Lecture 3: Tomasulo Algorithm, VLIW, Dynamic Branch Prediction, Software Pipelining, and Limits to ILP

Prof. Fred Chong
ECS 250A Computer Architecture
Winter 1999

Review: Summary

- Instruction Level Parallelism (ILP) in SW or HW
- Loop level parallelism is easiest to see
- SW parallelism dependencies defined for program, hazards if HW cannot resolve
- SW dependencies/compiler sophistication determine if compiler can unroll loops
  - Memory dependencies hardest to determine
- HW exploiting ILP
  - Works when can't know dependence at run time
  - Code for one machine runs well on another
- Key idea of Scoreboard: Allow instructions behind stall to proceed (Decode => Issue instr & read operands)
  - Enables out-of-order execution => out-of-order completion
  - ID stage checked both for structural

Review: Three Parts of the Scoreboard

1. Instruction status—which of 4 steps the instruction is in
2. Functional unit status—Indicates the state of the functional unit (FU). 9 fields for each functional unit
   - Busy—Indicates whether the unit is busy or not
   - Op—Operation to perform in the unit (e.g., + or –)
   - Fi—Destination register
   - Fj, Fk—Source-register numbers
   - Qj, Qk—Functional units producing source registers Fj, Fk
   - Rj, Rk—Flags indicating when Fj, Fk are ready
3. Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that register

Review: Scoreboard Example Cycle 3

Instruction status

Instruction	j	k	Issue	Read operands	Execution complete	Write Result
LD F6	34+	R2	1	2	3	
LD F2	45+	R3				
MULT 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 for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock	FU	F0	F2	F4	F6	F8	F10	F12	...	F30
3					Integer					

- Issue MULT? No, stall on structural hazard

Review: Scoreboard Example Cycle 9

Instruction status

Instruction	j	k	Issue	Read operands	Execution complete	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT F0	F2	F4	6	9		
SUBD F8	F6	F2	7	9		
DIVD F10	F0	F6	8			
ADDD F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU	F0	F2	F4	F6	F8	F10	F12	...	F30
9		Mult1				Add	Divide			

- Read operands for MULT & SUBD? Issue ADDD?

Review: Scoreboard Example Cycle 17

Instruction status

Instruction	j	k	Issue	Read operands	Execution complete	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT F0	F2	F4	6	9		
SUBD F8	F6	F2	7	9	11	12
DIVD F10	F0	F6	8			
ADDD F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU	F0	F2	F4	F6	F8	F10	F12	...	F30
17		Mult1				Add	Divide			

- Write result of ADDD? No, WAR hazard

Review: Scoreboard Example Cycle 62

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	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

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	No								

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
62	FU								

- In-order issue; out-of-order execute & commit

Review: Scoreboard Summary

- Speedup 1.7 from compiler; 2.5 by hand BUT slow memory (no cache)
- Limitations of 6600 scoreboard
  - No forwarding (First write regsiter then read it)
  - Limited to instructions in basic block (small *window*)
  - Number of functional units(structural hazards)
  - Wait for WAR hazards
  - Prevent WAW hazards

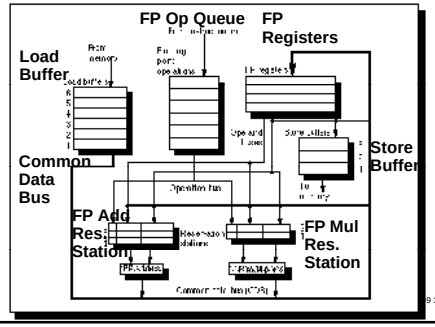
Another Dynamic Algorithm: Tomasulo Algorithm

- For IBM 360/91 about 3 years after CDC 6600 (1966)
- Goal: High Performance without special compilers
- Differences between IBM 360 & CDC 6600 ISA
  - IBM has only 2 register specifiers/instr vs. 3 in CDC 6600
  - IBM has 4 FP registers vs. 8 in CDC 6600
- Why Study? lead to Alpha 21264, HP 8000, MIPS 10000, Pentium II, PowerPC 604, …

Tomasulo Algorithm vs. Scoreboard

- Control & buffers distributed with Function Units (FU) vs. centralized in scoreboard;
  - FU buffers called "reservation stations"; have pending operands
- Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming ;
  - avoids WAR, WAW hazards
  - More reservation stations than registers, so can do optimizations compilers can't
- Results to FU from RS, not through registers, over Common Data Bus that broadcasts results to all FUs
- Load and Stores treated as FUs with RSs as well
- Integer instructions can go past branches, allowing FP ops beyond basic block in FP queue

Tomasulo Organization



Reservation Station Components

Op—Operation to perform in the unit (e.g., + or –)

Vj, Vk—Value of Source operands
- Store buffers has V field, result to be stored

Qj, Qk—Reservation stations producing source registers (value to be written)
- Note: No ready flags as in Scoreboard; Qj,Qk=0 => ready
- Store buffers only have Qi for RS producing result

Busy—Indicates reservation station or 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.

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

Tomasulo Example Cycle 0

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result
LD F6	34+	R2			
LD F2	45+	R3			
MULT F0	F2	F4			
SUBD F8	F6	F2			
DIVD F10	F0	F6			
ADDD F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU								

Tomasulo Example Cycle 1

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result
LD F6	34+	R2	1		
LD F2	45+	R3			
MULT F0	F2	F4			
SUBD F8	F6	F2			
DIVD F10	F0	F6			
ADDD F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU			Load1					

Tomasulo Example Cycle 2

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result
LD F6	34+	R2	1		
LD F2	45+	R3	2		
MULT F0	F2	F4			
SUBD F8	F6	F2			
DIVD F10	F0	F6			
ADDD F6	F8	F2			

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

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU	Load2		Load1					

Note: Unlike 6600, can have multiple loads outstanding

Tomasulo Example Cycle 3

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result
LD F6	34+	R2	1	3	
LD F2	45+	R3	2		
MULT F0	F2	F4	3		
SUBD F8	F6	F2			
DIVD F10	F0	F6			
ADDD F6	F8	F2			

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

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1				

- Note: registers names are removed (“renamed”) in Reservation Stations; MULT issued vs. scoreboard
- Load1 completing; what is waiting for Load1?

Tomasulo Example Cycle 4

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result
LD F6	34+	R2	1	3	4
LD F2	45+	R3	2	4	
MULT F0	F2	F4	3		
SUBD F8	F6	F2	4		
DIVD F10	F0	F6			
ADDD F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	Yes	SUBD	M(34+R2)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	Mult1	Load2		M(34+R2)	Add1			

- Load2 completing; what is waiting for it?

