Computer Architecture: Dynamic Scheduling: Tomasulo Hardware-Based Speculation

Berk Sunar and Thomas Eisenbarth

ECE 505

WPI

Outline

- 5 stages of RISC
- Type of hazards
- Static and Dynamic Branch Prediction
- Pipelining with Exceptions
- Pipelining with Floating-Point Operations
- Loop Unrolling
- Correlating and Tournament Branch Prediction
- Dynamic Scheduling: Scoreboard

- Dynamic Scheduling: Tomasulo: 3.5
- Hardware Based Speculation 3.6

- Multiple Issue

Last Time: Dynamic Scheduling w/ Scoreboard

+ Cheap in hardware
+ pretty good performance
- no bypassing
  - RAW dependencies handled through registers
- limited scheduling scope
  - WAW/structural hazards force in-order dispatch
  - WAR hazards delay writeback and issue of dependent operations

Problems can be solved with register renaming!

Tomasulo

- Scheme to allow out-of-order execution, invented by Robert Tomasulo for IBM 360/91 (1967)
  - Tracks when operands are available to minimize RAW hazards
  - Introduces **register renaming** in HW and thereby avoids WAW and WAR hazards
    - No more stalls due to WAW and WAR hazards
    - RAW hazards remain
  - Renaming provided by reservation stations:
    - more reservation stations than registers enable better naming dependence handling than can be done in SW

Hazards: Another Example and Tomasulo's solution

Example 2:

```
1      DIV.D   F0,F2,F4
2      ADD.D   F6,F0,F8
3      S.D     F6,0(R1)
4      SUB.D   F8,F10,F14
5      MUL.D   F6,F10,F8
```

- Antidependences: (WAR)
  - between 4 and 2, 5 and 3
- Output Dependence: (WAW)
  - between 2 and 5
- True data Dependences: (RAW)
  - Between 1 and 2, 2 and 3, and 4 and 5

After renaming:

```
1      DIV.D   F0,F2,F4
2      ADD.D   S,F0,F8
3      S.D     S,0(R1)
4      SUB.D   T,F10,F14
5      MUL.D   F6,F10,T
```

Resulting Hazards:

- WAR hazard on F8 by 2 and F6 by 3
- WAW hazard by F6 (5 might finish before 2)

- Name dependences removed by renaming
  - Requires free registers and subsequent renaming F8→T

Tomasulo based Processor: Example

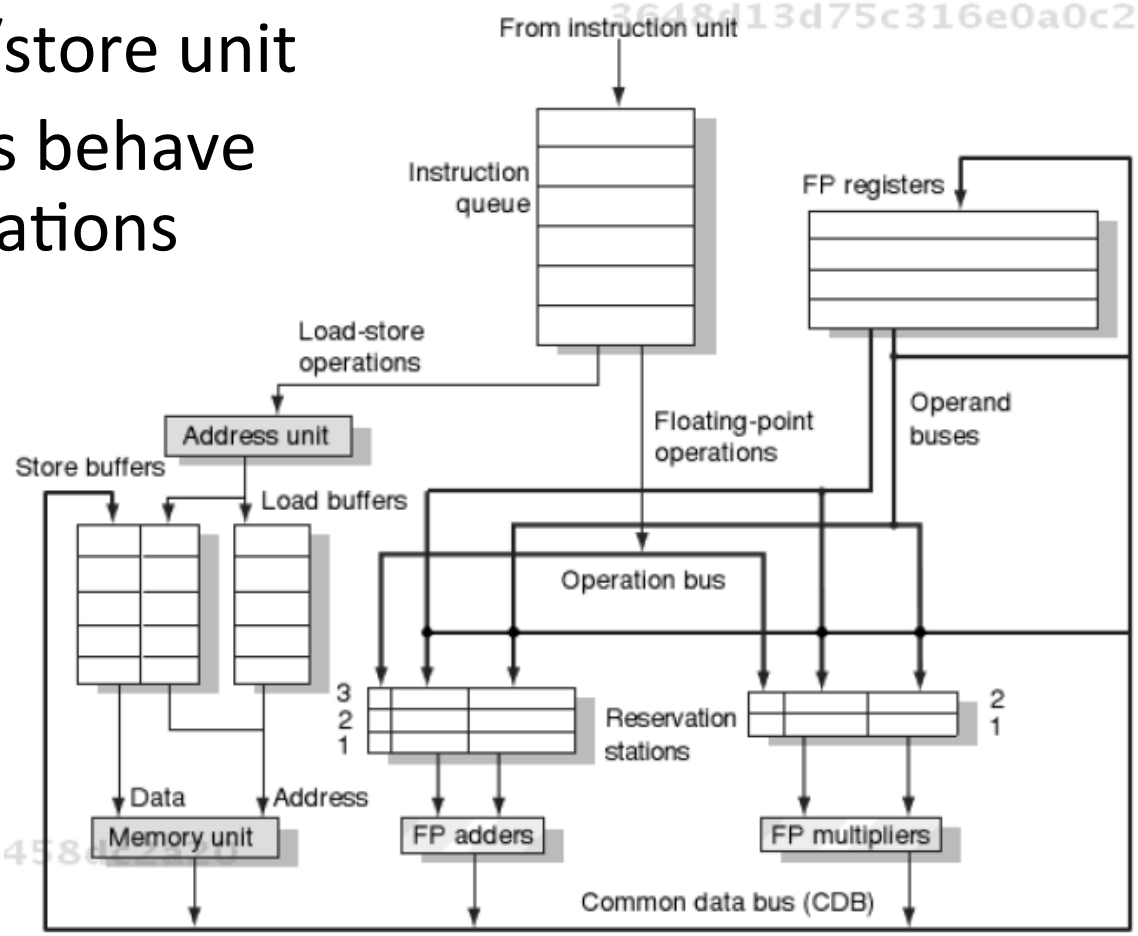
- FP units and load/store unit
- Load/store buffers behave like reservation stations

