Tomasulo's Algorithm for Dynamic Scheduling

- Developed for the IBM 360/91 in 1967 - about 3 years after CDC 6600
  - Goal: High performance without special compilers

- Differences between IBM 360 & CDC 6600
  - IBM 360 has only 2 register specifiers/instr vs. 3 in CDC 6600
  - IBM 360 has register-memory instructions
  - IBM 360 has 4 FP registers vs. 8 in CDC 6600
  - IBM 360 has pipelined functional units (3 adds, 2 multiplies)

- Tomasulo's algorithm is designed to handle name dependencies (WAW and WAR hazards) efficiently

SUB	F1, F2, F0	F1, F2, F0
DIVF	F2, F3, F2	T, F3, F2
ADDF	F3, F0, F0	S, F0, F0
MULF	F3, F1, F1	F3, F1, F1

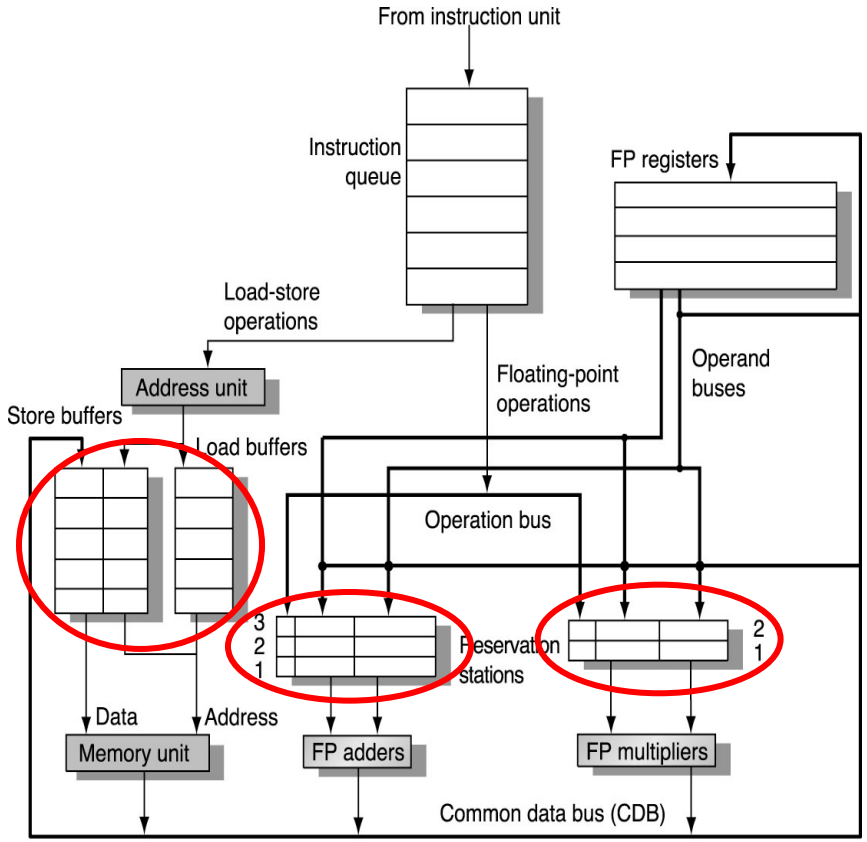
WAR hazard

WAW hazard

Both hazards can be avoided by register renaming

Tomasulo's Algorithm: Hardware Organization

- Led to concepts used in the Alpha 21264, HP 8000, MIPS 10000, Pentium II, PowerPC 604, …
- Key idea: Reservation Stations
  - Buffer operands for instructions waiting to issue
    - As soon as operand is available
  - Effectively provide an arbitrary number of (internal) registers
- Benefits
  1. HW renaming of registers to avoid WAR, WAW hazards
  2. Distributed hazard detection and resolution vs. centralized scoreboard and register file
     - Using the Common Data Bus
     - Bypassing of results to the FUs



Three Stages of Tomasulo's Algorithm

- Issue: Get instruction from the head of the instruction queue
  - If reservation station free, allocate and issue instruction
    - If operands are available, send them to the reservation station
    - If not, keep track of FUs that will produce the operands
  - If not free, stall issue

WAR, WAW hazards

Structural hazards

- Execution: Operate on operands (EX)
  - If both operands ready, execute
  - If not ready, watch CDB for result
  - Instructions along predicted paths not allowed to execute unless branch direction is resolved (simple implementation: prevent any instruction from leaving Issue if there are pending branch in the pipeline)

RAW hazards

- Write result: Finish execution (WB)
  - Write on CDB, mark reservation station available

Bypassing

Reservation Station Components

Op	Operation to perform in the unit (e.g., + or –)
Qj, Qk	Reservation stations producing source operands
Vj, Vk	**Value of Source operands**
Rj, Rk	Flags indicating when Vj, Vk are ready
Busy	Indicates reservation station and FU is busy

Register result status — Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Out-Of-Order Execution

- Instructions can become ready for execution out-of-order
- Dynamic scheduling is beneficial only if these instructions are allowed to “jump” over others

- FP units can pick any of the ready instructions to execute
  - All side effects in “registers”: can be tracked

- LD/ST instructions require additional support
  - Hardware support for dynamic memory disambiguation
    Allow memory operations to different addresses to proceed out-of-order
  - Can be achieved by noting that LD/ST instructions first need to compute address, then access memory
    - **Ensuring effective address calculation happens in program order; AND**
    - Not issuing a load (to memory unit) if detect a store to the same address
      Not issuing a store (to memory unit) if detect a load/store to same address

Example of Tomasulo's Algorithm

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)				2
L.D F2, 45(R3)				2
MUL.D F0, F2, F4				10
SUB.D F8, F6, F2				2
DIV.D F10, F0, F6				40
ADD.D F6, F8, F2				2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16

Example of Tomasulo's Algorithm: Cycle 1

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1			2
L.D F2, 45(R3)				2
MUL.D F0, F2, F4				10
SUB.D F8, F6, F2				2
DIV.D F10, F0, F6				40
ADD.D F6, F8, F2				2

Name	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
			Load1				

Example of Tomasulo's Algorithm: Cycle 2

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1			2
L.D F2, 45(R3)	2			2
MUL.D F0, F2, F4				10
SUB.D F8, F6, F2				2
DIV.D F10, F0, F6				40
ADD.D F6, F8, F2				2

Name	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
	Load2		Load1				

Example of Tomasulo's Algorithm: Cycle 3

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3		2
L.D F2, 45(R3)	2			2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2				2
DIV.D F10, F0, F6				40
ADD.D F6, F8, F2				2

