Course on: “Advanced Computer Architectures”

Instruction Level Parallelism
Part III

Prof. Cristina Silvano
Politecnico di Milano
email: cristina.silvano@polimi.it

Outline of Part III

Tomasulo Dynamic Scheduling Algorithm
Scoreboard vs Tomasulo

Tomasulo Dynamic Scheduling Algorithm

Tomasulo Algorithm

- Another dynamic scheduling algorithm:
  Enables instructions execution behind a stall to proceed
- Invented at IBM 3 years after CDC 6600 for the IBM 360/91
- Same goal: High performance without special compilers
- Lead to Alpha 21264, HP 8000, MIPS 10000, Pentium II, PowerPC 604

Tomasulo Algorithm vs. Scoreboard

- Control & buffers are **distributed** with Function Units (vs. centralized in scoreboard);
  - FU buffers called **"Reservation Stations"** have pending operands
- Registers in instructions replaced by values or pointers to reservation stations **(RS)** to enable **implicit Register Renaming**
  - **Avoids WAR, WAW hazards by renaming results by using RS numbers instead of RF numbers**
  - More reservation stations than registers, so can do optimizations compilers can't
- Basic idea: **Results to FU from RS, not through registers,** over **Common Data Bus** that broadcasts results to all Fus *(like a sort of forwarding)*
- 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

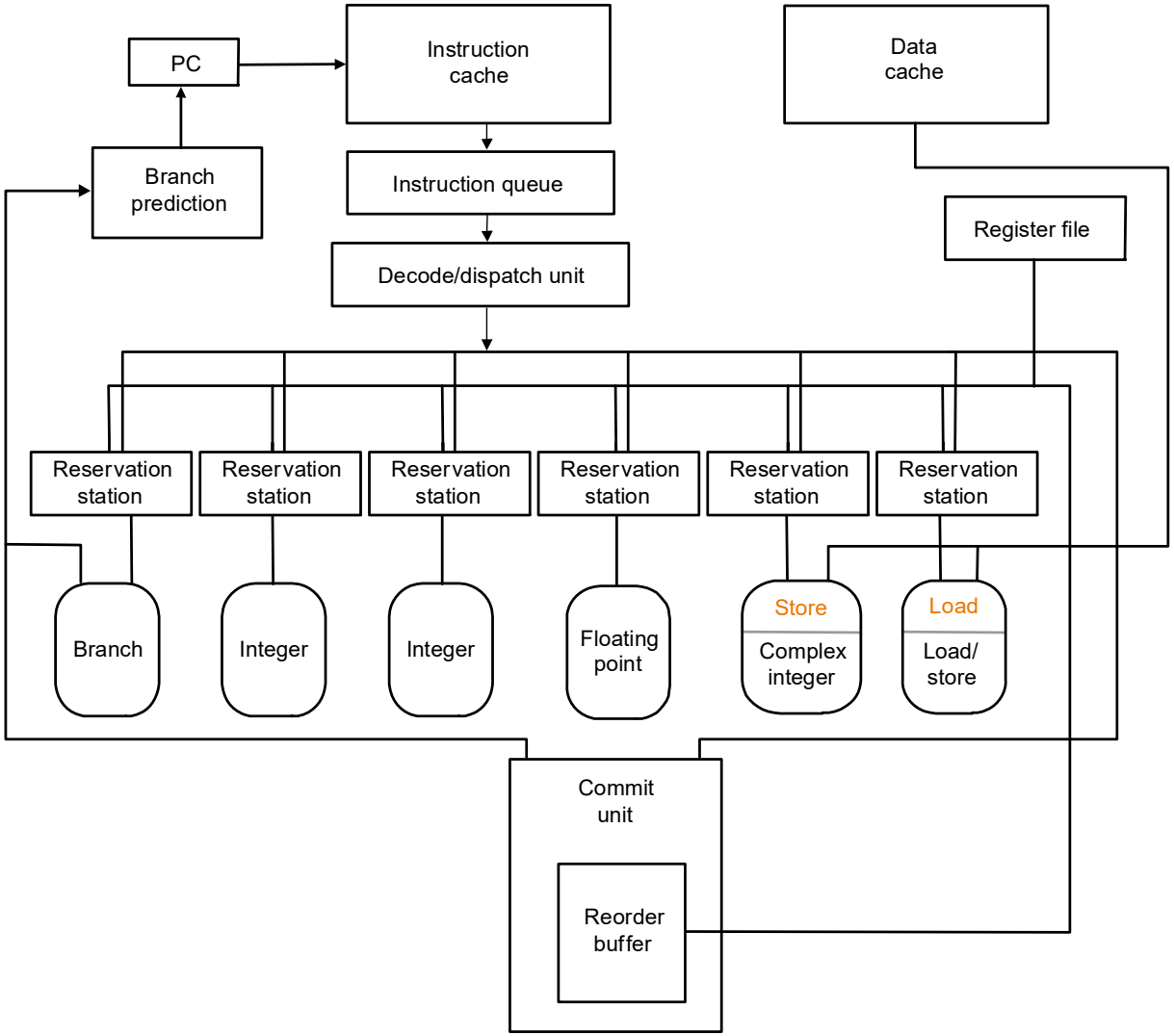
Tomasulo Algorithm Basics

- The control logic and the buffers are distributed with FUs (vs. centralized in Scoreboard)
- Operand buffers are called **Reservation Stations**
- Each instruction is an entry of a reservation station
- Instruction operands are replaced by values or pointers (**Register Renaming**)