Effective registers:

  - 5 ld/store buffers
  - 5 reservation stations

Still:

- in-order issue
- and single issue



Tomasulo Instruction Execution

Three steps replacing the **ID, EX** and **WB** steps:

1. Issue (aka Dispatch): get next instruction in FIFO order. Issue to next available matching reservation station.
   Read operands from register if available; else track FU that produces result (effective renaming→ avoids WAR and WAW hazards)
2. Execute: execution is delayed until all operands are available (avoids RAW hazards)
3. Write result: once ready, result is written to common data bus (CDB), and hence to registers and reservation stations.

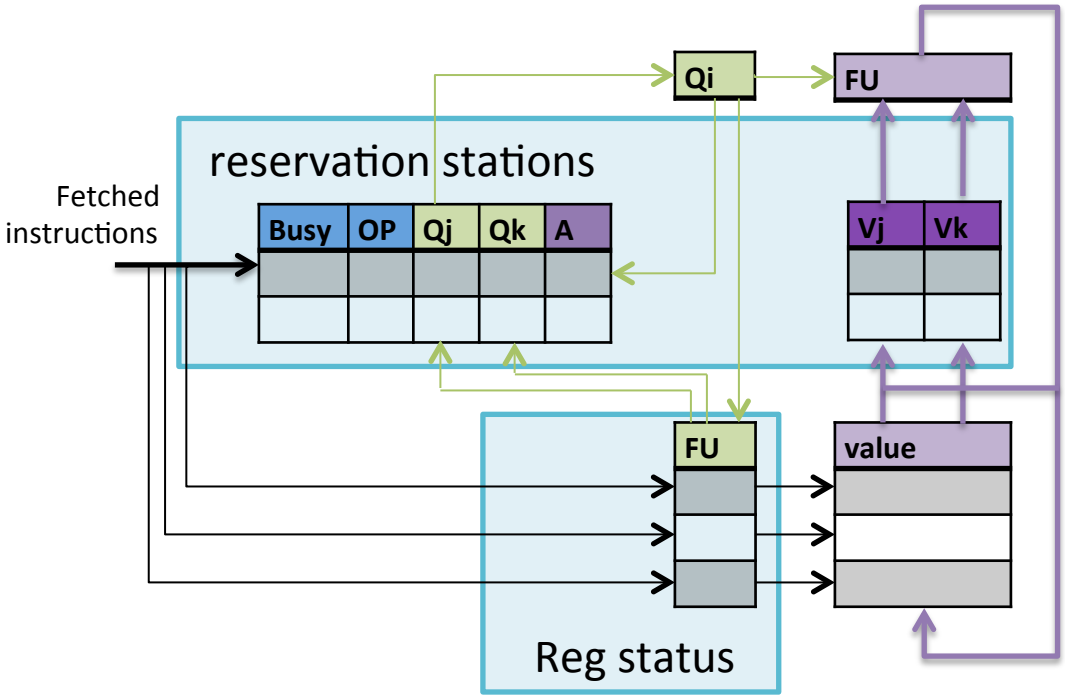
Implicit renaming

- Hazard detection and control done by reservation stations→ distributed approach
- Issued instructions refer to reservation station number that produces pending input→register renaming
- Common Data Bus (CDB) bypasses registers and allows simultaneous load of result for the FUs

