lecture 17a: Dataflow & Superscalar Execution

Prof. Onur Mutlu

ETH Zürich

Spring 2021

30 April 2021

Required Readings

- This week
  - Smith and Sohi, "The Microarchitecture of Superscalar Processors," Proceedings of the IEEE, 1995
  - H&H Chapters 7.8 and 7.9
  - McFarling, "Combining Branch Predictors," DEC WRL Technical Report, 1993.

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

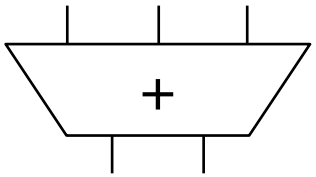
- Can we do it for the whole program?
  - In other words, how can we have a large instruction window?
- Can we do it efficiently with Tomasulo's algorithm?

Recall: State of RAT and RS in Cycle 7

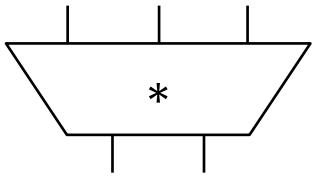
	Cycle	1	2	3	4	5	6	7
MUL R1, R2 → R3		F	D	E_1	E_2	E_3	E_4	E_5
ADD R3, R4 → R5			F	D	-	-	-	-
ADD R2, R6 → R7				F	D	E_1	E_2	E_3
ADD R8, R9 → R10					F	D	E_1	E_2
MUL R7, R10 → R11						F	D	-
ADD R5, R11 → R5							F	D

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	d	
R6	1		6
R7	0	b	
R8	1		8
R9	1		9
R10	0	c	
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	0	x		1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	0	b		0	c	
z						
t						



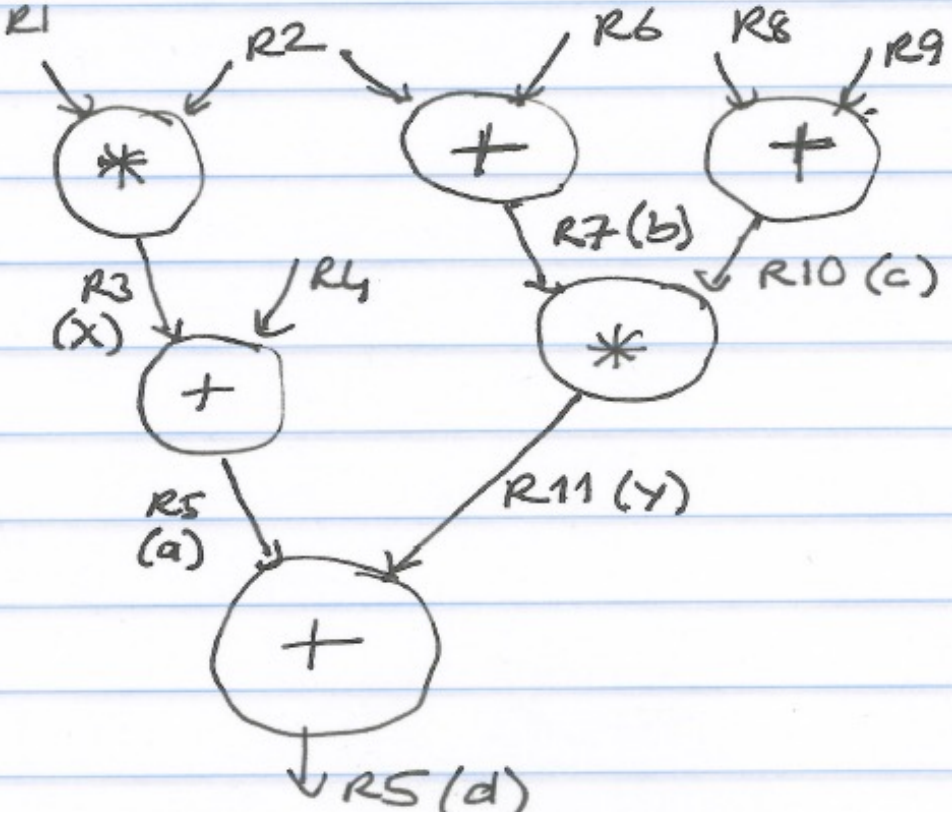
Recall: Dataflow Graph (Reverse-Engineered)

```
MUL  R1, R2 → R3  (x)
ADD  R3, R4 → R5  (a)
ADD  R2, R6 → R7  (b)
ADD  R8, R9 → R10 (c)
MUL  R7, R10 → R11 (y)
ADD  R5, R11 → R5 (d)
```

Dataflow graph

Nodes: operations performed by the instruction

Arcs: tags in Tomasulo's algorithm



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 Tradeoff: 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