Tomasulo Example Cycle 5

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULT	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4						
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2							

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	2	Add1	Yes	SUBD	M(34+R2)	M(45+R3)		
	0	Add2	No					
		Add3	No					
	10	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU	Mult1	M(45+R3)		M(34+R2)	Add1	Mult2		

FTC.W99 19

Tomasulo Example Cycle 6

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULT	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4						
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6						

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	1	Add1	Yes	SUBD	M(34+R2)	M(45+R3)		
	0	Add2	Yes	ADDD		M(45+R3)	Add1	
		Add3	No					
	9	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	M(45+R3)		Add2	Add1	Mult2		

- **Issue ADDD here vs. scoreboard?**

FTC.W99 20

Tomasulo Example Cycle 7

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULT	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	7					
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6						

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	Yes	SUBD	M(34+R2)	M(45+R3)		
	0	Add2	Yes	ADDD		M(45+R3)	Add1	
		Add3	No					
	8	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(45+R3)		Add2	Add1	Mult2		

- **Add1 completing; what is waiting for it?**

FTC.W99 21

Tomasulo Example Cycle 8

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULT	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6						

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	2	Add2	Yes	ADDD	M()-M()	M(45+R3)		
	0	Add3	No					
	7	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1	M(45+R3)		Add2	M()-M()	Mult2		

FTC.W99 22

Tomasulo Example Cycle 9

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULT	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6						

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	1	Add2	Yes	ADDD	M()-M()	M(45+R3)		
	0	Add3	No					
	6	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1	M(45+R3)		Add2	M()-M()	Mult2		

FTC.W99 23

Tomasulo Example Cycle 10

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULT	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10					

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	Yes	ADDD	M()-M()	M(45+R3)		
	0	Add3	No					
	5	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	Mult1	M(45+R3)		Add2	M()-M()	Mult2		

- **Add2 completing; what is waiting for it?**

FTC.W99 24

Tomasulo Example Cycle 11

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULTD	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10	11				

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	No					
	0	Add3	No					
	4	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock			F0	F2	F4	F6	F8	F10	F12	...	F30
11			FU	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2		

- **Write result of ADDD here vs. scoreboard?**

Tomasulo Example Cycle 12

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULTD	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	6	7				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10	11				

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	No					
	0	Add3	No					
	3	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock			F0	F2	F4	F6	F8	F10	F12	...	F30
12			FU	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2		

- **Note: all quick instructions complete already**

Tomasulo Example Cycle 13

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULTD	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10	11				

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	No					
	0	Add3	No					
	2	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock			F0	F2	F4	F6	F8	F10	F12	...	F30
13			FU	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2		

Tomasulo Example Cycle 14

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULTD	F0	F2	F4	3				Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10	11				

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	No					
	0	Add3	No					
	1	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock			F0	F2	F4	F6	F8	F10	F12	...	F30
14			FU	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2		

Tomasulo Example Cycle 15

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULTD	F0	F2	F4	3	15			Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10	11				

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	No					
		Add3	No					
	0	Mult1	Yes	MULTD	M(45+R3)	R(F4)		
	0	Mult2	Yes	DIVD		M(34+R2)	Mult1	

Register result status

Clock			F0	F2	F4	F6	F8	F10	F12	...	F30
15			FU	Mult1	M(45+R3)		(M–M)+M()	M()–M()	Mult2		

- **Mult1 completing; what is waiting for it?**

Tomasulo Example Cycle 16

Instruction status

Instruction		j	k	Issue	Execution complete	Write Result			Busy	Address
LD	F6	34+	R2	1	3	4		Load1	No	
LD	F2	45+	R3	2	4	5		Load2	No	
MULTD	F0	F2	F4	3	15	16		Load3	No	
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10	11				

Reservation Stations

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	0	Add1	No					
	0	Add2	No					
		Add3	No					
	0	Mult1	No					
	40	Mult2	Yes	DIVD	M*F4	M(34+R2)		

Register result status

Clock			F0	F2	F4	F6	F8	F10	F12	...	F30
16			FU	M*F4	M(45+R3)		(M–M)+M()	M()–M()	Mult2		

- **Note: Just waiting for divide**

Tomasulo Example Cycle 55

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result	Busy	Address
LD F6	34+	R2	1	3	4	Load1 No	
LD F2	45+	R3	2	4	5	Load2 No	
MULT F0	F2	F4	3	15	16	Load3 No	
SUBD F8	F6	F2	4	7	8		
DIVD F10	F0	F6	5				
ADDD F6	F8	F2	6	10	11		

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(34+R2)		

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
55	FU	M*F4	M(45+R3)			(M–M)+M()	M()–M()	Mult2	

Tomasulo Example Cycle 56

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result	Busy	Address
LD F6	34+	R2	1	3	4	Load1 No	
LD F2	45+	R3	2	4	5	Load2 No	
MULT F0	F2	F4	3	15	16	Load3 No	
SUBD F8	F6	F2	4	7	8		
DIVD F10	F0	F6	5	56			
ADDD F6	F8	F2	6	10	11		

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(34+R2)		

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(45+R3)			(M–M)+M()	M()–M()	Mult2	

- Mult 2 completing; what is waiting for it?

Tomasulo Example Cycle 57

Instruction status

Instruction	j	k	Issue	Execution complete	Write Result	Busy	Address
LD F6	34+	R2	1	3	4	Load1 No	
LD F2	45+	R3	2	4	5	Load2 No	
MULT F0	F2	F4	3	15	16	Load3 No	
SUBD F8	F6	F2	4	7	8		
DIVD F10	F0	F6	5	56	57		
ADDD F6	F8	F2	6	10	11		

Reservation Stations

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	No					
0	Mult2	No					

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
57	FU	M*F4	M(45+R3)			(M–M)+M()	M()–M()	M*F4/M	

- Again, in-oder issue, out-of-order execution, completion

Compare to Scoreboard Cycle 62

Instruction status

Instruction	j	k	Issue	Read operands	Execution complete	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULT 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

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	No								

Register result status

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
62	FU								

- Why takes longer on Scoreboard/6600?

Tomasulo v. Scoreboard (IBM 360/91 v. CDC 6600)

Pipelined Functional Units (6 load, 3 store, 3 +, 2 x/÷)	Multiple Functional Units (1 load/store, 1 + , 2 x, 1 ÷)
window size: 14 instructions	5 instructions
No issue on structural hazard	same
WAR: renaming avoids	stall completion
WAW: renaming avoids	stall completion
Broadcast results from FU	Write/read registers
Control: reservation stations	central scoreboard

Tomasulo Drawbacks

- Complexity
  - delays of 360/91, MIPS 10000, IBM 620?
- Many associative stores (CDB) at high speed
- Performance limited by Common Data Bus
  - Multiple CDBs => more FU logic for parallel assoc stores

Tomasulo Loop Example

```
Loop: LD      F0    0     R1
      MULTD   F4    F0    F2
      SD      F4    0     R1
      SUBI    R1    R1    #8
      BNEZ    R1    Loop
```

- Assume Multiply takes 4 clocks
- Assume first load takes 8 clocks (cache miss?), second load takes 4 clocks (hit)
- To be clear, will show clocks for SUBI, BNEZ
- Reality, integer instructions ahead

