Computer Science 146
Computer Architecture

Fall 2019

Harvard University

Instructor: Prof. David Brooks
dbrooks@eecs.harvard.edu

Lecture 8: Multiple Issue and Speculation

Lecture Outline

- Dynamic Branch Predictor Review
- Superscalar/Multiple-Issue Designs
- Speculative Execution
  - Tomasulo with ROB example

Dynamic Branch Prediction

- Branch History Table: 2-bits for good loop prediction
- Correlation: Recently executed branch give insight into the next branch
  - Different Branches or different executions of the same branch
- History can be global or per-branch PC (or per-set)
- Tournament Predictors – Combine many approaches
- Branch Target Buffer – Predicts *target* of branch

Return Address Stack

- Say foo() is called from many different locations in a program
- It will then return to many different locations!
- RAS can predict which location to return to because it stores the caller PC
- This is faster than having to load up indirect jumps (jump r31)
- If the call-depth doesn't exceed the size of the RAS, this prediction will always be correct

Multiple Issue

- Goal: Sustain a CPI of less than 1 by issuing and processing multiple instructions per cycle
- SuperScalar
  - Issue varying number of instructions per clock
    - Statically Scheduled
    - Dynamically Scheduled
- VLIW (EPIC)
  - Issue a fixed number of instructions formatted as one large instruction or instruction "packet"
  - Similar to static-scheduled superscalar

Multiple Issue Choices

Common Name	Issue Structure	Hazard Detection	Scheduling	Examples
Superscalar (static)	Dynamic	Hardware	Static	Sun UltraSPARC II/III
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	IBM POWER2
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Pentium III/4, MIPS R10K, Alpha 21264, IBM POWER4, HP PA8500
VLIW	Static	Software	Static	Trimedia, i860
EPIC	"mostly" static	mostly software	mostly static	Itanium (IA64)

Multiple Issue Example

	Single Issue Clock Cycle									
	1	2	3	4	5	6	7	8	9	10
i	IF	ID	EX	M	WB					
i+1		IF	ID	EX	M	WB				
i+2			IF	ID	EX	M	WB			
i+3				IF	ID	EX	M	WB		
i+4					IF	ID	EX	M	WB	
i+5						IF	ID	EX	M	WB

Multiple Issue Clock Cycle						
1	2	3	4	5	6	7
IF	ID	EX	M	WB		
IF	ID	EX	M	WB		
	IF	ID	EX	M	WB	
	IF	ID	EX	M	WB	
		IF	ID	EX	M	WB
		IF	ID	EX	M	WB

- Maybe 1 ALU + 1 FP
- 2 ALU + 2 LD/ST + 2 FP
- Many combinations possible – restriction ease implementation

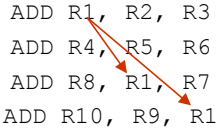
Multiple Issue: Hazards

- As usual, we have to deal with the big three hazards:
  - Structural Hazards
  - Data Hazards
  - Control Hazards
- Multiple issue gives:
  - More opportunity for hazards (why?)
  - Larger performance hit from hazards (why?)

Structural Hazards

- If both instructions per cycle are int/float we may need two int ALUs and two FP ALUs
- What about register files?
- This may lead to issue restrictions
  - Compiler/hardware has to manage these restrictions
- 2-issue machines typically do 1 INT/1 FP per cycle
  - Good performance for many apps (+)
  - Hazard Detection is easy (+)
  - No performance boost for non-FP apps (-)

Data Hazards



- Assume full-bypassing
  - How many stalls for single issue?
  - How many stalls for dual issue?
- Full bypassing?
  - Not easy…

Control Hazards

Branch → *i*

3 Branch Delay Slots: *i+1*, *i+2*, *i+3*

	Multiple Issue Clock Cycle									
	1	2	3	4	5	6	7	8	9	10
i	IF	ID	EX	M	WB					
i+1	IF	ID	EX	M	WB					
i+2		IF	ID	EX	M	WB				
i+3		IF	ID	EX	M	WB				
i+4			IF	ID	EX	M	WB			
i+5			IF	ID	EX	M	WB			

- Branch stalls bubbles are compounded in n-way machines

Example: Pipeline Problem

```
IF1   First part of instruction fetch (TLB access)
IF2   Instruction fetch completes (I-cache accessed)
RF    Instruction decoded and register file read
EX    Perform Operation; compute memory address
      (base+displacement); compute branch target
      address; compute branch condition
M1    First part of memory access (TLB access)
M2    Memory access completes (D-cache accessed)
WB    Write back results into register file
```

- How many read/write ports needed?

