CS252
Graduate Computer Architecture
Lecture 7

Dynamic Scheduling 2:
Precise Interrupts
February 9th, 2010

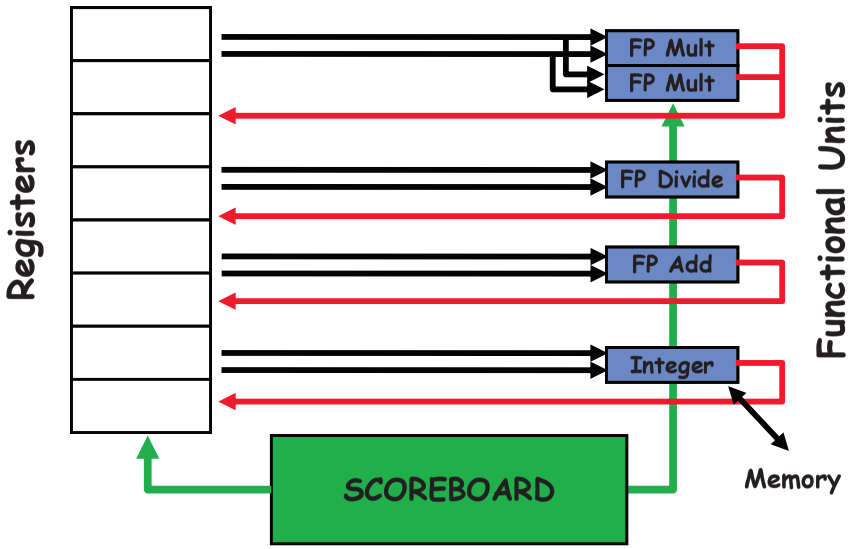
John Kubiatowicz

Electrical Engineering and Computer Sciences

University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

Review: Scoreboard (CDC 6600)



Review: Four Stages of Scoreboard Control

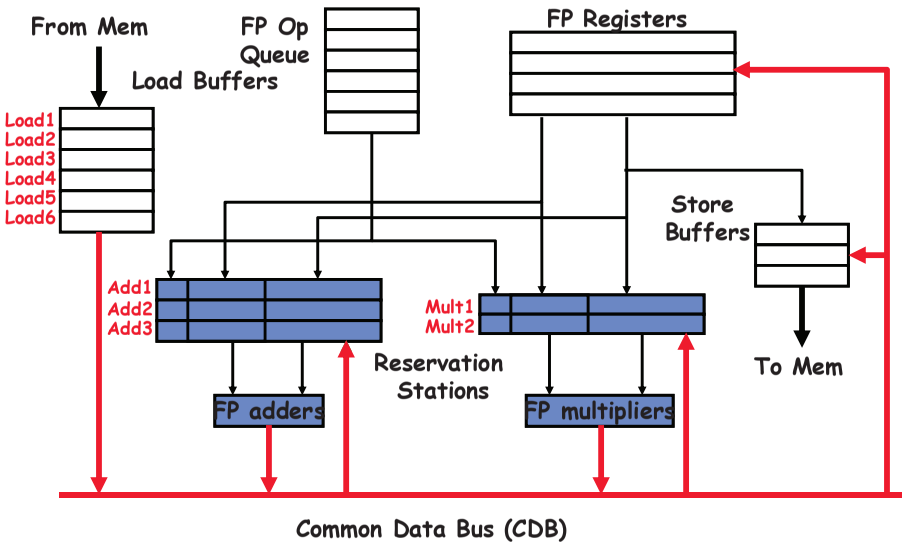
- **Issue**—decode instructions & check for structural hazards
  - Instructions issued in program order (for hazard checking)
  - Don't issue if structural hazard
  - Don't issue if instruction is output dependent on any previously issued but uncompleted instruction (no WAW hazards)
- **Read operands**—wait operands ready, then read them
  - All real dependencies (RAW hazards) resolved in this stage
  - No forwarding of data in this model!
- **Execution**—operate on operands
  - The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.
- **Write result**—finish execution
  - Stall if WAR hazards with previous instructions:

Example:

```
DIVD   F0,F2,F4
ADDD   F10,F0,F8
SUBD   F8,F8,F14
```

CDC 6600 scoreboard would stall SUBD until ADDD reads operands

Review: Tomasulo Organization



Review: Three Stages of Tomasulo Algorithm

1. **Issue**—get instruction from FP Op Queue
   - If reservation station free (no structural hazard), control issues instr & sends operands (renames registers).
2. **Execution**—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch Common Data Bus for result
3. **Write result**—finish execution (WB)
   - Write on Common Data Bus to all awaiting units; mark reservation station available

- Normal data bus: data + destination ("go to" bus)
- Common data bus: data + source ("come from" bus)
  - 64 bits of data + 4 bits of Functional Unit source address
  - Write if matches expected Functional Unit (produces result)
  - Does the broadcast

Review: Compare to Scoreboard Cycle 62

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Issue	Exec Comp	Write Result
1	3	4
2	4	5
3	15	16
4	7	8
5	56	57
6	10	11

- Why take longer on scoreboard/6600?
  - Structural Hazards
  - Lack of forwarding

Review: Loop Example Cycle 9

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6		
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Code
	Add1	No						LD F0 0 R1
	Add2	No						MULTD F4 F0 F2
	Add3	No						SD F4 0 R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI R1 R1 #8
	Mult2	Yes	Multd		R(F2)	Load2		BNEZ R1 Loop

Register result status

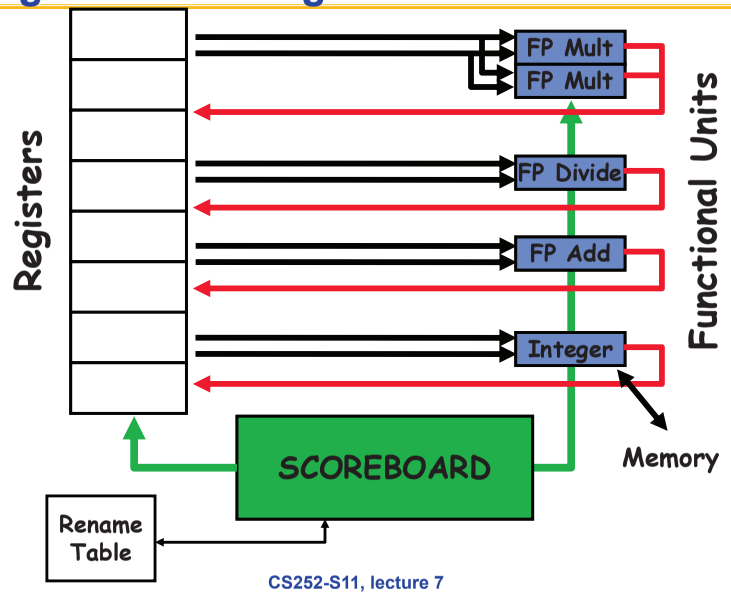
Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
9	72	Fu	Load2		Mult2						

