Computer Architecture (Fall 2022)

Tomasulo's Algorithm

Dr. Duo Liu (刘铎)
Office: Main Building 0626
Email: liuduo@cqu.edu.cn

Review: Reducing CPI (or Increasing IPC)

$$CPI = CPI_{ideal} + stalls_{structural} + stalls_{dataHazard} + stalls_{control}$$

technique	reduces
forwarding/ bypassing	potential data-hazard stalls
delayed branches	control-hazard stalls
basic dynamic scheduling (scoreboarding)	data-hazard stalls from true dependencies
dynamic scheduling with register renaming	data-hazard, anti-dep. & output dep. stalls
dynamic branch prediction	control stalls
issuing multiple instruction per clock cycle	ideal CPI
speculation	data-hazard and control-hazard stalls
dynamic memory disambiguation	data-hazard stalls with memory
loop unrolling	control hazard stalls
basic compiler pipeline scheduling	data-hazard stalls
compiler dependency analysis	ideal CPI, data-hazard stalls
software pipelining & trace scheduling	ideal CPI, data-hazard stalls
compiler speculation	ideal CPI, data-hazard stalls

Review: Name Dependences and the New Hazards due to Out-of-Order Execution

- True data dependency: an instruction produces a value for a following instruction
  - may lead to a RAW hazard
- Name dependence: no real value must be transmitted between two instructions, but they both use the same register or memory location
  - anti-dependence
    - may lead to a WAR hazard
  - output dependence
    - may lead to a WAW hazard

```
DIV.D F0, F2, F4
ADD.D F6, F0, F8
```

```
ADD.D F6, F0, F8
SUB.D F8, F8, F14
```

```
ADD.D F6, F0, F8
MUL.D F6, F10, F8
```

Review: Dynamic HW Scheduling

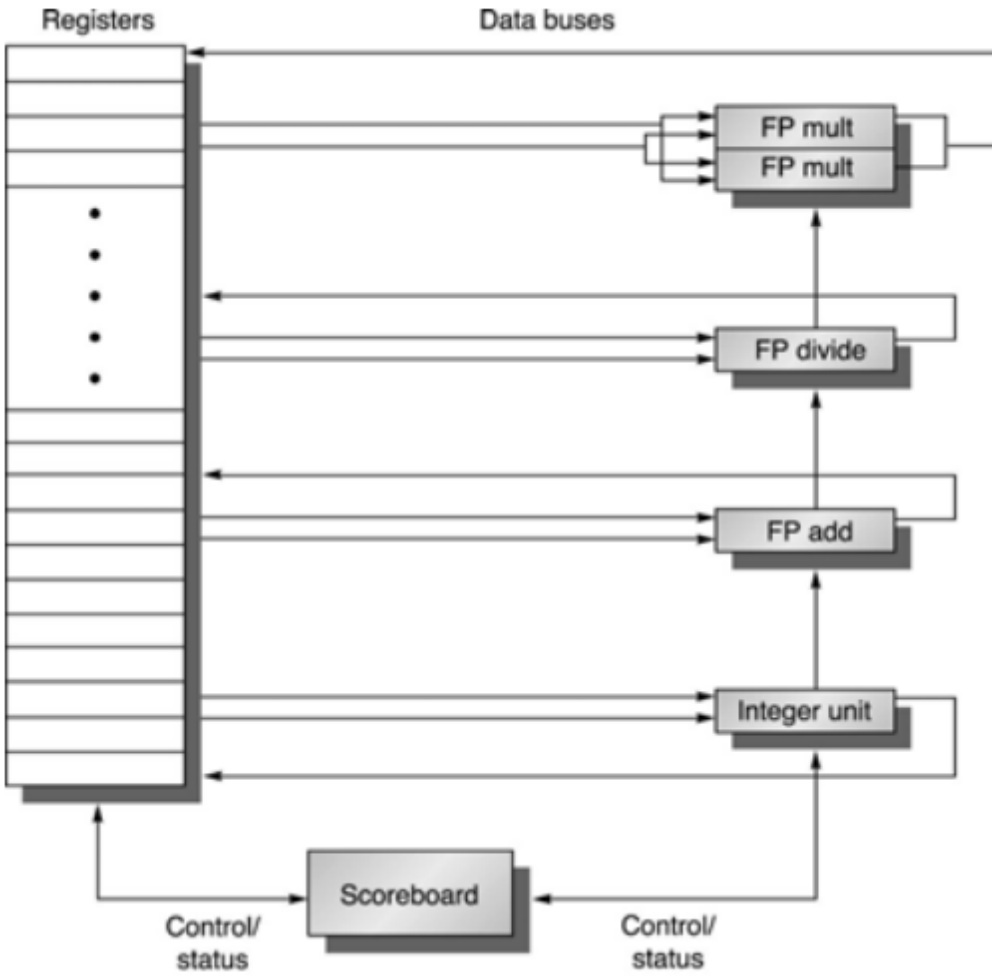

```
DIV.D F0, F2, F4
ADD.D F6, F0, F8
SUB.D F8, F8, F14
```

