15-740/18-740
Computer Architecture
Lecture 11: More Out-of-Order Execution

Prof. Onur Mutlu
Carnegie Mellon University
Fall 2011, 10/5/2011

Last Lecture

- Downsides of static scheduling
- Out of order execution (Dynamic scheduling)
  - Data stored in many places (reservation stations) vs.
  - Central physical register file

Today

- More on out-of-order execution
- An example code execution
- Issues in out-of-order processing

Reviews

- Due next Monday
  - Mutlu et al., "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors," HPCA 2003.
  - Mutlu et al., "Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance," IEEE Micro Top Picks 2006.

- Due next Wednesday
- Chrysos and Emer, "Memory Dependence Prediction Using Store Sets," ISCA 1998.

Out-of-order Execution (Dynamic Scheduling)

- Idea: Move the dependent instructions out of the way of independent ones
  - Rest areas for dependent instructions: Reservation stations

- Monitor the source "values" of each instruction in the resting area
- When all source "values" of an instruction are available, "fire" (i.e. dispatch) the instruction
  - Instructions dispatched in dataflow (not control-flow) order

- Benefit:
  - Latency tolerance: Allows independent instructions to execute and complete in the presence of a long latency operation

In-order vs. Out-of-order Dispatch

- In order dispatch:

F	D	E	E	E	E	R	W								
	F	D		STALL		E	R	W							
		F		STALL		D	E	R	W						
						F	D	E	E	E	E	E	R	W	
							F	D		STALL			E	R	W

IMUL R3 ← R1, R2
ADD R3 ← R3, R1
ADD R1 ← R6, R7
IMUL R3 ← R6, R8
ADD R7 ← R3, R9

- Tomasulo + precise exceptions:

F	D	E	E	E	E	R	W				
	F	D		WAIT		E	R	W			
		F	D	E	R				W		
			F	D	E	E	E	E	R	W	
				F	D		WAIT		E	R	W

- 16 vs. 12 cycles

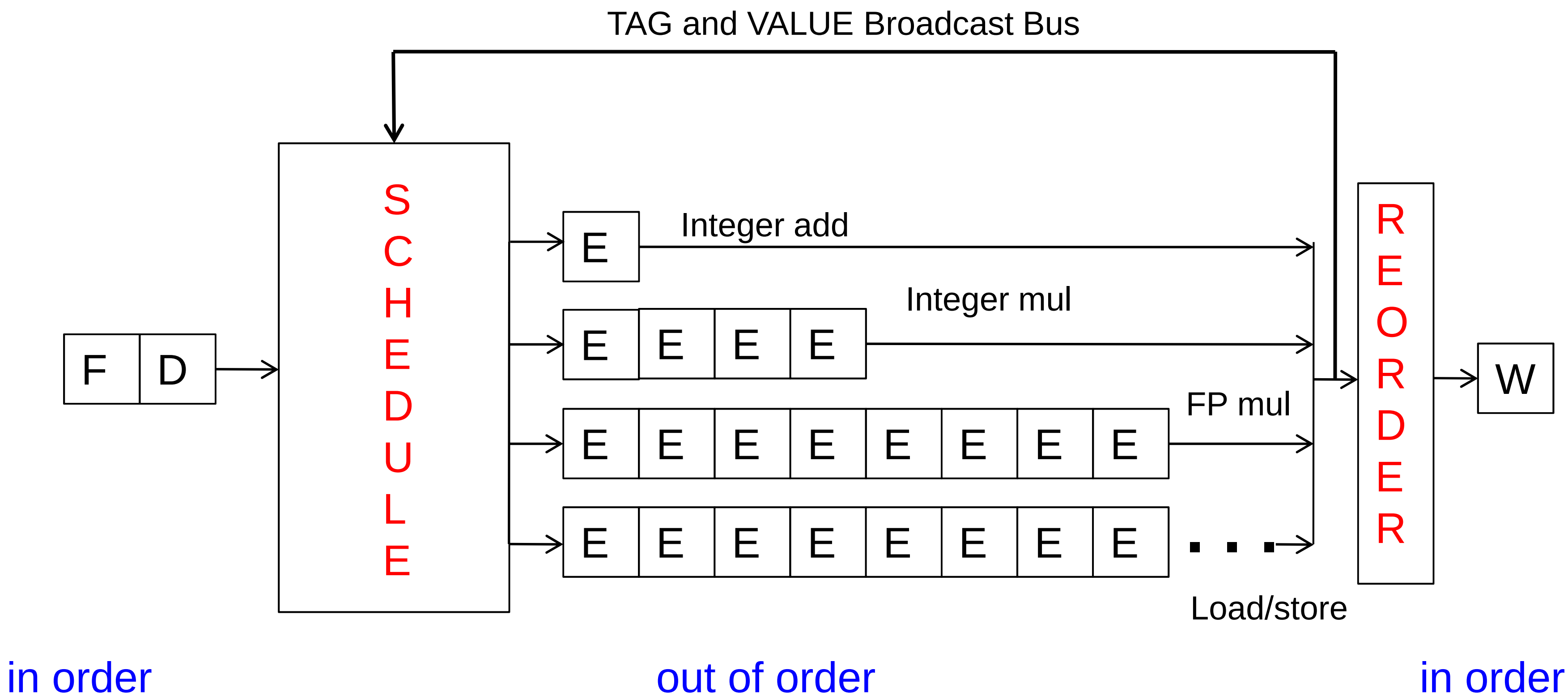
Enabling OoO Execution

1. Need to link the consumer of a value to the producer
   - Register renaming: Associate a "tag" with each data value
2. Need to buffer instructions until they are ready
   - Insert instruction into reservation stations after renaming
3. Instructions need to keep track of readiness of source values
   - Broadcast the "tag" when the value is produced
   - Instructions compare their "source tags" to the broadcast tag → if match, source value becomes ready
4. When all source values of an instruction are ready, dispatch the instruction to functional unit (FU)
   - What if more instructions become ready than available FUs?

Summary of OOO Execution Concepts

- Renaming eliminates false dependencies
- Tag broadcast enables value communication between instructions → dataflow
- An out-of-order engine dynamically builds the dataflow graph of a piece of the program
  - which piece?
    - Limited to the instruction window
  - Can we do it for the whole program? Why would we like to?
    - How can we have a large instruction window efficiently?

Two Humps in a Modern Pipeline



- Hump 1: Reservation stations (scheduling window)
- Hump 2: Reordering (reorder buffer, aka instruction window or active window)

Tomasulo's Algorithm: Renaming

- Register rename table (register alias table)

	tag	value	valid?
R0			1
R1			1
R2			1
R3			1
R4			1
R5			1
R6			1
R7			1
R8			1
R9			1

Tomasulo's Algorithm

- If reservation station available before renaming
  - Instruction + renamed operands (source value/tag) inserted into the reservation station
  - Only rename if reservation station is available
- Else stall
- While in reservation station, each instruction:
  - Watches common data bus (CDB) for tag of its sources
  - When tag seen, grab value for the source and keep it in the reservation station
  - When both operands available, instruction ready to be dispatched
- Dispatch instruction to the Functional Unit when instruction is ready
- After instruction finishes in the Functional Unit
  - Arbitrate for CDB
  - Put tagged value onto CDB (tag broadcast)
  - Register file is connected to the CDB
    - Register contains a tag indicating the latest writer to the register
    - If the tag in the register file matches the broadcast tag, write broadcast value into register (and set valid bit)
  - Reclaim rename tag
    - no valid copy of tag in system!

Register Renaming and OoO Execution

- Architectural registers dynamically renamed
  - Mapped to reservation stations

	tag	value	valid?
R0	-	V0	1
R1	S2	new1V1 / V1	0 / 1
R2	-	V2	1
R3	S0 / S3	new3V3 / V3	0 / 1
R4	-	V4	1
R5	-	V5	1
R6	-	V6	1
R7	S4	V7	0
R8	-	V8	1
R9	-	V9	1

Instruction	Renamed
IMUL R3 ← R1, R2	IMUL S0 ← V1, V2
ADD R3 ← R3, R1	ADD S1 ← S0, V4
ADD R1 ← R6, R7	ADD S2 ← V6, V7
IMUL R3 ← R6, R8	IMUL S3 ← V6, V8
ADD R7 ← R3, R9	ADD S4 ← S3, V9

	Src1 tag	Src1 value	V?	Src2 tag	Src2 value	V?	Ctl	S?
S0	-	Retired --- Entry Deallocated				ıent	nul	1
S1	S0	Retired --- Entry Deallocated				ent	add	1
S2	-	Completed --- Wait for Retirement					add	1
S3	-	BROADCAST S3 and new3V3				1	add	1
S4	S3	new3V3 / V3	0 / 1	-	V9	1	add	

An Exercise

```
MUL   R3 ← R1, R2
ADD   R5 ← R3, R4
ADD   R7 ← R2, R6
ADD   R10 ← R8, R9
MUL   R11 ← R7, R10
ADD   R5 ← R5, R11
```

F	D	E	R	W

- Assume ADD (4 cycle execute), MUL (6 cycle execute)
- Assume one adder and one multiplier
- How many cycles
  - in a non-pipelined machine
  - in an in-order-dispatch pipelined machine with future file and reorder buffer
  - in an out-of-order dispatch pipelined machine with future file and reorder buffer

An Exercise (II)

- Execution of the previous example on a machine with register data values distributed across future file, reservation stations, reorder buffer, and architectural register files

- Execution of the previous example on a machine with register data values consolidated in a centralized physical register file

- Think of the tradeoffs between the two designs
- Understand how each design works