- Dataflow graph constructed completely in hardware
- Renaming detaches early iterations from registers

Explicit Register Renaming

- Tomasulo provides *Implicit Register Renaming*
  - User registers renamed to reservation station tags
- Explicit Register Renaming:
  - Use *physical* register file that is larger than number of registers specified by ISA
- Keep a translation table:
  - ISA register => physical register mapping
  - When register is written, replace table entry with new register from freelist.
  - Physical register becomes free when not being used by any instructions in progress.
- Pipeline can be exactly like "standard" DLX pipeline
  - IF, ID, EX, etc….
- Advantages:
  - Removes all WAR and WAW hazards
  - Like Tomasulo, good for allowing full out-of-order completion
  - Allows data to be fetched from a single register file
  - Makes speculative execution/precise interrupts easier:
    - » All that needs to be "undone" for precise break point is to undo the table mappings

Question: Can we use explicit register renaming with scoreboard?



Scoreboard Example

Instruction status:

Instruction	j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+ R2				
LD	F2	45+ R3				
MULTD	F0	F2 F4				
SUBD	F8	F6 F2				
DIVD	F10	F0 F6				
ADDD	F6	F8 F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	P0	P2	P4	P6	P8	P10	P12		P30

- Initialized Rename Table

Renamed Scoreboard 1

Instruction status:

Instruction	j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+ R2	1			
LD	F2	45+ R3				
MULTD	F0	F2 F4				
SUBD	F8	F6 F2				
DIVD	F10	F0 F6				
ADDD	F6	F8 F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
1 FU	P0	P2	P4	P32	P8	P10	P12		P30

- Each instruction allocates free register
- Similar to single-assignment compiler transformation

Renamed Scoreboard 2

Instruction status:

Instruction	j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+ R2	1	2		
LD	F2	45+ R3	2			
MULTD	F0	F2 F4				
SUBD	F8	F6 F2				
DIVD	F10	F0 F6				
ADDD	F6	F8 F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
2 FU	P0	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 3

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3	2	3		
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	P36	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 4

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	Yes	Sub	P38	P32	P34		Int2	Yes	No
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	P36	P34	P4	P32	P38	P10	P12		P30

Renamed Scoreboard 5

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 6

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
10	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 7

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
9	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 8

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
8	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 9

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
7	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 10

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10			

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
6	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Addd	P42	P38	P4			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

WAR Hazard gone!

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	P36	P34	P4	P42	P38	P40	P12		P30

- Notice that P32 not listed in Rename Table
  - Still live. Must not be reallocated by accident

Renamed Scoreboard 11

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
5	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 12

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
4	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 13

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
3	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 14

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
2	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 15

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
1	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 16

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
0	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 17

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
17	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 18

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5	18		
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
40	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
18	FU	P36	P34	P4	P42	P38	P40	P12		P30

Explicit Renaming Support Includes:

- **Rapid access to a table of translations**
- **A physical register file that has more registers than specified by the ISA**
- **Ability to figure out which physical registers are free.**
  - **No free registers ⇒ stall on issue**
- **Thus, register renaming doesn't require reservation stations. However:**
  - **Many modern architectures use explicit register renaming + Tomasulo-like reservation stations to control execution.**

How many instructions can be in the pipeline?

Which features of an ISA limit the number of instructions in the pipeline?

Number of Registers

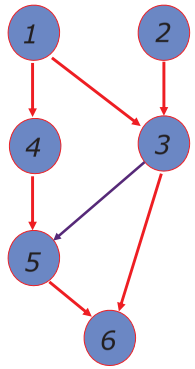
Which features of a program limit the number of instructions in the pipeline?

Control transfers

Out-of-order dispatch by itself does not provide any significant performance improvement !

How important is renaming? Consider execution without it

					latency
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		long
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4,	F2,	F8	4
6	ADDD	F10,	F6,	F4	1

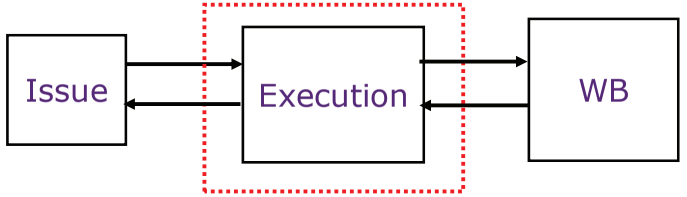


In-order: 1 (2,1) . . . . . 2 3 4 4 3 5 . . . 5 6 6
Out-of-order: 1 (2,1) 4 4 . . . . 2 3 . . 3 5 . . . 5 6 6

Out-of-order execution did not allow any significant improvement!

Little's Law

Throughput (T) = Number in Flight (N) / Latency (L)



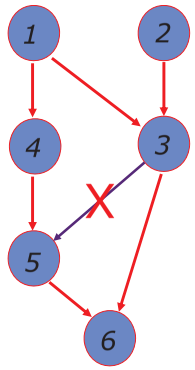
Example:

- *4 floating point registers*
- *8 cycles per floating point operation*

⇒ *maximum of ½ issue per cycle without renaming!*

Instruction-level Parallelism via Renaming

					latency
1	LD	F2,	34(R2)		1
2	LD	F4,	45(R3)		long
3	MULTD	F6,	F4,	F2	3
4	SUBD	F8,	F2,	F2	1
5	DIVD	F4',	F2,	F8	4
6	ADDD	F10,	F6,	F4'	1



In-order: 1 (2,1) . . . . . 2 3 4 4 3 5 . . . 5 6 6
Out-of-order: 1 (2,1) 4 4 5 . . . 2 (3,5) 3 6 6

Any antidependence can be eliminated by renaming.
(renaming ⇒ additional storage)
Can be done either in Software or Hardware

Administrative

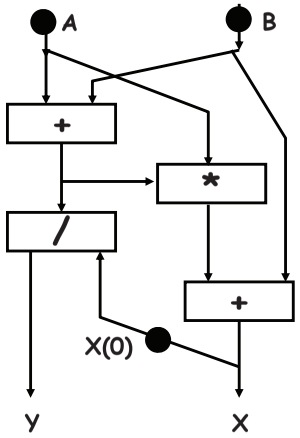
- **Midterm I: Wednesday 3/14**
  Location: 320 Soda Hall
  TIME: 2:30-5:30
  - **Can have 1 sheet of 8½x11 *handwritten* notes – both sides**
  - **No microfiche of the book!**
- **This info is on the Lecture page**
- **Meet at LaVal's afterwards for Pizza and Beverages**
  - **Great way for me to get to know you better**
  - **I'll Buy!**
- **Don't quite have new set of papers up. Will do that by tomorrow at the latest….**