Pipeline Problem Cont.

	Single Issue Clock Cycle									
	1	2	3	4	5	6	7	8	9	10
i	IF1	IF2	ID	EX	M1	M2	WB			
i+1		IF1	IF2	ID	EX	M1	M2	WB		
i+2			IF1	IF2	ID	EX	M1	M2	WB	
i+3				IF1	IF2	ID	EX	M1	M2	WB
i+4					IF1	IF2	ID	EX	M1	M2
i+5						IF1	IF2	ID	EX	M1

- What is the branch delay?
- What is the load delay?
- How many adders are needed to prevent structural hazards?
- How many destination RegIDs and comparators are needed for forwarding?

Pipeline Problem Cont.

	Multiple Issue Clock Cycle									
	1	2	3	4	5	6	7	8	9	10
i	IF1	IF2	ID	EX	M1	M2	WB			
i+1	IF1	IF2	ID	EX	M1	M2	WB			
i+2		IF1	IF2	ID	EX	M1	M2	WB		
i+3		IF1	IF2	ID	EX	M1	M2	WB		
i+4			IF1	IF2	ID	EX	M1	M2	WB	
i+5			IF1	IF2	ID	EX	M1	M2	WB	
i+6				IF1	IF2	ID	EX	M1	M2	WB
i+7				IF1	IF2	ID	EX	M1	M2	WB
i+8					IF1	IF2	ID	EX	M1	M2
i+9					IF1	IF2	ID	EX	M1	M2

Branch Delay? Load Delay? Forwarding IDs? Read/Write ports?

Putting things together

- Talked about these things independently…
- Instruction Fetch
  - Branch Prediction (fill scheduler with instruction + multiple instruction per cycle)
- Scheduling/Hazard elimination
  - Dynamic Scheduling with Tomasulo (RAW Hazards)
  - Register Renaming (WAR and WAW Hazards)
- Multiple functional units, register file ports
  - Potentially can reduce CPI < 1
- Speculative Execution
- Precise Interrupts
- Memory systems (later this semester)

Focus on Speculation/Interrupts

- Precise Interrupts
  - All instructions before interrupt must complete
  - All instructions after interrupt must seem to never start
- Speculation (similar problem!)
  - If branch prediction is wrong, could update state incorrectly leading to wrong program behavior
- Out-of-Order *completion*
  - Post-interrupt/mispredict writebacks change state
  - Does Out-of-Order scheduling require this?

Solving both problems with one solution

- Need the ability to squash/restart *any* instruction
  - Gives us precise state
    - Need for memory ops (page faults, etc)
    - Need for FP ops (divide by 0)
  - Gives us ability to recover mis-speculations
    - Need for branches
- Providing precise state solves both these problems

How to get precise state?

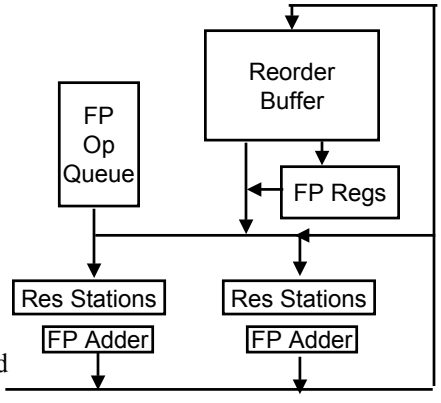
- Imprecise state
  - As we've said this is a bad idea
  - For speculation it is unacceptable
- Force in-order completion at WB (stall when necessary)
- Precise state in software: save recovery info for traps
  - Traps on all faulting memory, FP, and mis-predicted branch ops?
- Precise state in hardware: save recovery info online

Solution: Writeback and Commit

- Allow out of order issue/writeback
  - Require in-order commit when instruction is no longer speculative
  - Prevent speculative changes from changing state
    - e.g. memory write or register write
- Collect pre-commit instructions
  - in a reorder buffer
    - holds completed but not committed instruction
  - Effectively contains a set of virtual registers
    - similar to a reservation station
    - and becomes a bypass (forwarding) source