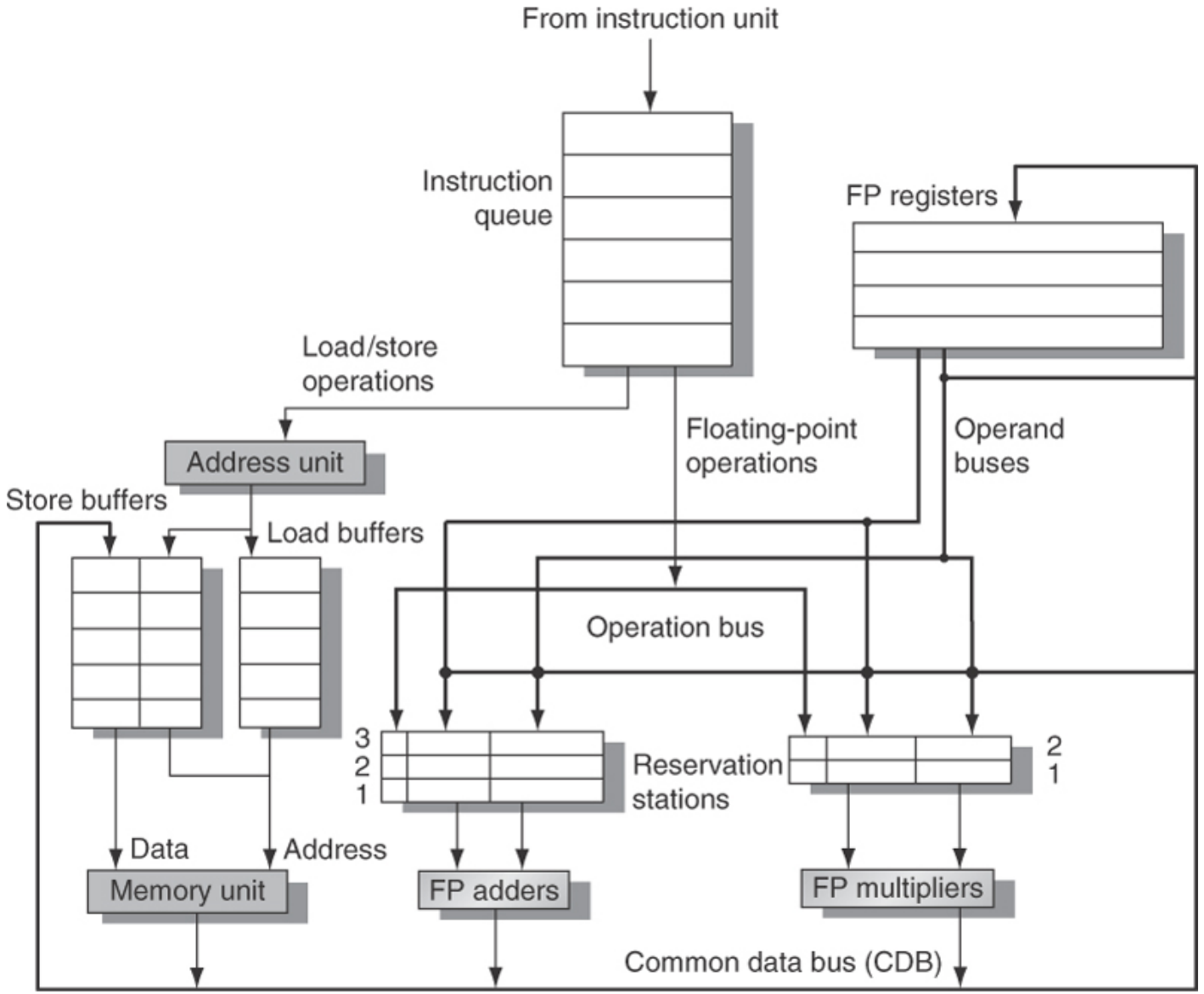
Tomasulo Algorithm Basics

- **Register Renaming** allows to:
  - **Avoid WAR and WAW hazards**
  - Reservation stations are more than registers (so can do better optimizations than a compiler).
- Results are dispatched to other FUs through a Common Data Bus (CDB)
- Load/Stores treated as FUs

Tomasulo Architecture



Tomasulo Architecture for an FPU



Reservation Station Components

- **Tag** identifying the RS
- **Busy** = Indicates RS Busy
- **OP** = Type of operation to perform on the component.
- **V_j, V_k** = Value of the source operands
  - **V_j** holds offset for loads
- **Q_j,Q_k** = Pointers to RS that produce V_j,V_k
  - Zero value = Source op. is already available in V_j or V_k

- Note: Only one of V-field or Q-field is valid for each operand

Register File and Load/Store Buffers

- **RF and the Store buffers have a Value (V) and a Pointer (Q) field.**
  - Pointer (Q) field corresponds to number of reservation station producing the result to be stored in RF (or store buffer)
    If zero ⇒ no active instructions producing the result (RF or store buffer content is the correct value).
- **Load buffers have an Address field and a Busy field. Store buffers have only an Address field**
  - Address field: To hold info for memory address calculation for load/store. Initially contains the instruction offset (immediate field); after address calculation stores the effective address.

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

First stage of Tomasulo Algorithm

ISSUE

- Get an instruction ***I*** from the head of instruction queue (maintained in FIFO order to ensure **in-order issue**).
- **Check if an RS is empty (i.e., check for structural hazards in RS) otherwise stalls.** If operands are not in RF, keep track of FU that will produce the operands (Q pointers).
- If there is not an empty RS $\Rightarrow$ structural hazard in RS and the instruction stalls.

First stage of Tomasulo Algorithm

ISSUE

- **Rename registers**

- **WAR resolution:** If ***I*** writes ***Rx,*** read by an instruction ***K*** already issued, ***K*** knows already the value of ***Rx*** read in RS buffer or knows what instruction will write it. So the RF can be linked to ***I***.

- **WAW resolution:** Since we use in-order issue, the RF can be linked to ***I.***

Second stage of Tomasulo Algorithm

Execution

- **When both operands are ready and execution unit available, then start execution.**
- If not ready, watch the Common Data Bus for results.
- By delaying execution until operands are available, RAW hazards are avoided at this stage.
- Notice that several instructions could become ready in the same clock cycle for the same FU (need to check if execution unit is available).
- Notice that usually RAW hazards are shorter because operands are given directly by RS without waiting for RF write back *(sort of forwarding).*

Second stage of Tomasulo Algorithm

Execution

- ***Load and Stores:*** Two-step execution process:
  - First step: compute effective address when base register is available, place it in load or store buffer.
  - Loads in Load Buffer execute as soon as memory unit is available; stores in store buffer wait for the value to be stored before being sent to memory unit.
- ***Loads and Stores:*** Kept in program order through effective address calculation – helps in preventing hazards through memory.

- **To preserve exception behavior**:
  - No instruction can initiate execution until all branches preceding it in program order have completed.
  - If branch prediction is used, CPU must know prediction correctness before beginning execution of following instructions. (Speculation allows more brilliant results!)