Name	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F4)	Load2		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	Load2		Load1				

Example of Tomasulo's Algorithm: Cycle 4

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4		2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4			2
DIV.D F10, F0, F6				40
ADD.D F6, F8, F2				2

Name	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	Yes	SUB	M(…)			Load2	Yes	No
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F4)	Load2		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	Load2		M(R2+34)	Add1			

Example of Tomasulo's Algorithm: Cycle 5

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4			2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2				2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
2	Add1	Yes	SUB	M(…)	M(…)		Load2	Yes	Yes
	Add2	No							
	Add3	No							
10	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		M(R2+34)	Add1	Mult2		

Example of Tomasulo's Algorithm: Cycle 6

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4			2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6			2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
1	Add1	Yes	SUB	M(…)	M(…)		Load2	Yes	Yes
	Add2	Yes	ADD		M(…)	Add1		No	Yes
	Add3	No							
9	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		Add2	Add1	Mult2		

Example of Tomasulo's Algorithm: Cycle 7

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4	7		2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6			2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
0	Add1	Yes	SUB	M(…)	M(…)		Load2	Yes	Yes
	Add2	Yes	ADD		M(…)	Add1		No	Yes
	Add3	No							
8	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		Add2	Add1	Mult2		

Example of Tomasulo's Algorithm: Cycle 8

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6			2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
2	Add2	Yes	ADD	M() – M()	M(…)	Add1		Yes	Yes
	Add3	No							
7	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		Add2	M() – M()	Mult2		

Example of Tomasulo's Algorithm: Cycle 9

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6			2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
1	Add2	Yes	ADD	M()-M()	M(…)	Add1		Yes	Yes
	Add3	No							
6	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		Add2	M() – M()	Mult2		

Example of Tomasulo's Algorithm: Cycle 10

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6	10		2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
0	Add2	Yes	ADD	M()-M()	M(…)	Add1		Yes	Yes
	Add3	No							
5	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		Add2	M() – M()	Mult2		

Example of Tomasulo's Algorithm: Cycle 11

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3			10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6	10	11	2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
4	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		(M-M)+M(…)	M() – M()	Mult2		

Example of Tomasulo's Algorithm: Cycle 15

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3	15		10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6	10	11	2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
0	Mult1	Yes	MUL	M(…)	R(F4)	Load2		Yes	Yes
	Mult2	Yes	DIV		M(…)	Mult1		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Mult1	M(R3+45)		(M-M)+M	M() – M()	Mult2		

Example of Tomasulo's Algorithm: Cycle 16

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3	15	16	10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6	10	11	2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
40	Mult2	Yes	DIV	M*R(F4)	M(…)	Mult1		Yes	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
M*R(F4)	M(R3+45)		(M-M)+M	M() – M()	Mult2		

Example of Tomasulo's Algorithm: Cycle 56

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3	15	16	10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5	56		40
ADD.D F6, F8, F2	6	10	11	2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
0	Mult2	Yes	DIV	M*R(F4)	M(…)	Mult1		Yes	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
M*R(F4)	M(R3+45)		(M-M)+M	M() – M()	Mult2		

Example of Tomasulo's Algorithm: Cycle 57

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3	15	16	10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5	56	57	40
ADD.D F6, F8, F2	6	10	11	2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
M*R(F4)	M(R3+45)		(M-M)+M	M() – M()	M*R(F)/M		

Tomasulo's Algorithm (IBM 360/91) vs. Scoreboarding (CDC 6600)

Instruction	Issue	Exec.	Write
L.D F6, 34(R2)	1	3	4
L.D F2, 45(R3)	2	4	5
MUL.D F0, F2, F4	3	15	16
SUB.D F8, F6, F2	4	7	8
DIV.D F10, F0, F6	5	56	57
ADD.D F6, F8, F2	6	10	11

Pipelined Functional Units
(6 load, 3 store, 3 +, 2 x/÷)

WAR: renaming avoids
WAW: renaming avoids
Broadcast results from FU
Control: reservation stations
High hardware complexity

Instruction	Issue	Read	Exec.	Write
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	**9**	19	20
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8	21	61	62
ADD.D F6, F8, F2	13	14	16	22

Multiple Functional Units
(1 load/store, 1 + , 2 x, 1 ÷)

stall completion
stall issue
Write/read registers
Control: central scoreboard
Low hardware complexity

Tomasulo Loop Example

- Code fragment (multiply array elements by a scalar in F2)

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

- Consider how the loop will execute
  - Branch predicted as taken (allows loop to be unrolled dynamically)
  - Integer operations complete early, so we are left with a stream of the L.D, MUL.D, and S.D instructions
- Track execution of these instructions for two iterations
  - Multiply takes 4 clocks
  - First load takes 8 clocks, second takes 4 cycles

Tomasulo Loop Example: Cycle 1 (Assume R1=80)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1		
MUL.D F4, F0, F2			
S.D F4, 0(R1)			
L.D F0, 0(R1)			
MUL.D F4, F0, F2			
S.D F4, 0(R1)			

Name	Busy	Addr.	Qi
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load1							

Tomasulo Loop Example: Cycle 2

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1		
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)			
L.D F0, 0(R1)			
MUL.D F4, F0, F2			
S.D F4, 0(R1)			

Name	Busy	Addr.	Qi
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load1		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load1		Mult1					

Tomasulo Loop Example: Cycle 3

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1		
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)			
MUL.D F4, F0, F2			
S.D F4, 0(R1)			

Name	Busy	Addr.	Qi
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load1		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load1		Mult1					

Tomasulo Loop Example: Cycle 6
(DADD issued in Cycle 4, BNE in Cycle 5, predicted taken)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1		
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6		
MUL.D F4, F0, F2			
S.D F4, 0(R1)			

Name	Busy	Addr.	Qi
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load1		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load2		Mult1					

Tomasulo Loop Example: Cycle 7

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1		
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6		
MUL.D F4, F0, F2	7		
S.D F4, 0(R1)			

Name	Busy	Addr.	Qi
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load1		No	Yes
	Mult2	Yes	MUL		R(F2)	Load2		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load2		Mult2					

Tomasulo Loop Example: Cycle 8

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1		
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6		
MUL.D F4, F0, F2	7		
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
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	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load1		No	Yes
	Mult2	Yes	MUL		R(F2)	Load2		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load2		Mult2					