SUB.D does not depend on the previous instructions but it can not execute because ADD.D is stalling due to a RAW hazard

- With dynamic scheduling, the HW rearranges the instruction execution to reduce the stalls while maintaining data flow and exception behavior
  - it simplifies compiling
    - code compiled for a given pipeline can be run efficiently on another
  - it handles cases where dependencies are unknown at compile time (due to memory references)
  - it is the basis for hardware speculation

Review: Dynamic Scheduling with Scoreboard

- Goal
  - when possible, execute instructions following a stall
- Strategy
  - in-order issuing
  - out-of-order execution
  - out-of-order completion
- Implementation
  - centralized control
  - read operands together when they are all available
- Hazard handling
  - structural and WAW
    - stall issuing
  - RAW
    - solved dynamically
  - WAR
    - stall committing



MIPS Pipeline Stages with Scoreboard

- All hazard detection and resolution is centralized in the scoreboard module
  - when operands can be read
  - when execution can start
  - when result can be written
- In-order issuing, out-of-order execution and commit
- The ID, EX, and WB stages are replaced by four new stages

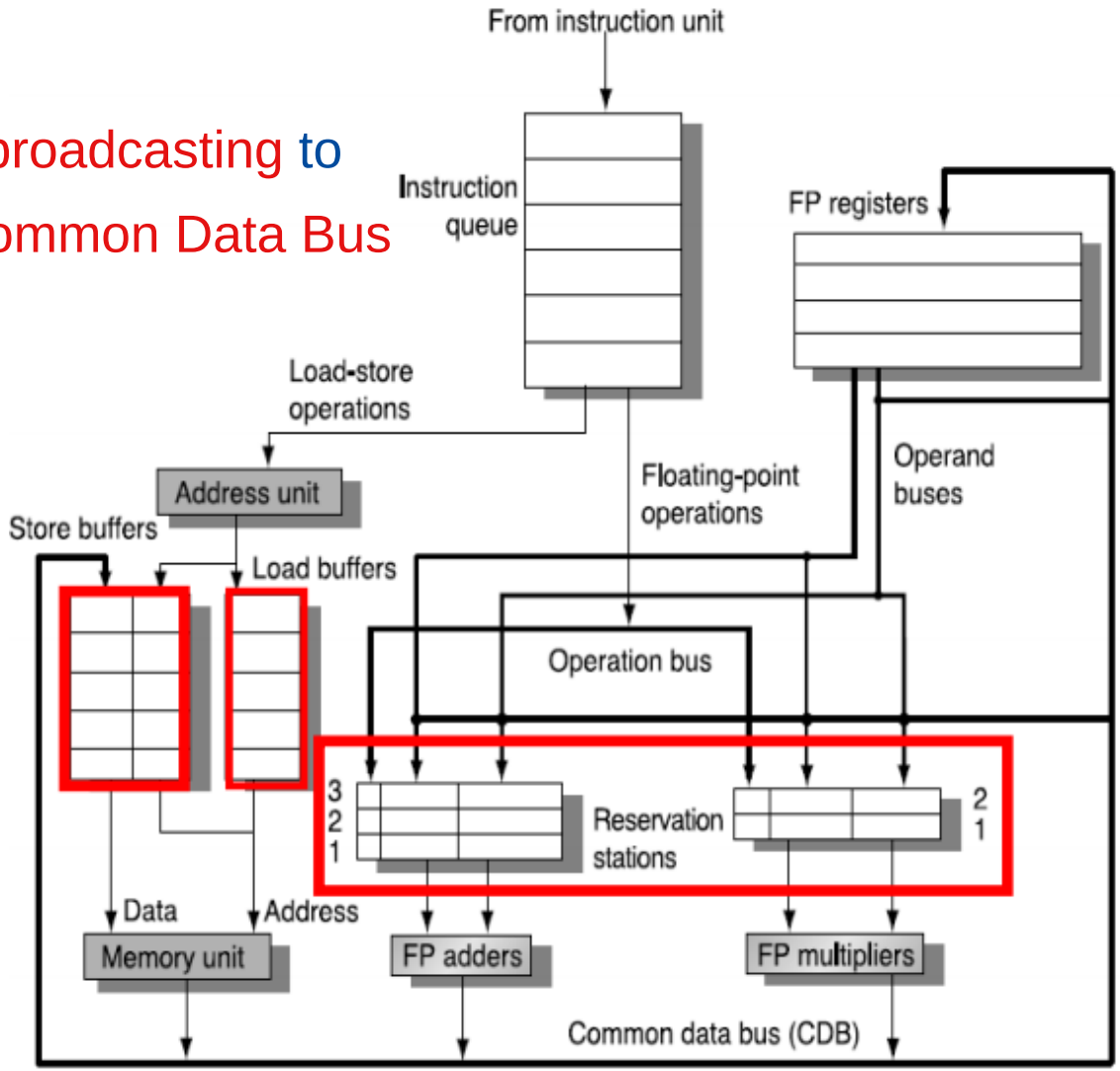
1. Issue (ID-1)
   - in-order instruction decoding
   - check/stall for structural hazards
   - check/stall for WAW hazards
2. Read Operands (ID-2)
   - wait for each operand availability
   - resolve RAW hazards dynamically
   - no forwarding, but wait for WB
3. Execution (EX)
   - out-of-order execution
   - inform scoreboard upon completion
4. Write Result (WB)
   - check/stall for WAR hazards
   - write result into register asap
   - no statically-assigned write slot