Data-Flow Architectures

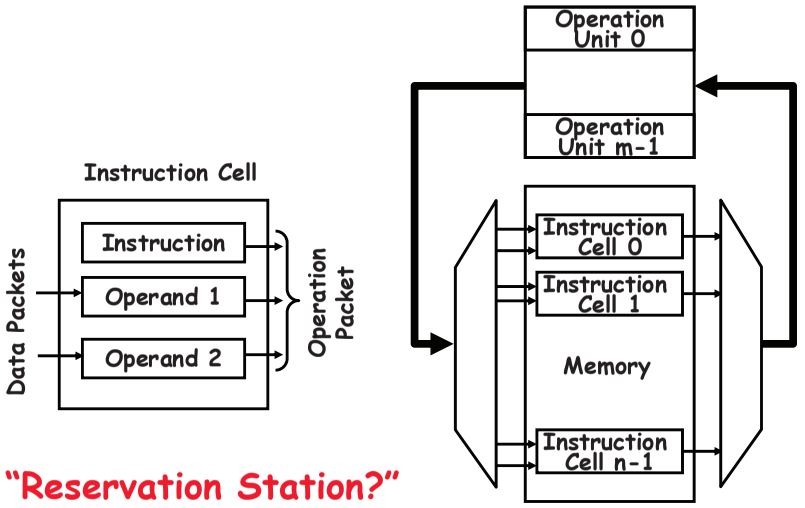
- **Basic Idea: Hardware respresents direct encoding of compiler dataflow graphs:**

```
Input: a,b
   y:= (a+b)/x
   x:= (a*(a+b))+b
output: y,x
```

- **Data flows along arcs in "Tokens".**
- **When two tokens arrive at compute box, box "fires" and produces new token.**
- **Split operations produce copies of tokens**



Paper by Dennis and Misunas

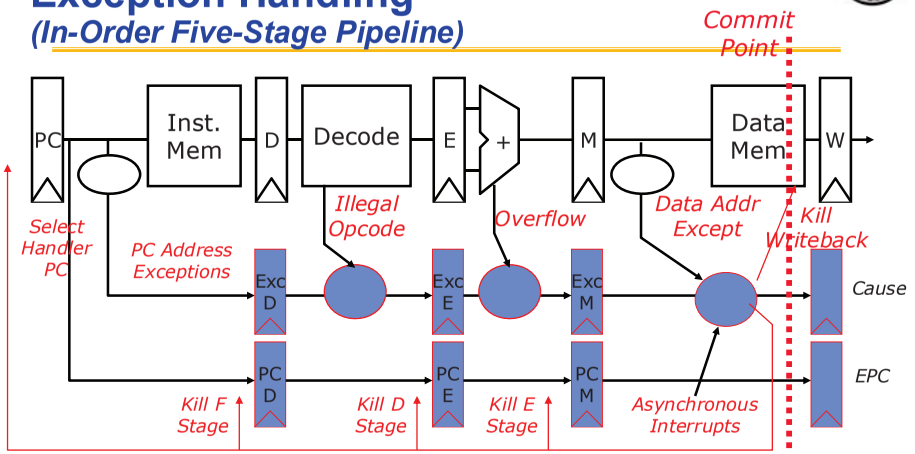


"Reservation Station?"

What about Precise Exceptions/Interrupts?

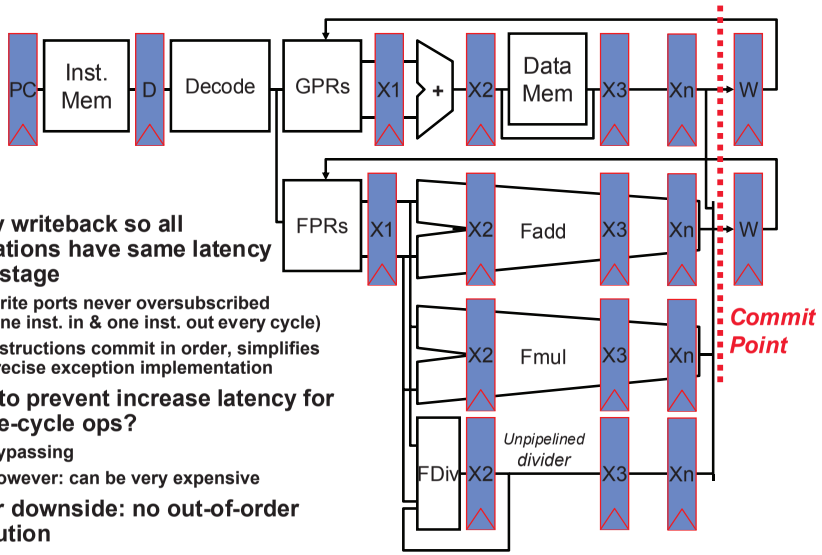
- Both Scoreboard and Tomasulo have:
  - *In-order issue, out-of-order execution, out-of-order completion*
- **Recall:** An interrupt or exception is *precise* if there is a single instruction for which:
  - All instructions before that have committed their state
  - No following instructions (including the interrupting instruction) have modified any state.
- **Need way to resynchronize execution with instruction stream (I.e. with issue-order)**
  - Easiest way is with *in-order completion (i.e. reorder buffer)*
  - Other Techniques (Smith paper): Future File, History Buffer

Exception Handling
(In-Order Five-Stage Pipeline)



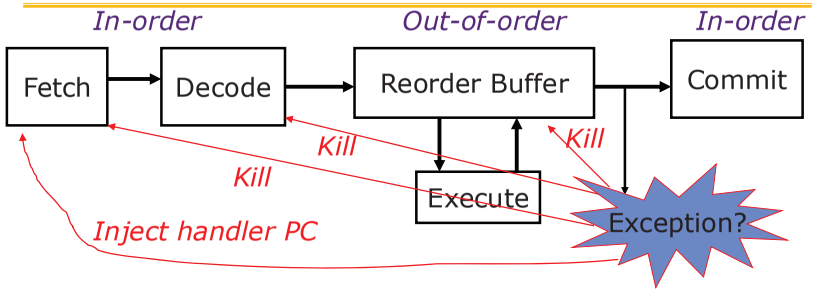
- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions
- Inject external interrupts at commit point (override others)
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

Complex In-Order Pipeline: Precise Exceptions



- Delay writeback so all operations have same latency to W stage
  - Write ports never oversubscribed (one inst. in & one inst. out every cycle)
  - Instructions commit in order, simplifies precise exception implementation
- How to prevent increase latency for single-cycle ops?
  - Bypassing
  - However: can be very expensive
- Other downside: no out-of-order execution