Reorder Buffer: HW buffer for results of uncommitted instructions

- 3 fields: instr, destination, value
- Reorder buffer can be operand source => more registers like RS
- Use reorder buffer number instead of reservation station when execution completes
- Supplies operands between execution complete & commit
- Once operand commits, result is put into register
- Instructions commit
- As a result, its easy to undo speculated instructions on mispredicted branches or on exceptions



Four Steps of Speculative Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue

   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")

2. Execution—operate on operands (EX)

   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")

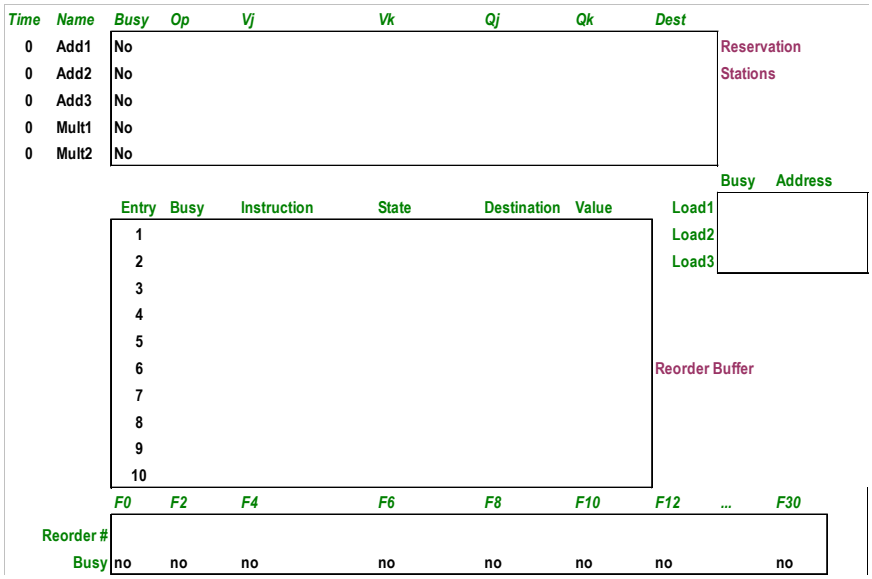
3. Write result—finish execution (WB)

   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available. (tags are now ROB #s not RS #s)

4. Commit—update register with reorder result

   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

Tomasulo With Reorder Buffer - Cycle 0



Tomasulo With Reorder Buffer - Cycle 1

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Stations

	Busy	Address
Load1	Yes	34+Regs[R2]
Load2		
Load3		

Entry	Busy	Instruction	State	Destination	Value
1	Yes	LD F6, 34(R2)	Issue	F6	
2					
3					
4					
5					
6					
7					
8					
9					
10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#1					
Busy	no	no	no	Yes	no	no	no		no

Tomasulo With Reorder Buffer - Cycle 2

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	No						

Reservation Stations

	Busy	Address
Load1	Yes	34+Regs[R2]
Load2	Yes	45+Regs[R3]
Load3		

	Entry	Busy	Instruction	State	Destination	Value
head →	1	Yes	LD F6, 34(R2)	Ex1	F6	
tail →	2	Yes	LD F2, 45(R3)	Issue	F2	
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #		#2		#1					
Busy	no	Yes	no	Yes	no	no	no		no

Tomasulo With Reorder Buffer - Cycle 3

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult		Regs[F4]	#2		#3
0	Mult2	No						

Reservation Stations

	Busy	Address
Load1	No	
Load2	Yes	45+Regs[R3]
Load3		

	Entry	Busy	Instruction	State	Destination	Value
head →	1	Yes	LD F6, 34(R2)	write	F6	Mem[load1]
	2	Yes	LD F2, 45(R3)	Ex1	F2	
tail →	3	Yes	MULT F0, F2, F4	Issue	F0	
	4					
	5					
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3	#2		#1					
Busy	Yes	Yes	no	Yes	no	no	no		no

Tomasulo With Reorder Buffer - Cycle 4

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+Regs[R3]]			#4
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	No						

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
head →	2	Yes	LD F2, 45(R3)	write	F2	Mem[load2]
	3	Yes	MULT F0, F2, F4	EX1	F0	
tail →	4	Yes	SUBD F8, F6, F2	Issue	F8	
	5					
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3	#2			#4				
Busy	Yes	Yes	no	no	Yes	no	no		no