Tomasulo Components

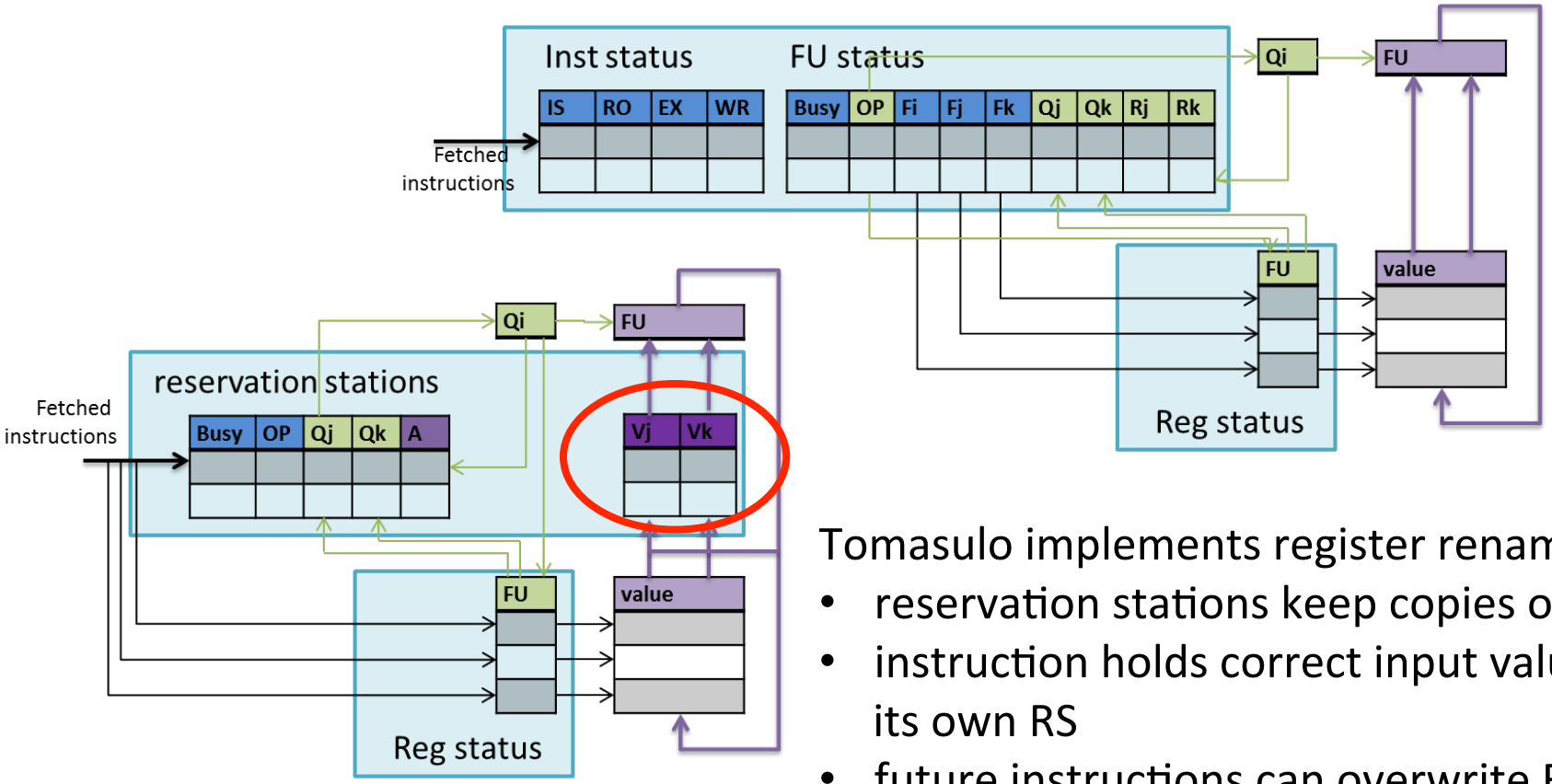
- ~~Instruction Status:~~ No longer needed → distributed approach
- Reservation Station:
  - Busy: Reservation Station and associated FU are busy
  - Op: operation to perform on S1 and S2
  - Qj,Qk: Reservation stations producing source operands
  - Vj, Vk: Value of source operands (either Vx or Qx is valid)
  - A: for memory address calculation; contains immediate, then resulting effective address (load-store buffers only)
- Register File: has only one field:
  - Qi: tag of reservation station whose output should be stored in that register. Blank if no active instruction targets register.
- CDB: common data bus
  - broadcasts <value, Qi> of completed instructions

Simple Tomasulo



- Instruction fields and status bits
- Tags
- Values

Tomasulo vs. Scoreboard



Tomasulo implements register renaming:

- reservation stations keep copies of input
- instruction holds correct input values in its own RS
- future instructions can overwrite RF master copy
  → doesn't affect other inputs

Tomasulo Data Structure

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 34(R2)			
L.D F2, 45(R3)			
MUL.D F0, F2, F4			
SUB.D F8, F6, F2			
DIV.D F10,F0, F6			
ADD.D F6, F8, F2			

Register Status

Register	Qi
F0	
F2	
F4	
F6	
F8	
F10	

CDB

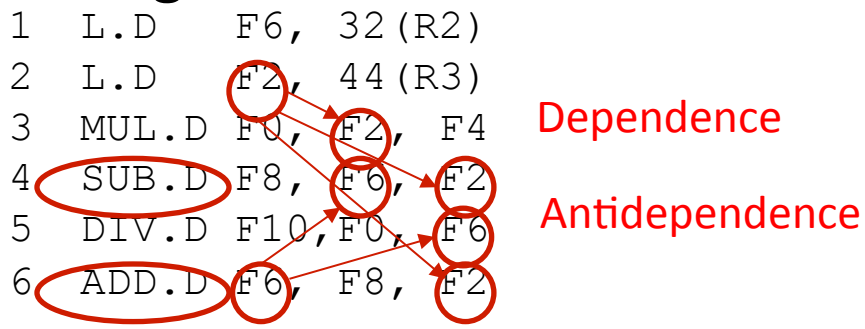
Register	Qi
F0	

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1							
Load2							
Add1							
Add2							
Add3							
Mul1							
Mul2							

Tomasulo Data Structure: Example

Example Program:



- EX latencies: load: 1cc, Add: 2cc, Mul: 6cc, Div: 12cc (by MUL FU)
- Which hazards exist?
  - RAW hazards: from 2 to 3, 4, and 6, from 3 to 5, and from 4 to 6
  - WAR hazards: from 4 to 6, and 5 to 6
  - Functional hazard: on Add FU for 4 and 6, on MUL FU for 3 and 5

Q: Information tables after 1 has finished?