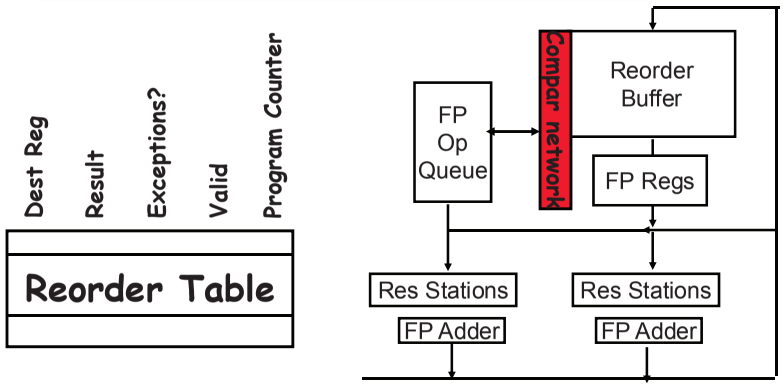
In-Order Commit for Precise Exceptions



- Instructions fetched and decoded into instruction reorder buffer in-order
- Execution is out-of-order (⇒ out-of-order completion)
- *Commit* (write-back to architectural state, i.e., regfile & memory) is in-order

Temporary storage needed to hold results before commit (shadow registers and store buffers)

What are the hardware complexities with reorder buffer (ROB)?



- How do you find the latest version of a register?
  - As specified by Smith paper, need associative comparison network
  - Could use future file or just use the register result status buffer to track which specific reorder buffer has received the value
- Need as many ports on ROB as register file

Four Steps of *Speculative* Tomasulo

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")

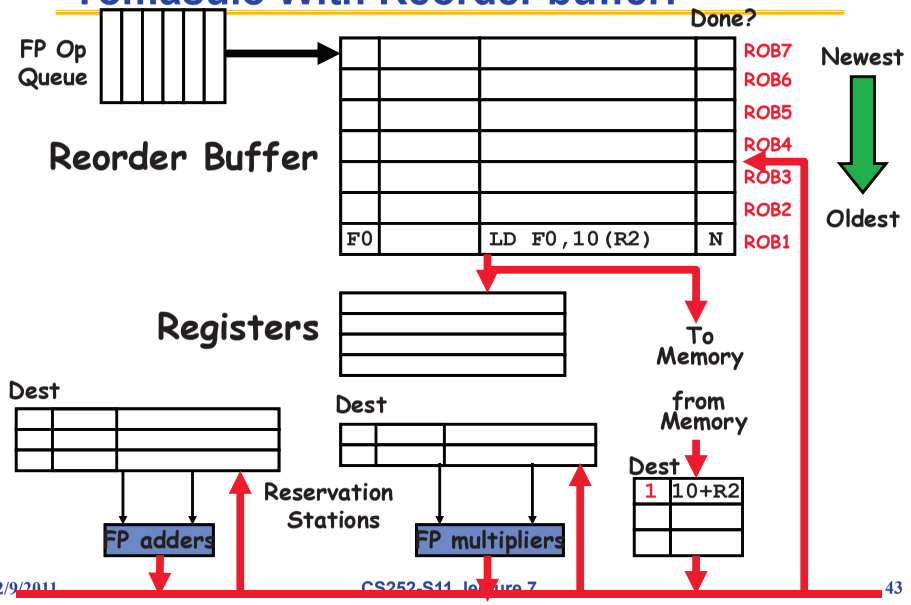
3. **Write result**—finish execution (WB)

   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

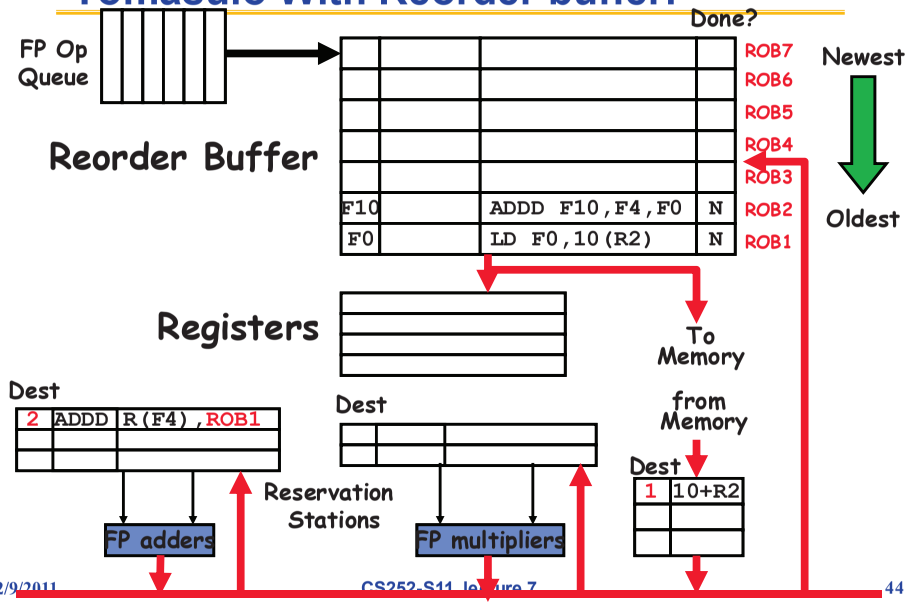
4. **Commit**—update register with reorder result

   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")

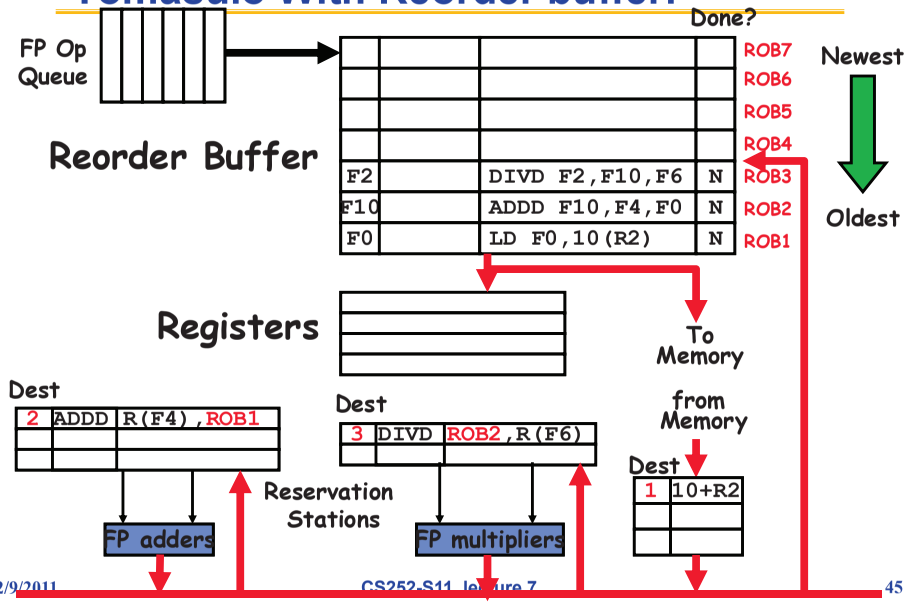
Tomasulo With Reorder buffer:



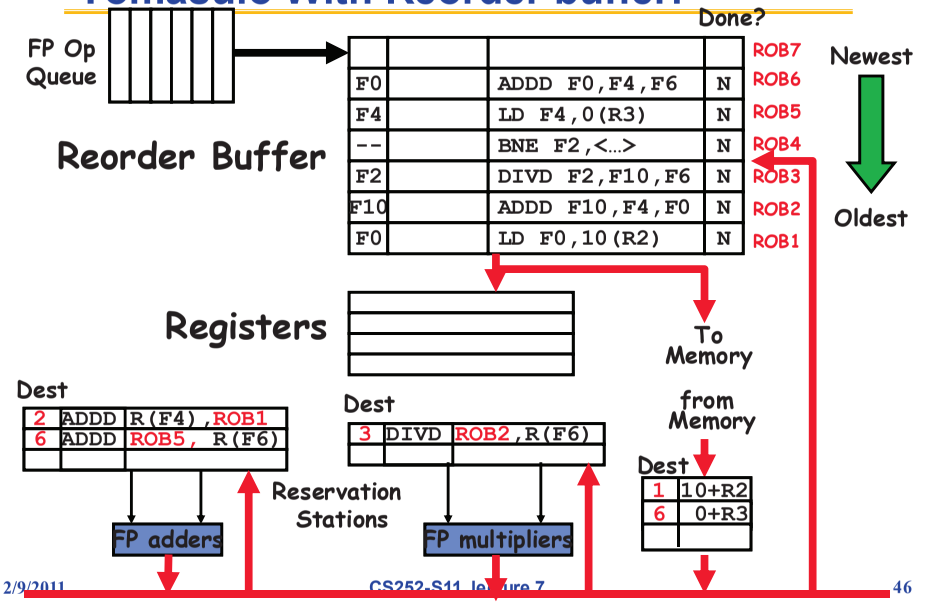
Tomasulo With Reorder buffer:



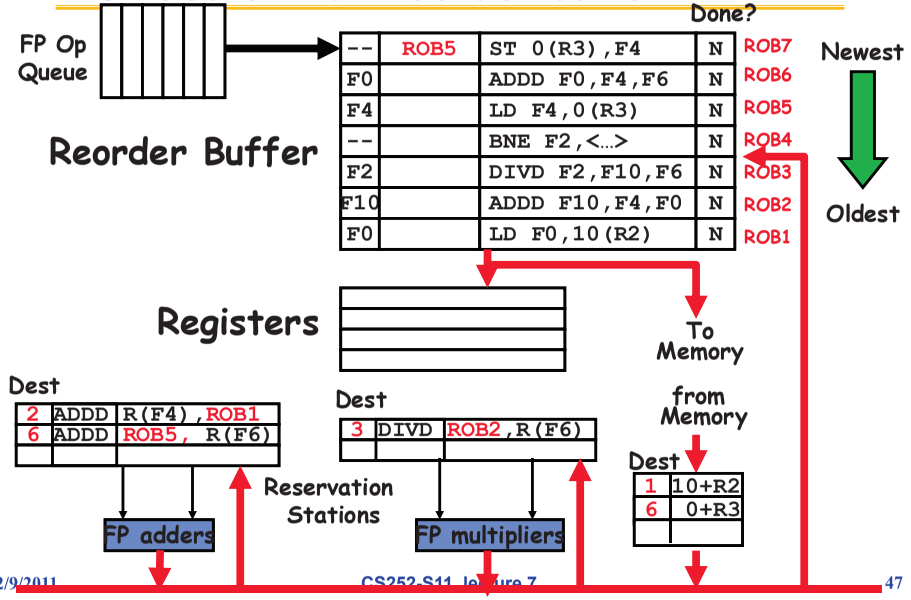
Tomasulo With Reorder buffer:



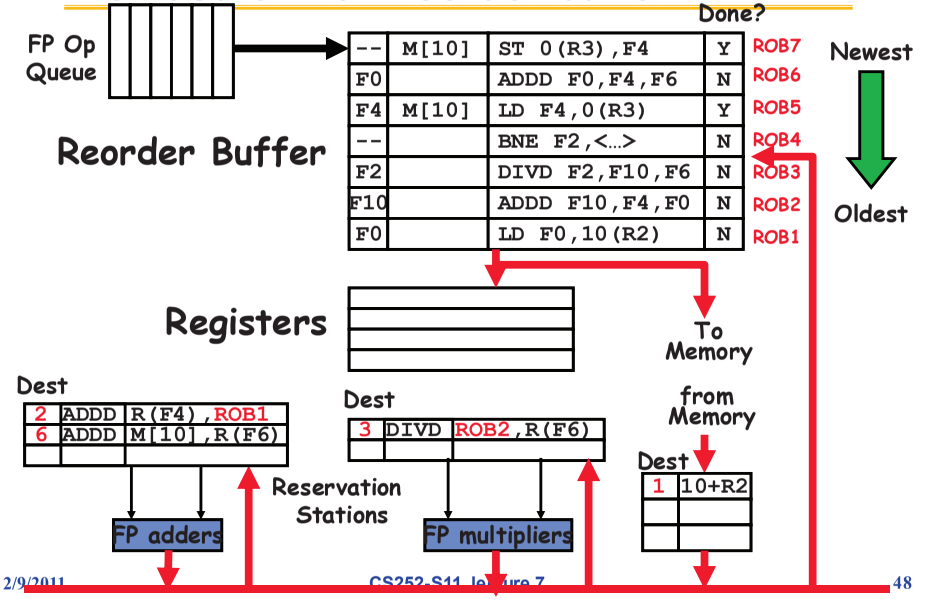
Tomasulo With Reorder buffer:



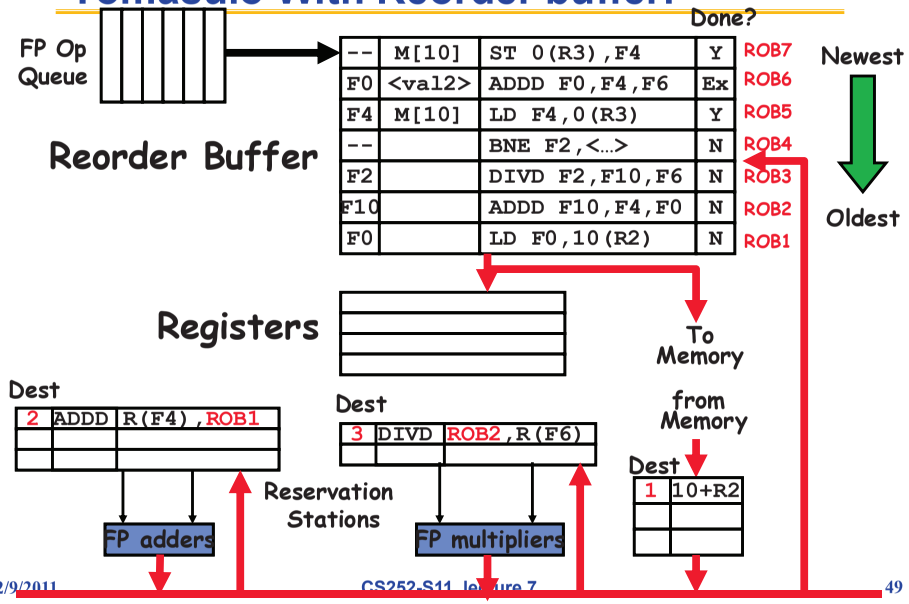
Tomasulo With Reorder buffer:



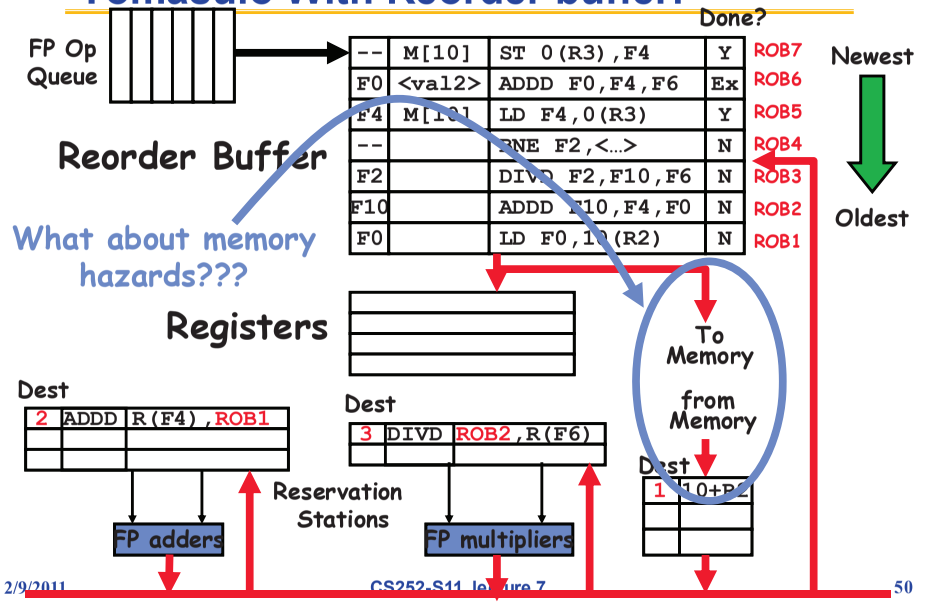
Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



Memory Disambiguation: Sorting out RAW Hazards in memory

- Question: Given a load that follows a store in program order, are the two related?
  - (Alternatively: is there a RAW hazard between the store and the load)?

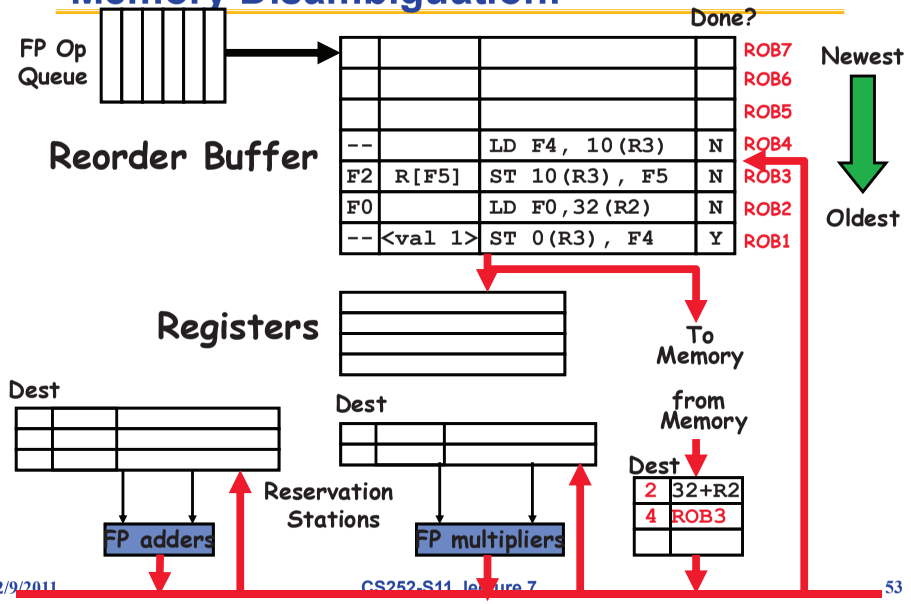
```
Eg:   st    0(R2),R5
      ld    R6,0(R3)
```

- Can we go ahead and start the load early?
  - Store address could be delayed for a long time by some calculation that leads to R2 (divide?).
  - We might want to issue/begin execution of both operations in same cycle.
  - Today: Answer is that we are not allowed to start load until we know that address 0(R2) ≠ 0(R3)
  - Next Week: We might guess at whether or not they are dependent (called "dependence speculation") and use reorder buffer to fixup if we are wrong.

Hardware Support for Memory Disambiguation

- Need buffer to keep track of all outstanding stores to memory, in program order.
  - Keep track of address (when becomes available) and value (when becomes available)
  - FIFO ordering: will retire stores from this buffer in program order
- When issuing a load, record current head of store queue (know which stores are ahead of you).
- When have address for load, check store queue:
  - If *any* store prior to load is waiting for its address, stall load.
  - If load address matches earlier store address (associative lookup), then we have a *memory-induced RAW hazard*:
    - » store value available ⇒ return value
    - » store value not available ⇒ return ROB number of source
  - Otherwise, send out request to memory
- Actual stores commit in order, so no worry about WAR/WAW hazards through memory.

