Course on: “Advanced Computer Architectures”

Tomasulo Dynamic Scheduling Algorithm

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

Tomasulo Algorithm

- Another dynamic scheduling algorithm:
  Tomasulo again enables instructions execution behind a stall to proceed
- **Tomasulo introduces the Implicit Register Renaming to avoid WAR & WAW hazards**
- Same goal: To get high performance at runtime without special compilers
- Invented at IBM 3 years after CDC 6600 for the IBM 360/91
- Lead to Alpha 21264, HP 8000, MIPS 10000, Pentium II, PowerPC 604 and may other recent microprocessors.

How to use Register Renaming statically by the compiler to avoid WAR & WAW hazards

```
DIV.D F0,F2,F4
ADD.D F6,F0,F8      # RAW F0
S.D F6,0(R1)        # RAW F6
MUL.D F6,F10,F8     # WAW F6 with ADD.D,
                    # WAR F6 with Store
```

Register Renaming introduces a register **S** to avoid WAR and WAW hazards:

```
DIV.D F0,F2,F4
ADD.D S,F0,F8       # RAW F0
S.D S,0(R1)         # RAW S
MUL.D F6,F10,F8
```

How to use Implicit Register Renaming to avoid dynamically WAR & WAW hazards

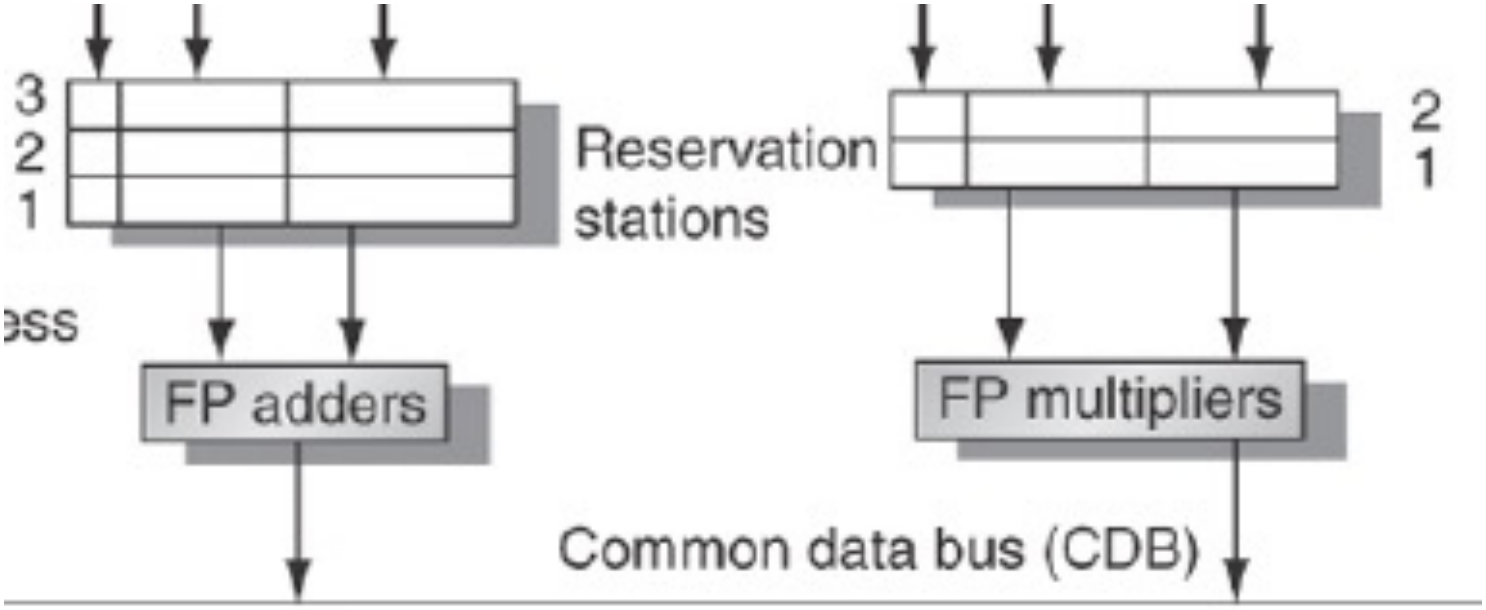
```
DIV.D F0,F2,F4
ADD.D F6,F0,F8      # RAW F0
S.D F6,0(R1)        # RAW F6
MUL.D F6,F10,F8     # WAW F6 with ADD.D,
                    # WAR F6 with S.D
```

Implicit Register Renaming uses the Reservation Station **RS1** to avoid WAR and WAW hazards:

```
DIV.D F0,F2,F4
ADD.D RS1,F0,F8     # RAW F0
S.D RS1,0(R1)       # RAW RS1
MUL.D F6,F10,F8
```

Tomasulo basic concepts (1)