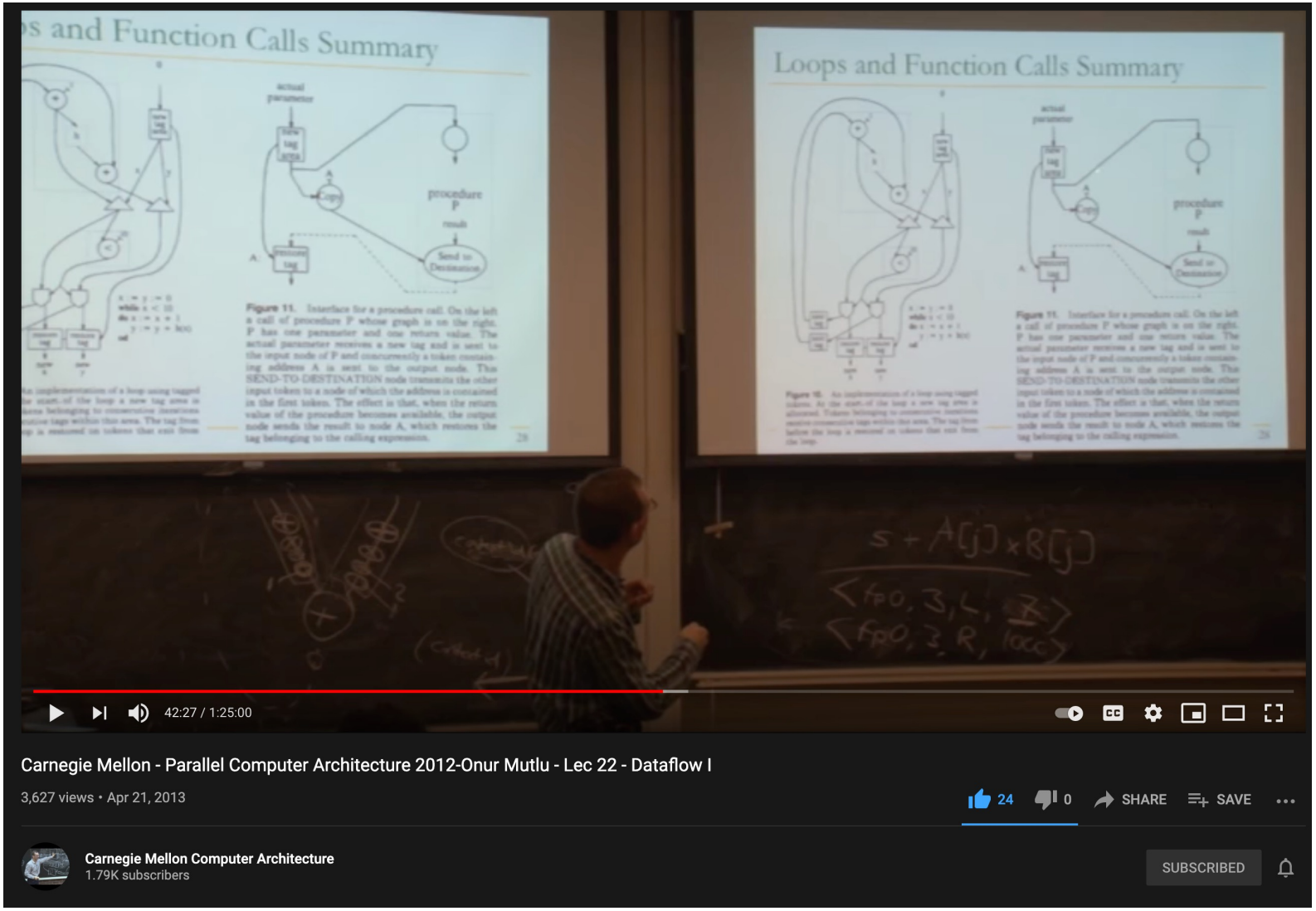
- Microarchitecture: How the **underlying implementation actually executes** instructions
  - Microarchitecture can execute instructions in any order as long as it obeys the semantics specified by the ISA when making the instruction results visible to software
    - Programmer should see the order specified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

Readings & Lectures on Data Flow Model

- Dennis and Misunas, "A preliminary architecture for a basic data-flow processor," ISCA 1974.
- Gurd et al., "The Manchester prototype dataflow computer," CACM 1985.