Third stage of Tomasulo Algorithm

Write result

- When result is available, write on Common Data Bus and from there into RF and into all RSs (including store buffers) waiting for this result; stores also write data to memory during this stage.

- Mark reservation stations available.

The Common Data Bus

- A common data bus is a data + source bus.
- In the IBM 360/91: Data=64 bits, Source=4 bits
- FU must perform associative lookup in the RS.

Tomasulo Algorithm (some details)

- Loads and stores go through a functional unit for effective address computation before proceeding to effective load and store buffers
- Loads take a second execution step to access memory, then go to Write Result to send the value from memory to RF and/or RS
- Stores complete their execution in their Write Result stage (writes data to memory)
- **All writes occur in Write Result** – simplifying Tomasulo algorithm.

Tomasulo Algorithm (some details)

- **A Load and a Store can be done in different order, *provided they access different memory locations*;** otherwise, a WAR (interchange load-store sequence) or a RAW (interchange store-load sequence) may result (WAW if two stores are interchanged). Loads can be reordered freely.
- To detect such hazards: Data memory addresses associated with *any earlier memory operation* must have been computed by the CPU (e.g.: address computation executed in program order)

Tomasulo Algorithm (some details)

- Load executed out of order with previous store: Assume address computed in program order. When Load address has been computed, it can be compared with A fields in active Store buffers: In the case of a match, Load is not sent to Load buffer until conflicting store completes.
- Stores must check for matching addresses in both Load and Store buffers (*dynamic disambiguation,* alternative to static disambiguation performed by the compiler)
- Drawback: Amount of hardware required.
- Each RS must contain a fast associative buffer; single CDB may limit performance.

TOMASULO BASIC SCHEME

- IN-ORDER ISSUE
- OUT-OF-ORDER EXECUTION
- OUT-OF-ORDER COMPLETION
- REGISTER RENAMING based on Reservation Stations to avoid WAR and WAW hazards
- Results dispatched to RESERVATION STATIONS and to RF through the Common Data Bus
- Control is distributed on Reservation Stations
- *Reservation Stations offer a sort of data forwarding!*

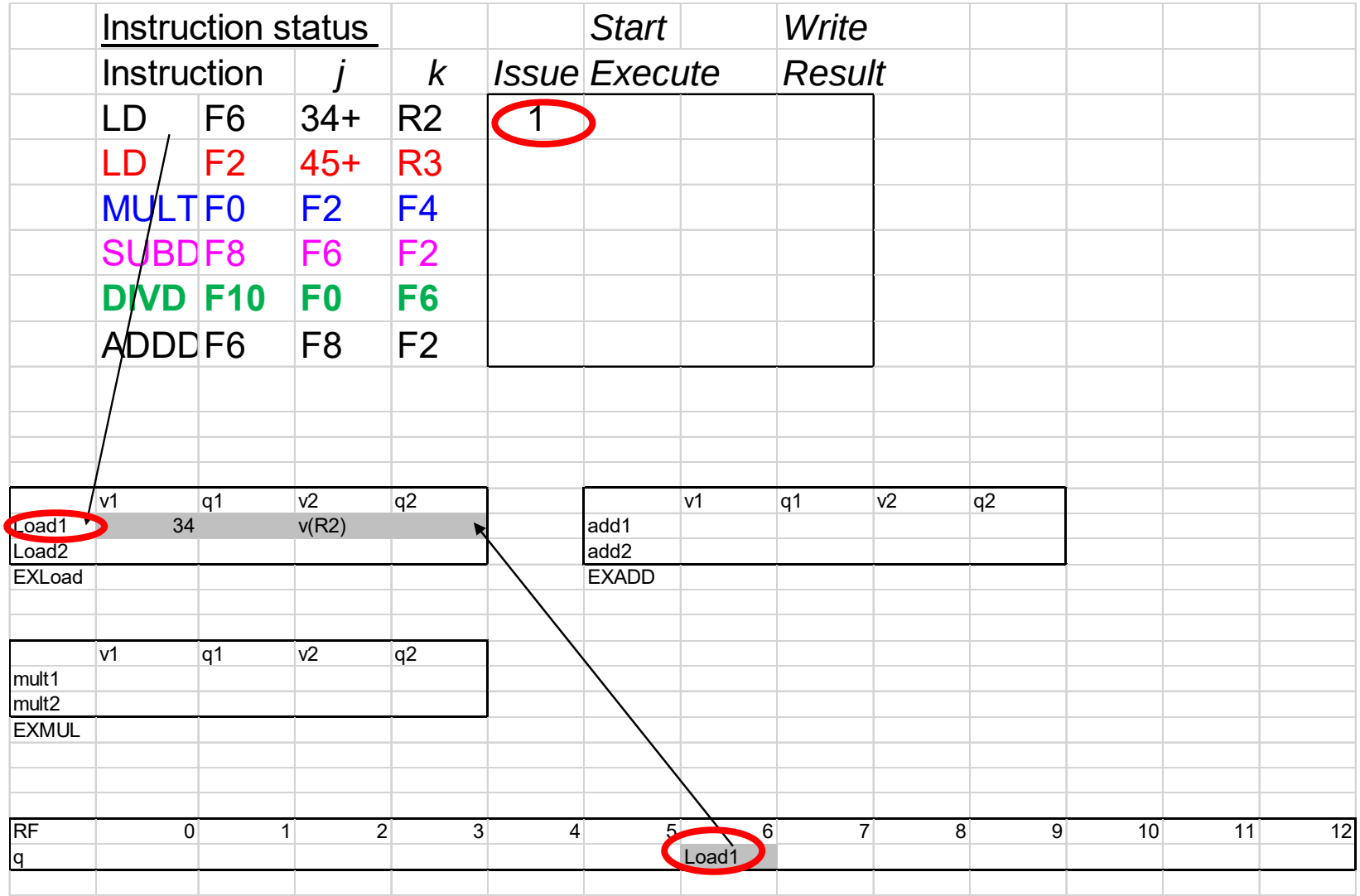
TOMASULO STAGES

- **ISSUE (IN-ORDER):**
  - Check for structural hazards in RESERVATION STATIONS (not in FU)
- **START EXECUTE (OUT-OF-ORDER)**
  - When operands ready (Check for RAW hazards solved)
  - When FU available (Check for structural hazards in FU)
- **WRITE RESULTS (OUT-OF-ORDER)**
  - Execution completion depends on latency of FUs
  - Execution completion of LD/ST depends on cache hit/miss latencies
  - Write results on Common Data Bus to Reservations Stations, Store Buffers and RF