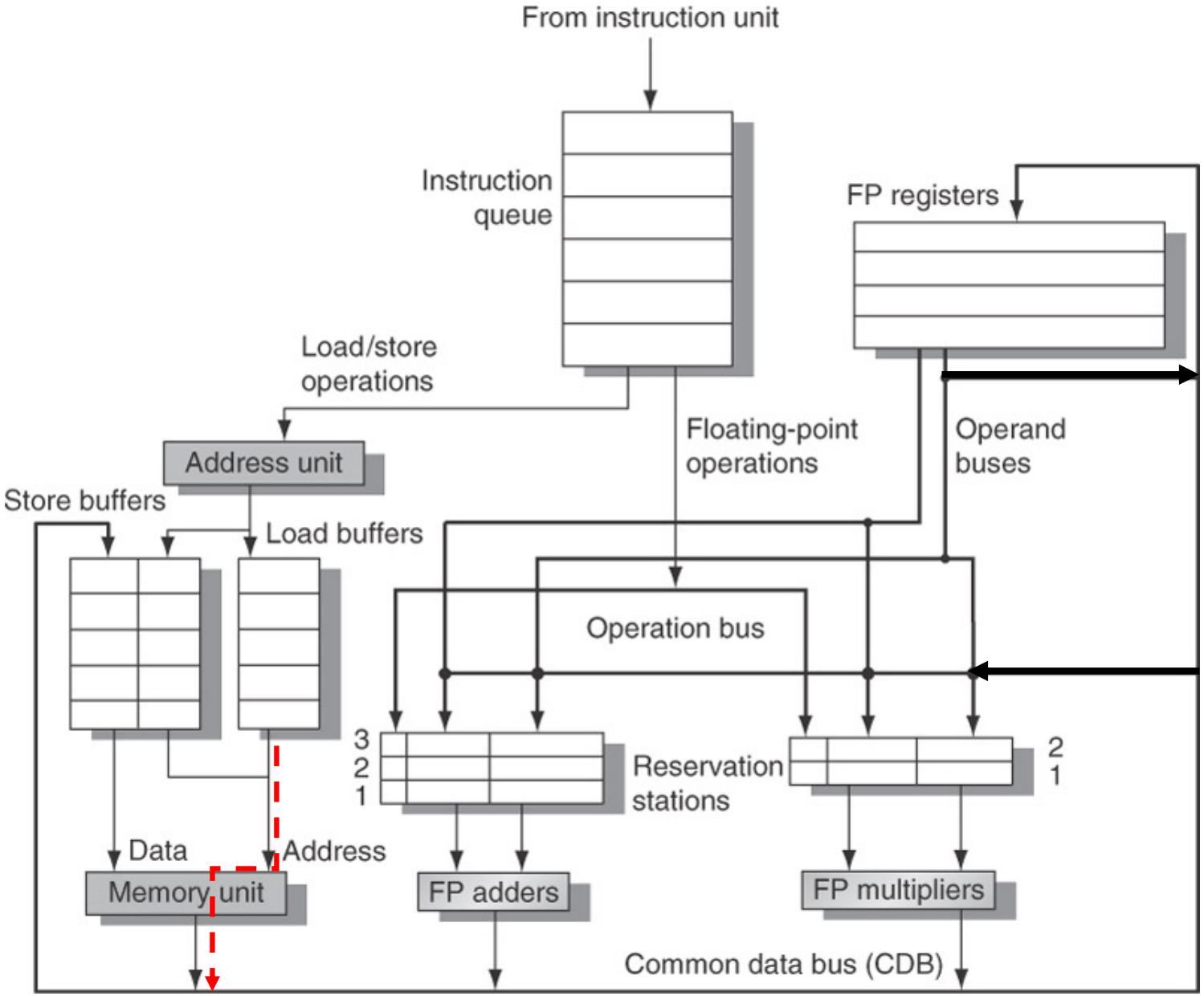
- Tomasulo introduces some FU buffers called **"Reservation Stations"** in front of the FUs to keep pending operands
- The control logic & RSs are **distributed** with the Function Units (vs. centralized in scoreboard);



Tomasulo basic concepts (2)

- Registers in instructions are replaced by their 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 are passed to the FUs from Reservation Stations, not through Registers, over to Common Data Bus that broadcasts results to all Fus and to Store buffers** *(like a sort of forwarding)*
- **Store buffers** are treated as a sort of RSs as well

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 j and k
  - **V_j** holds memory address for loads/stores
- **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: Either V-field or Q-field is valid for each operand

Register File and Store Buffers

- **Each entry in the RF and in the Store buffers have a Value (Vi) and a Pointer (Qi) field.**
  - The **Value (Vi)** field holds the register/buffer content;
  - The **Pointer (Qi)** field corresponds to the number of the RS producing the result to be stored in this register (or store buffer);
  - If the pointer is zero means that the value is available in the register/buffer content (no active instruction is computing the result);

Load/Store Buffers

- **Load/Store buffers have Busy and Address field.**
- **Address field:** To hold info for memory address calculation for load/stores.
  Initially it contains the instruction offset (immediate field); after address calculation, it stores the effective address.

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

	Busy	*Addr*	*Fu*
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Store instructions in the **Store Buffers** wait for the value given by the RF or the FUs to be sent to the memory unit.