Tomasulo Loop Example: Cycle 9
(also issue DADD of iteration 2)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6		
MUL.D F4, F0, F2	7		
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
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	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load1		No	Yes
	Mult2	Yes	MUL		R(F2)	Load2		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load2		Mult2					

Tomasulo Loop Example: Cycle 10
(also issue BNE of iteration 2, predicted taken)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6	10	
MUL.D F4, F0, F2	7		
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
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	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
4	Mult1	Yes	MUL	M(80)	R(F2)	Load1		Yes	Yes
	Mult2	Yes	MUL		R(F2)	Load2		No	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load2		Mult2					

Tomasulo Loop Example: Cycle 11
(also issue L.D of iteration 3)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7		
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
3	Mult1	Yes	MUL	M(80)	R(F2)	Load1		Yes	Yes
4	Mult2	Yes	MUL	M(72)	R(F2)	Load2		Yes	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult2					

Tomasulo Loop Example: Cycle 12
(fail to issue MUL of iteration 3: Structural Hazard)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2		
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7		
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
2	Mult1	Yes	MUL	M(80)	R(F2)	Load1		Yes	Yes
3	Mult2	Yes	MUL	M(72)	R(F2)	Load2		Yes	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult2					

Tomasulo Loop Example: Cycle 14

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2	14	
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7		
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3			

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
0	Mult1	Yes	MUL	M(80)	R(F2)	Load1		Yes	Yes
1	Mult2	Yes	MUL	M(72)	R(F2)	Load2		Yes	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult2					

Tomasulo Loop Example: Cycle 15

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2	14	15
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7	15	
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	M()*F2
Store2	Yes	72	Mult2
Store3			

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
0	Mult2	Yes	MUL	M(72)	R(F2)	Load2		Yes	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult2					

Tomasulo Loop Example: Cycle 16
(can now issue MUL of iteration 3)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2	14	15
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7	15	16
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	M()*F2
Store2	Yes	72	M()*F2
Store3			

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load3		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult1					

Tomasulo Loop Example: Cycle 17
(issue S.D of iteration 3)

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2	14	15
S.D F4, 0(R1)	3		
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7	15	16
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	M()*F2
Store2	Yes	72	M()*F2
Store3	Yes	64	Mult1

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load3		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult1					

Tomasulo Loop Example: Cycle 19

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2	14	15
S.D F4, 0(R1)	3	19	
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7	15	16
S.D F4, 0(R1)	8		

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	M()*F2
Store2	Yes	72	M()*F2
Store3	Yes	64	Mult1

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load3		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult1					

Tomasulo Loop Example: Cycle 20

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2	14	15
S.D F4, 0(R1)	3	19	20
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7	15	16
S.D F4, 0(R1)	8	20	

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	Yes	72	M()*F2
Store3	Yes	64	Mult1

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load3		No	Yes
	Mult2	No							

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult1					

Tomasulo Loop Example: Cycle 21

Instructions

Instruction	Issue	Execute	Write
L.D F0, 0(R1)	1	9	10
MUL.D F4, F0, F2	2	14	15
S.D F4, 0(R1)	3	19	20
L.D F0, 0(R1)	6	10	11
MUL.D F4, F0, F2	7	15	16
S.D F4, 0(R1)	8	20	21

Name	Busy	Addr.	Qi
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	No		
Store3	Yes	64	Mult1

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	Yes	MUL		R(F2)	Load3		No	Yes
	Mult2	No							

10 instructions complete in 12 cycle interval

Registers

F0	F2	F4	F6	F8	F10	F12	F16
Load3		Mult1					

Tomasulo's Algorithm: Summary

- Reservations stations: renaming to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of Scoreboard

- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation

- 360/91 descendants are Pentium II; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264

HW Support for Reducing Stalls

- Branch prediction schemes help minimize wasted issue effort
- Branch prediction becomes even more important in ISAs with dynamic scheduling

Can we do better?

Yes, with more hardware support

- Hardware speculation
  - Execute the (dependent) instruction, but state does not get “committed”

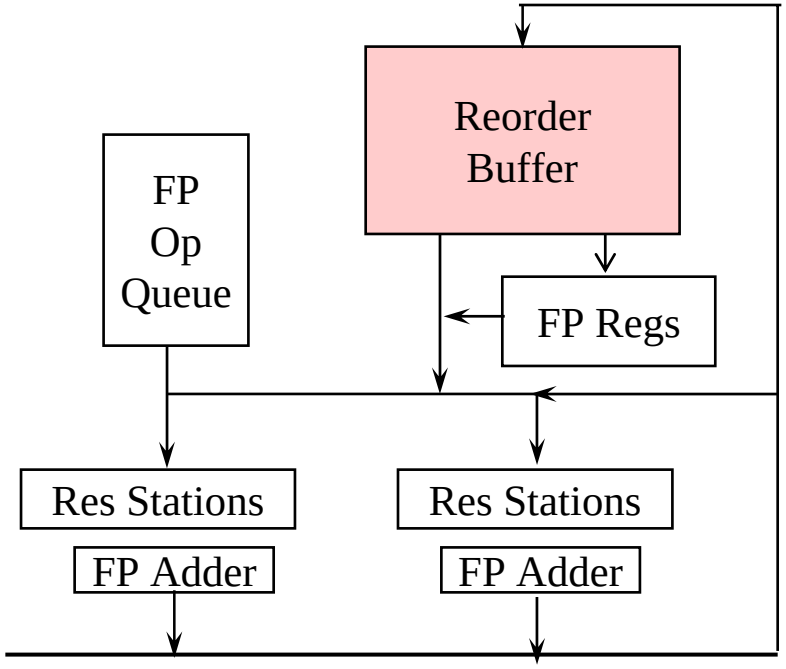
Hardware Support: Speculative Execution

- *Main idea*: allow execution of an instruction dependent on a predicted-taken branch such that there are no consequences (including exceptions) if branch is not actually taken
  - Hardware needs to undo the instruction - hard to do if there are exceptions
- Don't want a speculative instruction to cause exceptions that stop programs (i.e. memory violation)
- Hardware support for speculation should buffer results and exceptions from instructions until known that the instruction would execute
  - Reorder Buffer in extended Tomasulo's algorithm

Hardware-based Speculation

- Modify dynamic scheduling logic to also support speculative execution
- In Tomasulo's algorithm, separate speculative bypassing of results from real bypassing of results
  - When instruction no longer speculative, write results (instruction commit)
  - Execute out-of-order but commit (write results) in order