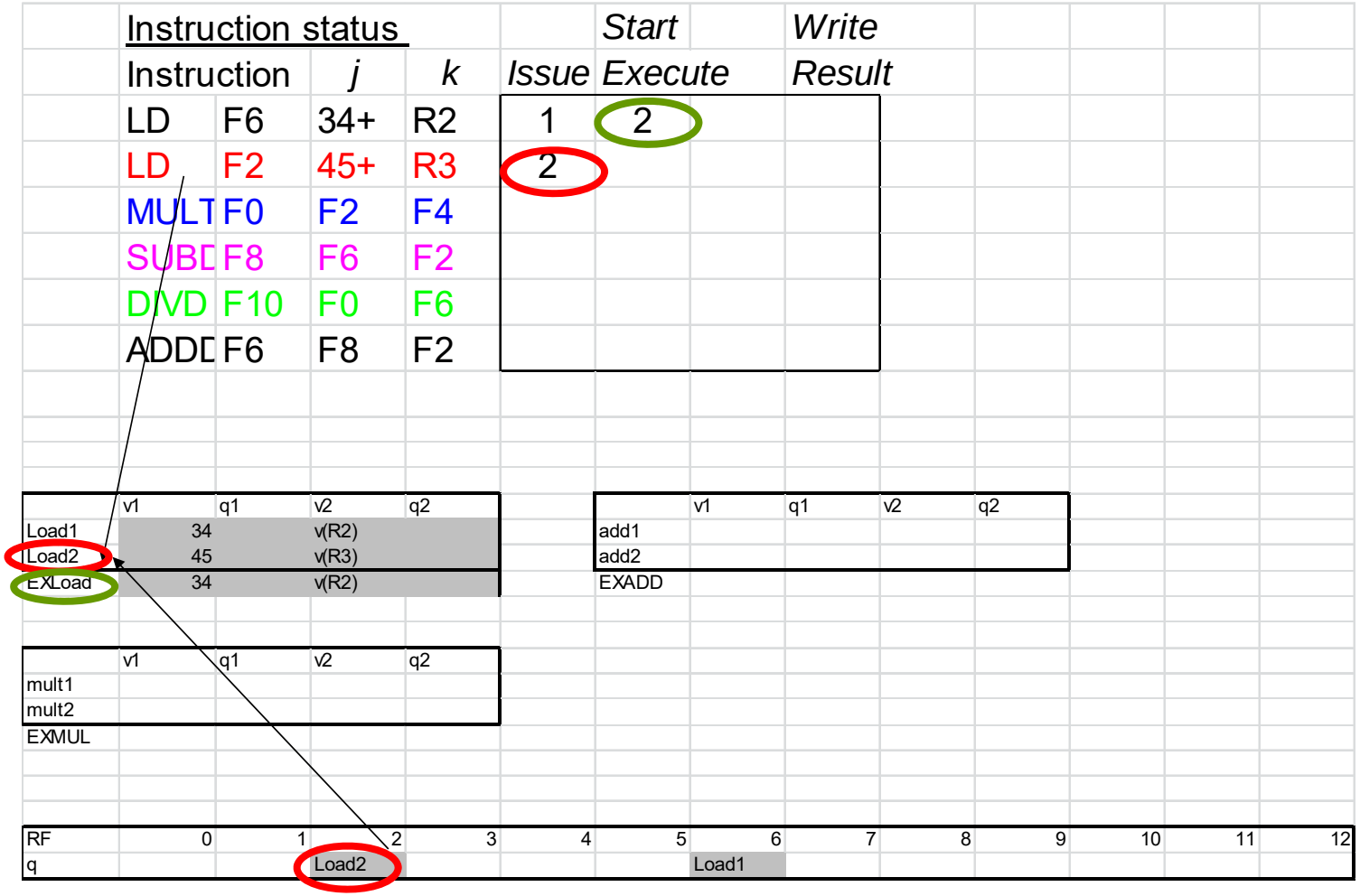
Tomasulo Example: Analysis of dependences and hazards

```
LD F6, 34(R2)
LD F2, 45(R3)
MULTD F0, F2, F4   # RAW F2
SUBD F8, F6, F2    # RAW F2,RAW F6
DIVD F10, F0, F6   # RAW F0,RAW F6
ADDD F6, F8, F2    # WAR F6,RAW F8,RAW F2
```