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 there is a RS empty (i.e., check for structural hazards in RS) otherwise instruction *stalls.***
- If operands are not ready in RF, keep track of FU that will produce them **(Q pointers)** – this step ***renames registers,*** eliminating WAR, WAW hazards

First stage of Tomasulo Algorithm

ISSUE (cont'd)

- **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 (previously issued) 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

Start Execution

- **When both operands ready (Check for RAW hazards solved)**
- **When FU available (Check for structural hazards in FU)**
- If not ready, monitor 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 (we need to check if execution unit is available: critical choice for loads/stores to be kept in program order!)
- 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

Start Execution (cont'd)

- ***Load and Stores:*** Two-step execution process:
  - **First step:** compute effective address when base register is available, place it in load / store buffer.
  - **Second step:**
    - **Loads** in Load Buffers 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 are k*ept in program order** through effective address calculation – helps in preventing hazards through memory.

Second stage of Tomasulo Algorithm

Start Execution (cont'd)

- **To preserve exception behavior**:
  - No instruction can initiate execution until all branches preceding it in program order have completed;
  - This restriction guarantees that an instruction that generates an exception really would have been executed;
  - 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 it on Common Data Bus and from there into Register File and into all RSs (including store buffers)** waiting for this result;
- **Stores** also write data to memory unit during this stage (when memory address and result data are available);
- Mark reservation station available.

TOMASULO BASIC SCHEME

- IN-ORDER ISSUE
- OUT-OF-ORDER EXECUTION
- OUT-OF-ORDER COMPLETION
- IMPLICIT 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.

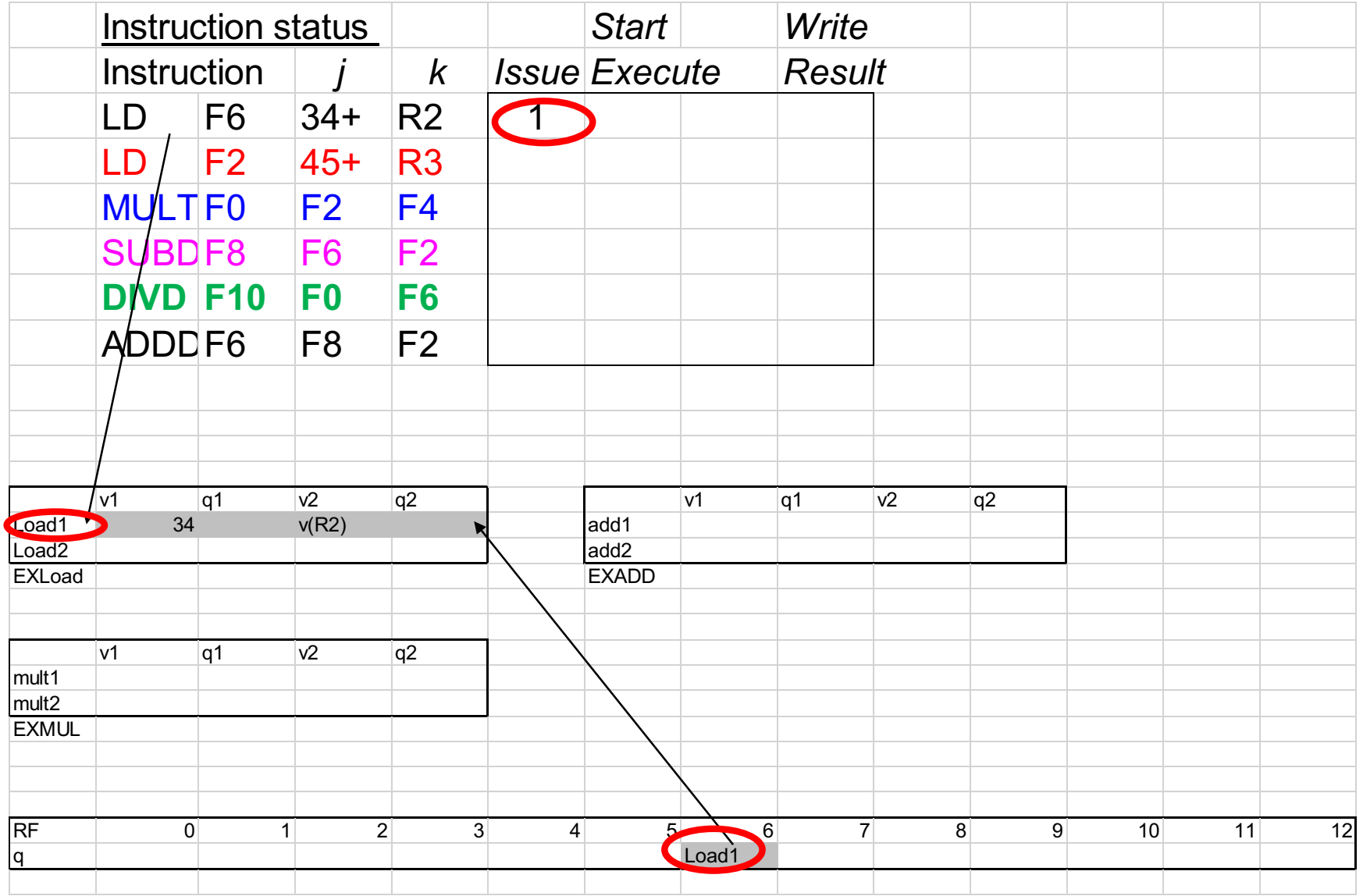
Tomasulo Example: Analysis of dependences and hazards

```
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
                    # WAW F6
```