Dynamic Scheduling with Tomasulo's Algorithm

- Motivations
  - need for improving performance of floating-point unit for the whole IBM 360 family (without specialized compilers)
  - IBM 360 had only 4 DP FP registers (limiting the effectiveness of compiler scheduling)
  - IBM 360 had long memory access and long FP delays
- Main features
  - distributed control and buffering with reservation stations
  - dynamically replace register specification in instruction with value or pointer to functional unit producing the value (register renaming)
    - handle efficiently WARs and WAWs
  - tracking when operands are available to handle RAWs
  - more reservation stations than registers
    - eliminates name dependencies that compiler can't eliminate
- Basic concepts reused in many other architectures
  - Alpha 21264, Pentium II, III, IV, PowerPC, MIPS R10000, AMD K5

Assumption: Basic Architecture of MIPS FP Unit with Tomasulo's Algorithm

- Distributed control based on reservation station tag fields
- Register file bypassed in broadcasting to reservation stations via Common Data Bus (CDB)
- Precise exception
  - stall issuing operations when there is a pending branch in the pipeline
    - speculation will remove this restriction
- IBM 360 had pipelined functional units
  - but we describe it with multiple functional units



MIPS Pipeline Stages with Tomasulo's Algorithm

- Distributed hazard detection and resolution
- In-order issuing, out-of-order execution and commit
- ID, EX, and WB stages replaced by three steps that can take an arbitrary number of clock cycles

1. Issue (ID-1)
   - get new instruction from queue
   - check for availability of matching reservation station (structural hazards)
   - check for operands in the register file and read them if available (register renaming handles WAR and WAW hazards)
   - if operands are not available, keep track of functional units that will produce them
2. Execution (EX)
   - monitor CDB for missing operands
   - start execution when operands are available in reservation stations (RAW hazards)
3. Write Result (WB)
   - use CDB to broadcast the result to register file and reservation stations (including store buffers to write memory)

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)

