Digital Design & Computer Arch.

Lecture 16a: Dataflow & Superscalar Execution

Prof. Onur Mutlu

ETH Zürich

Spring 2020

23 April 2020

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

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

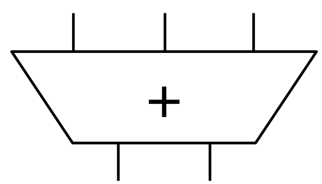
- Can we do it for the whole program?
  - In other words, how can we have a large instruction window?
- Can we do it efficiently with Tomasulo's algorithm?

Recall: State of RAT and RS in Cycle 7

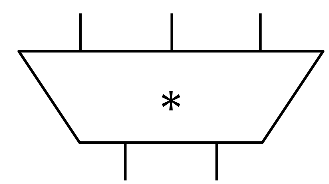
Instruction	Cycle 1	2	3	4	5	6	7
MUL R1, R2 → R3	F	D	E_1	E_2	E_3	E_4	E_5
ADD R3, R4 → R5		F	D	-	-	-	-
ADD R2, R6 → R7			F	D	E_1	E_2	E_3
ADD R8, R9 → R10				F	D	E_1	E_2
MUL R7, R10 → R11					F	D	-
ADD R5, R11 → R5						F	D

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						



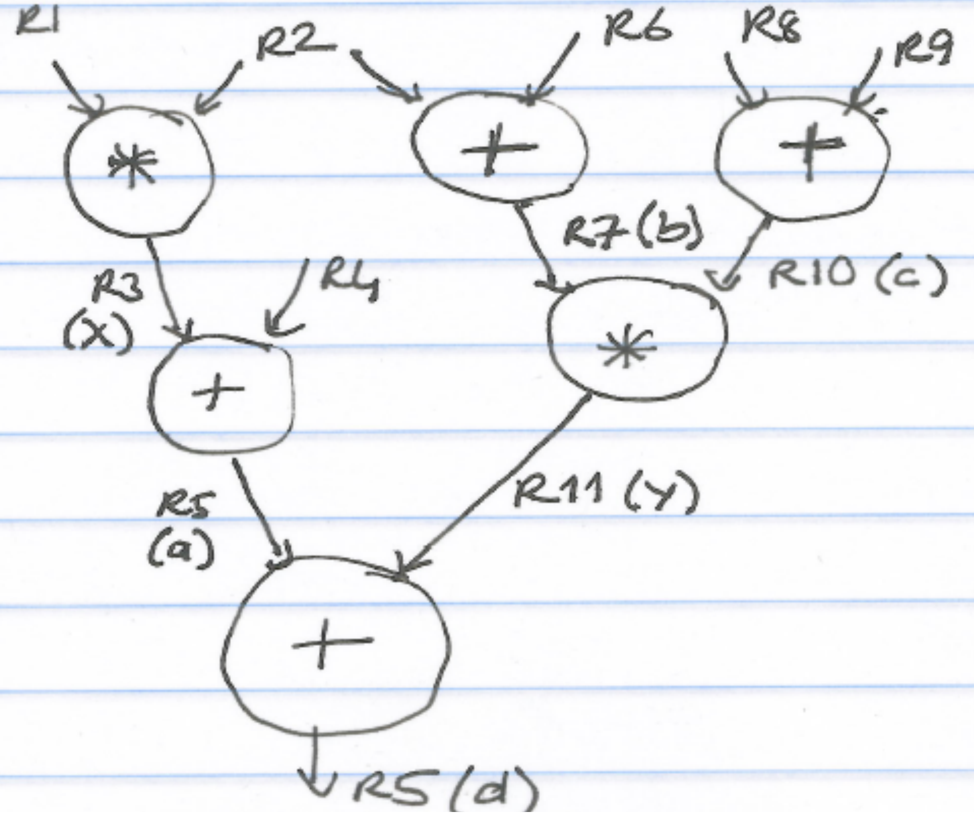
Recall: Dataflow Graph

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



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 → N instructions per cycle

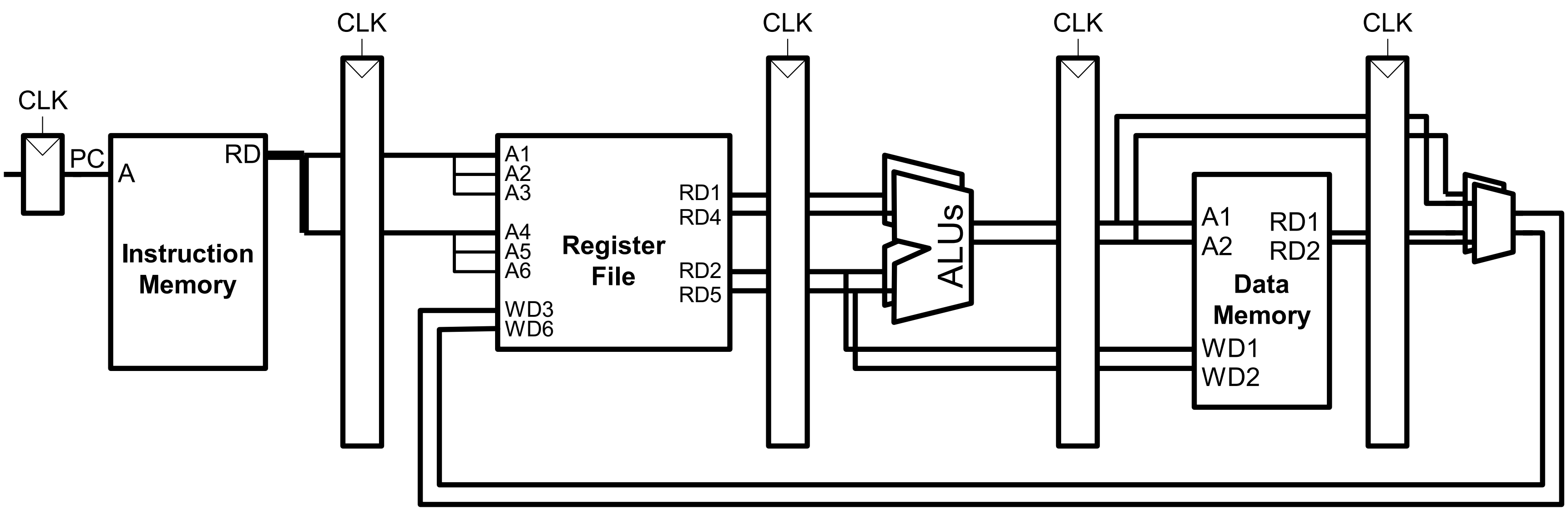
- Need to add the hardware resources for doing so