Tomasulo Example 1

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok
L.D F2, 44(R3)	ok	ok	
MUL.D F0, F2, F4	ok		
SUB.D F8, F6, F2	ok		
DIV.D F10,F0, F6	ok		
ADD.D F6, F8, F2	ok		

Register Status

Register	Qi
F0	Mult1
F2	Load2
F4	
F6	Add2
F8	Add1
F10	Mult2

CDB

Register	Qi
F6	Load1

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44+Re[R3]
Add1	Yes	SUB	Mem[32+Re[R2]]			Load2	
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mul1	Yes	MUL		Re[F4]	Load2		
Mul2	Yes	DIV		Mem[32 + Re[R2]]	Mult1		

Tomasulo Example : After cc1

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok
L.D F2, 44(R3)	ok	ok	**cc1**
MUL.D F0, F2, F4	ok		
SUB.D F8, F6, F2	ok		
DIV.D F10,F0, F6	ok		
ADD.D F6, F8, F2	ok		

Register Status

Register	Qi
F0	Mult1
F2	~~Load2~~
F4	
F6	Add2
F8	Add1
F10	Mult2

CDB

Register	Qi
F2	Load2

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	~~Yes~~ NO	~~Load~~					~~44+Re[R3]~~
Add1	Yes	SUB	Mem[32+Re[R2]]	Mem[44+Re[R3]]		~~Load2~~	
Add2	Yes	ADD		Mem[44+Re[R3]]	Add1	~~Load2~~	
Add3	No						
Mul1	Yes	MUL	Mem[44+Re[R3]]	Re[F4]	~~Load2~~		
Mul2	Yes	DIV		Mem[32 + Re[R2]]	Mult1		

Tomasulo Example : After cc2

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok
L.D F2, 44(R3)	ok	ok	**cc1**
MUL.D F0, F2, F4	ok	**cc2+(cc7)**	
SUB.D F8, F6, F2	ok	**cc2+(cc3)**	
DIV.D F10,F0, F6	ok		
ADD.D F6, F8, F2	ok		

Register Status

Register	Qi
F0	Mult1
F2	~~Load2~~
F4	
F6	Add2
F8	Add1
F10	Mult2

CDB

Register	Qi
~~F2~~	~~Load2~~

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	~~Yes~~ NO	~~Load~~					~~44+Re[R3]~~
Add1	Yes	SUB	Mem[32+Re[R2]]	Mem[44+Re[R3]]		~~Load2~~	
Add2	Yes	ADD		Men[44+Re[R3]]	Add1	~~Load2~~	
Add3	No						
Mul1	Yes	MUL	Mem[44+Re[R3]]	Re[F4]	~~Load2~~		
Mul2	Yes	DIV		Mem[32 + Re[R2]]	Mult1		

Tomasulo Example : After cc4

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok
L.D F2, 44(R3)	ok	ok	**cc1**
MUL.D F0, F2, F4	ok	**cc2+(cc7)**	
SUB.D F8, F6, F2	ok	**cc2+(cc3)**	**cc4**
DIV.D F10,F0, F6	ok		
ADD.D F6, F8, F2	ok		

Register Status

Register	Qi
F0	Mult1
F2	
F4	
F6	Add2
F8	~~Add1~~
F10	Mult2

CDB

Register	Qi
F8	Add1

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	~~Yes~~ No	~~SUB~~	~~Mem[32+Re[R2]]~~	~~Mem[44+Re[R3]]~~			
Add2	Yes	ADD	Result[SUB]	Mem[44+Re[R3]]	~~Add1~~		
Add3	No						
Mul1	Yes	MUL	Mem[44+Re[R3]]	Re[F4]			
Mul2	Yes	DIV		Mem[32 + Re[R2]]	Mult1		

Tomasulo Example : After cc5

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok
L.D F2, 44(R3)	ok	ok	**cc1**
MUL.D F0, F2, F4	ok	**cc2+(cc7)**	
SUB.D F8, F6, F2	ok	**cc2+(cc3)**	**cc4**
DIV.D F10,F0, F6	ok		
ADD.D F6, F8, F2	ok	**Cc5+(cc6)**	

Register Status

Register	Qi
F0	Mult1
F2	
F4	
F6	Add2
F8	
F10	Mult2

CDB

Register	Qi
~~F8~~	~~Add1~~

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1							
Add2	Yes	ADD	Result[SUB]	Mem[44+Re[R3]]			
Add3	No						
Mul1	Yes	MUL	Mem[44+Re[R3]]	Re[F4]			
Mul2	Yes	DIV		Mem[32 + Re[R2]]	Mult1		