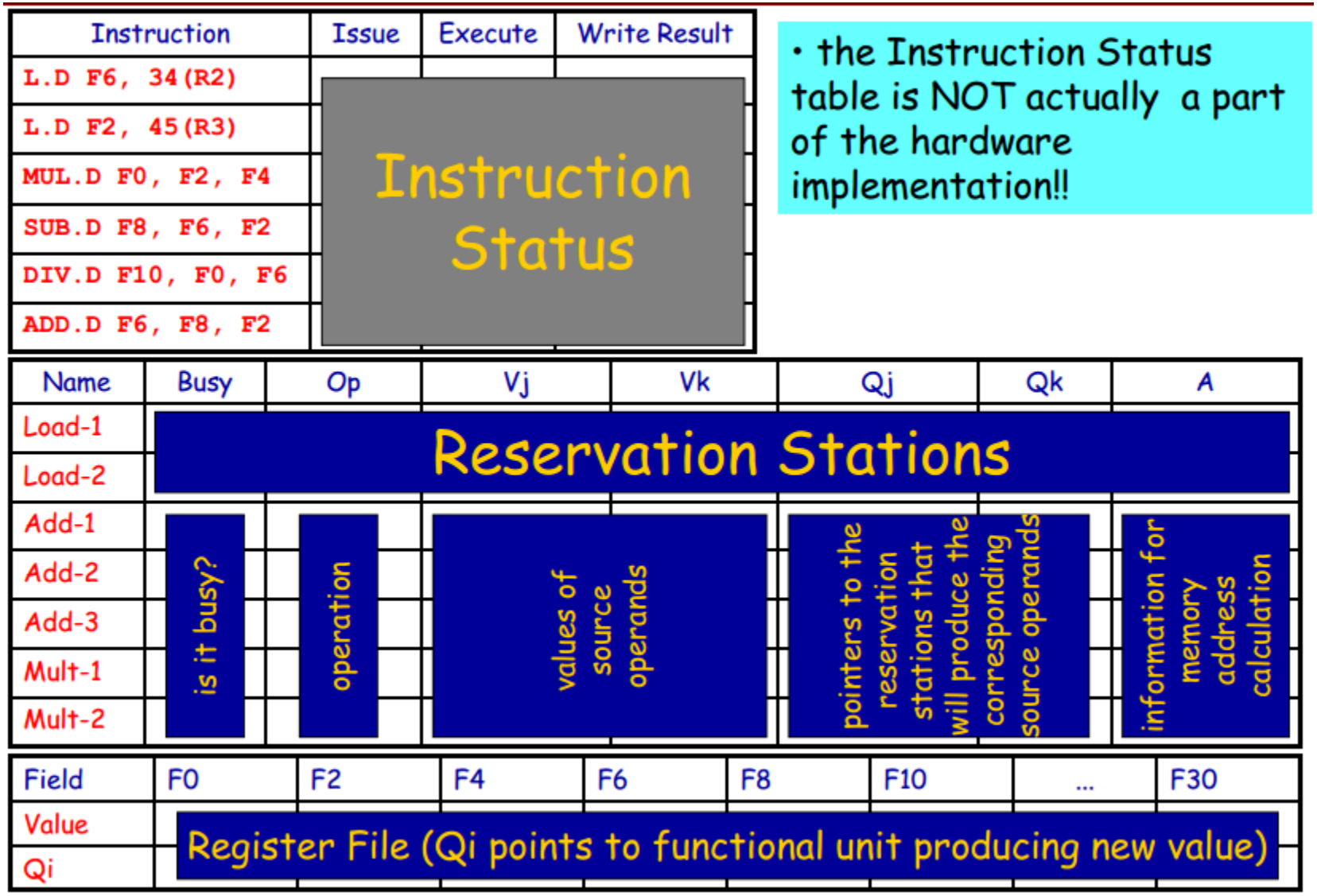
- Note: No ready flags as in Scoreboard; Qj,Qk=0 => ready
- Store buffers only have Qi for RS producing result

A: to hold memory address for load and store instructoins

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.

Tomasulo's Algorithm Data Structure



Tomasulo's Algorithm: Checking & Bookkeeping per Instruction (op D,S1,S2)

Inst. Status	Wait until	Bookkeeping

1. Issue	(RS[r].busy == no) where r is the reservation station for the functional unit matching op	if (Regs[S1].Qi ≠0)
		RS.[r].Qj ← Regs[S1].Qi else
		RS.[r].Vj ← Regs[S1].value
		if (Regs[S2].Qi ≠0)
		RS.[r].Qk ← Regs[S2].Qi else
		RS.[r].Vk ← Regs[S2].value
		RS[r].busy ← yes ; Regs[D].Qi ← r;