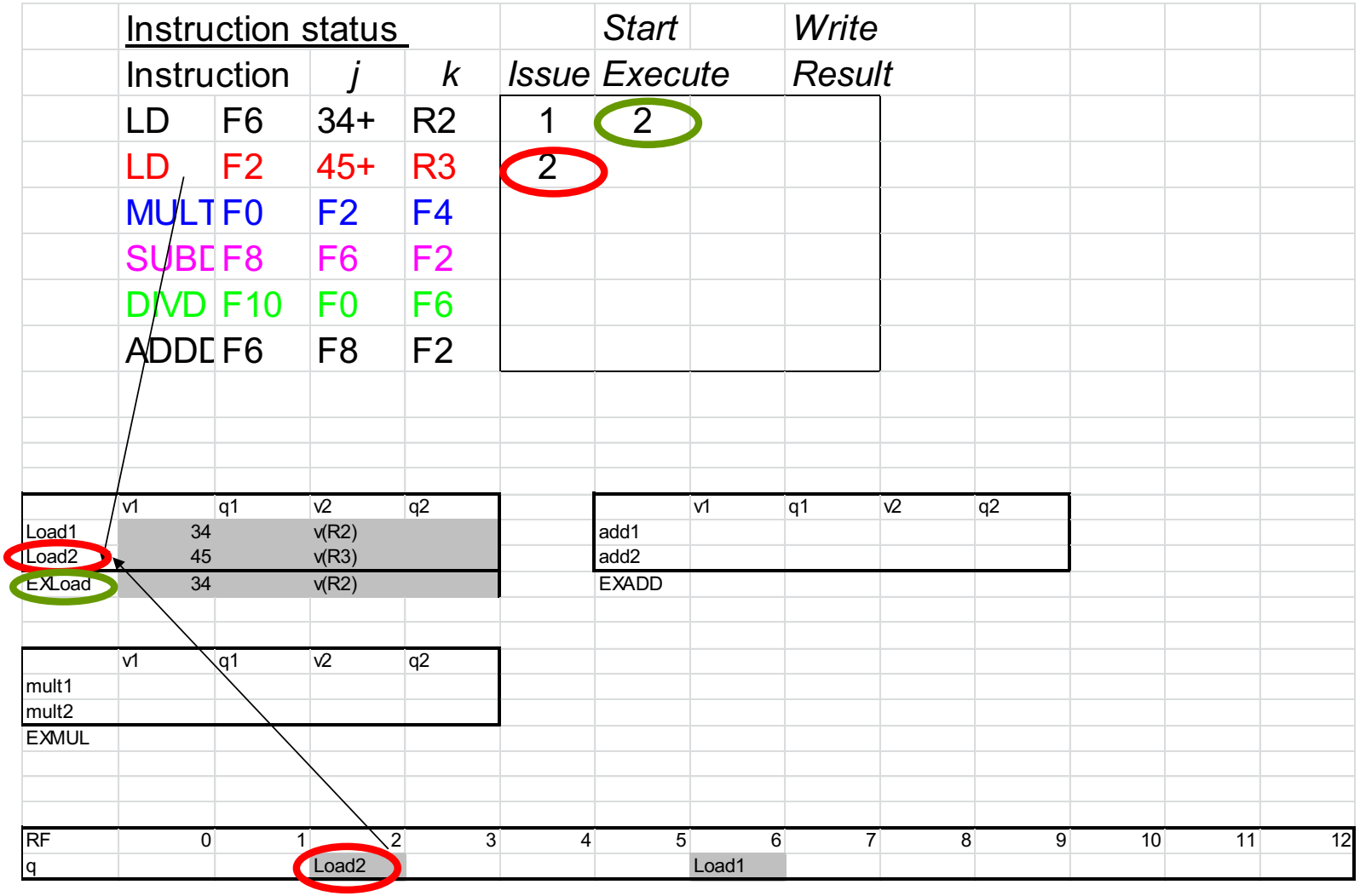
Tomasulo Example: Implicit Register Renaming to avoid WAR & WAW

```
LD Load1, 34(R2)
LD F2, 45(R3)
MULTD F0, F2, F4          # RAW F2
SUBD F8, Load1, F2        # RAW F2, RAW Load1
DIVD F10, F0, Load1       # RAW F0, RAW Load1
ADDD F6, F8, F2           # RAW F8, RAW F2
```