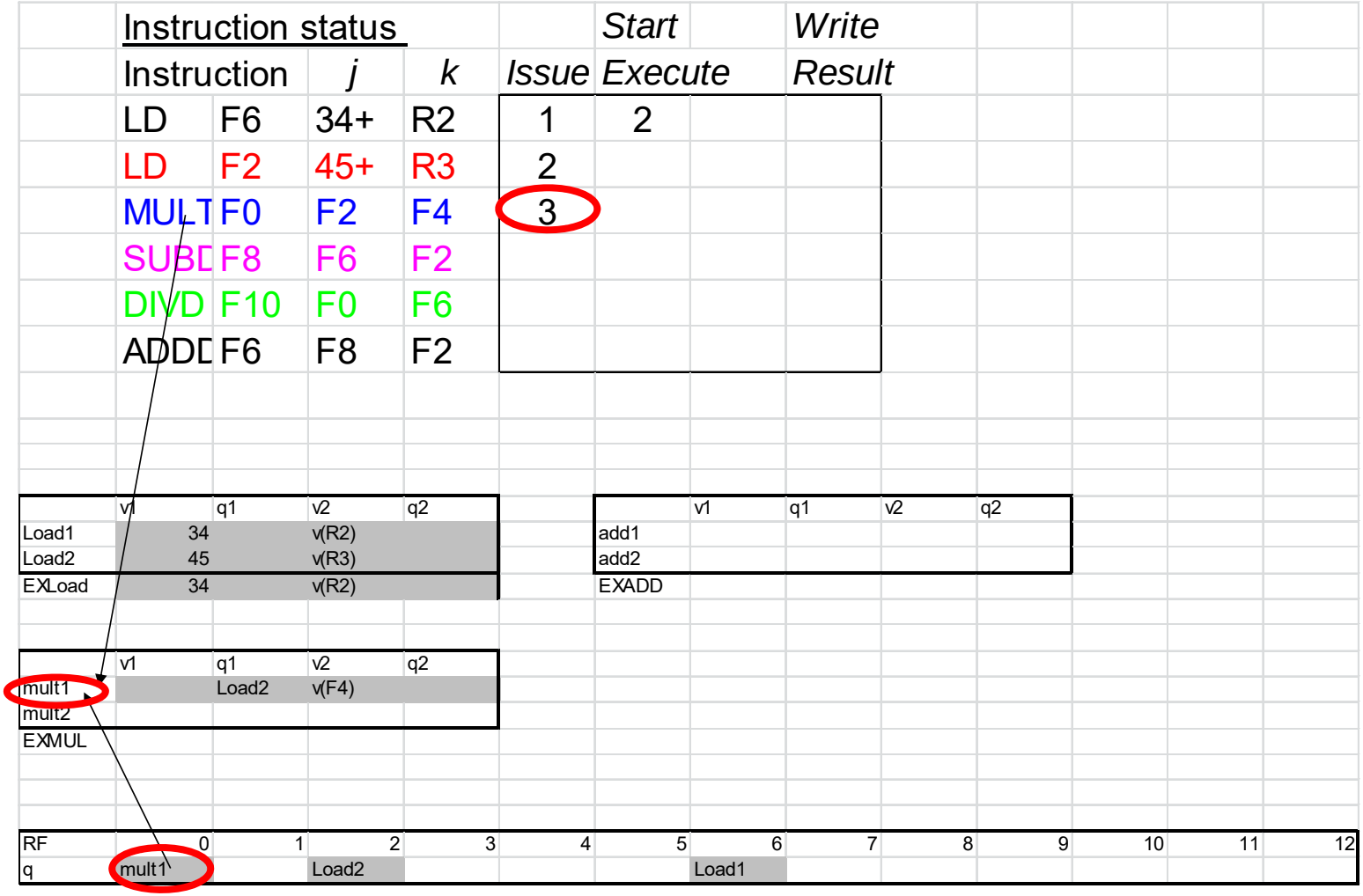
Tomasulo's example Cycle 1



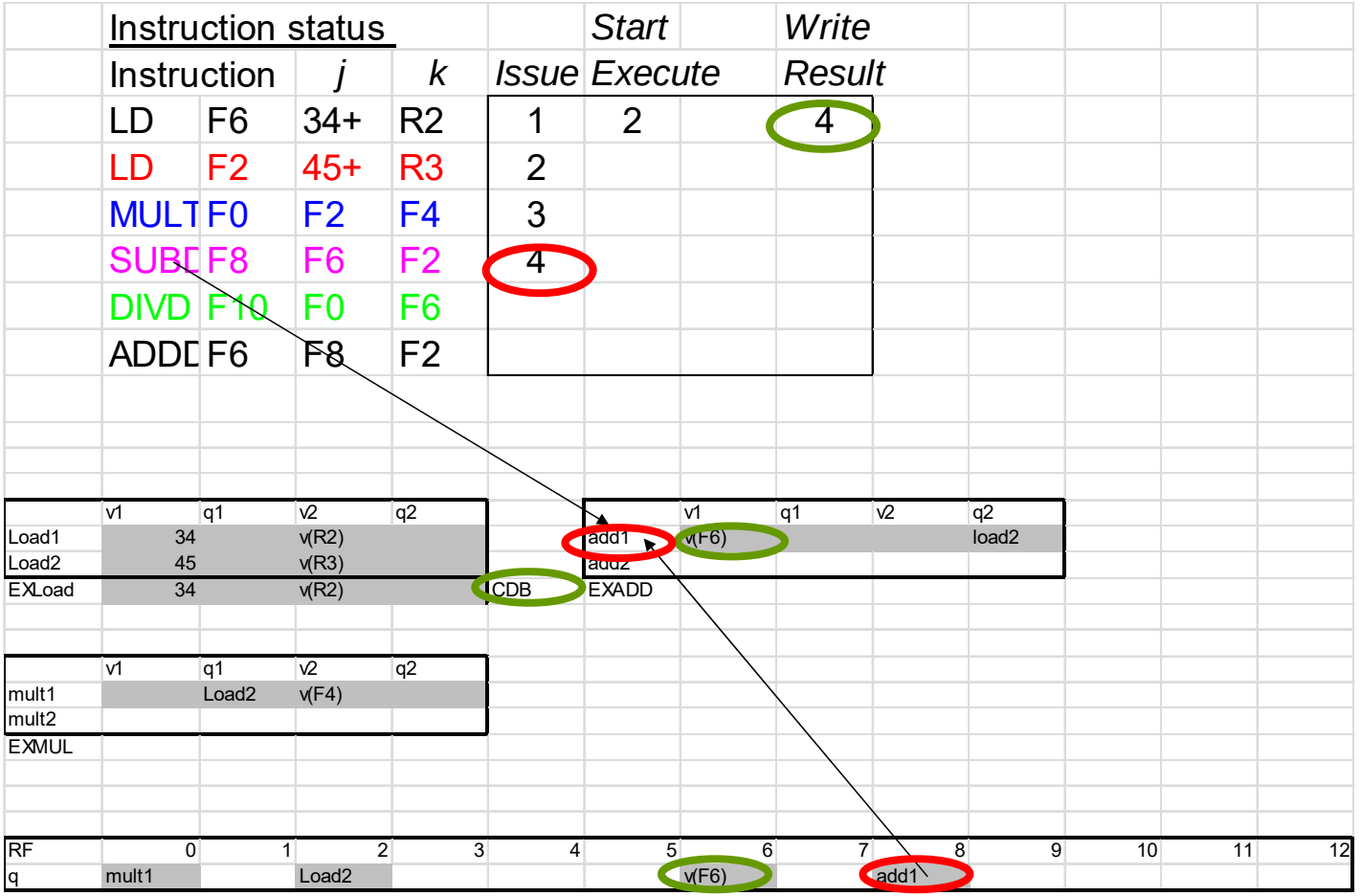
Tomasulo's example Cycle 2



Tomasulo's example Cycle 3

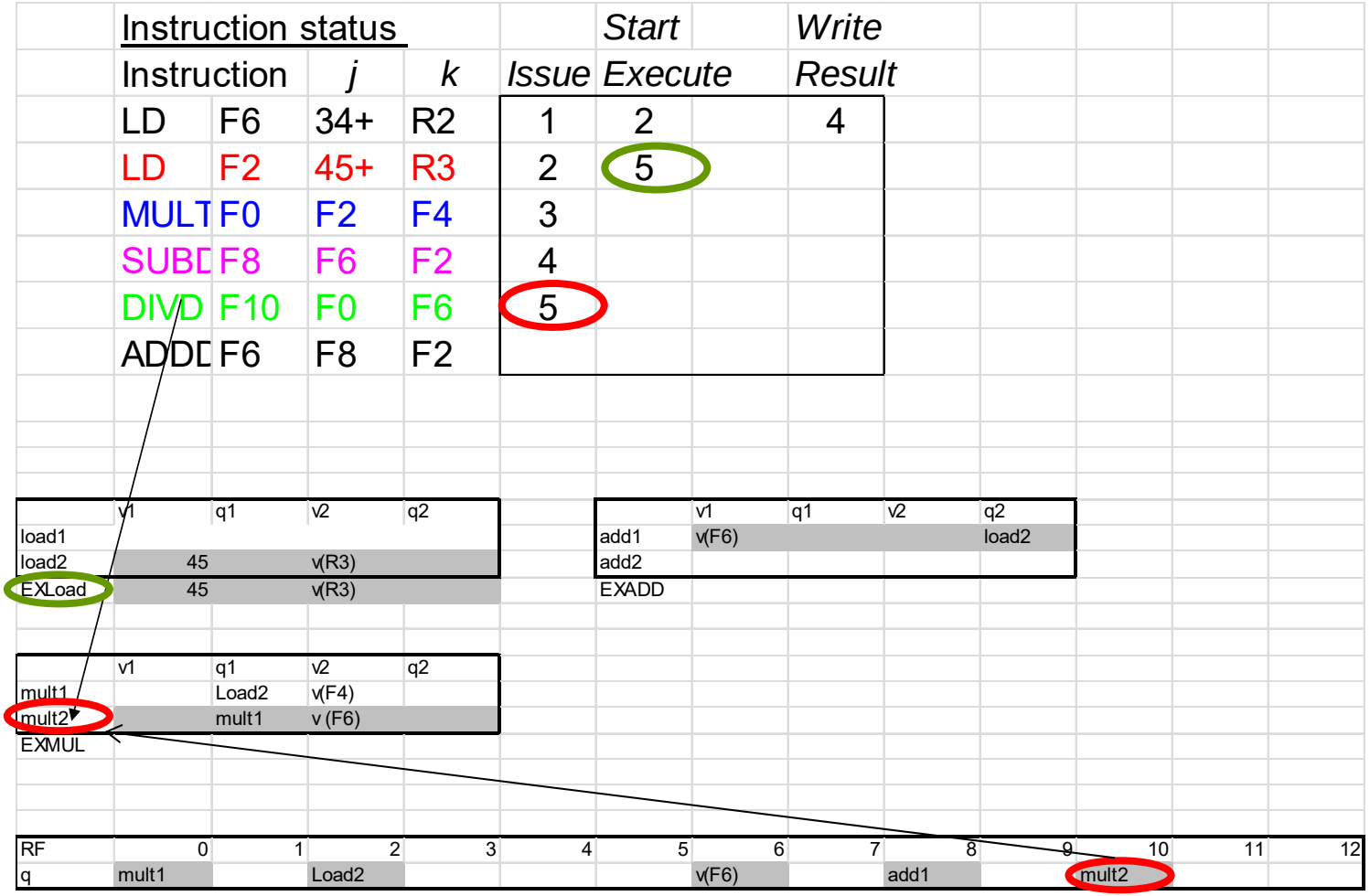


Tomasulo's example Cycle 4



Forwarding is provided
Writes on RF (F6) and RS of ADD1 through CDB

Tomasulo's example Cycle 5



Instruction status				Issue	Start Execute	Write Result
LD	F6	34+	R2	1	2	4
LD	F2	45+	R3	2	5	
MULT	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