FTC.W99 37

Loop Example Cycle 0

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result
LD F0		0 R1	1			
MULT F4		F0 F2	1			
SD F4		0 R1	1			
LD F0		0 R1	2			
MULT F4		F0 F2	2			
SD F4		0 R1	2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	
Store1	No	
Store2	No	
Store3	No	

Qi:

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						LD F0 0 R1
0	Add2	No						MULT F4 F0 F2
0	Add3	No						SD F4 0 R1
0	Mult1	No						SUBI R1 R1 #8
0	Mult2	No						BNEZ R1 Loop

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
0	80	Qi							

FTC.W99 38

Loop Example Cycle 1

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result
LD F0		0 R1	1	1		
MULT F4		F0 F2	1			
SD F4		0 R1	1			
LD F0		0 R1	2			
MULT F4		F0 F2	2			
SD F4		0 R1	2			

	Busy	Address
Load1	Yes	80
Load2	No	
Load3	No	
Store1	No	
Store2	No	
Store3	No	

Qi:

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						LD F0 0 R1
0	Add2	No						MULT F4 F0 F2
0	Add3	No						SD F4 0 R1
0	Mult1	No						SUBI R1 R1 #8
0	Mult2	No						BNEZ R1 Loop

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
1	80	Qi	Load1						

FTC.W99 39

Loop Example Cycle 2

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result
LD F0		0 R1	1	1		
MULT F4		F0 F2	1	2		
SD F4		0 R1	1			
LD F0		0 R1	2			
MULT F4		F0 F2	2			
SD F4		0 R1	2			

	Busy	Address
Load1	Yes	80
Load2	No	
Load3	No	
Store1	No	
Store2	No	
Store3	No	

Qi:

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						LD F0 0 R1
0	Add2	No						MULT F4 F0 F2
0	Add3	No						SD F4 0 R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI R1 R1 #8
0	Mult2	No						BNEZ R1 Loop

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
2	80	Qi	Load1		Mult1				

FTC.W99 40

Loop Example Cycle 3

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result
LD F0		0 R1	1	1		
MULT F4		F0 F2	1	2		
SD F4		0 R1	1	3		
LD F0		0 R1	2			
MULT F4		F0 F2	2			
SD F4		0 R1	2			

	Busy	Address
Load1	Yes	80
Load2	No	
Load3	No	
Store1	Yes	80
Store2	No	
Store3	No	

Qi: Mult1

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						LD F0 0 R1
0	Add2	No						MULT F4 F0 F2
0	Add3	No						SD F4 0 R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI R1 R1 #8
0	Mult2	No						BNEZ R1 Loop

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
3	80	Qi	Load1		Mult1				

- Note: MULT1 has no registers names in RS

FTC.W99 41

Loop Example Cycle 4

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result
LD F0		0 R1	1	1		
MULT F4		F0 F2	1	2		
SD F4		0 R1	1	3		
LD F0		0 R1	2			
MULT F4		F0 F2	2			
SD F4		0 R1	2			

	Busy	Address
Load1	Yes	80
Load2	No	
Load3	No	
Store1	Yes	80
Store2	No	
Store3	No	

Qi: Mult1

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:
0	Add1	No						LD F0 0 R1
0	Add2	No						MULT F4 F0 F2
0	Add3	No						SD F4 0 R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI R1 R1 #8
0	Mult2	No						BNEZ R1 Loop

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12 ...	F30
4	72	Qi	Load1		Mult1				

FTC.W99 42

Loop Example Cycle 5

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0		0 R1	1	1			Load1	Yes	80	
MULT F4		F0 F2	1	2			Load2	No		
SD F4		0 R1	1	3			Load3	No		Qi
LD F0		0 R1	2				Store1	Yes	80	Mult1
MULT F4		F0 F2	2				Store2	No		
SD F4		0 R1	2				Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
0	Add1	No						LD	F0	0	R1
0	Add2	No						MULT	F4	F0	F2
0	Add3	No						SD	F4	0	R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI	R1	R1	#8
0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
5	72	Qi	Load1		Mult1					

Loop Example Cycle 6

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0		0 R1	1	1			Load1	Yes	80	
MULT F4		F0 F2	1	2			Load2	Yes	72	
SD F4		0 R1	1	3			Load3	No		Qi
LD F0		0 R1	2	6			Store1	Yes	80	Mult1
MULT F4		F0 F2	2				Store2	No		
SD F4		0 R1	2				Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
0	Add1	No						LD	F0	0	R1
0	Add2	No						MULT	F4	F0	F2
0	Add3	No						SD	F4	0	R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI	R1	R1	#8
0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
6	72	Qi	Load2		Mult1					

• Note: F0 never sees Load1 result

Loop Example Cycle 7

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0		0 R1	1	1			Load1	Yes	80	
MULT F4		F0 F2	1	2			Load2	Yes	72	
SD F4		0 R1	1	3			Load3	No		Qi
LD F0		0 R1	2	6			Store1	Yes	80	Mult1
MULT F4		F0 F2	2	7			Store2	No		
SD F4		0 R1	2				Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
0	Add1	No						LD	F0	0	R1
0	Add2	No						MULT	F4	F0	F2
0	Add3	No						SD	F4	0	R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI	R1	R1	#8
0	Mult2	Yes	MULTD		R(F2)	Load2		BNEZ	R1	Loop	

Register result status:

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
7	72	Qi	Load2		Mult2					

• Note: MULT2 has no registers names in RS

Loop Example Cycle 8

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0		0 R1	1	1			Load1	Yes	80	
MULT F4		F0 F2	1	2			Load2	Yes	72	
SD F4		0 R1	1	3			Load3	No		Qi
LD F0		0 R1	2	6			Store1	Yes	80	Mult1
MULT F4		F0 F2	2	7			Store2	Yes	72	Mult2
SD F4		0 R1	2	8			Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
0	Add1	No						LD	F0	0	R1
0	Add2	No						MULT	F4	F0	F2
0	Add3	No						SD	F4	0	R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI	R1	R1	#8
0	Mult2	Yes	MULTD		R(F2)	Load2		BNEZ	R1	Loop	

Register result status:

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
8	72	Qi	Load2		Mult2					

Loop Example Cycle 9

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0		0 R1	1	1	9		Load1	Yes	80	
MULT F4		F0 F2	1	2			Load2	Yes	72	
SD F4		0 R1	1	3			Load3	No		Qi
LD F0		0 R1	2	6			Store1	Yes	80	Mult1
MULT F4		F0 F2	2	7			Store2	Yes	72	Mult2
SD F4		0 R1	2	8			Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
0	Add1	No						LD	F0	0	R1
0	Add2	No						MULT	F4	F0	F2
0	Add3	No						SD	F4	0	R1
0	Mult1	Yes	MULTD		R(F2)	Load1		SUBI	R1	R1	#8
0	Mult2	Yes	MULTD		R(F2)	Load2		BNEZ	R1	Loop	

Register result status:

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
9	64	Qi	Load2		Mult2					

• Load1 completing; what is waiting for it?