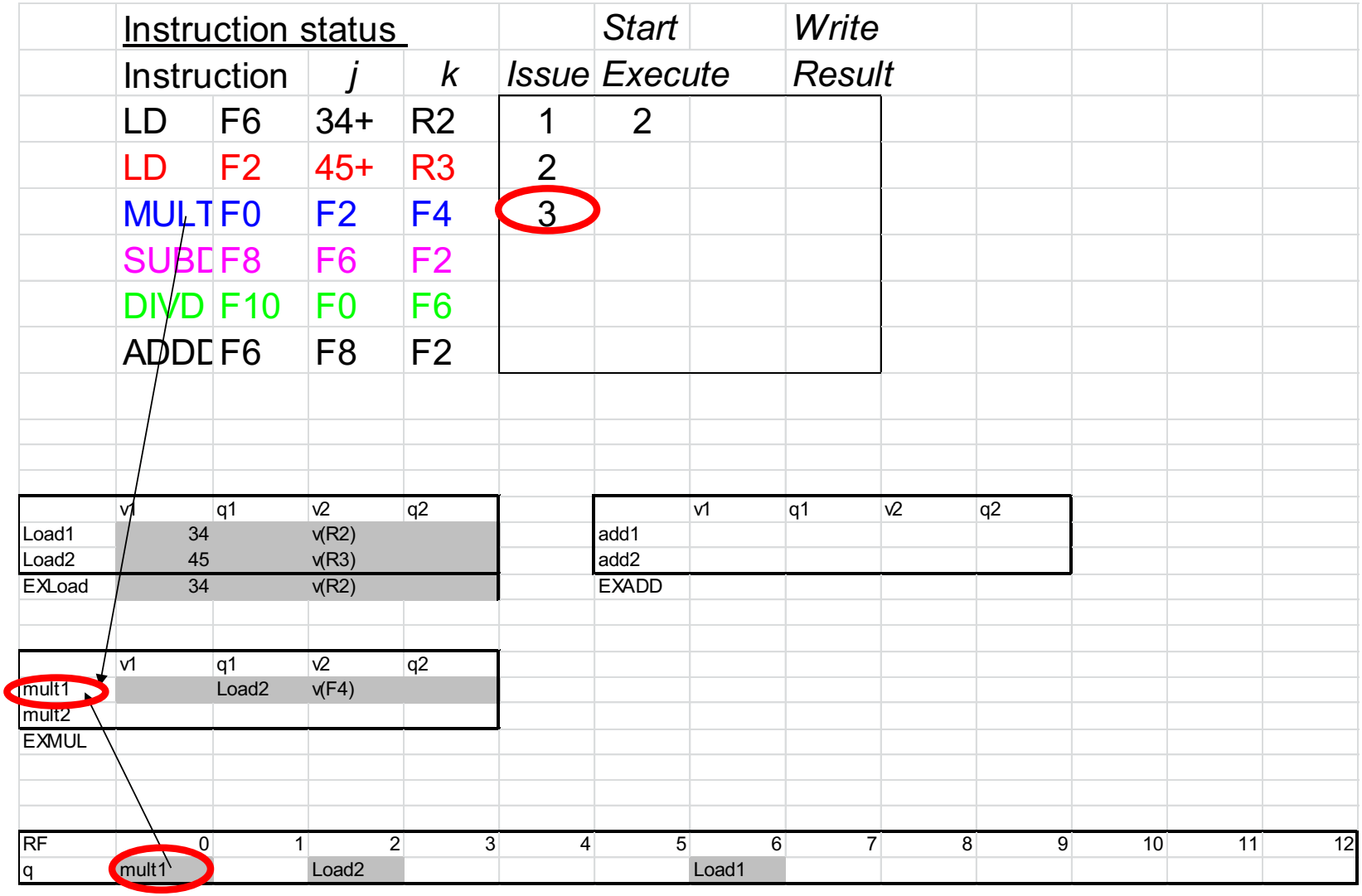
Tomasulo's example Cycle 1



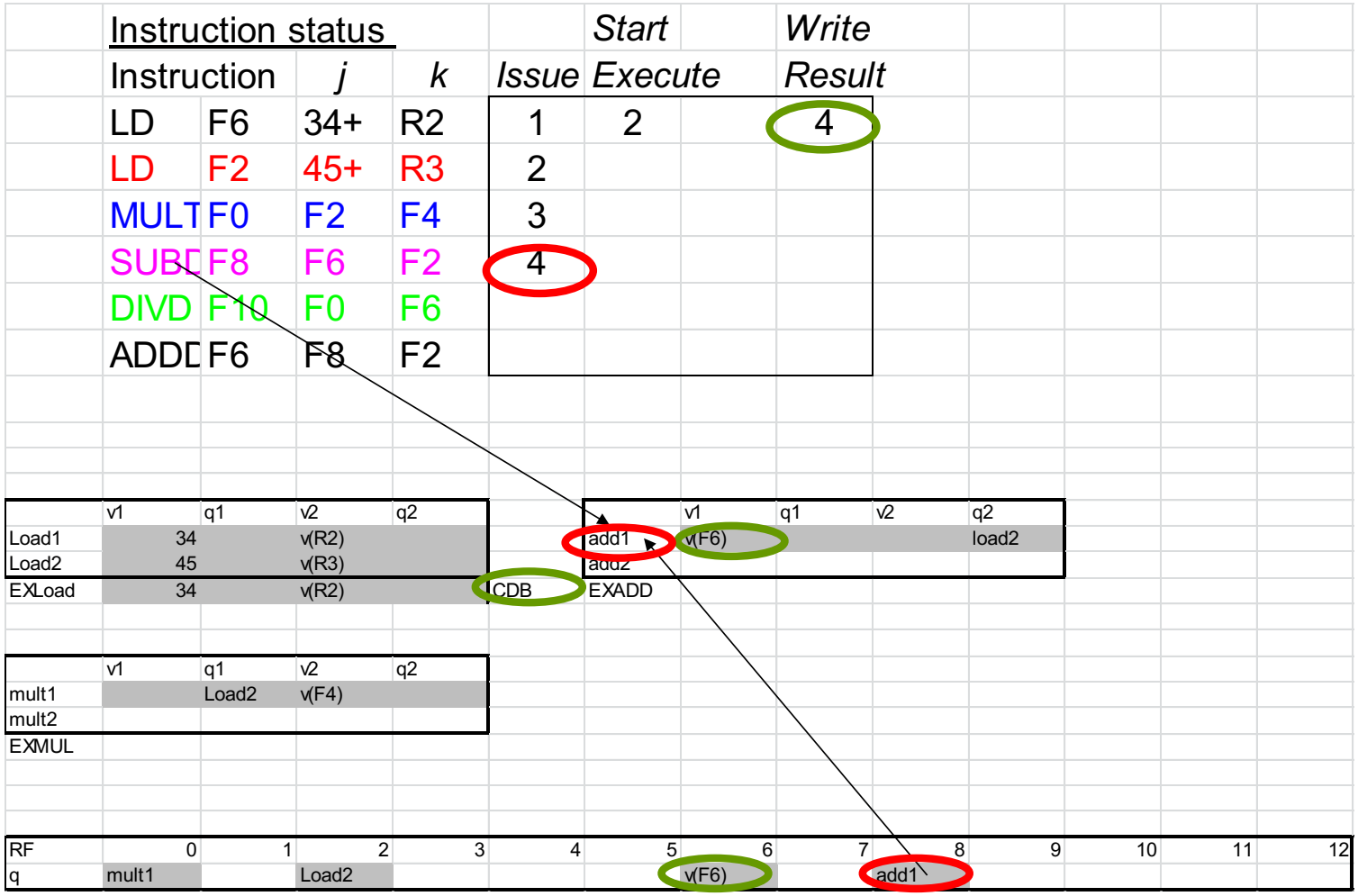
Tomasulo's example Cycle 2



Tomasulo's example Cycle 3

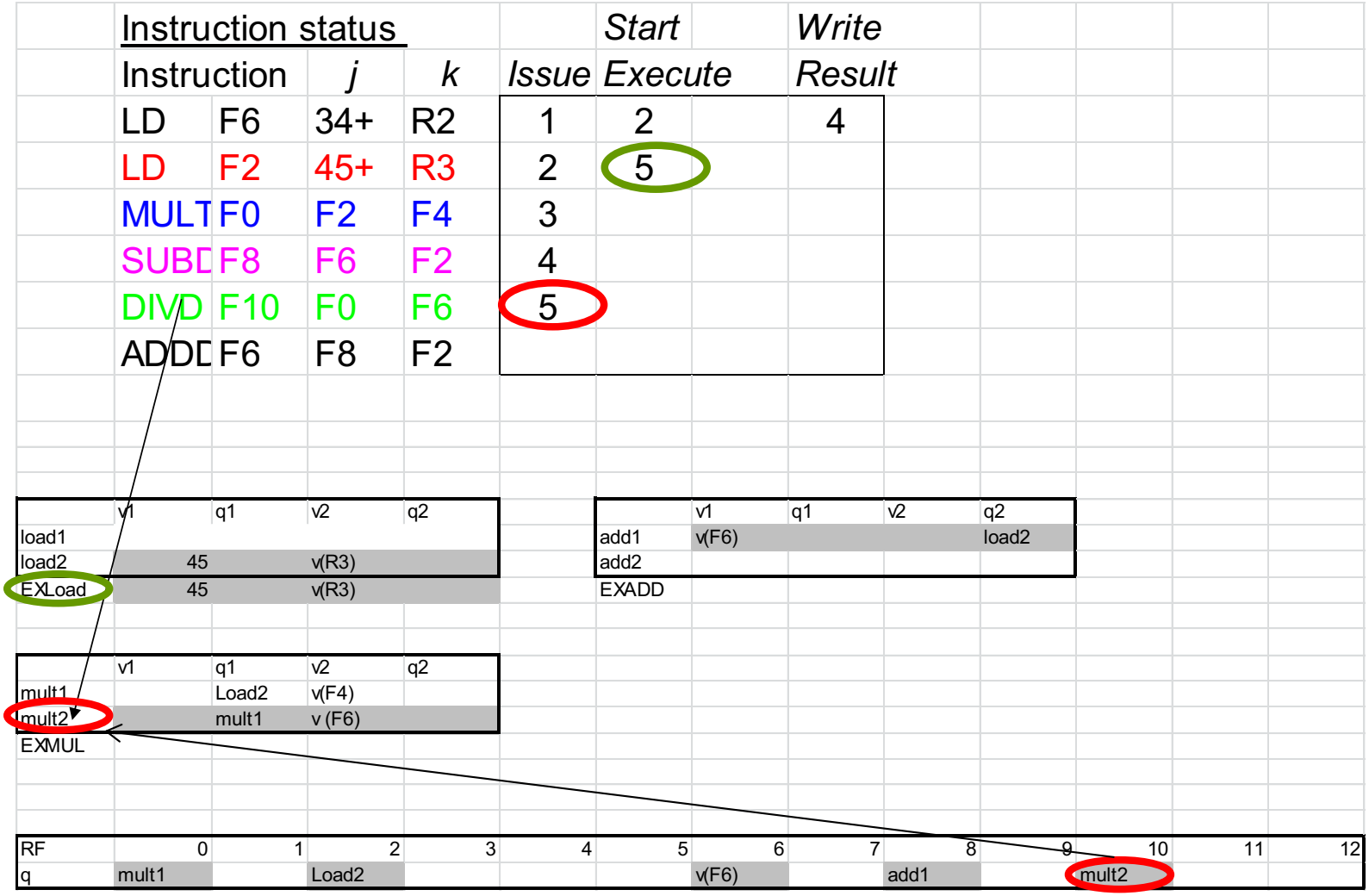


Tomasulo's example Cycle 4

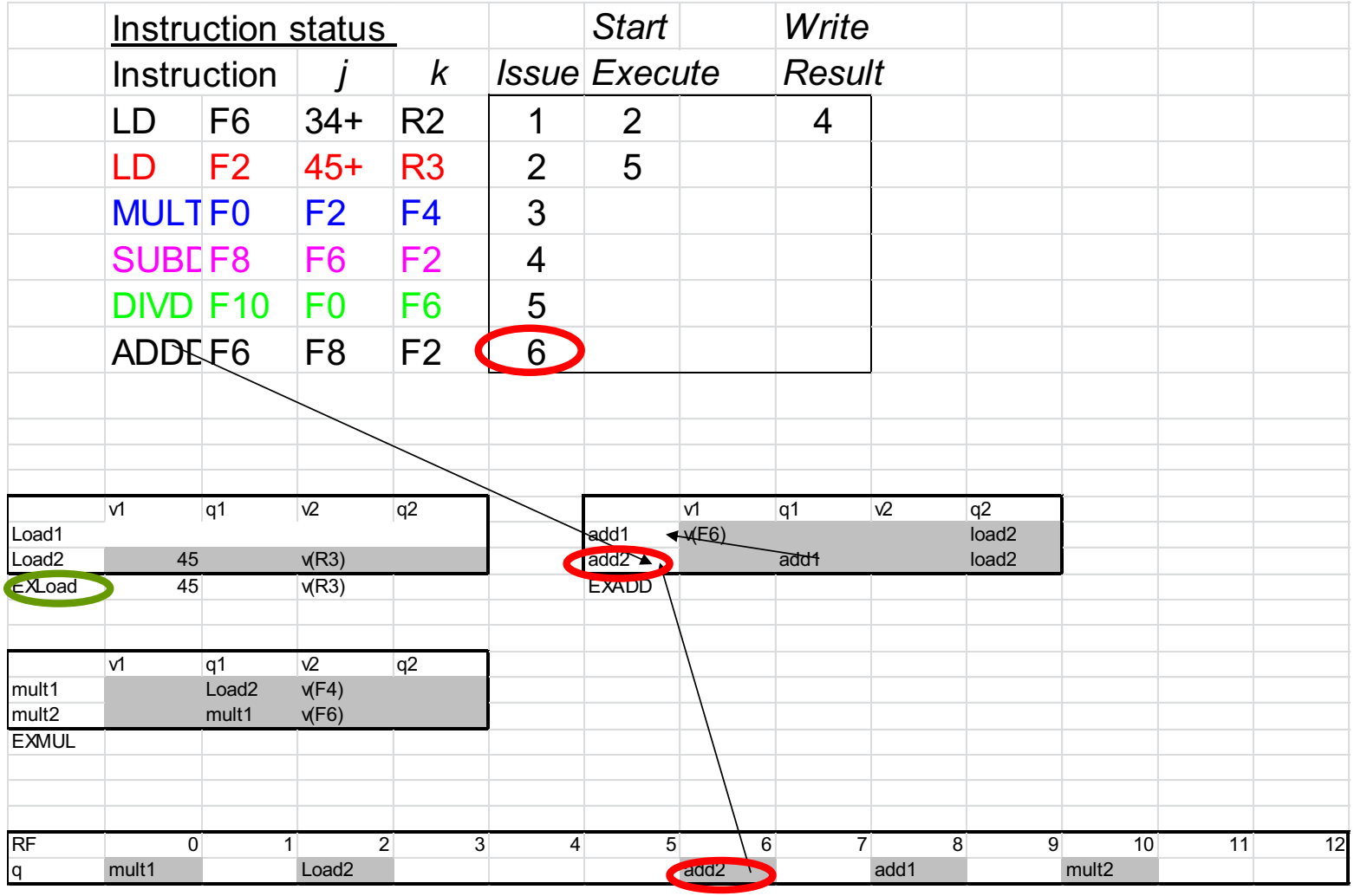


Forwarding is provided