- Key mechanism: Reorder Buffer
  - Hardware buffer for uncommitted results



Hardware-based Speculation: Reorder Buffer

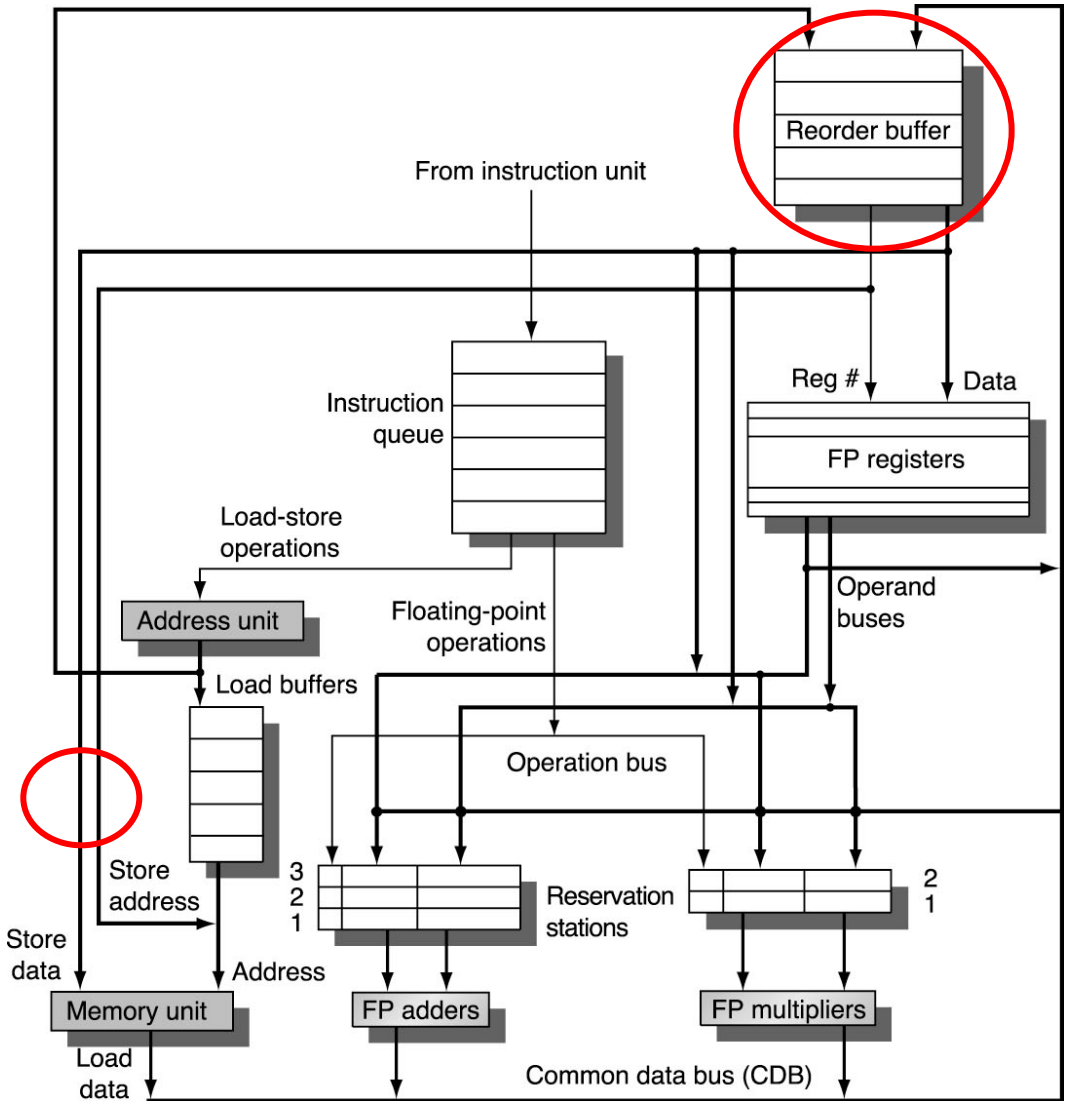
- Reorder buffer can be operand source, if value not yet committed
- Once operand commits, result is found in register file

- Mechanism
  - At issue time, allocate an entry in the ROB to hold result
    - Each entry has 4 fields: instruction type, destination register, value, ready flag
  - Use ROB entry number instead of reservation station to rename
    - Since each value has a location in the ROB anyways
    - Can use additional registers for renaming, and ROB only for tracking commits
  - Instruction results commit to register set in order
    - Simple if ROB implemented as a queue

  - Undoing speculated instructions on mispredicted branches or on exceptions just requires throwing away uncommitted entries
    - Exceptions are not recognized until an instruction becomes ready to commit

Extended Tomasulo's Algorithm

- Force instructions to "commit" in order
  - i.e., processor register state is seen in order, supporting precise exceptions
- Typically implemented using a ReOrder Buffer
  - Holds results till they are ready to be written to registers
  - Instructions can use ROB values



Four Steps of Speculative Tomasulo's Algorithm

- Issue: Get instruction from the head of the instruction queue
  - If reservation station and ROB slot free, allocate and issue instruction
    - If operands are available, send them to the reservation station
    - If not, keep track of ROB entry that will produce the operands
  - If not free, stall issue
- Execution: Operate on operands (EX)
  - If both operands ready, execute
  - If not ready, watch CDB for result
- Write result: Finish execution (WB)
  - Write on CDB, mark reservation station available
  - Result picked up by ROB entry
- Commit: Update register with ROB result
  - When instruction at head of ROB and its result is present, update register (or send store to memory), free up ROB slot
  - If ROB head is an incorrectly predicted branch, flush ROB

(From first example)
Example of Tomasulo's Algorithm: Cycle 16

Instructions

Instruction	Issue	Execute	Write	
L.D F6, 34(R2)	1	3	4	2
L.D F2, 45(R3)	2	4	5	2
MUL.D F0, F2, F4	3	15	16	10
SUB.D F8, F6, F2	4	7	8	2
DIV.D F10, F0, F6	5			40
ADD.D F6, F8, F2	6	10	11	2

Name	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Load/Store Units

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk
	Add1	No							
	Add2	No							
	Add3	No							
	Mult1	No							
40	Mult2	Yes	DIV	M*R(F4)	M(…)	Mult1		Yes	Yes