Memory Disambiguation:

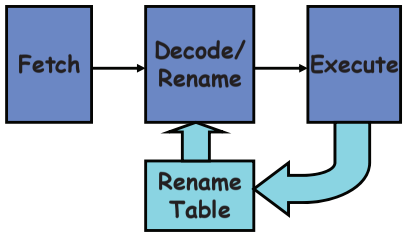


Relationship between precise interrupts and speculation:

- Speculation is a form of guessing
  - Branch prediction, data prediction
  - If we speculate and are wrong, need to back up and restart execution to point at which we predicted incorrectly
  - This is exactly same as precise exceptions!
- Branch prediction is a very important!
  - Need to "take our best shot" at predicting branch direction.
  - If we issue multiple instructions per cycle, lose lots of potential instructions otherwise:
    - » Consider 4 instructions per cycle
    - » If take single cycle to decide on branch, waste from 4 - 7 instruction slots!
- Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit*
  - This is why reorder buffers in all new processors

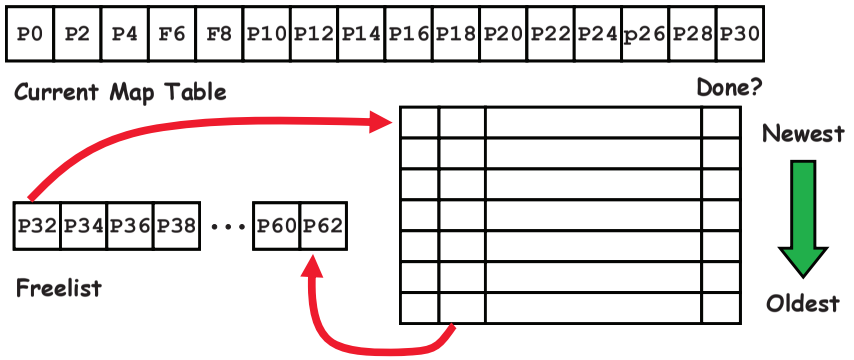
Quick Recap: Explicit Register Renaming

- Make use of a *physical* register file that is larger than number of registers specified by ISA
- Keep a translation table:
  - ISA register => physical register mapping
  - When register is written, replace table entry with new register from freelist.
  - Physical register becomes free when not being used by any instructions in progress.



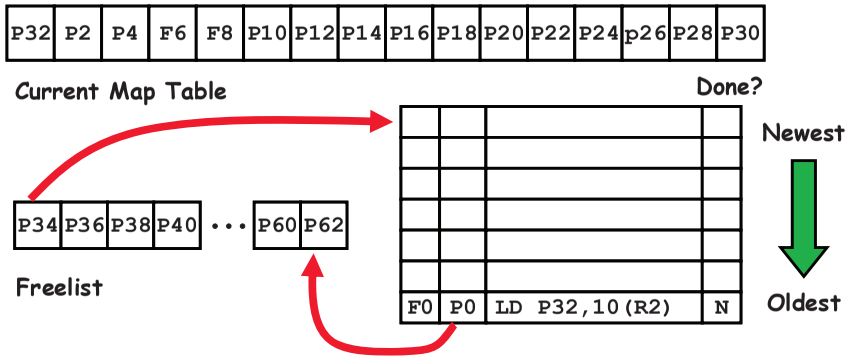
Explicit register renaming:

R10000 Freelist Management



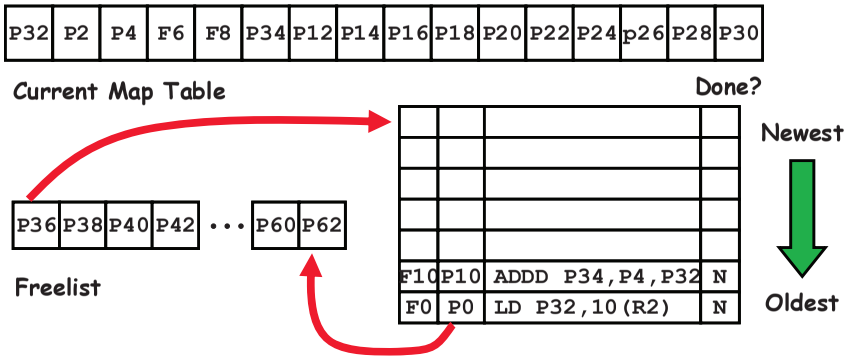
- Physical register file larger than ISA register file
- On issue, each instruction that modifies a register is allocated new physical register from freelist
- Used on: R10000, Alpha 21264, HP PA8000

Explicit register renaming:
R10000 Freelist Management

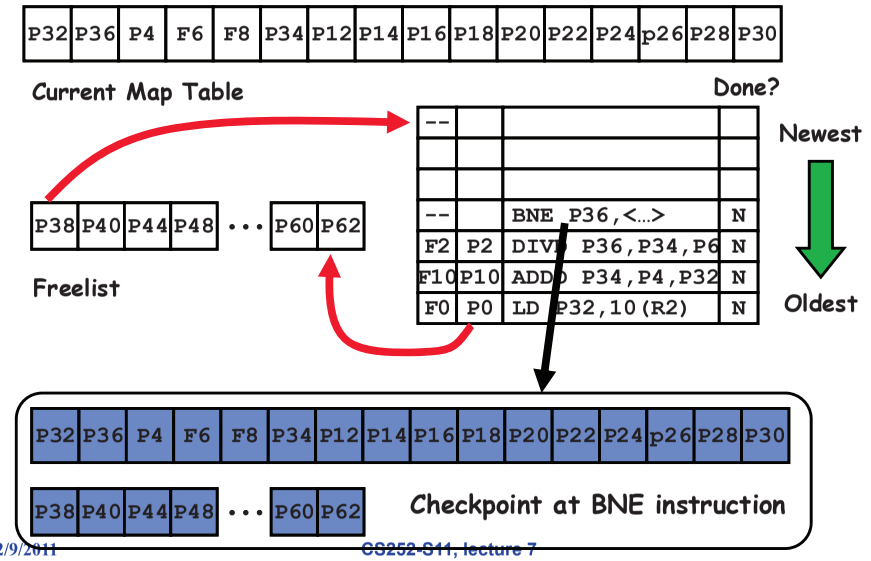


- Note that physical register P0 is "dead" (or not "live") past the point of this load.
  - When we go to commit the load, we free up

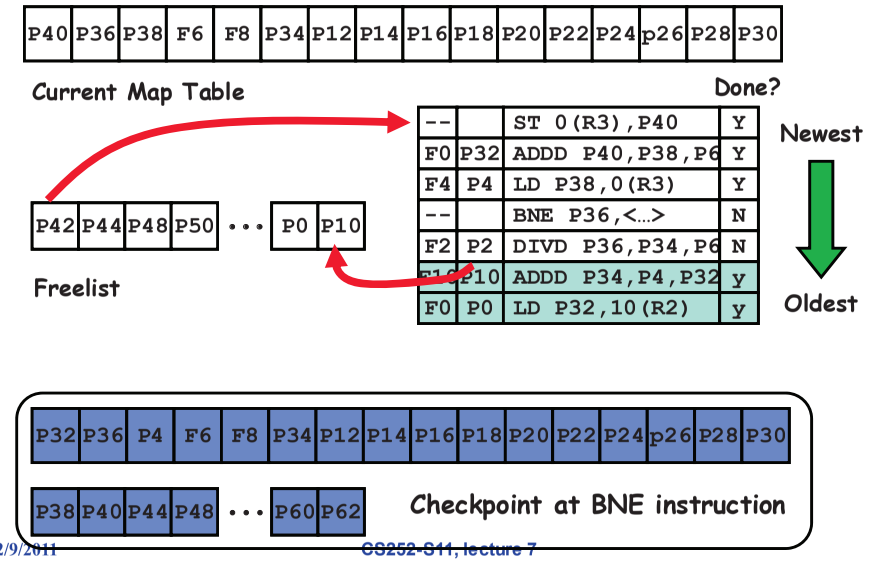
Explicit register renaming:
R10000 Freelist Management