2. Execution	(RS.[r].Qj == 0) and (RS.[r].Qk == 0)	execute operation using operands in RS[r].Vj and RS[r].Vk and producing res
3. Write Result	RS.[r] is done & CDB is available	∀x (if (Regs[x].Qi == r) Regs[x].val ← res; Regs[x].Qi ← 0) ∀f (if (RS[f].Qj == r) RS[f].Vj ← res; RS[f].Qj ← 0) ∀f (if (RS[f].Qk == r) RS[f].Vk ← res; RS[f].Qk ← 0) RS[r].busy ← no

Tomasulo's Algorithm: Execution of Memory Accesses

Loads and Stores executed in program order based on a two-step process

1. compute the effective address when base register is available
2. then
   - if load execute as soon as the memory unit is available
   - if store, wait for the value to be stored and send it to memory unit

Inst. Status	Wait until	Bookkeeping
2. Execution	(RS.[r].Qj == 0) and r being head of load/store queue	RS[r].A ← RS[r].Vj + RS[r].A if (op is load) do res ← Mem[RS[r].A]

3. Write Result	RS.[r] is done &
	CDB is available
	
	if (op is store)
	RS.[r] is done & wait for RS[r].Qk=0
		if (op is load) same steps as before…
		
		if (op is store) do
		Mem[RS[r].A] ← RS[r].Vk
		RS[r].busy ← no

Tomasulo's Algorithm Ex.: Clock Cycle = 1

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

- issuing first load instruction

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	yes	L.D					34+R[R2]
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	no						
Mult-2	no						

Field	F0	F2	F4	F6	F8	F10	...	F30
Value								
Qi				Load-1				

Tomasulo's Algorithm Ex.: Clock Cycle = 2

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

- executing first load instr.
- issuing second load instr.
- differently from CDC 6600, multiple loads can be outstanding

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	yes	L.D					34+R[R2]
Load-2	yes	L.D					45+R[R3]
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	no						
Mult-2	no						

Field	F0	F2	F4	F6	F8	F10	...	F30
Value								
Qi		Load-2		Load-1				

Tomasulo's Algorithm Ex.: Clock Cycle = 3

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

- completing first load instr.
- executing second load instr.
- issuing multiplication,
- register F4 read asap and Mult-1 reservation station stores its value

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	yes	L.D					45+R[R3]
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D		R[F4]	Load-2		
Mult-2	no						

Field	F0	F2	F4	F6	F8	F10	...	F30
Value				M[34(R2)]				
Qi	Mult-1	Load-2						

Tomasulo's Algorithm Ex.: Clock Cycle = 4

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

- issuing subtraction
- completing second load, whose result is broadcasted to Add-1 and Mult-1
- stalling multiplication
- reservation station Vj gets value loaded by second load

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	yes	SUB.D	M[45(R3)]	M[34(R2)]			
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	no						

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		M[34(R2)]				
Qi	Mult-1				Add-1			

Tomasulo's Algorithm Ex.: Clock Cycle = 5

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

- executing multiplication and subtraction (with 10- and 2-cycle execution latency, respectively)
- issuing division

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	yes	SUB.D	M[45(R3)]	M[34(R2)]			
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		M[34(R2)]				
Qi	Mult-1				Add-1	Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 6

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

- completed execution of subtraction
- executing multiplication (still 8 cycles to go)
- stalling division (waiting for multiplication)
- issuing addition

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	yes	SUB.D	M[45(R3)]	M[34(R2)]			
Add-2	yes	ADD.D		M[45(R3)]	Add-1		
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		M[34(R2)]				
Qi	Mult-1			Add-2	Add-1	Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 7

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

- committing subtraction, broadcasting result to ADD-2
- executing multiplication (still 7 cycles to go)
- stalling division (waiting for multiplication)

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	yes	ADD.D	res{SUB.D}	M[45(R3)]			
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		M[34(R2)]	res{SUB.D}			
Qi	Mult-1			Add-2		Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 8

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

- executing multiplication (still 6 cycles to go)
- stalling division (waiting for multiplication)
- executing addition (1 more cycle to go); register renaming avoids WAR hazard!