Registers

F0	F2	F4	F6	F8	F10	F12	F16
M*R(F4)	M(R3+45)		(M-M)+M	M() – M()	Mult2		

Speculative Tomasulo's Algorithm: Cycle 16

Instructions

Instruction	Issue	Execute	Write	Commit
L.D F6, 34(R2)	1	3	4	5
L.D F2, 45(R3)	2	4	5	6
MUL.D F0, F2, F4	3	15	16	
SUB.D F8, F6, F2	4	7	8	
DIV.D F10, F0, F6	5			
ADD.D F6, F8, F2	6	10	11	

Reorder Buffer

	?	Instr.	Reg	Value
1	N	L	F6	M(R2+34)
2	N	L	F2	M(R3+45)
3	Y	MUL	F0	#2 x R(F4)
4	Y	SUB	F8	#1 - #2
5	Y	DIV	F10	
6	Y	ADD	F6	#4 + #2

Reservation stations

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Rj	Rk	Dest.
	Add1	No								
	Add2	No								
	Add3	No								
	Mult1	No								
40	Mult2	Yes	DIV	#2xR(F4)	M(…)	#3		Yes	Yes	#5

Registers

F0	F2	F4	F6	F8	F10	F12	F16
#3	M(R3+45)		#6	#4	#5		

Hardware-based Speculation

- Hardware-based speculation offers many advantages
  - Can incorporate hardware-based branch prediction
  - Maintains a precise exception model even for speculative instructions, since results don’t commit early
  - Does not require additional bookkeeping code
  - Does not depend on a good compiler
- Its main disadvantage is that it has substantial hardware requirements
- Schemes similar to what has been described have been implemented in the PowerPC 620, MIPS R10000, Intel P6, and AMD K5

Multiple Issue Processors

Issuing Multiple Instructions Per Cycle

- Why?
  - All of the schemes described so far can at best achieve 1 instruction/cycle
  - Increasing transistor budgets, parallelism in instruction streams (independent instructions) have pushed for multiple instructions/cycle

Two variations

- Superscalar: varying number of instructions/cycle (1 to 8),
  - static (requires compiler support for most benefit), or dynamic
  - with or without speculation (implies hardware scheduling)
  - IBM Power2, Sun UltraSPARC, Pentium III/4, DEC Alpha, HP 8000
- (Very) Long Instruction Words (V)LIW: fixed set of instructions (4-16)
  - scheduled by the compiler: put ops into wide templates
  - i860, IA64 (with some hardware support)

- New metric of performance: Instructions Per Clock cycle (IPC) vs. CPI

Statically Scheduled Superscalar MIPS Processor

Superscalar MIPS: 2 instructions; 1 FP op, 1 other

- Instruction issue
  - Fetch 64-bits/clock cycle
    - Need to handle cache-line complications
  - Hardware determines whether 0, 1, or 2 instructions can be issued
    - Can only issue 2nd instruction if 1st instruction issues
- Hazard detection
  - Likelihood of hazards between two instructions in a packet
    - Simple solution: treat this as a structural hazard (issue only 1 of them)
  - 1-instruction load delay can expand to 3-instruction delay in 2-way SS
    - 2^{nd} instruction in the pair can't use it without stall, nor 2 instructions in next slot
  - Branch delay becomes 2 cycles (2^{nd} inst. is branch) or 3 cycles (1^{st} inst. is branch)
- Execution
  - Additional (or pipelined) functional units to derive benefit
  - Additional port for FP registers to do FP load or FP store and FP op
  - More bypass paths

2-way Static Superscalar MIPS Pipeline

Instruction Type	Pipe Stages							
Integer	IF	ID	EX	MEM	WB			
FP	IF	ID	EX	MEM	WB			
Integer		IF	ID	EX	MEM	WB		
FP		IF	ID	EX	MEM	WB		
Integer			IF	ID	EX	MEM	WB	
FP			IF	ID	EX	MEM	WB	
Integer				IF	ID	EX	MEM	WB
FP				IF	ID	EX	MEM	WB

Pipeline Scheduling and Loop Unrolling

- Rare to find the ideal instruction mix of the previous slide
- Modern-day compilers apply several optimizations so as to expose Instruction Level Parallelism (ILP)

```
for (i=1000; i>0; i--)
    x[i] = x[i] + s

L1: L.D       F0, 0(R1)
    ADD.D     F4, F0, F2
    S.D       F4, 0(R1)
    DADDUI    R1, R1, #-8
    BNE       R1, R2, L1
```

	Instruction	Issue Cycle
L1	L.D F0, 0(R1)	1
	stall	2
	ADD.D F4, F0, F2	3
	stall	4
	stall	5
	S.D F4, 0(R1)	6
	DADDUI R1, R1, #-8	7
	stall	8
	BNE R1, R2, L1	9
	stall	10

3 cycles

Pipeline Scheduling and Loop Unrolling (cont'd)

- Loop unrolling optimization: Replicate loop body multiple times, adjusting the loop termination code

```
L1: L.D      F0, 0(R1)
    ADD.D    F4, F0, F2
    S.D      F4, 0(R1)
    L.D      F6, -8(R1)
    ADD.D    F8, F6, F2
    S.D      F8, -8(R1)
    L.D      F10, -16(R1)
    ADD.D    F12, F10, F2
    S.D      F12, -16(R1)
    L.D      F14, -24(R1)
    ADD.D    F16, F14, F2
    S.D      F16, -24(R1)
    DADDUI   R1, R1, #-32
    BNE      R1, R2, L1
```

	Instruction	Issue Cycle
L1	L.D F0, 0(R1)	1
	L.D F6, -8(R1)	2
	L.D F10, -16(R1)	3
	L.D F14, -24(R1)	4
	ADD.D F4, F0, F2	5
	ADD.D F8, F6, F2	6
	ADD.D F12, F10, F2	7
	ADD.D F16, F14, F2	8
	S.D F4, 0(R1)	9
	S.D F4, -8(R1)	10
	DADDUI R1, R1, #-32	11
	S.D F12, 16(R1)	12
	BNE R1, R2, L1	13
	S.D F16, 8(R1)	14

Loop Unrolling for Superscalar Processors

- Unroll loop 5 times to avoid extra 1-cycle delays in 2-way SS

```
L1: L.D       F0, 0(R1)
    ADD.D     F4, F0, F2
    S.D       F4, 0(R1)
    L.D       F6, -8(R1)
    ADD.D     F8, F6, F2
    S.D       F8, -8(R1)
    L.D       F10, -16(R1)
    ADD.D     F12, F10, F2
    S.D       F12, -16(R1)
    L.D       F14, -24(R1)
    ADD.D     F16, F14, F2
    S.D       F16, -24(R1)
    L.D       F18, -32(R1)
    ADD.D     F20, F18, F2
    S.D       F20, -32(R1)
    DADDUI    R1, R1, #-40
    BNE       R1, R2, L1
```