Tomasulo Example : After cc7

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok
L.D F2, 44(R3)	ok	ok	**cc1**
MUL.D F0, F2, F4	ok	**cc2+(cc7)**	
SUB.D F8, F6, F2	ok	**cc2+(cc3)**	**cc4**
DIV.D F10,F0, F6	ok		
ADD.D F6, F8, F2	ok	Cc5+(cc6)	**cc7**

Register Status

Register	Qi
F0	Mult1
F2	
F4	
F6	~~Add2~~
F8	
F10	Mult2

CDB

Register	Qi
F6	Add2

Write can proceed (no WAR hazard)

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	No						
Add2	~~Yes~~ No	~~ADD~~	~~Result[SUB]~~	~~Mem[44+Re[R3]]~~			
Add3	No						
Mul1	Yes	MUL	Mem[44+Re[R3]]	Re[F4]			
Mul2	Yes	DIV		Mem[32 + Re[R2]]	Mult1		

Tomasulo Example : After cc8

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F6, 32(R2)	ok	ok	ok
L.D F2, 44(R3)	ok	ok	**cc1**
MUL.D F0, F2, F4	ok	cc2+(cc7)	**cc8**
SUB.D F8, F6, F2	ok	cc2+(cc3)	**cc4**
DIV.D F10,F0, F6	ok		
ADD.D F6, F8, F2	ok	Cc5+(cc6)	cc7

Register Status

Register	Qi
F0	Mult1
F2	
F4	
F6	
F8	
F10	Mult2

CDB

Register	Qi
F0	Mult1

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	No						
Add2	No						
Add3	No						
Mul1	~~Yes~~ No	~~MUL~~	~~Mem[44+Re[R3]]~~	~~Re[F4]~~			
Mul2	Yes	DIV	Result[Mult1]	Mem[32 + Re[R2]]	~~Mult1~~		

Tomasulo: Example 2

Program with loop:

```
Loop:   L.D     F0, 0(R1)
        MUL.D   F4, F0, F2
        S.D     F4, 0(R1)
        DADDIU  R1, R1, -8
        BNE     R1, R2, Loop
```

- EX latencies: As before:
  - load: 1cc, Add: 2cc, Mul: 6cc, Div: 12cc (by MUL FU)
  - 2 load, 2 store, 3 add , 2 mult FUs
- Assume branch is taken multiple times: Tomasulo does dynamic unrolling

Q: Assume 2 iterations have been scheduled, but FP loads have not completed. We only look at FP operations.

Tomasulo Example 2

Instruction Status (not part of the hardware!!!)

Instruction	Issue	Execute complete	Write result
L.D F0, 0(R1)	ok	ok	
MUL.D F4, F0, F2	ok		
S.D F4, 0(R1)	ok		
L.D F0, 0(R1)	ok	ok	
MUL.D F4, F0, F2	ok		
S.D F4, 0(R1)	ok		

Register Status

Register	Qi
F0	Load2
F2	
F4	Mult2
F6	
F8	
F10	

CDB

Register	Qi

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	Yes	Load					Re[R0]+0
Load2	Yes	Load					Re[R0]-8
Add i	No						
Mul1	Yes	MUL		Re[F2]	Load1		
Mul2	Yes	MUL		Re[F2]	Load2		
Store1	Yes	Store	Re[R1]			Mult1	
Store2	Yes	Store	Re[R1]-8			Mult2	

Tomasulo: Overview

+ Prevents stalls due to WAW WAR hazards with renaming
+ Can achieve close to 1 IPC, or higher with multiple instruction issue
– Has high cost in hardware
– CDB is potential bottleneck

- Has become popular with caches, due to unpredictable delays of cache miss
- Achieves high performance without an optimized compiler

Outline

- 5 stages of RISC
- Type of hazards
- Static and Dynamic Branch Prediction
- Pipelining with Exceptions
- Pipelining with Floating-Point Operations
- Loop Unrolling
- Correlating and Tournament Branch Prediction
- Dynamic Scheduling: Scoreboard
- Dynamic Scheduling: Tomasulo: 3.5