Loop Example Cycle 10

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0		0 R1	1	1	9	10	Load1	No		
MULT F4		F0 F2	1	2			Load2	Yes	72	
SD F4		0 R1	1	3			Load3	No		Qi
LD F0		0 R1	2	6	10		Store1	Yes	80	Mult1
MULT F4		F0 F2	2	7			Store2	Yes	72	Mult2
SD F4		0 R1	2	8			Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
0	Add1	No						LD	F0	0	R1
0	Add2	No						MULT	F4	F0	F2
0	Add3	No						SD	F4	0	R1
4	Mult1	Yes	MULTD	M(80)	R(F2)			SUBI	R1	R1	#8
0	Mult2	Yes	MULTD		R(F2)	Load2		BNEZ	R1	Loop	

Register result status:

Clock	R1		F0	F2	F4	F6	F8	F10	F12 ...	F30
10	64	Qi	Load2		Mult2					

• Load2 completing; what is waiting for it?

Loop Example Cycle 11

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address		
LD F0	0	R1	1	1	9	10	Load1	No			
MULT F4	F0	F2	1	2			Load2	No			
SD F4	0	R1	1	3			Load3	Yes	64	Qi	
LD F0	0	R1	2	6	10	11	Store1	Yes	80	Mult1	
MULT F4	F0	F2	2	7			Store2	Yes	72	Mult2	
SD F4	0	R1	2	8			Store3	No			

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	3	Mult1	Yes	MULTD	M(80)	R(F2)			SUBI	R1	R1	#8
	4	Mult2	Yes	MULTD	M(72)	R(F2)			BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
11	64	Qi	Load3		Mult2					

FTC.W99 49

Loop Example Cycle 12

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address		
LD F0	0	R1	1	1	9	10	Load1	No			
MULT F4	F0	F2	1	2			Load2	No			
SD F4	0	R1	1	3			Load3	Yes	64	Qi	
LD F0	0	R1	2	6	10	11	Store1	Yes	80	Mult1	
MULT F4	F0	F2	2	7			Store2	Yes	72	Mult2	
SD F4	0	R1	2	8			Store3	No			

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	2	Mult1	Yes	MULTD	M(80)	R(F2)			SUBI	R1	R1	#8
	3	Mult2	Yes	MULTD	M(72)	R(F2)			BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
12	64	Qi	Load3		Mult2					

FTC.W99 50

Loop Example Cycle 13

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address		
LD F0	0	R1	1	1	9	10	Load1	No			
MULT F4	F0	F2	1	2			Load2	No			
SD F4	0	R1	1	3			Load3	Yes	64	Qi	
LD F0	0	R1	2	6	10	11	Store1	Yes	80	Mult1	
MULT F4	F0	F2	2	7			Store2	Yes	72	Mult2	
SD F4	0	R1	2	8			Store3	No			

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	1	Mult1	Yes	MULTD	M(80)	R(F2)			SUBI	R1	R1	#8
	2	Mult2	Yes	MULTD	M(72)	R(F2)			BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
13	64	Qi	Load3		Mult2					

FTC.W99 51

Loop Example Cycle 14

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address		
LD F0	0	R1	1	1	9	10	Load1	No			
MULT F4	F0	F2	1	2	14		Load2	No			
SD F4	0	R1	1	3			Load3	Yes	64	Qi	
LD F0	0	R1	2	6	10	11	Store1	Yes	80	Mult1	
MULT F4	F0	F2	2	7			Store2	Yes	72	Mult2	
SD F4	0	R1	2	8			Store3	No			

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	Yes	MULTD	M(80)	R(F2)			SUBI	R1	R1	#8
	1	Mult2	Yes	MULTD	M(72)	R(F2)			BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
14	64	Qi	Load3		Mult2					

- **Mult1 completing; what is waiting for it?**

FTC.W99 52

Loop Example Cycle 15

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address		
LD F0	0	R1	1	1	9	10	Load1	No			
MULT F4	F0	F2	1	2	14	15	Load2	No			
SD F4	0	R1	1	3			Load3	Yes	64	Qi	
LD F0	0	R1	2	6	10	11	Store1	Yes	80	M(80)*R(F	
MULT F4	F0	F2	2	7	15		Store2	Yes	72	Mult2	
SD F4	0	R1	2	8			Store3	No			

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	No						SUBI	R1	R1	#8
	0	Mult2	Yes	MULTD	M(72)	R(F2)			BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
15	64	Qi	Load3		Mult2					

- **Mult2 completing; what is waiting for it?**

FTC.W99 53

Loop Example Cycle 16

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address		
LD F0	0	R1	1	1	9	10	Load1	No			
MULT F4	F0	F2	1	2	14	15	Load2	No			
SD F4	0	R1	1	3			Load3	Yes	64	Qi	
LD F0	0	R1	2	6	10	11	Store1	Yes	80	M(80)*R(F	
MULT F4	F0	F2	2	7	15	16	Store2	Yes	72	M(72)*R(7	
SD F4	0	R1	2	8			Store3	No			

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	Yes	MULTD		R(F2)	Load3		SUBI	R1	R1	#8
	0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
16	64	Qi	Load3		Mult1					

FTC.W99 54

Loop Example Cycle 17

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0	0	R1	1	1	9	10	Load1	No		
MULT F4	F0	F2	1	2	14	15	Load2	No		
SD F4	0	R1	1	3			Load3	Yes	64	Qi
LD F0	0	R1	2	6	10	11	Store1	Yes	80	M(80)*R(F
MULT F4	F0	F2	2	7	15	16	Store2	Yes	72	M(72)*R(7
SD F4	0	R1	2	8			Store3	Yes	64	Mult1

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	Yes	MULTD		R(F2)	Load3		SUBI	R1	R1	#8
	0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
17	64	Qi	Load3		Mult1					

FTC.W99 55

Loop Example Cycle 18

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0	0	R1	1	1	9	10	Load1	No		
MULT F4	F0	F2	1	2	14	15	Load2	No		
SD F4	0	R1	1	3	18		Load3	Yes	64	Qi
LD F0	0	R1	2	6	10	11	Store1	Yes	80	M(80)*R(F
MULT F4	F0	F2	2	7	15	16	Store2	Yes	72	M(72)*R(7
SD F4	0	R1	2	8			Store3	Yes	64	Mult1

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	Yes	MULTD		R(F2)	Load3		SUBI	R1	R1	#8
	0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
18	56	Qi	Load3		Mult1					

FTC.W99 56

Loop Example Cycle 19

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0	0	R1	1	1	9	10	Load1	No		
MULT F4	F0	F2	1	2	14	15	Load2	No		
SD F4	0	R1	1	3	18	19	Load3	Yes	64	Qi
LD F0	0	R1	2	6	10	11	Store1	No		
MULT F4	F0	F2	2	7	15	16	Store2	Yes	72	M(72)*R(7
SD F4	0	R1	2	8			Store3	Yes	64	Mult1

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	Yes	MULTD		R(F2)	Load3		SUBI	R1	R1	#8
	0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
19	56	Qi	Load3		Mult1					