	Integer Instruction	FP Instruction	
L1	L.D F0, 0(R1)		1
	L.D F6, -8(R1)		2
	L.D F10, -16(R1)	ADD.D F4, F0, F2	3
	L.D F14, -24(R1)	ADD.D F8, F6, F2	4
	L.D F18, -32(R1)	ADD.D F12, F10, F2	5
	S.D F4, 0(R1)	ADD.D F16, F14, F2	6
	S.D F4, -8(R1)	ADD.D F20, F18, F2	7
	S.D F12, -16(R1)		8
	DADDUI R1, R1, #-40		9
	S.D F16, 16(R1)		10
	BNE R1, R2, L1		11
	S.D F20, 8(R1)		12

Loop unrolling: 10 to 3.5 cycles/iteration
SS: 3.5 cycles/iteration to 2.4 (1.5 times improvement)

Dynamic Scheduling in Superscalar Processors

- How to extend Tomasulo's algorithm?

- General solution:

  – Allow issue stage to work faster than rest of architecture
    - Achieved by both pipelining and widening the issue logic
    - Instructions issued, reservation stations allocated in order

  – Rest of the design already supports overlapped execution

  – Need wider CDB to store multiple results/cycle
    - Need to allow multiple instruction commits per clock cycle

Limits of Superscalar Processors

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations
  - No hazards

- Need more instructions to issue at the same time to get improved performance
  - However, greater difficulty of decode and issue
  - Even 2-way superscalar => examine 2 opcodes, 6 register specifiers, and decide if 1 or 2 instructions can issue
- Issue rates of modern processors vary between 2—8 instructions/cycle

- Motivation for VLIW and EPIC processors …

VLIW/EPIC Architectures

- Very Long Instruction Word (VLIW)
  - processor can initiate multiple operations per cycle

`r1 = L r4`	`r2 = Add r1,M`	`f1 = Mul f1,f2`	`r5 = Add r5,4`

  - Specified completely by the compiler (unlike superscalar machines)
  - Hardware is simple: issues the packet given it by the compiler
- Explicitly Parallel Instruction Computing (EPIC)
  - VLIW + new features
    - predication, rotating registers, speculations, etc.
- More later about compiling for VLIW/EPIC

Studies of the Limitations of ILP

- **Is there that much parallelism in programs?**
- Start off with a hardware model of an ideal processor

  1. Register renaming – infinite virtual registers and all WAW & WAR hazards are avoided

  2. Branch prediction – perfect; no mispredictions

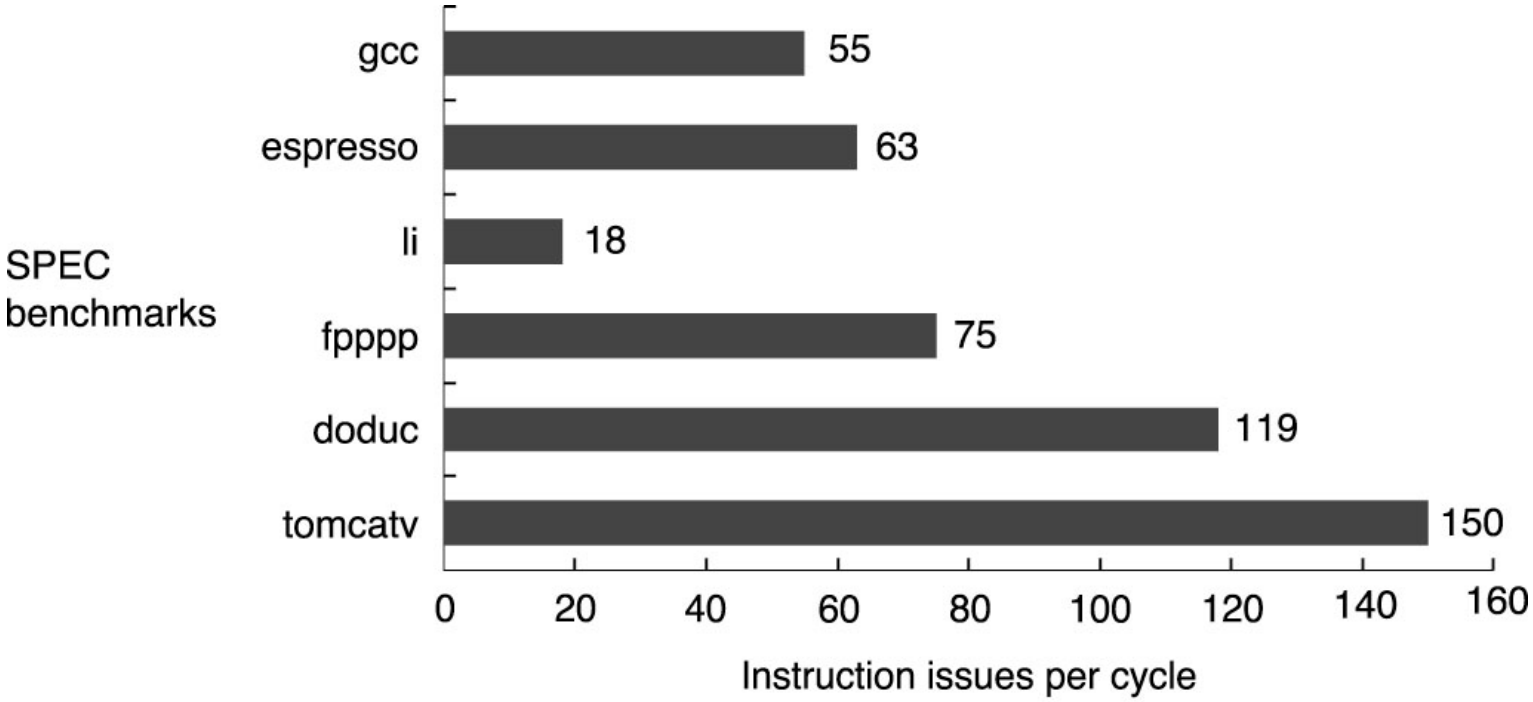
  3. Jump prediction – all jumps perfectly predicted

  2 and 3 => no control dependencies == machine with perfect speculation == an unbounded buffer of instructions available for execution

  4. Memory-address alias analysis –addresses are known and a load can be moved before a store provided addresses not equal