- Hardware Based Speculation 3.6

Hardware-Based Speculation

Idea: to overcome control dependences

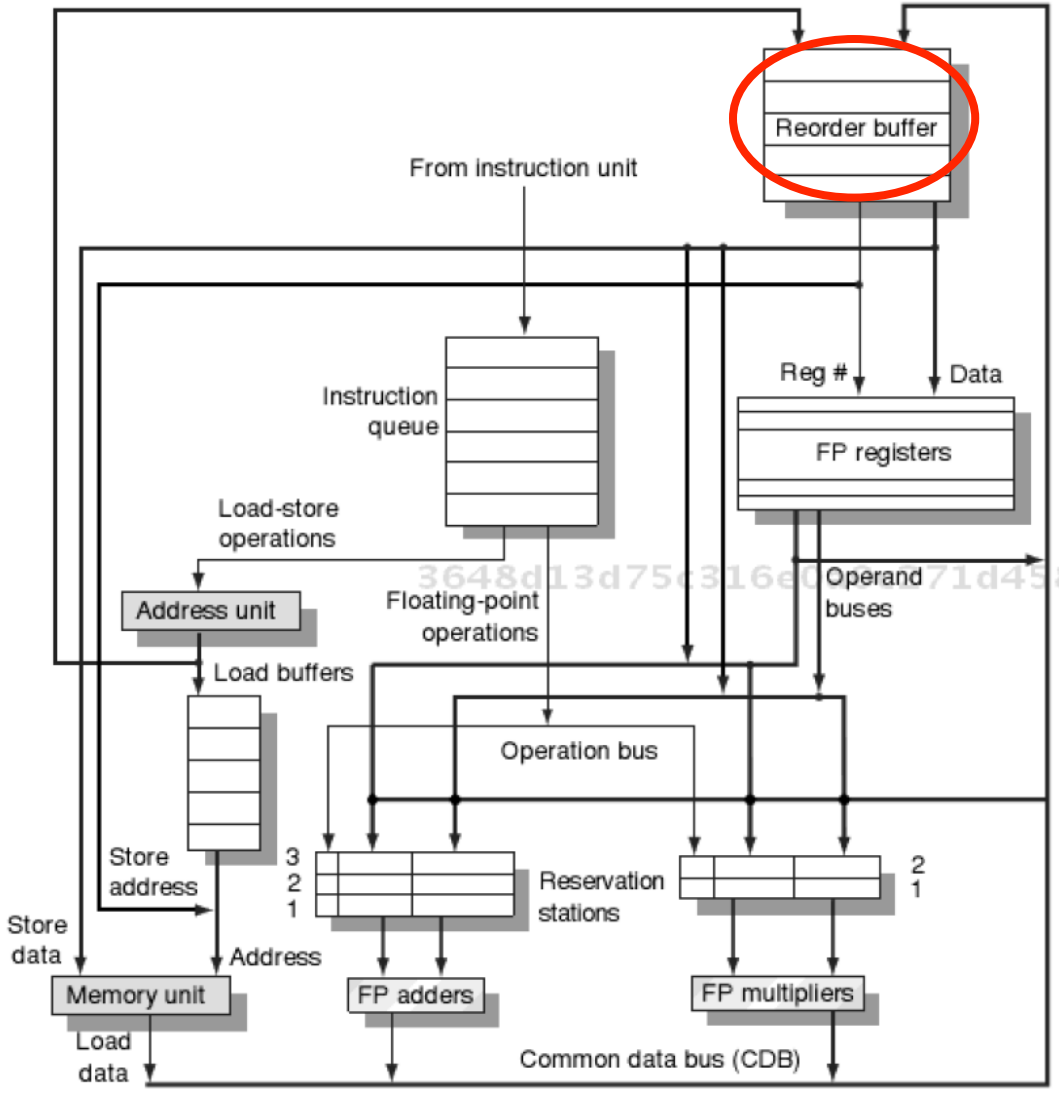
- Execute branches assuming branch prediction was correct.
  - Requires mechanism to handle wrong speculation

Hardware-based speculation based on:

1. Dynamic Branch Prediction
2. Speculation: allows execution before control dependences are resolved
3. Dynamic Scheduling to schedule combinations of basic blocks

Tomasulo + Speculation

- Reorder Buffer
  - enables speculation by separating bypassing of operands from instruction completion
  - replaces store buffer
  - Also provides additional registers
- Regs and memory only updated if execution is no longer speculative



Instruction Execution w/ Speculation

1. Issue: get next instruction from queue. Issue if matching reservation station *and* ROB slot are available.
   Read operands from register or ROB if available; RS tags result to ROB slot.
2. Execute: execution is delayed until all operands are available (monitoring CDB).
3. Write result: once ready, result is written to common data bus (CDB), and hence to ROB and reservation stations (but not registers).
4. Commit: Either update the register (normal commit) or memory (store commit) when instruction reaches head of ROB *or* if branch with incorrect prediction reaches head of ROB, the ROB is flushed and execution is restarted.

Tomasulo+Speculation Components

- Reorder Buffer: ROB entries have four fields per instruction:
  - Instruction: instruction + type (branch (no dest. result) or store (memory) or ALU/LD (reg))
  - State: state of the instruction
  - Destination: either register number or memory address (for stores)
  - Value: value of instruction result (waiting for commit)
- Reservation Station:
  - Busy: Reservation Station and associated FU are busy
  - Op: operation to perfrom on S1 and S2
  - Qj,Qk: ROB entry number producing source operands
  - Vj, Vk: Value of source operands (either Vx or Qx is valid)
  - Destination (Qi): tag of ROB entry number that should receive the result
  - A: for memory address calculation; contains immediate, then resulting effective address (load-store buffers only)
- Register File: has only one field:
  - Qi: tag of ROB entry number whose output should be stored in that register. Blank if no active instruction targets register.
- CDB: common data bus
  - broadcasts <value, Qi> of completed instructions

Tomasulo+Speculation Components

Reorder Buffer

Entry	Busy	Instruction	State	Dest	Value
1					
2					
3					
4					
5					
6					

Register Status

Register	Busy	Qi
F0		
F2		
F4		
F6		
F8		
F10		

CDB

Register	Qi

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1								
Load2								
Add1								
Add2								
Add3								
Mul1								
Mul2								

Tomasulo Data Structure: Example

Example Program:

```
1  L.D   F6, 32(R2)
2  L.D   F2, 44(R3)
3  MUL.D F0, F2, F4
4  SUB.D F8, F6, F2
5  DIV.D F10,F0, F6
6  ADD.D F6, F8, F2
```

- EX latencies: As before:
  - load: 1cc, Add: 2cc, Mul: 6cc, Div: 12cc (by MUL FU)
  - 2 load, 2 store, 3 add , 2 mult FUs

Q: Information in tables after 3 has finished?