FTC.W99 57

Loop Example Cycle 20

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0	0	R1	1	1	9	10	Load1	No		
MULT F4	F0	F2	1	2	14	15	Load2	No		
SD F4	0	R1	1	3	18	19	Load3	Yes	64	Qi
LD F0	0	R1	2	6	10	11	Store1	No		
MULT F4	F0	F2	2	7	15	16	Store2	Yes	72	M(72)*R(7
SD F4	0	R1	2	8	20		Store3	Yes	64	Mult1

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	Yes	MULTD		R(F2)	Load3		SUBI	R1	R1	#8
	0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
20	56	Qi	Load3		Mult1					

FTC.W99 58

Loop Example Cycle 21

Instruction status:

Instruction	j	k	iteration	Issue	Execution complete	Write Result		Busy	Address	
LD F0	0	R1	1	1	9	10	Load1	No		
MULT F4	F0	F2	1	2	14	15	Load2	No		
SD F4	0	R1	1	3	18	19	Load3	Yes	64	Qi
LD F0	0	R1	2	6	10	11	Store1	No		
MULT F4	F0	F2	2	7	15	16	Store2	No		
SD F4	0	R1	2	8	20	21	Store3	Yes	64	Mult1

Reservation Stations:

	Time	Name	Busy	Op	S1 Vj	S2 Vk	RS for j Qj	RS for k Qk	Code:			
	0	Add1	No						LD	F0	0	R1
	0	Add2	No						MULT	F4	F0	F2
	0	Add3	No						SD	F4	0	R1
	0	Mult1	Yes	MULTD		R(F2)	Load3		SUBI	R1	R1	#8
	0	Mult2	No						BNEZ	R1	Loop	

Register result status:

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
21	56	Qi	Load3		Mult1					

FTC.W99 59

Tomasulo Summary

- Reservations stations: renaming to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of Scoreboard
  - Allows loop unrolling in HW
- Not limited to basic blocks (integer units gets ahead, beyond branches)
- Helps cache misses as well
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 descendants are Pentium II; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264

FTC.W99 60

Superscalar Processors

- What support do we need to fetch & issue multiple instructions/cycle? (that we have already seen)
  - 1. Keep track of dependencies between instructions
  - 2. Out-of-order execution

Review: Unrolled Loop that Minimizes Stalls for Scalar

```
1 Loop: LD     F0,0(R1)
2       LD     F6,-8(R1)
3       LD     F10,-16(R1)
4       LD     F14,-24(R1)
5       ADDD   F4,F0,F2
6       ADDD   F8,F6,F2
7       ADDD   F12,F10,F2
8       ADDD   F16,F14,F2
9       SD     0(R1),F4
10      SD     -8(R1),F8
11      SD     -16(R1),F12
12      SUBI   R1,R1,#32
13      BNEZ   R1,LOOP
14      SD     8(R1),F16    ; 8-32 = -24
```

LD to ADDD: 1 Cycle
ADDD to SD: 2 Cycles

14 clock cycles, or 3.5 per iteration

Loop Unrolling in Superscalar

	Integer instruction	*FP instruction*	*Clock cycle*
Loop:	LD F0,0(R1)		1
	LD F6,-8(R1)		2
	LD F10,-16(R1)	ADDD F4,F0,F2	3
	LD F14,-24(R1)	ADDD F8,F6,F2	4
	LD F18,-32(R1)	ADDD F12,F10,F2	5
	SD 0(R1),F4	ADDD F16,F14,F2	6
	SD -8(R1),F8	ADDD F20,F18,F2	7
	SD -16(R1),F12		8
	SD -24(R1),F16		9
	SUBI R1,R1,#40		10
	BNEZ R1,LOOP		11
	SD -32(R1),F20		12

- Unrolled 5 times to avoid delays (+1 due to SS)
- 12 clocks, or 2.4 clocks per iteration (1.5X)

Dynamic Branch Prediction

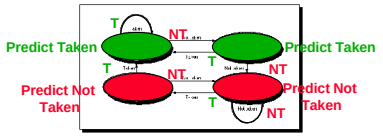
- With out-of-order or superscalars, branches affect performance more. Why?
  - Because even though a branch only wastes, say, 2 cycles, that now means 2*(width) instructions rather than just 2.
- How do we judge different prediction schemes?
  - Space
  - Performance - f(accuracy, cost of misprediction)

One-bit predictor

- Keep 1-bit for each branch
  - tells whether it was taken last time
- Store lower bits of address in table
  - no address check - may be incorrect branch!
- If it stores lower 8 bits, what is the table size?
  - 2^8
- What is the accuracy of a nine-iteration loop?
  - 7/9 (It mispredicts on first and last iterations)
- What is the penalty in DLX?
  - 1 cycle

Dynamic Branch Prediction

- Solution: 2-bit scheme where change prediction only if get misprediction *twice:* (Figure 4.13, p. 264)



- Red: stop, not taken
- Green: go, taken

Analyze BHT

- How does it do with the 9-iteration loop?
  - 1/9 accuracy - half the mispredictions as 1-bit!
- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when index the table
- 4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%
- 4096 about as good as infinite table (in Alpha 211164)

N-bit saturating counter

- N bit counter can take from 0 to (2^n)-1
- if count >= 2^(n-1), branch predict taken
- if taken, increment; if untaken, decrement
- How does 2-bit counter differ from 2-bit BHT?
  - 2-bit BHT must be wrong twice to switch predictions. For the 2-bit counter, it could switch predictions every time for a branch that switches every time.

Can we do better?

- If (d==0)
- { …….; d = 1;……}
- if (d==1)
- { blah, blah, blah}

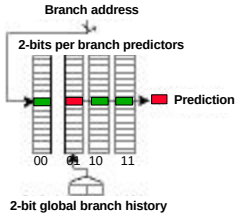
Correlating Branches

- Hypothesis: recent branches are correlated; that is, behavior of recently executed branches affects prediction of current branch
- Idea: record m most recently executed branches as taken or not taken, and use that pattern to select the proper branch history table
- In general, (m,n) predictor means record last m branches to select between 2^m history tables each with n-bit counters
  - What, then was the 1-bit predictor? (0,1)
  - 2-bit predictor? (0,2)

Correlating Branches

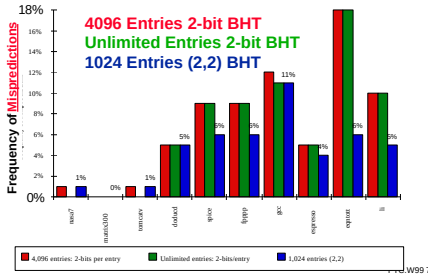
(2,2) predictor

- Then behavior of recent branches selects between, say, four predictions of next branch, updating just that prediction
- If b3-not taken, b4-taken, this shows entry for b5.
- Where is entry for b5 if b3-taken, b4-not taken?
- What's relationship between this and 2-bit?