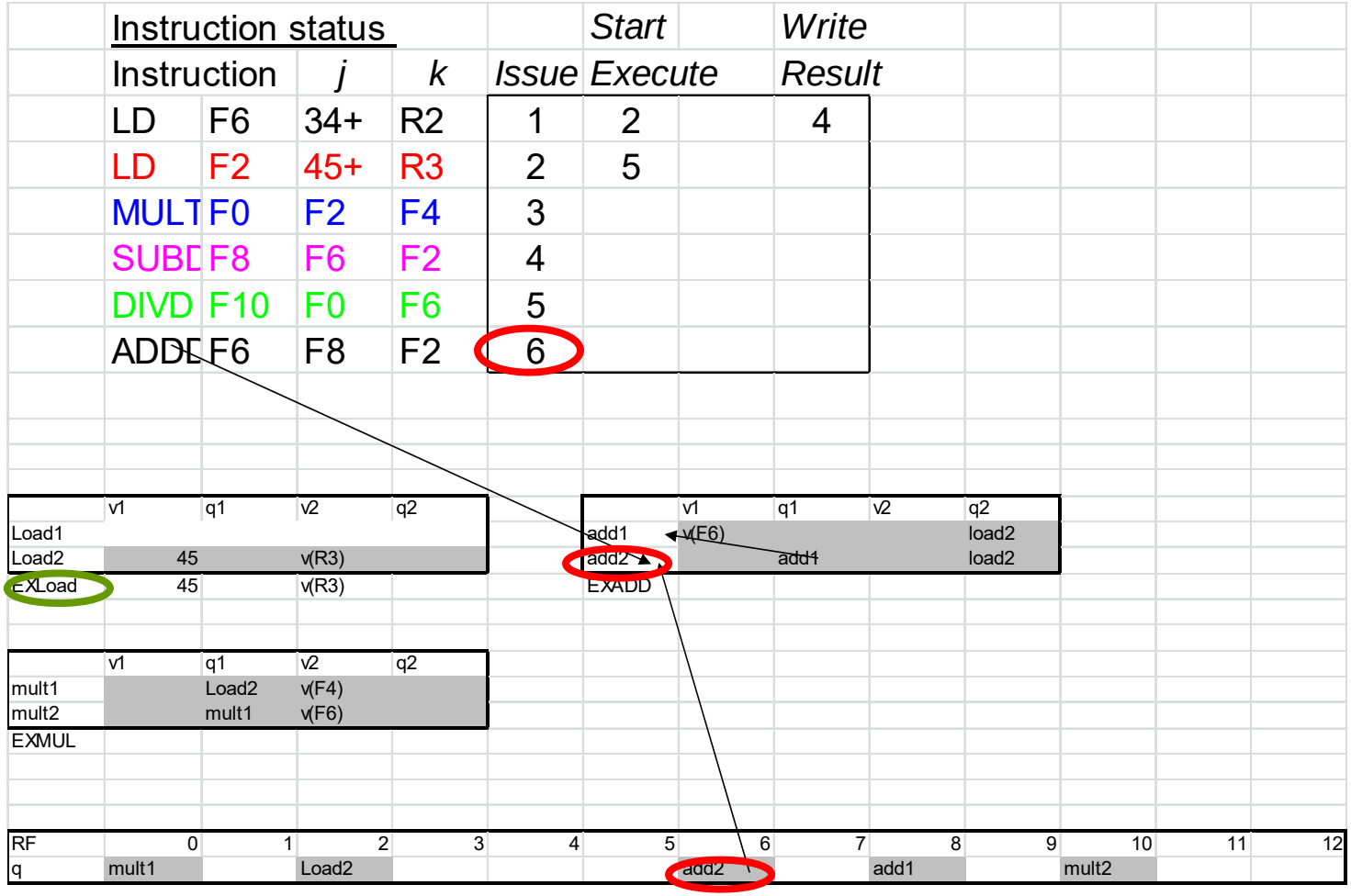
	v1	q1	v2	q2
load1				
load2	45		v(R3)	
EXLoad	45		v(R3)	

	v1	q1	v2	q2
add1	v(F6)			load2
add2				
EXADD				

	v1	q1	v2	q2
mult1		Load2	v(F4)	
mult2		mult1	v (F6)	
EXMUL				

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	mult1		Load2				v(F6)		add1		mult2		

Tomasulo's example Cycle 6



WAR on F6 has been eliminated: ADDD will write in F6
DIVD has already read v(F6) as v2 RS buffer @ Cycle 5
SUBD has already read v(F6) as v1 RS buffer @ Cycle 4

Tomasulo's example Cycle 7

Instruction status					Start		Write
Instruction		j	k	Issue	Execute		Result
LD	F6	34+	R2	1	2		4
LD	F2	45+	R3	2	5		7
MULT	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6			

	v1	q1	v2	q2
Load1				
Load2	45		v(R3)	
EXLoad	45		v(R3)	

CDB

	v1	q1	v2	q2
add1	v(F6)		v(F2)	
add2		add1	v(F2)	
EXADD				

	v1	q1	v2	q2
mult1	v(F2)		v(F4)	
mult2		mult1	v(F6)	
EXMUL				

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	mult1		v(F2)				add2		add1		mult2		

Forwarding is provided
Writes on RF (F2) and RSs through CDB

Tomasulo's example Cycle 8

Instruction status					Start	Write
Instruction		j	k	Issue	Execute	Result
LD	F6	34+	R2	1	2	4
LD	F2	45+	R3	2	5	7
MULT	F0	F2	F4	3	8	
SUBD	F8	F6	F2	4	8	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	v1	q1	v2	q2
Load1				
Load2				
EXLoad				

	v1	q1	v2	q2
add1	v(F6)		v(F2)	
add2		add1	v(F2)	
EXADD	v(F6)		v(F2)	

	v1	q1	v2	q2
mult1	v(F2)		v(F4)	
mult2		mult1	v(F6)	
EXMUL	v(F2)		v(F4)	

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	mult1		v(F2)				add2		add1		mult2		

Tomasulo's example Cycle 10

Instruction status					Start		Write
Instruction		j	k	Issue	Execute		Result
LD	F6	34+	R2	1	2		4
LD	F2	45+	R3	2	5		7
MULT	F0	F2	F4	3	8		
SUBD	F8	F6	F2	4	8		10
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6			

Latency MULTD: 10 cycles

Latency SUBD: 2 cycles

	v1	q1	v2	q2
Load1				
Load2				
EXLoad				

	v1	q1	v2	q2
add1	v(F6)		v(F2)	
add2	v(F8)		v(F2)	
EXADD	v(F6)		v(F2)	CDB

	v1	q1	v2	q2
mult1	v(F2)		v(F4)	
mult2		mult1	v(F6)	
EXMUL	v(F2)		v(F4)	

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	mult1		v(F2)				add2		v(F8)		mult2		

Tomasulo's example Cycle 11

Instruction status					Start		Write
Instruction		j	k	Issue	Execute		Result
LD	F6	34+	R2	1	2		4
LD	F2	45+	R3	2	5		7
MULT	F0	F2	F4	3	8		
SUBD	F8	F6	F2	4	8		10
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	11		

MULTD: 7 cycles remaining

	v1	q1	v2	q2
Load1				
Load2				
EXLoad				

	v1	q1	v2	q2
add1				
add2	v(F8)		v(F2)	
EXADD	v(F8)		v(F2)	

	v1	q1	v2	q2
mult1	**v(F2)**		**v(F4)**	
mult2		mult1	v(F6)	
EXMUL	v(F2)		v(F4)	

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	mult1		v(F2)				add2		v(F8)		mult2		

Tomasulo's example Cycle 13

Instruction status					Start		Write
Instruction		j	k	Issue	Execute		Result
LD	F6	34+	R2	1	2		4
LD	F2	45+	R3	2	5		7
MULT	F0	F2	F4	3	8		
SUBD	F8	F6	F2	4	8		10
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	11		13

MULTD: 5 cycles remaining

Latency ADDD: 2 cycles

	v1	q1	v2	q2
Load1				
Load2				
EXLoad				

	v1	q1	v2	q2
add1				
add2	v(F8)		v(F2)	
EXADD	v(F8)		v(F2)	