Tomasulo+Speculation: Example1

Reorder Buffer

Entry	Busy	Instruction	State	Dest	Value
1	No	L.D F6, 32(R2)	Commit	F6	Mem[32+Re[R2]]
2	No	L.D F2, 44(R3)	Commit	F2	Mem[44+Re[R3]]
3	Yes	MUL.D F0, F2, F4	Write Res	F0	#2xRegs[F4]
4	Yes	SUB.D F8, F6, F2	Write Res	F8	Re[F6]-#2
5	Yes	DIV.D F10,F0, F6	Execute	F10	
6	Yes	ADD.D F6, F8, F2	Write Res	F6	#4+#2

Register Status

Register	Busy	Qi
F0	Yes	3
F2	No	
F4	No	
F6	Yes	6
F8	Yes	4
F10	Yes	5

CDB

ROB	Qi
#3	Mul1

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	No							
Load2	No							
Add1	No							
Add2	No							
Add3	No							
Mul1	No	MUL.D	Mem[44+Re[R3]]	Re[F4]			#3	
Mul2	Yes	DIV		Mem[32+Re[R2]	#3		#5	

Only top of ROB commits!
SUB and ADD will not commit until MUL has committed

➔ No instruction after earliest uncompleted instruction is allowed to complete! (Yields precise exceptions)

Speculation: Example 2

Program with loop:

```
Loop:   L.D     F0, 0(R1)
        MUL.D   F4, F0, F2
        S.D     F4, 0(R1)
        DADDIU  R1, R1, -8
        BNE     R1, R2, Loop
```

- EX latencies: As before:
  - load: 1cc, Add: 2cc, Mul: 6cc, Div: 12cc (by MUL FU)
  - 2 load, 2 store, 3 add , 2 mult FUs

Q: Assume 2 iterations have been scheduled, but only first L.D and MUL.D have been commited.

Tomasulo+Speculation: Example2

Reorder Buffer

Ent	Busy	Instruction	IS	Ex	WR	Co	Dest	Value
1	No	L.D F0, 0(R1)	ok	ok	ok		F0	Mem[0+Re[R1]]
2	Yes	MUL.D F4, F0, F2	ok				F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	ok				0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	ok				R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	ok					
6	Yes	L.D F0, 0(R1)	ok				F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	ok				F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	ok				0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8					R1	#4-8
10	Yes	BNE R1, R2, Loop						

Register Status

	Busy	Qi
	Yes	6
	No	
	No	
	Yes	7
	No	
	No	

Speculation Example2: Cycle 1

Reorder Buffer

Ent	Busy	Instruction	IS	Ex	WR	Co	Dest	Value
1	No	L.D F0, 0(R1)	ok	ok	ok	cc1	F0	Mem[0+Re[R1]]
2	Yes	MUL.D F4, F0, F2	ok	cc1			F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	ok				0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	ok	cc1			R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	ok					
6	Yes	L.D F0, 0(R1)	ok				F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	ok				F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	ok				0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8	cc1				R1	#4-8
10	Yes	BNE R1, R2, Loop						

Speculation Example2: Cycle 2

Reorder Buffer

Ent	Busy	Instruction	IS	Ex	WR	Co	Dest	Value
1	No	L.D F0, 0(R1)	ok	ok	ok	cc1	F0	Mem[0+Re[R1]]
2	Yes	MUL.D F4, F0, F2	ok	cc1-4			F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	ok				0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	ok	cc1	cc2		R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	ok					
6	Yes	L.D F0, 0(R1)	ok				F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	ok				F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	ok				0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8	cc1				R1	#4-8
10	Yes	BNE R1, R2, Loop	cc2					

Speculation Example2: Cycle 3

Reorder Buffer

Ent	Busy	Instruction	IS	Ex	WR	Co	Dest	Value
1	No	L.D F0, 0(R1)	ok	ok	ok	cc1	F0	Mem[0+Re[R1]]
2	Yes	MUL.D F4, F0, F2	ok	cc1-4			F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	ok	cc3			0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	ok	cc1	cc2		R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	ok	cc3				
6	Yes	L.D F0, 0(R1)	ok	cc3			F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	ok				F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	ok	cc3			0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8	cc1	cc3			R1	#4-8
10	Yes	BNE R1, R2, Loop	cc2					