Accuracy of Different Schemes

(Figure 4.21, p. 272)



Re-evaluating Correlation

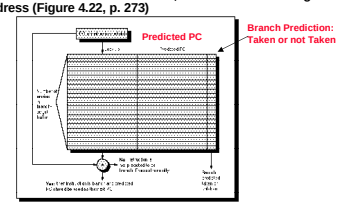
- Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:

program	branch %	static	# = 90%
compress	14%	236	13
eqntott	25%	494	5
gcc	15%	9531	2020
mpeg	10%	5598	532
real gcc	13%	17361	3214

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?
- What does this say about benchmarks?
  - Not always representative of real programs

Need Address at Same Time as Prediction

- Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)
  - Note: must check for branch match now, since can't use wrong branch address (Figure 4.22, p. 273)



- Return instruction addresses predicted with stack

Dynamic Branch Prediction Summary

- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch
- Branch Target Buffer: include branch address & prediction

Predicated instructions

- Useful for superscalar processors or not-taken code with only one line:
  - Why is a superscalar any different?
    - » Because we're trying to put several instructions in line, and branches let us only put one inst in that line.
  - If (a==0) b=c;
- Create single instruction which means:
  - if (cond) exp else nop;
  - Takes this branch out completely - no misprediction
  - What is net effect if cond = false? nop
- Drawbacks:
  - Stall if cond evaluated late

Predicated Instructions

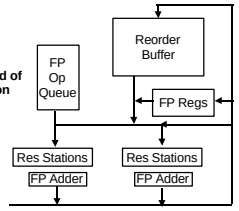
- Superscalar - instruction reordering
  - 2 instructions/cycle - one memory, one ALU
  - 1 instruction if it is a branch
  - LW R1, 40(R2) ADD R3, R4, R5
  - ADD R6, R3, R7
  - BEQZ R10, L
  - LW R8, 20(R10)
  - LW R9, 0(R8)
- How can we get rid of stall between two LW's?
  - LW R1, 40(R2) ADD R3, R4, R5
  - if(R10) LW R8,20(R10) ADD R6, R3, R7
  - BEQZ R10, L
  - LW R9, 0(R8)

Speculation

- *Speculation*: allow an instruction to execute *without* any consequences (including exceptions) if branch is not actually taken ("HW undo"); called "boosting"
- Combine branch prediction with dynamic scheduling to execute before branches resolved
- Separate *speculative* bypassing of results from real bypassing of results
  - When instruction no longer speculative, write boosted results (*instruction commit*) or discard boosted results
  - execute out-of-order but commit in-order to prevent irrevocable action (update state or exception) until instruction commits

HW support for Speculation

- Need HW buffer for results of uncommitted instructions: *reorder buffer*
  - 3 fields: instr, destination, value
  - Reorder buffer can be operand source => more registers like RS
  - Use reorder buffer number instead of reservation station when execution completes
  - Supplies operands between execution complete & commit
  - Once operand commits, result is put into register
  - Instructions commit
  - As a result, it's easy to undo speculated instructions on mispredicted branches or on exceptions - What do you do?
    - » You flush the reorder buffer & pipeline



Four Steps of Speculative Tomasulo Algorithm

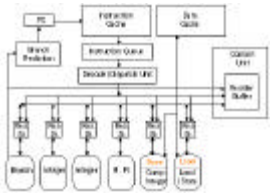
1. Issue—get instruction from FP Op Queue
   - If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")
2. Execution—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")
3. Write result—finish execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
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

Renaming Registers

- Common variation of speculative design
- Reorder buffer keeps instruction information but not the result
- Extend register file with extra renaming registers to hold speculative results
- Rename register allocated at issue; result into rename register on execution complete; rename register into real register on commit
- Operands read either from register file (real or speculative) or via Common Data Bus
- Advantage: operands are always from single source (extended register file)

Dynamic Scheduling in PowerPC 604 and Pentium Pro

- Both In-order Issue, Out-of-order execution, In-order Commit



Pentium Pro more like a scoreboard since central control vs. distributed

Dynamic Scheduling in PowerPC 604 and Pentium Pro

Parameter	PPC	PPro
Max. instructions issued/clock	4	3
Max. instr. complete exec./clock	6	5
Max. instr. commited/clock	6	3
Window (Instrs in reorder buffer)	16	40
Number of reservations stations	12	20
Number of rename registers	8int/12FP	40
No. integer functional units (FUs)	2	2
No. floating point FUs	1	1
No. branch FUs	1	1
No. complex integer FUs	1	0
No. memory FUs	1 1 load +1 store	

Q: How pipeline 1 to 17 byte x86 instructions?

Dynamic Scheduling in Pentium Pro

- PPro doesn't pipeline 80x86 instructions
- PPro decode unit translates the Intel instructions into 72-bit micro-operations (- DLX)
- Sends micro-operations to reorder buffer & reservation stations
- Takes 1 clock cycle to determine length of 80x86 instructions + 2 more to create the micro-operations
- 12-14 clocks in total pipeline (- 3 state machines)
- Many instructions translate to 1 to 4 micro-operations
- Complex 80x86 instructions are executed by a conventional microprogram (8K x 72 bits) that issues long sequences of micro-operations

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Two variations
- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, HP 8000
- (Very) Long Instruction Words (V)LIW: fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates
  - Joint HP/Intel agreement in 1999/2000?
  - Intel Architecture-64 (IA-64) 64-bit address
  - Style: "Explicitly Parallel Instruction Computer (EPIC)"
- Anticipated success lead to use of Instructions Per Clock cycle (IPC) vs. CPI

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar DLX: 2 instructions, 1 FP & 1 anything else
  - Fetch 64-bits/clock cycle; Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues
  - More ports for FP registers to do FP load & FP op in a pair

Type	*PipeStages*					
Int. instruction	IF	ID	EX	MEM	WB	
FP instruction	IF	ID	EX	MEM	WB	
Int. instruction		IF	ID	EX	MEM	WB
FP instruction		IF	ID	EX	MEM	WB
Int. instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

- 1 cycle load delay expands to 3 instructions in SS
  - instruction in right half can't use it, nor instructions in next slot

Multiple Issue Challenges

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations
  - No hazards
- If more instructions issue at same time, greater difficulty of decode and issue
  - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue
- VLIW: tradeoff instruction space for simple decoding
  - The long instruction word has room for many operations
  - By definition, all the operations the compiler puts in the long instruction word are independent => execute in parallel
  - E.g., 2 integer operations, 2 FP ops, 2 Memory refs, 1 branch
    - » 16 to 24 bits per field => 7*16 or 112 bits to 7*24 or 168 bits wide
  - Need compiling technique that schedules across several branches

Loop Unrolling in VLIW

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP op. 2*	*Int. op/ branch*	*Clock*
LD F0,0(R1)	LD F6,-8(R1)				1
LD F10,-16(R1)	LD F14,-24(R1)				2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2		4
		ADDD F20,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16				7
SD -32(R1),F20	SD -40(R1),F24			SUBI R1,R1,#48	8
SD -0(R1),F28				BNEZ R1,LOOP	9