Explicit register renaming:
R10000 Freelist Management

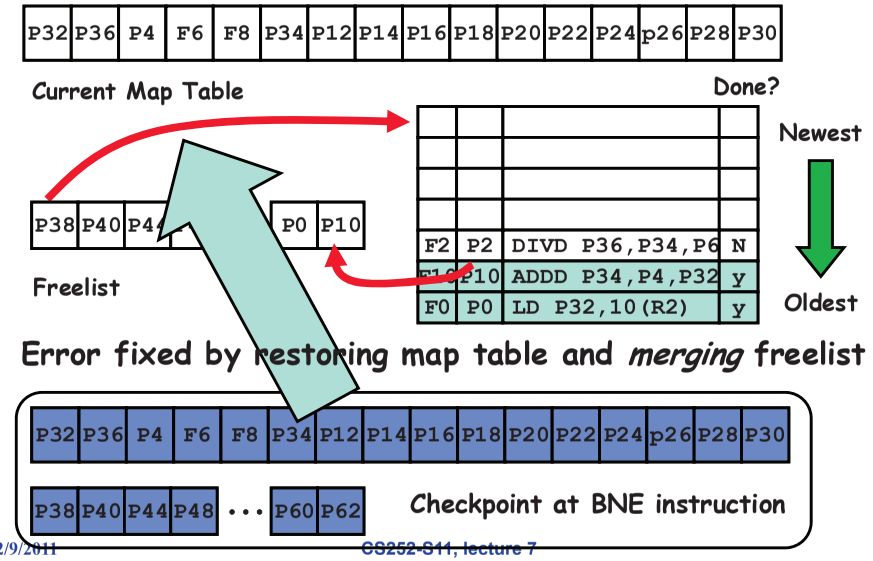


Explicit register renaming:
R10000 Freelist Management



Explicit register renaming:

R10000 Freelist Management

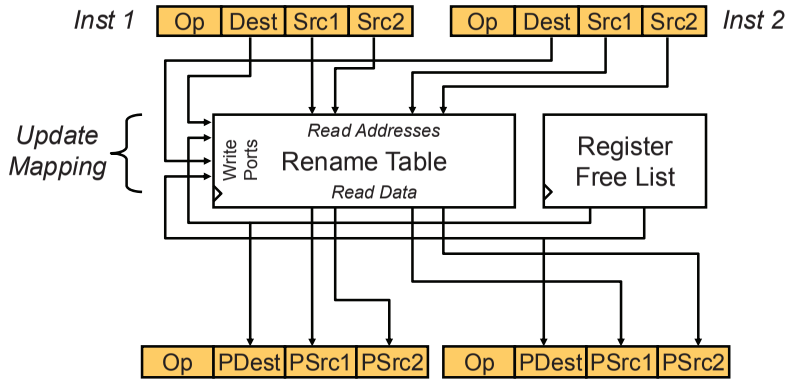


Advantages of Explicit Renaming

- Decouples *renaming* from *scheduling:*
  - Pipeline can be exactly like "standard" DLX pipeline (perhaps with multiple operations issued per cycle)
  - Or, pipeline could be tomasulo-like or a scoreboard, etc.
  - Standard forwarding or bypassing could be used
- Allows data to be fetched from single register file
  - No need to bypass values from reorder buffer
  - This can be important for balancing pipeline
- Many processors use a variant of this technique:
  - R10000, Alpha 21264, HP PA8000
- Another way to get precise interrupt points:
  - All that needs to be "undone" for precise break point is to undo the table mappings
  - Provides an interesting mix between reorder buffer and future file
    - » Results are written immediately back to register file
    - » Registers *names* are "freed" in program order (by ROB)

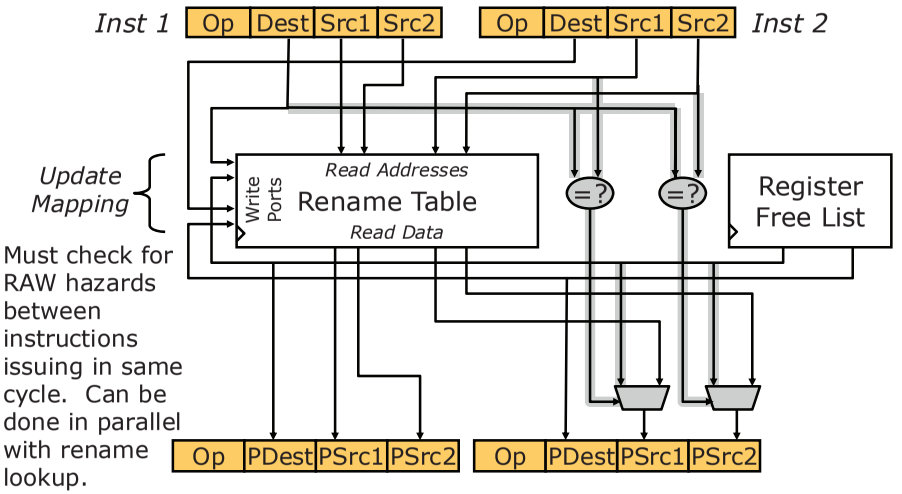
Superscalar Register Renaming

- During decode, instructions allocated new physical destination register
- Source operands renamed to physical register with newest value
- Execution unit only sees physical register numbers



Does this work?

Superscalar Register Renaming (Try #2)



Must check for RAW hazards between instructions issuing in same cycle. Can be done in parallel with rename lookup.

MIPS R10K renames 4 serially-RAW-dependent insts/cycle

Summary

- DataFlow view:
  - Data triggers execution rather than instructions triggering data
- Dynamic hardware schemes can unroll loops dynamically in hardware
  - Form of limited dataflow
  - Register renaming is essential
- Explicit Renaming: more physical registers than needed by ISA.
  - Rename table: tracks current association between architectural registers and physical registers
  - Uses a translation table to perform compiler-like transformation on the fly
- Precise Interrupts:
  - Must commit things back in order
  - Reorder buffer: temporarily holds results until commit possible
  - Toss out things to achieve precise interrupt point