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	yes	ADD.D	res{SUB.D}	M[45(R3)]			
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		M[34(R2)]	res{SUB.D}			
Qi	Mult-1			Add-2		Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 9

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3		
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	

- executing multiplication (still 5 cycles to go)
- stalling division (waiting for multiplication)
- completed addition execution

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	yes	ADD.D	res{SUB.D}	M[45(R3)]			
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		M[34(R2)]	res{SUB.D}			
Qi	Mult-1			Add-2		Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 10

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3		
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	10

- executing multiplication (still 4 cycles to go)
- stalling division (waiting for multiplication)
- committed addition (without WAR hazard with previous division)

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi	Mult-1					Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 11

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3		
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	10

- executing multiplication (still 3 cycles to go)
- stalling division (waiting for multiplication)

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi	Mult-1					Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 12

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3		
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	10

- executing multiplication (still 2 cycles to go)
- stalling division (waiting for multiplication)

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi	Mult-1					Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 13

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3		
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	10

- executing multiplication (still 1 cycle to go)
- stalling division (waiting for multiplication)

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi	Mult-1					Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 14

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3	14	
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	10

- completed execution of multiplication
- stalling division (waiting for multiplication)

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	yes	MUL.D	M[45(R3)]	R[F4]			
Mult-2	yes	DIV.D		M[34(R2)]	Mult-1		

Field	F0	F2	F4	F6	F8	F10	...	F30
Value		M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi	Mult-1					Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 15

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3	14	15
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	10

- committing multiplication and broadcasting the result to division, which at the next cycle can finally start the execution

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	no						
Mult-2	yes	DIV.D	res{MUL.D}	M[34(R2)]			

Field	F0	F2	F4	F6	F8	F10	...	F30
Value	res{MUL.D}	M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi						Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 16

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3	14	15
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5		
ADD.D F6, F8, F2	6	9	10

- starting division execution, it will take 40 cycles

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	no						
Mult-2	yes	DIV.D	res{MUL.D}	M[34(R2)]			

Field	F0	F2	F4	F6	F8	F10	...	F30
Value	res{MUL.D}	M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi						Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 55

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3	14	15
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5	55	
ADD.D F6, F8, F2	6	9	10

- completed division execution

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	no						
Mult-2	yes	DIV.D	res{MUL.D}	M[34(R2)]			

Field	F0	F2	F4	F6	F8	F10	...	F30
Value	res{MUL.D}	M[45(R3)]		res{ADD.D}	res{SUB.D}			
Qi						Mult-2		

Tomasulo's Algorithm Ex.: Clock Cycle = 56

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3	14	15
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5	55	56
ADD.D F6, F8, F2	6	9	10

- committing division

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	no						
Load-2	no						
Add-1	no						
Add-2	no						
Add-3	no						
Mult-1	no						
Mult-2	no						

Field	F0	F2	F4	F6	F8	F10	...	F30
Value	res{MUL.D}	M[45(R3)]		res{ADD.D}	res{SUB.D}	res{DIV.D}		
Qi								

Recalling…
Scoreboard Example: Clock Cycle = 62

Instruction	Issue	Read Operands	Exec. Complete	Write Result
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

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	no								
Mult-2	no								
Add	no								
Divide	no								

	F0	F2	F4	F6	F8	F10	...	F30
FU								

- 62 cycles necessary (in-order issue; out-of-order execution & commit)

Tomasulo's Algorithm vs. Scoreboard: the Example Results

Instruction	Issue	Execute	Write Result
L.D F6, 34(R2)	1	2	3
L.D F2, 45(R3)	2	3	4
MUL.D F0, F2, F4	3	14	15
SUB.D F8, F6, F2	4	6	7
DIV.D F10, F0, F6	5	55	56
ADD.D F6, F8, F2	6	9	10

- Tomasulo's algorithm did better because we had:
  - less structural hazards (two parallel loads)
  - CDB-based bypassing (division starts 1 cycle after multiplication)
  - no stalling for WAR hazards (addition starts earlier)

Instruction	Issue	Read Operands	Exec. Complete	Write Result
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

Tomasulo's Algorithm vs. Scoreboard

	Tomasulo's Algorithm	Scoreboard
introduced with	IBM 360/91 in 1967	CDC 6600 in 1964
resources	3 adders, 2 mul/div, 6 load, 3 store	7 int units, 4 FP units, 5 mem. reference
max window size	14	5
structural hazards	stall issuing	stall issuing
WAR hazards	avoided by register renaming	stall committing
WAW hazards	avoided by register renaming	stall issuing
control	based on reservation stations	scoreboard module
communication	broadcasting to reservation stations (and registers)	buses to/from registers

- with scoreboard both operands are read at once when they are both ready in the register file, while Tomasulo`s algorithm reads each operand asap