Unrolled 7 times to avoid delays

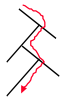
7 results in 9 clocks, or 1.3 clocks per iteration (1.8X)

Average: 2.5 ops per clock, 50% efficiency

Note: Need more registers in VLIW (15 vs. 6 in SS)

Trace Scheduling

- Parallelism across IF branches vs. LOOP branches
- Two steps:
  - *Trace Selection*
    - » Find likely sequence of basic blocks (*trace*) of (statically predicted or profile predicted) long sequence of straight-line code
  - *Trace Compaction*
    - » Squeeze trace into few VLIW instructions
    - » Need bookkeeping code in case prediction is wrong
- Compiler undoes bad guess (discards values in registers)
- Subtle compiler bugs mean wrong answer vs. pooer performance; no hardware interlocks



Advantages of HW (Tomasulo) vs. SW (VLIW) Speculation

- HW determines address conflicts
- HW better branch prediction
- HW maintains precise exception model
- HW does not execute bookkeeping instructions
- Works across multiple implementations
- SW speculation is much easier for HW design

Superscalar v. VLIW

- Smaller code size
- Binary compatability across generations of hardware
- Simplified Hardware for decoding, issuing instructions
- No Interlock Hardware (compiler checks?)
- More registers, but simplified Hardware for Register Ports (multiple independent register files?)

Intel/HP "Explicitly Parallel Instruction Computer (EPIC)"

- 3 Instructions in 128 bit "groups"; field determines if instructions dependent or independent
  - Smaller code size than old VLIW, larger than x86/RISC
  - Groups can be linked to show independence > 3 instr
- 64 integer registers + 64 floating point registers
  - Not separate files per functional unit as in old VLIW
- Hardware checks dependencies (interlocks => binary compatibility over time)
- Predicated execution (select 1 out of 64 1-bit flags) => 40% fewer mispredictions?
- IA-64 : name of instruction set architecture; EPIC is type
- Merced is name of first implementation (1999/2000?)
- LIW = EPIC?

Dynamic Scheduling in Superscalar

- Dependencies stop instruction issue
- Code compiler for old version will run poorly on newest version
  - May want code to vary depending on how superscalar

Dynamic Scheduling in Superscalar

- How to issue two instructions and keep in-order instruction issue for Tomasulo?
  - Assume 1 integer + 1 floating point
  - 1 Tomasulo control for integer, 1 for floating point
- Issue 2X Clock Rate, so that issue remains in order
- Only FP loads might cause dependency between integer and FP issue:
  - Replace load reservation station with a load queue; operands must be read in the order they are fetched
  - Load checks addresses in Store Queue to avoid RAW violation
  - Store checks addresses in Load Queue to avoid WAR,WAW
  - Called "decoupled architecture"

Performance of Dynamic SS

Iteration no.	*Instructions*	*Issues*	*Executes*	*Writes result*
		clock-cycle number		
1	LD F0,0(R1)	1	2	4
1	ADDD F4,F0,F2	1	5	8
1	SD 0(R1),F4	2	9	
1	SUBI R1,R1,#8	3	4	5
1	BNEZ R1,LOOP	4	5	
2	LD F0,0(R1)	5	6	8
2	ADDD F4,F0,F2	5	9	12
2	SD 0(R1),F4	6	13	
2	SUBI R1,R1,#8	7	8	9
2	BNEZ R1,LOOP	8	9	

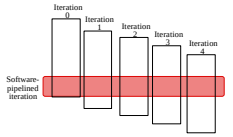
- 4 clocks per iteration; only 1 FP instr/iteration

Branches, Decrements issues still take 1 clock cycle

How get more performance?

Software Pipelining

- Observation: if iterations from loops are independent, then can get more ILP by taking instructions from different iterations
- Software pipelining: reorganizes loops so that each iteration is made from instructions chosen from different iterations of the original loop (- Tomasulo in SW)



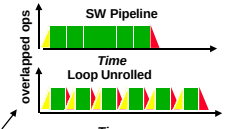
Software Pipelining Example

Before: Unrolled 3 times

```
1  LD    F0,0(R1)
2  ADDD  F4,F0,F2
3  SD    0(R1),F4
4  LD    F6,-8(R1)
5  ADDD  F8,F6,F2
6  SD    -8(R1),F8
7  LD    F10,-16(R1)
8  ADDD  F12,F10,F2
9  SD    -16(R1),F12
10 SUBI  R1,R1,#24
11 BNEZ  R1,LOOP
```

After: Software Pipelined

```
1  SD    0(R1),F4 ; Stores M[i]
2  ADDD  F4,F0,F2 ; Adds to M[i-1]
3  LD    F0,-16(R1); Loads M[i-2]
4  SUBI  R1,R1,#8
5  BNEZ  R1,LOOP
```



- Symbolic Loop Unrolling
  - Maximize result-use distance
  - Less code space than unrolling
  - Fill & drain pipe only once per loop vs. once per each unrolled iteration in loop unrolling

Limits to Multi-Issue Machines

- Inherent limitations of ILP
  - 1 branch in 5: How to keep a 5-way VLIW busy?
  - Latencies of units: many operations must be scheduled
  - Need about Pipeline Depth x No. Functional Units of independent Types. Difficulties in building HW
  - Easy: More instruction bandwidth
  - Easy: Duplicate FUs to get parallel execution
  - Hard: Increase ports to Register File (bandwidth)
    - » VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
  - Harder: Increase ports to memory (bandwidth)
  - Decoding Superscalar and impact on clock rate, pipeline depth?

Limits to Multi-Issue Machines

- Limitations specific to either Superscalar or VLIW implementation
  - Decode issue in Superscalar: how wide practical?
  - VLIW code size: unroll loops + wasted fields in VLIW
    - » IA-64 compresses dependent instructions, but still larger
  - VLIW lock step => 1 hazard & all instructions stall
    - » IA-64 not lock step? Dynamic pipeline?
  - VLIW & binary compatibility. IA-64 promises binary compatibility

Limits to ILP

- Conflicting studies of amount
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?

Limits to ILP

Initial HW Model here; MIPS compilers.