Tomasulo With Reorder Buffer - Cycle 5

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+Regs[R3]]			#4
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex2	F0	
	4	Yes	SUBD F8, F6, F2	Ex1	F8	
tail →	5	Yes	DIVD F10, F0, F6	Issue	F10	
	6					
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3				#4	#5			
Busy	Yes	no	no	no	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 6

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	Yes	SUB	Regs[F6]	Mem[45+Regs[R3]]			#4
0	Add2	Yes	Add		Regs[F2]	#4		#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex3	F0	
	4	Yes	SUBD F8, F6, F2	Ex2	F8	
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	Issue	F6	
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 7

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	
0	Add1	No							Reservation
0	Add2	Yes	Add	#4	Regs[F2]			#6	Stations
0	Add3	No							
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3	
0	Mult2	Yes	DIV		Regs[F6]	#3		#5	

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value	
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]	Load1
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]	Load2
head →	3	Yes	MULT F0, F2, F4	Ex4	F0		Load3
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2	
	5	Yes	DIVD F10, F0, F6	Issue	F10		
tail →	6	Yes	ADDD F6, F8, F2	EX1	F6		Reorder Buffer
	7						
	8						
	9						
	10						

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 8

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	
0	Add1	No							Reservation
0	Add2	Yes	Add	#4	Regs[F2]			#6	Stations
0	Add3	No							
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3	
0	Mult2	Yes	DIV		Regs[F6]	#3		#5	

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value	
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]	Load1
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]	Load2
head →	3	Yes	MULT F0, F2, F4	Ex5	F0		Load3
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2	
	5	Yes	DIVD F10, F0, F6	Issue	F10		
tail →	6	Yes	ADDD F6, F8, F2	Ex2	F6		Reorder Buffer
	7						
	8						
	9						
	10						

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 9

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	Yes	Add	#4	Regs[F2]			#6
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex6	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 10

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex7	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 11

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex8	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 12

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	Yes	Mult	Mem[45+Regs[R3]]	Regs[F4]			#3
0	Mult2	Yes	DIV		Regs[F6]	#3		#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	Ex9	F0	
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Issue	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 13

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
head →	3	Yes	MULT F0, F2, F4	write	F0	#2 x Regs[F4]
	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Ex1	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	#3			#6	#4	#5			
Busy	Yes	no	no	Yes	Yes	Yes	no		no

Figure 3.30

P 230

Tomasulo With Reorder Buffer - Cycle 14

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest
0	Add1	No						
0	Add2	No						
0	Add3	No						
0	Mult1	No						
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5

Reservation Stations

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]
head →	4	Yes	SUBD F8, F6, F2	write	F8	F6 - #2
	5	Yes	DIVD F10, F0, F6	Ex2	F10	
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2
	7					
	8					
	9					
	10					

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#6	#4	#5			
Busy	No	no	no	Yes	Yes	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 15

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	
0	Add1	No							Reservation
0	Add2	No							Stations
0	Add3	No							
0	Mult1	No							
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5	

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value	
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]	
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]	
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]	
	4	No	SUBD F8, F6, F2	commit	F8	F6 - #2	
head →	5	Yes	DIVD F10, F0, F6	Ex3	F10		
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2	Reorder Buffer
	7						
	8						
	9						
	10						

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#6		#5			
Busy	no	no	no	Yes	no	Yes	no		no

Tomasulo With Reorder Buffer - Cycle 16

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	
0	Add1	No							Reservation
0	Add2	No							Stations
0	Add3	No							
0	Mult1	No							
0	Mult2	Yes	DIV	#2xRegs[F4]	Regs[F6]			#5	

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value	
	1	No	LD F6, 34(R2)	commit	F6	Mem[load1]	
	2	No	LD F2, 45(R3)	commit	F2	Mem[load2]	
	3	No	MULT F0, F2, F4	commit	F0	#2 x Regs[F4]	
	4	No	SUBD F8, F6, F2	commit	F8	F6 - #2	
head →	5	Yes	DIVD F10, F0, F6	Ex4	F10		
tail →	6	Yes	ADDD F6, F8, F2	write	F6	#4 + F2	Reorder Buffer
	7						
	8						
	9						
	10						

Need 36 more EX cycles for DIV to finish…

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #				#6		#5			
Busy	no	no	no	Yes	no	Yes	no		no