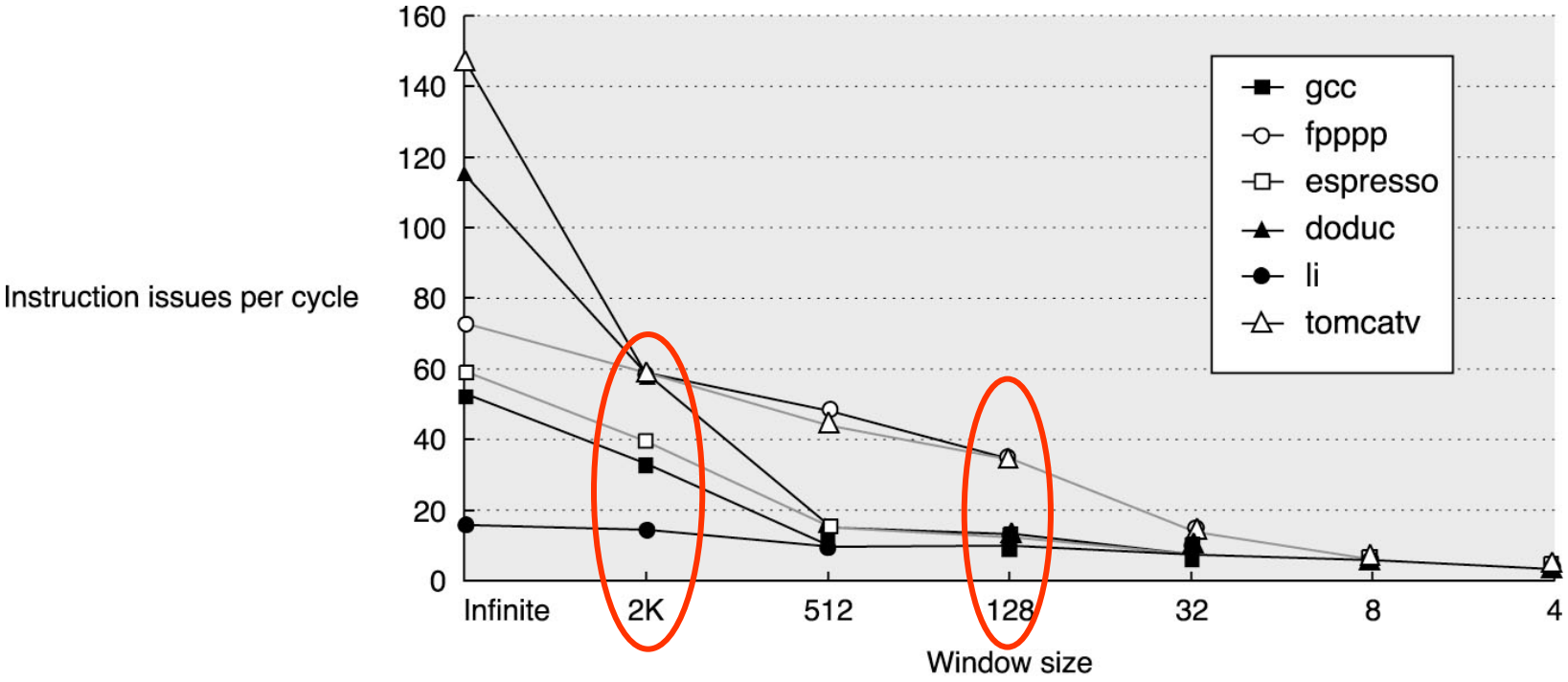
  1 and 4 => only true data dependencies
- 1 cycle latency for all instructions
- Perfect caches (loads and stores complete in one cycle)

ILP Limit for Six SPEC92 Benchmarks



- Fair bit of instruction-level parallelism, but how much of this stems from the ideal nature of assumptions
  - Infinite registers, perfect jump/branch prediction, perfect alias analysis

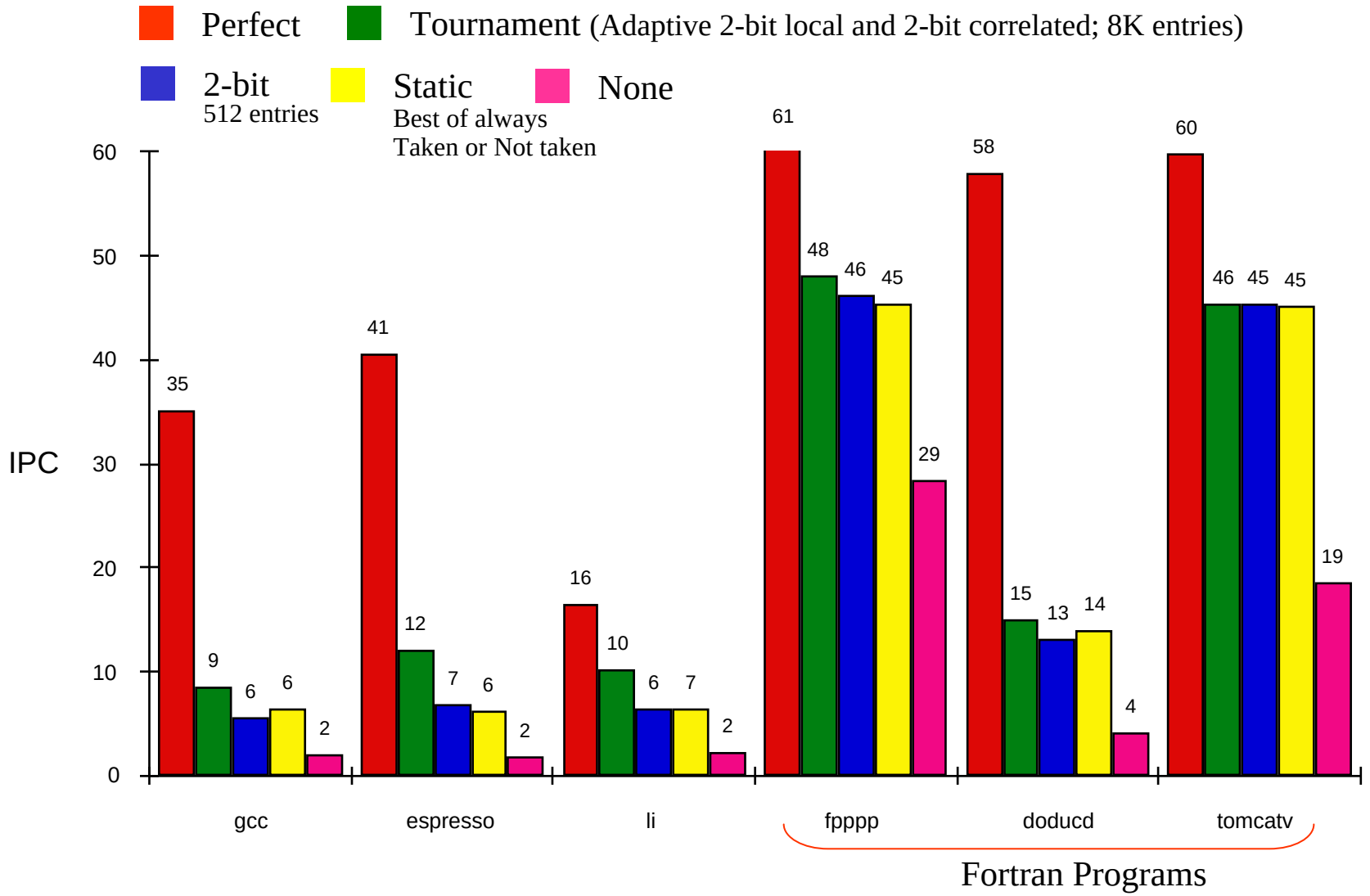
More Realistic Hardware: Limiting the Instruction Window