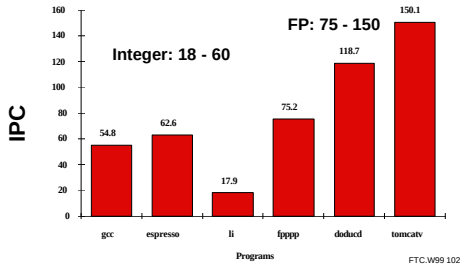
Assumptions for ideal/perfect machine to start:

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 => machine with perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis*–addresses are known & a store can be moved before a load provided addresses not equal

1 cycle latency for all instructions; unlimited number of instructions issued per clock cycle

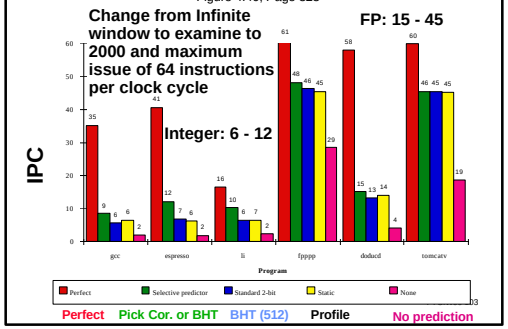
Upper Limit to ILP: Ideal Machine

(Figure 4.38, page 319)



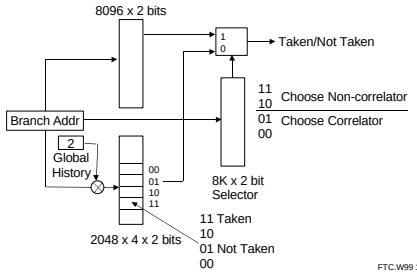
More Realistic HW: Branch Impact

Figure 4.40, Page 323



Perfect Pick Cor. or BHT BHT (512) Profile No prediction

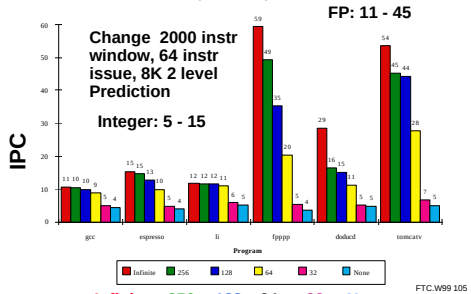
Selective History Predictor



FTC.W99 104

More Realistic HW: Register Impact

Figure 4.44, Page 328

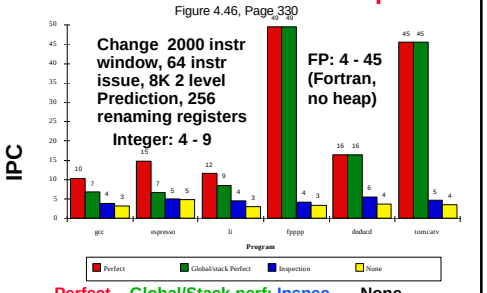


Infinite 256 128 64 32 None

FTC.W99 105

More Realistic HW: Alias Impact

Figure 4.46, Page 330

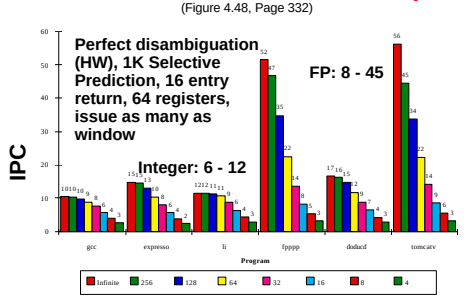


Perfect Global/Stack perf; heap conflicts Inspec. Assem. None

FTC.W99 106

Realistic HW for '9X: Window Impact

(Figure 4.48, Page 332)

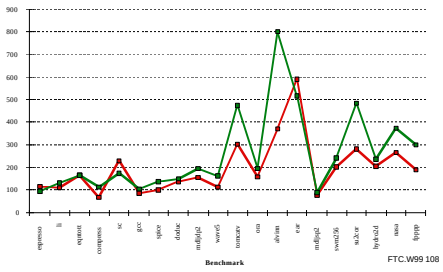


Infinite 256 128 64 32 16 8 4

FTC.W99 107

Braniac vs. Speed Demon(1993)

- 8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe) vs. 2-scalar Alpha @ 200 MHz (7 stage pipe)



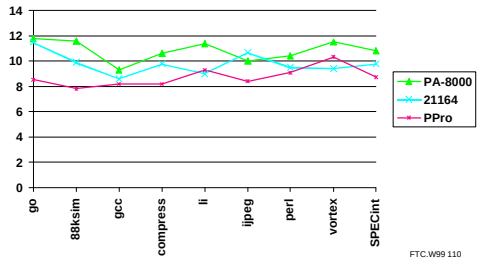
FTC.W99 108

3 1996 Era Machines

	Alpha 21164	PPro	HP PA-8000
Year	1995	1995	1996
Clock	400 MHz	200 MHz	180 MHz
Cache	8K/8K/96K/2M	8K/8K/0.5M	0/0/2M
Issue rate	2int+2FP	3 instr (x86)	4 instr
Pipe stages	7-9	12-14	7-9
Out-of-Order	6 loads	40 instr (μop)	56 instr
Rename regs	none	40	56

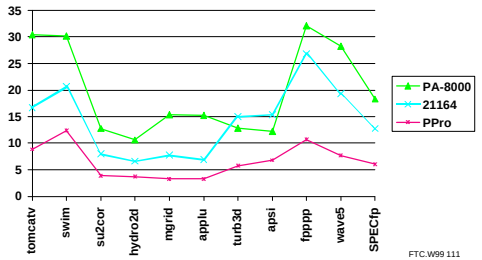
FTC.W99 109

SPECint95base Performance (July 1996)



FTC.W99 110

SPECfp95base Performance (July 1996)



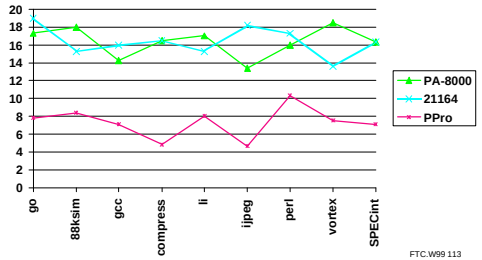
FTC.W99 111

3 1997 Era Machines

	Alpha 21164	Pentium II	HP PA-8000
Year	1995	1996	1996
Clock	600 MHz ('97)	300 MHz ('97)	236 MHz ('97)
Cache	8K/8K/96K/2M	16K/16K/0.5M	0/0/4M
Issue rate	2int+2FP	3 instr (x86)	4 instr
Pipe stages	7-9	12-14	7-9
Out-of-Order	6 loads	40 instr (μop)	56 instr
Rename regs	none	40	56

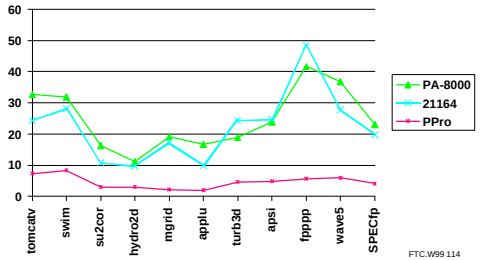
FTC.W99 112

SPECint95base Performance (Oct. 1997)



FTC.W99 113

SPECfp95base Performance (Oct. 1997)



FTC.W99 114

Summary

- **Branch Prediction**
  - Branch History Table: 2 bits for loop accuracy
  - Recently executed branches correlated with next branch?
  - Branch Target Buffer: include branch address & prediction
  - Predicated Execution can reduce number of branches, number of mispredicted branches
- **Speculation: Out-of-order execution, In-order commit (reorder buffer)**
- **SW Pipelining**
  - Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead
- **Superscalar and VLIW: CPI < 1 (IPC > 1)**
  - Dynamic issue vs. Static issue
  - More instructions issue at same time => larger hazard penalty*