Tomasulo With Reorder Buffer: Summary

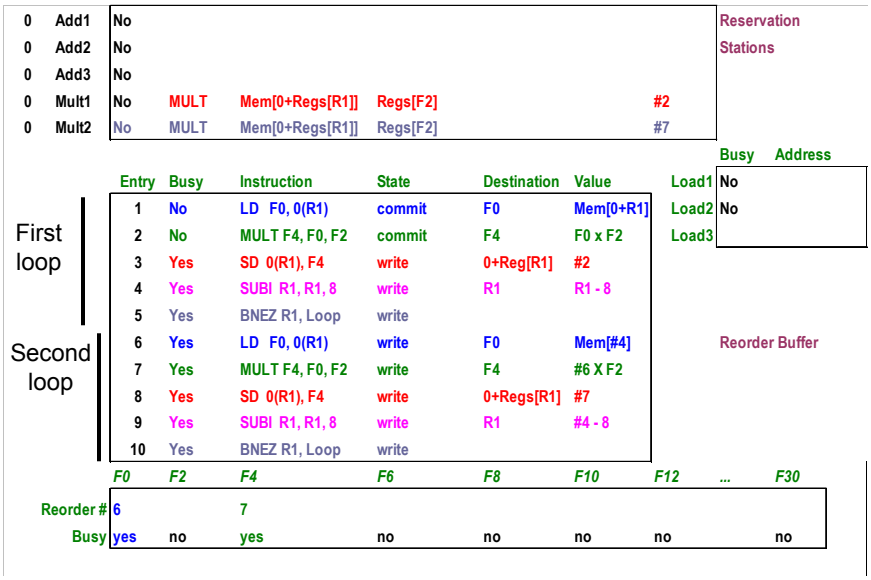
Instruction	Issue	Exec Comp	Writeback	Commit
LD F6, 34(R2)	1	2	3	4
LD F2, 45(R3)	2	3	4	5
MULT F0, F2, F4	3	12	13	14
SUBD F8, F6, F2	4	6	7	15
DIVD F10, F0, F6	5	52	53	54
ADDD F6, F8, F2	6	8	9	55

In-order Issue/Commit, Out-of-Order Execution/Writeback

Precise State with ROB

- ROB maintains precise state and allows speculation
  - Waits until precise condition reaches retire/commit stage
  - (Or until branch is noted mis-predicted)
  - Clear ROB, RS, and register status table (Flush)
  - Service exception/Restart from True Branch target
- Need to do similar things with memory ops
  - Called Memory Ordering Buffer (MOB)
    - Completed stores write to MOB then complete (write to memory) in-order (when they read head of buffer)

Example of Speculative State of Reorder Buffer



0	Add1	No					Reservation Stations
0	Add2	No					
0	Add3	No					
0	Mult1	No	MULT	Mem[0+Regs[R1]]	Regs[F2]	#2	
0	Mult2	No	MULT	Mem[0+Regs[R1]]	Regs[F2]	#7	

	Busy	Address
Load1	No	
Load2	No	
Load3		

	Entry	Busy	Instruction	State	Destination	Value
First loop	1	No	LD F0, 0(R1)	commit	F0	Mem[0+R1]
	2	No	MULT F4, F0, F2	commit	F4	F0 x F2
	3	Yes	SD 0(R1), F4	write	0+Reg[R1]	#2
	4	Yes	SUBI R1, R1, 8	write	R1	R1 - 8
	5	Yes	BNEZ R1, Loop	write		
Second loop	6	Yes	LD F0, 0(R1)	write	F0	Mem[#4]
	7	Yes	MULT F4, F0, F2	write	F4	#6 X F2
	8	Yes	SD 0(R1), F4	write	0+Regs[R1]	#7
	9	Yes	SUBI R1, R1, 8	write	R1	#4 - 8
	10	Yes	BNEZ R1, Loop	write		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	F12	...	F30
Reorder #	6		7						
Busy	yes	no	yes	no	no	no	no		no

Multiply has just reached commit, so other instructions can start committing

Tomasulo + ROB Summary

- Many implementations are very similar
  - Pentium III, PowerPC, etc
- Some limitations
  - Too many value copy operations
    - Register file => RS => ROB => Register File
  - Too many muxes/busses (CDB)
    - Values are coming from everywhere to everywhere else!
  - Reservation Stations mix values(data) and tags(control)
    - Slows down the max clock frequency

For next time

- Case Studies (P6, Pentium 4, MIPS R10K)
- Limits of ILP