Writes on RF (F6) and RS of ADD1 through CDB

Tomasulo's example Cycle 5

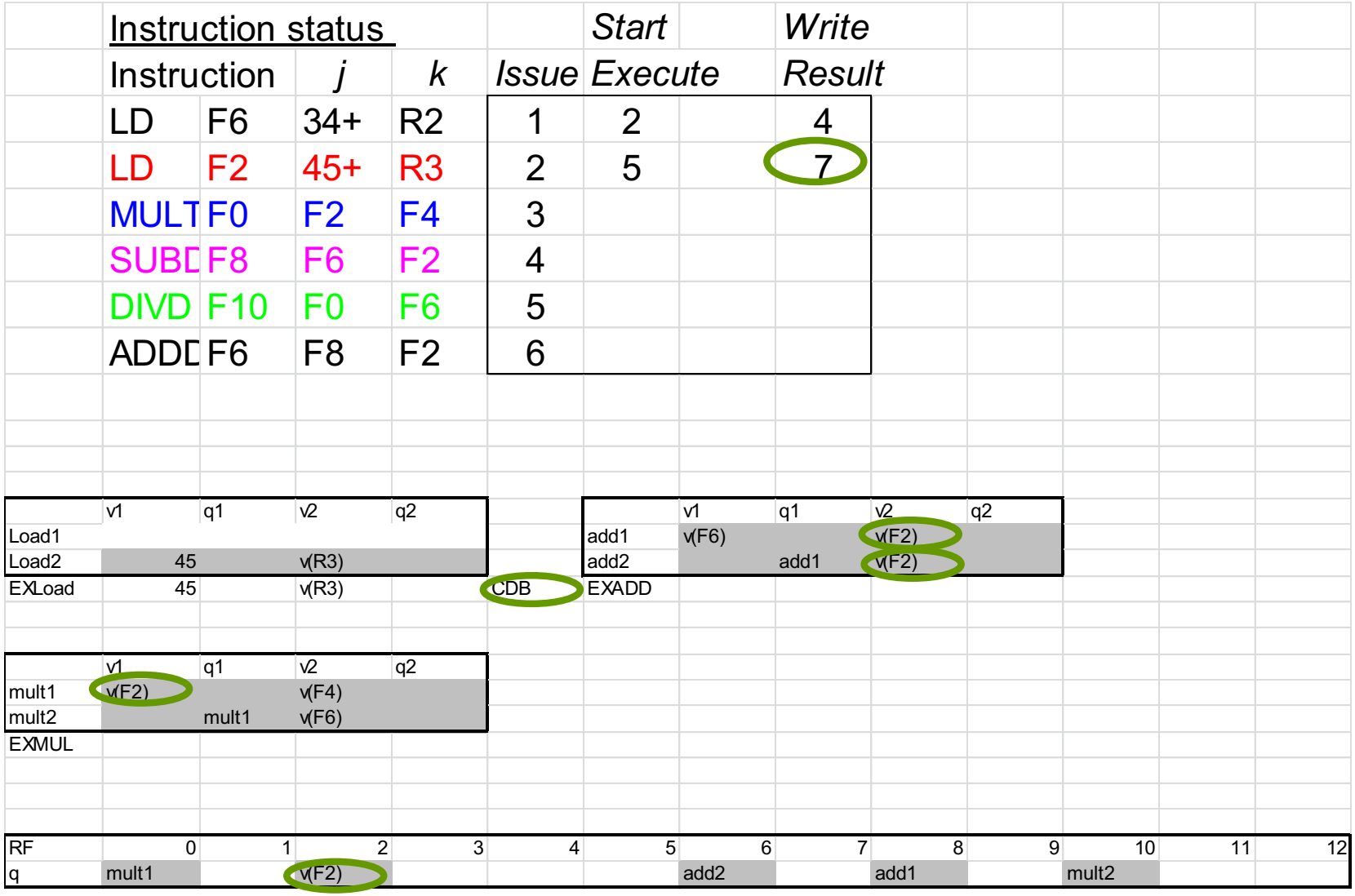


Tomasulo's example Cycle 6



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				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	
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				Issue	Start Execute		Write Result
Instruction		j	k				
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					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			
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.

Tomasulo Drawbacks

- Complexity
  - Large amount of hardware and power dissipation
- 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)

- **Hardware exploiting ILP dynamically.**
  - 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:** *Implicit register 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 and many other modern microprocessors.

Dynamic SchedulingTechniques: Recap 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