- More detailed Lecture Video & Slides on DataFlow:
  - http://www.youtube.com/watch?v=D2uue7izU2c
  - http://www.ece.cmu.edu/~ece740/f13/lib/exe/fetch.php?media=onur-740-fall13-module5.2.1-dataflow-part1.ppt

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

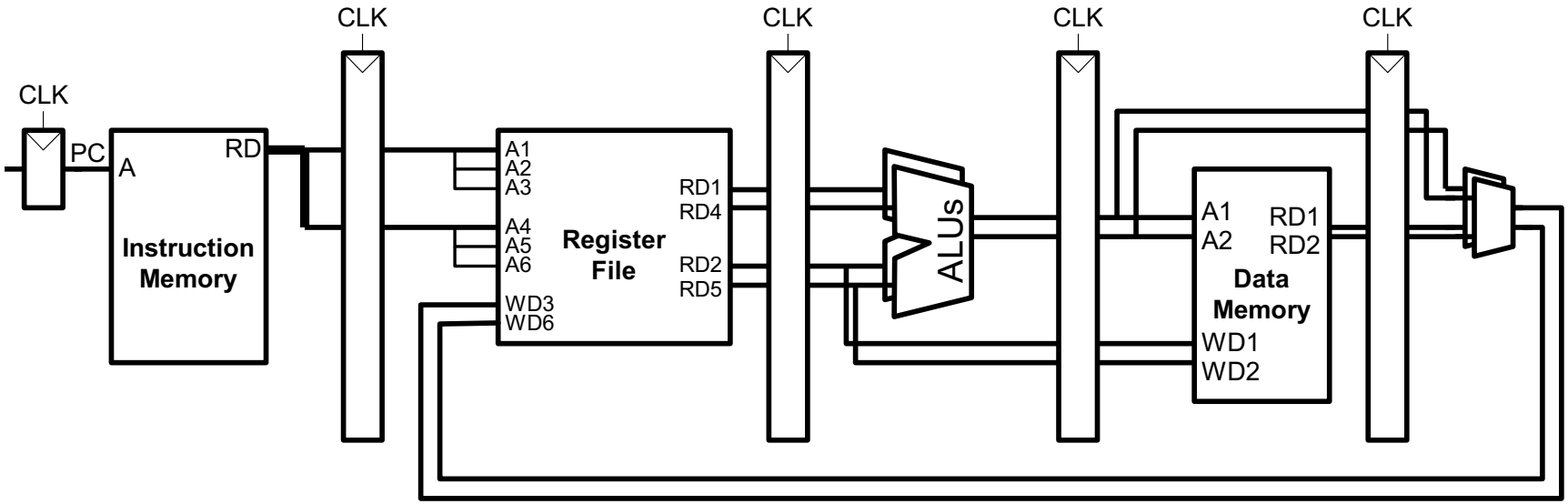
Superscalar Execution

- 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

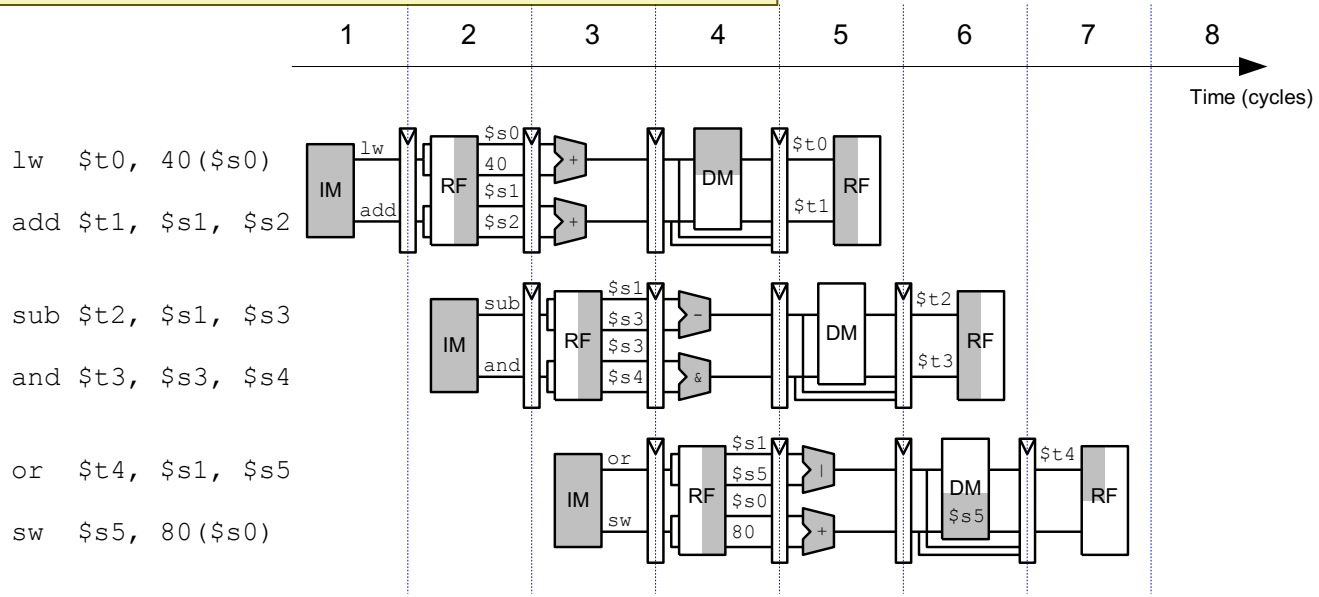


Here: Ideal IPC = 2

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

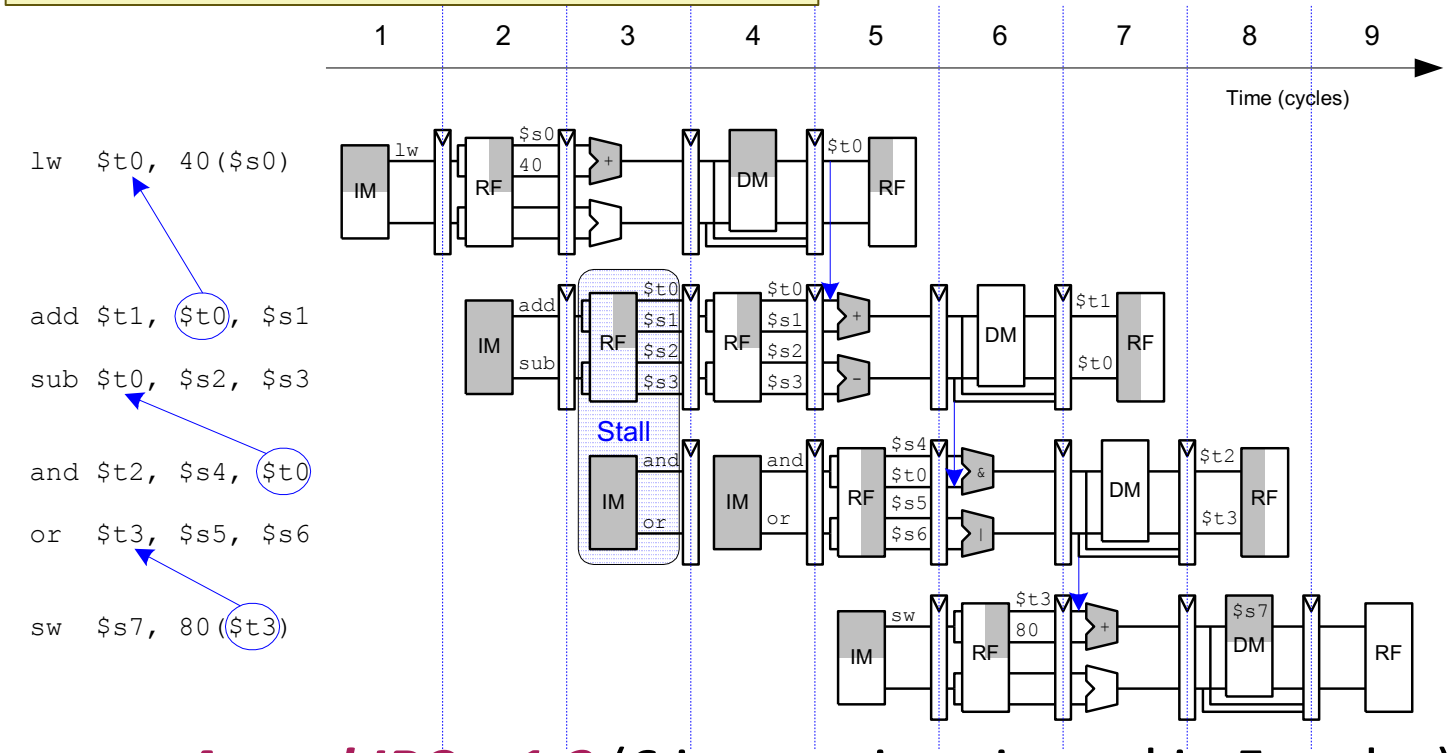


Actual IPC = 2 (6 instructions issued in 3 cycles)

Superscalar Performance with Dependences

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Ideal IPC = 2



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

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 → clock cycle time
  - More hardware resources needed

- Recall: Execution time of an entire program
  - **{# of instructions} x {Average CPI} x {clock cycle time}**

Digital Design & Computer Arch.

Lecture 17a: Dataflow & Superscalar Execution

Prof. Onur Mutlu

ETH Zürich

Spring 2021

30 April 2021