CDB

	v1	q1	v2	q2
mult1	**v(F2)**		**v(F4)**	
mult2		mult1	v(F6)	
EXMUL	v(F2)		v(F4)	

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	mult1		v(F2)				v(F6)		v(F8)		mult2		

WAR on F6 has already been eliminated:
ADDD writes result in CDB and in F6 (DIVD which has already read v(F6) at cycle 5)

Tomasulo's example Cycle 18

Instruction status

Instruction		j	k	Issue	Start Execute		Write Result
LD	F6	34+	R2	1	2		4
LD	F2	45+	R3	2	5		7
MULT	F0	F2	F4	3	8		18
SUBD	F8	F6	F2	4	8		10
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	11		13

	v1	q1	v2	q2
Load1				
Load2				
EXLoad				

	v1	q1	v2	q2
add1				
add2				
EXADD				

	v1	q1	v2	q2	
mult1	**v(F2)**		**v(F4)**		
mult2	v(F0)		v(F6)		
EXMUL	v(F2)		v(F4)		CDB

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	v(F0)		v(F2)				v(F6)		v(F8)		mult2		

Tomasulo's example Cycle 19

Instruction status					Start		Write
Instruction		*j*	*k*	*Issue*	*Execute*		*Result*
LD	F6	34+	R2	1	2		4
LD	F2	45+	R3	2	5		7
MULT	F0	F2	F4	3	8		18
SUBD	F8	F6	F2	4	8		10
DIVD	F10	F0	F6	5	19		
ADDD	F6	F8	F2	6	11		13

	v1	q1	v2	q2
Load1				
Load2				
EXLoad				

	v1	q1	v2	q2
add1				
add2				
EXADD				

	v1	q1	v2	q2
mult1				
mult2	v(F0)		v(F6)	
EXMUL	v(F0)		v(F6)	

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	v(F0)		v(F2)				v(F6)		v(F8)		mult2		

Tomasulo's example Cycle 59

Instruction status					Start		Write
Instruction		j	k	Issue	Execute		Result
LD	F6	34+	R2	1	2		4
LD	F2	45+	R3	2	5		7
MULT	F0	F2	F4	3	8		18
SUBD	F8	F6	F2	4	8		10
DIVD	F10	F0	F6	5	19		59
ADDD	F6	F8	F2	6	11		13

Latency DIVD: 40 cycles

	v1	q1	v2	q2
Load1				
Load2				

EXLoad

	v1	q1	v2	q2
add1				
add2				

EXADD

	v1	q1	v2	q2
mult1				
mult2	v(F0)		v(F6)	

EXMUL v(F0) v(F(6)) CDB

RF	0	1	2	3	4	5	6	7	8	9	10	11	12
q	v(F0)		v(F2)				v(F6)		v(F8)		v(F10)		

Compare Scoreboard vs Tomasulo

Instruction status:

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

Issue	*Start Exec*	*Write Result*
1	2	4
2	5	7
3	8	18
4	8	10
5	19	59
6	11	13

Tomasulo (IBM) versus Scoreboard (CDC)

- Issue window size=5
- No issue on structural hazards in RS
- WAR, WAW avoided with renaming
- Broadcast results from FU
- Control distributed on RS
- Allows loop unrolling in HW

- Issue window size=12
- No issue on structural hazards in FU
- Stall the completion for WAW and WAR hazards
- Results written back on registers.
- Control centralized through the Scoreboard.

Limits to the Instruction Level Parallelism

- Branches
- Exceptions
  - (non-)Precise: operand integrity for the exception handler
  - (non-)Exact: handler modifications are seen by instructions after the exception

Tomasulo Drawbacks

- Complexity
  - Large amount of hardware
  - Delays of 360/91, MIPS 10000, IBM 620?
- Many associative stores (CDB) at high speed
- Performance limited by Common Data Bus
  - Multiple CDBs $\Rightarrow$ More FU logic for parallel assoc stores

Summary (1)

- **HW exploiting ILP**
  - Works when can't know dependence at compile 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 & data dependencies
  - Original version didn't handle forwarding
  - No automatic register renaming

Summary (2)

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

Dynamic Scheduling Techniques: Scoreboard vs. Tomasulo

SCOREBOARD BASIC SCHEME

- IN-ORDER ISSUE
- OUT-OF-ORDER READ OPERANDS
- OUT-OF-ORDER EXECUTION
- OUT-OF-ORDER COMPLETION
- NO FORWARDING
- Control is centralized into the Scoreboard

SCOREBOARD STAGES

- **ISSUE (IN-ORDER):**
  - Check for structural hazards
  - Check for WAW hazards on destination ops
- **READ OPERANDS (OUT-OF-ORDER)**
  - Check for RAW hazards
  - Check for structural hazards in reading RF
- **EXECUTION (OUT-OF-ORDER)**
  - Execution completion depends on latency of FUs
  - Execution completion of LD/ST depends on cache hit/miss latencies)
- **WRITE RESULTS (OUT-OF-ORDER)**
  - Check for WAR hazards on destionation ops
  - Check for structural hazards in writing RF

SCOREBOARD optimisations

- Check for WAW postponed in WRITE stage instead of in ISSUE stage
- Forwarding

TOMASULO BASIC SCHEME

- IN-ORDER ISSUE
- OUT-OF-ORDER EXECUTION
- OUT-OF-ORDER COMPLETION
- REGISTER RENAMING based on Reservation Stations to avoid WAR and WAW hazards
- Results dispatched to RESERVATION STATIONS and to RF through the Common Data Bus
- Control is distributed on Reservation Stations
- *Reservation Stations offer a sort of data forwarding!*

TOMASULO STAGES

- **ISSUE (IN-ORDER):**
  - Check for structural hazards in Reservation Stations (not in FU)
- **START EXECUTE (OUT-OF-ORDER)**
  - When operands ready (Check for RAW hazards solved)
  - When FU available (Check for structural hazards in FU)
- **WRITE RESULTS (OUT-OF-ORDER)**
  - Execution completion depends on latency of FUs
  - Execution completion of LD/ST depends on cache hit/miss latencies
  - Write results on Common Data Bus to Reservations Stations, Store Buffers and RF