- The set of instructions that is examined for simultaneous executions is called the window
  - Limiting factor: operand checking
    - Scales as # of inst. completing/cycle * window size * # operands/inst.
  - FP programs have loop level parallelism which benefit from large window

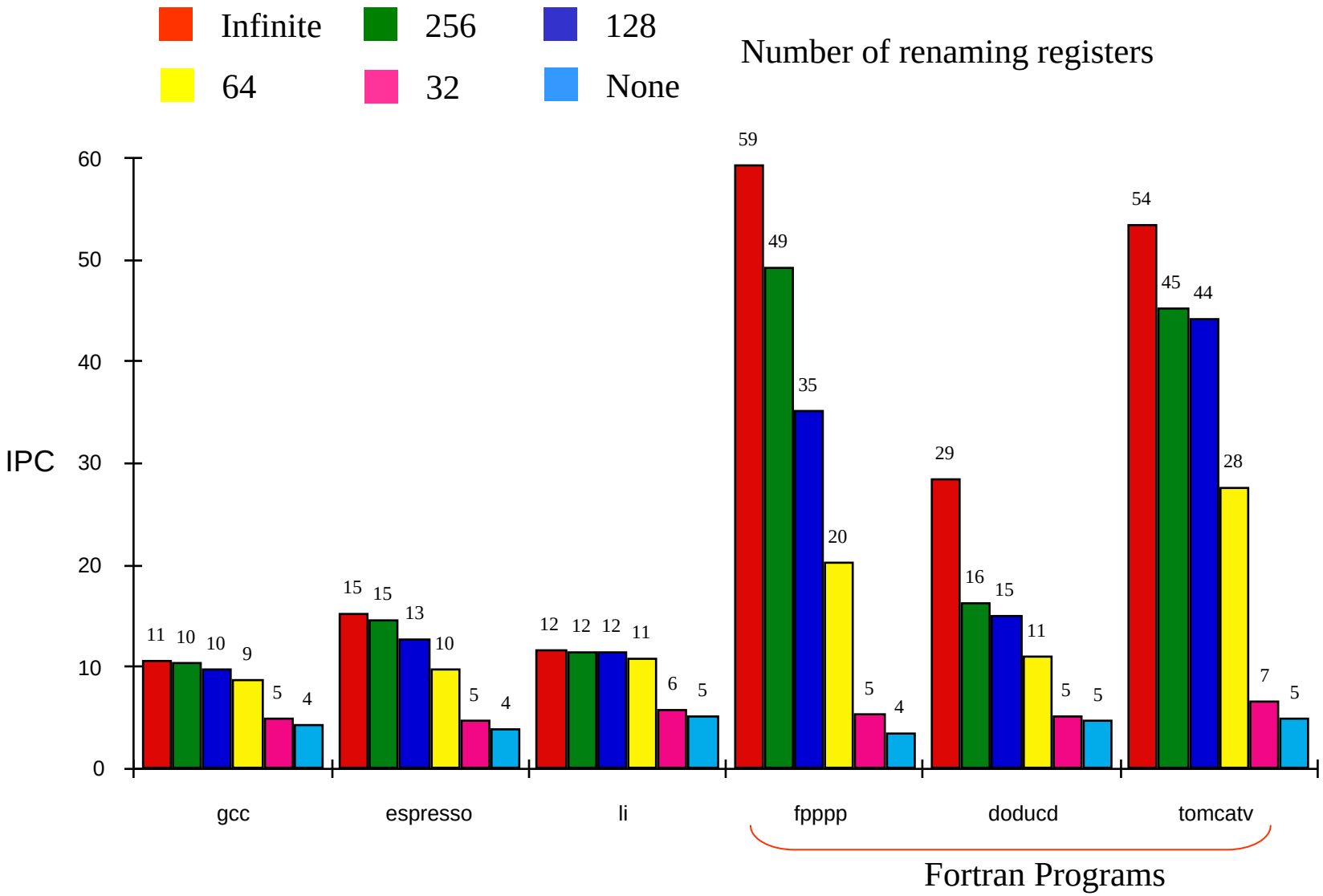
More Realistic Hardware: Branch Impact

Instr. window = 2000, issue width = 64



More Realistic HW: Register Impact

Instr. window = 2000, issue width = 64, bpred = 8K adaptive



Beyond the Limits of the Study

- More aggressive optimizations
  - Address value prediction and speculation (for memory accesses)
    - Can help achieve results similar to near-perfect alias analysis
  - Speculating on multiple paths
    - Reduces recovery costs (hopefully some path is useful), and exposes more ILP

- Even perfect model has some limitations
  - WAR and WAW hazards through memory
    - Can arise due to reuse of stack locations
  - Unnecessary dependences (i.e., compilers can do better than assumed)
    - E.g., dependence on loop control variable can be eliminated by loop unrolling
  - Overcoming the data flow limit
    - Recent idea: Value Prediction
    - Speculate that a register will have a certain value, and then recover if this speculation turns out to be false
      - Can speculate both data values and address values (for alias elimination)

A Different Perspective

- So far: Parallelism among instructions in a single thread of control
- What if we interleave instructions from multiple threads of control?
  - These instructions are independent (modulo thread synchronization)
    - Different register sets per thread
  - Overall program finishes earlier
    - Note that behavior of a single thread has not been improved
- Multithreading (later)