- 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

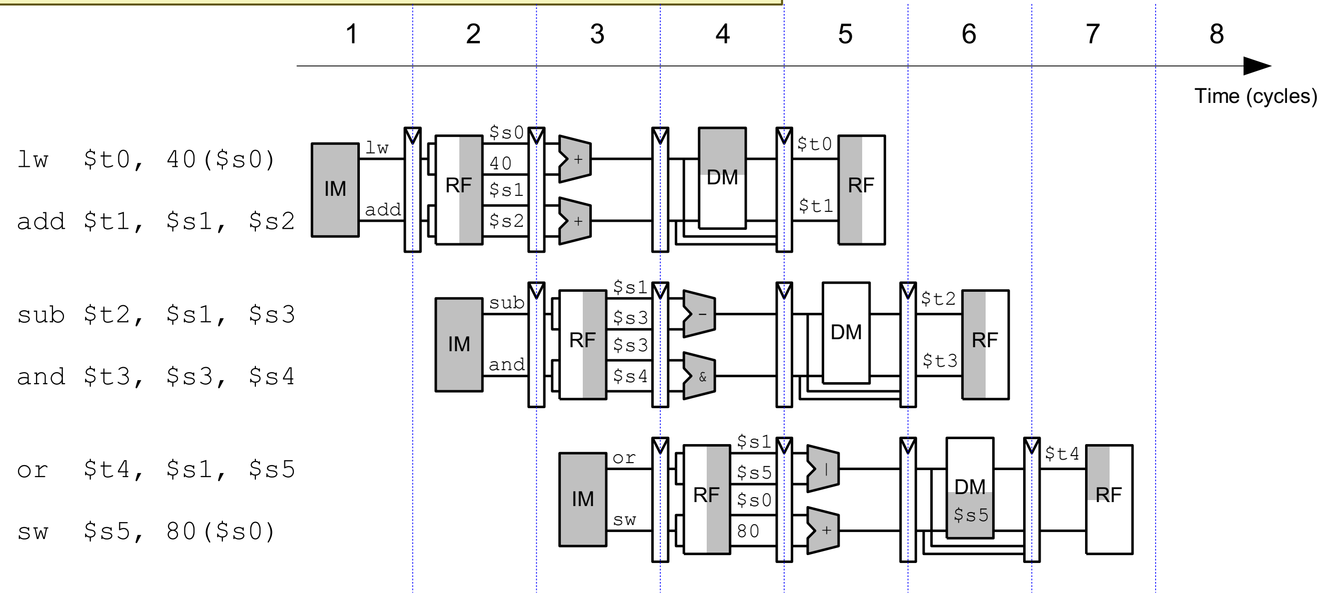


Here: Ideal IPC = 2

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

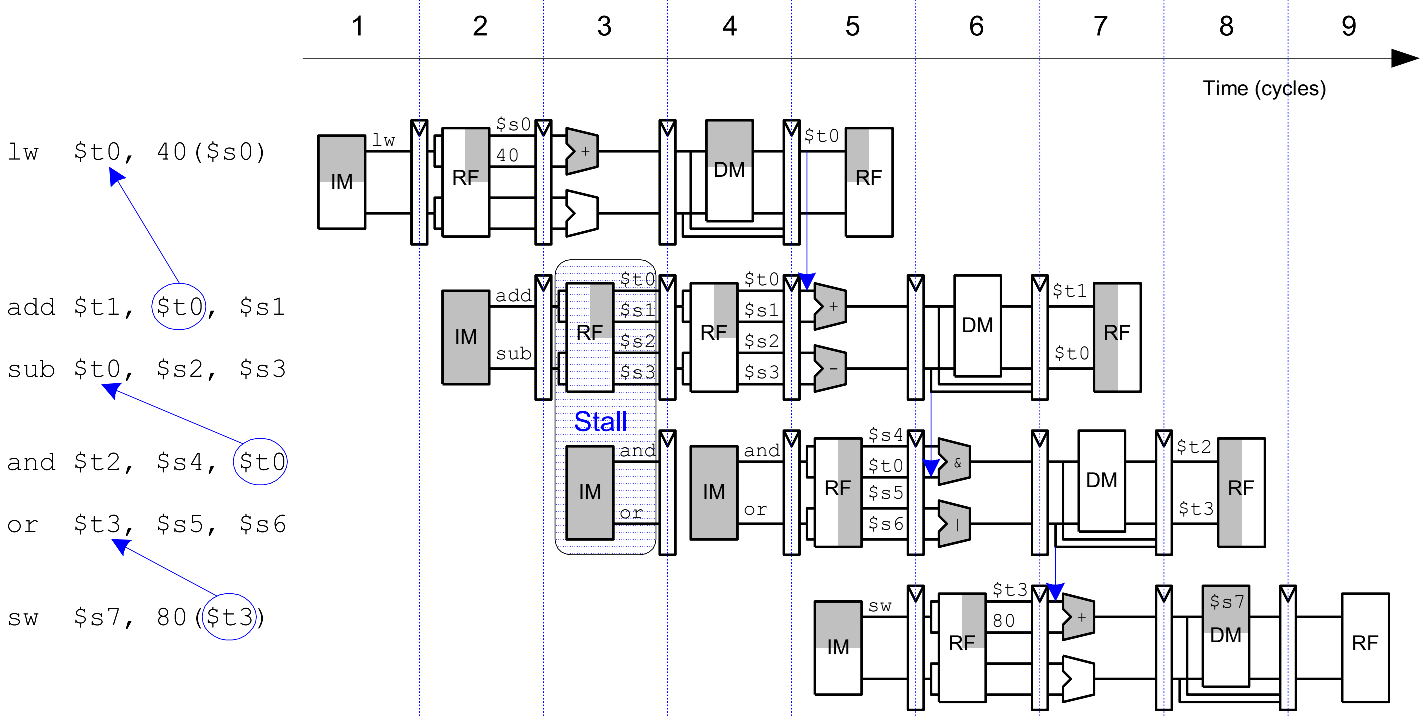


Actual IPC = 2 (6 instructions issued in 3 cycles)

Superscalar Performance with Dependencies

```
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)

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

Digital Design & Computer Arch.

Lecture 16a: Dataflow & Superscalar Execution

Prof. Onur Mutlu

ETH Zürich

Spring 2020

23 April 2020