Speculation Example2: Cycle 4

Reorder Buffer

Ent	Busy	Instruction	IS	Ex	WR	Co	Dest	Value
1	No	L.D F0, 0(R1)	ok	ok	ok	cc1	F0	Mem[0+Re[R1]]
2	Yes	MUL.D F4, F0, F2	ok	cc1-4			F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	ok	cc3			0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	ok	cc1	cc2		R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	ok	cc3	cc4			
6	Yes	L.D F0, 0(R1)	ok	cc3	cc4		F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	ok				F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	ok	cc3			0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8	cc1	cc3	cc4		R1	#4-8
10	Yes	BNE R1, R2, Loop	cc2					

Speculation Example2: Cycle 5

Reorder Buffer

Ent	Busy	Instruction	IS	Ex	WR	Co	Dest	Value
1	No	L.D F0, 0(R1)	ok	ok	ok	cc1	F0	Mem[0+Re[R1]]
2	Yes	MUL.D F4, F0, F2	ok	cc1-4	cc5		F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	ok	cc3			0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	ok	cc1	cc2		R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	ok	cc3	cc4			
6	Yes	L.D F0, 0(R1)	ok	cc3	cc4		F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	ok	cc5-8			F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	ok	cc3			0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8	cc1	cc3	cc4		R1	#4-8
10	Yes	BNE R1, R2, Loop	cc2	cc5				

Speculation Example2: Cycle 6

Reorder Buffer

Ent	Busy	Instruction	IS	Ex	WR	Co	Dest	Value
1	No	L.D F0, 0(R1)	ok	ok	ok	cc1	F0	Mem[0+Re[R1]]
2	Yes	MUL.D F4, F0, F2	ok	cc1-4	cc5	cc6	F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	ok	cc3	cc6		0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	ok	cc1	cc2		R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	ok	cc3	cc4			
6	Yes	L.D F0, 0(R1)	ok	cc3	cc4		F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	ok	cc5-8			F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	ok	cc3			0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8	cc1	cc3	cc4		R1	#4-8
10	Yes	BNE R1, R2, Loop	cc2	cc5	cc6			

Only top of ROB commits!
All later instructions wait for MUL to be committed

All other instructions will commit as early as possible.

Tomasulo+Speculation: Example2

Reorder Buffer

Entry	Busy	Instruction	State	Dest	Value
1	No	L.D F0, 0(R1)	Commit	F0	Mem[0+Re[R1]]
2	No	MUL.D F4, F0, F2	Commit	F4	#1xRegs[F2]
3	Yes	S.D F4, 0(R1)	Write Res	0+Re[R1]	#2
4	Yes	DADDIU R1, R1, -8	Write Res	R1	Regs[R1]-8
5	Yes	BNE R1, R2, Loop	Write Res		
6	Yes	L.D F0, 0(R1)	Write Res	F0	Mem[#4]
7	Yes	MUL.D F4, F0, F2	Write Res	F4	#6xRegs[F2]
8	Yes	S.D F4, 0(R1)	Write Res	0+Re[R1]	#7
9	Yes	DADDIU R1, R1, -8	Write Res	R1	#4-8
10	Yes	BNE R1, R2, Loop	Write Res		

Register Status

Register	Busy	Qi
F0	Yes	6
F2	No	
F4	No	
F6	Yes	7
F8	No	
F10	No	

Only top of ROB commits!
All later instructions wait for MUL to be committed
All other instruction will commit as early as possible.

What happens if 5 was mispredicted?

Speculation: Hazards through Memory

- WAW and WAR hazards avoided, as memory updated occur in order
- RAW hazards maintained through restriction:
  - Prevent load second step if A field of load is also target of a store in ROB
  - Maintain program order for computation of effective load address wrt. earlier stores

Speculation Overview

- Speculative actions easily undone due to ROB
- Writes only happen during commit (unlike Tomasulo)
  - ➢ Precise exception handling possible during commit.
- Performance highly dependent on branch prediction quality
  - With correct prediction close to 1 IPC, due to eliminated data and control stalls
  - Multiple issue (i.e. allowing multiple instructions to be issued and committed per cycle) can further tweak performance

Outlook

Next Time:

Multiple Issue

Multiple Issue Processors

Last time:

- data and control stalls eliminated with dynamic scheduling and speculative execution
  - Performance close to 1 IPC
- To go beyond 1 IPC, more than 1 instruction must be issued (and completed) per cycle

Three major flavors of multiple issue:

1. Statically scheduled superscalar processors
2. VLIW (very long instruction word) processors
3. Dynamically scheduled superscalar processors

Characteristics

- Superscalar Processors issue varying number of instructions per clock
  - Either in-order execution (statically scheduled) or out-of-order execution (dynamically scheduled)
- VLIW Processors issue fixed number of instruction, formatted as one large instruction, with parallelism explicitly indicated by inst.
  - Inherently statically scheduled by compiler
  - High similarity to static superscalar!

Overview of Multiple-Issue Processors

Common Name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly Embedded : MIPS, ARM (e.g. Cortex-A8)
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None so far
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core iX, AMD Phenom, IBM POWER7
VLIW/LIW	Static	Primarily software	Static	All hazards determined by compiler (often implicitly)	Mostly signal processing, e.g. TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by compiler	Itanium