Superscalar Architecture

Muhammad Tahir

Lecture 14-15

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

Multiple-Issue Processor

- The processor performance

$$\begin{aligned} \text{Execution Time} &= N \times CPI \times TPC \\ N &\sim \text{number of instructions} \\ CPI &\sim \text{cycles per instructions} \\ TPC &\sim \text{time per cycle} \end{aligned}$$

- Pipelining provides
  - Instruction-level-parallelism (ILP) with scalar processor
  - CPI ≈ 1 (practically) OR $= 1$ (ideally) with TPC $\downarrow$
- Further performance improvement requires CPI < 1
- CPI < 1 requires multiple-issue architecture

Multiple-Issue Processor Cont'd

Different types of multiple-issue processors

- **Superscalar** processor
  - Statically scheduled superscalar processor implements in-order execution
  - Dynamically scheduled superscalar processor implements out-of-order execution
- **VLIW** (Very Long Instruction Word) processor

Program Execution Order

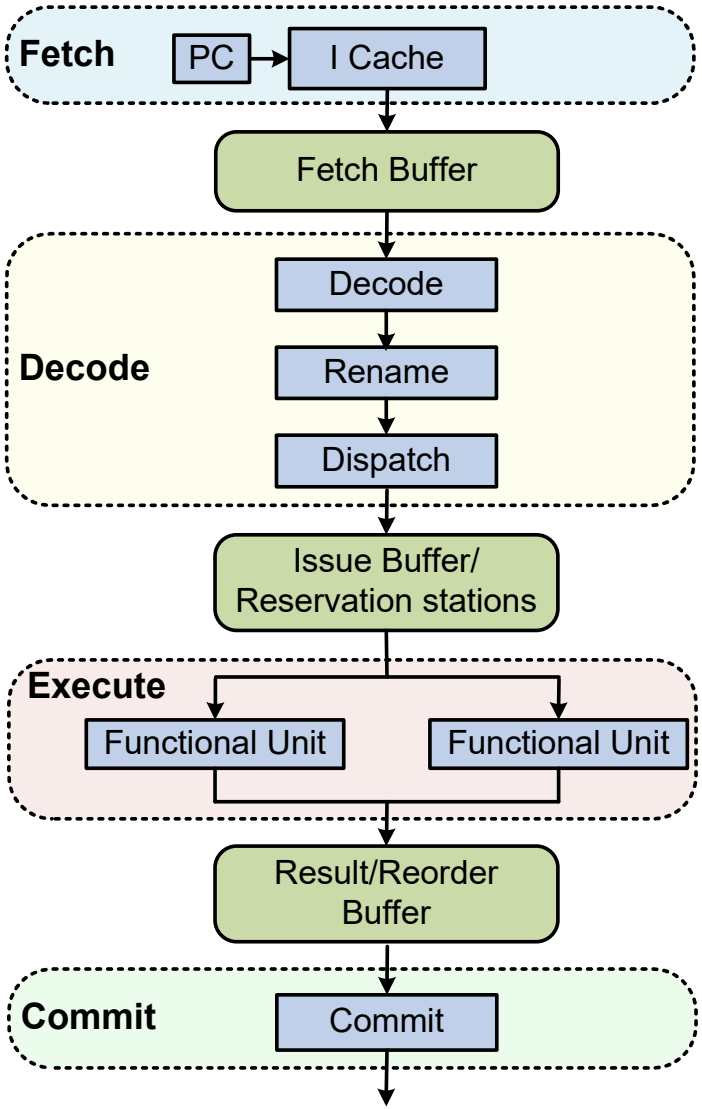
- Program Order: The order created by the compiler
- Fetch Order: The order of instruction fetching for execution, can be different from Program Order due to prefetching and trace-execution
- Execution Order: The order in which instructions are executed, can be in-order or out of order
- Commit Order: The order of commitment or retiring of instructions (order of updating registers and/or memory)

Instruction Execution Order

- Orders associated with instruction execution
  - Order in which instructions are **Issued**
  - Order in which instructions are **Completed**
- Useful execution ordering combinations
  - In-order Issue with In-order Completion
  - In-order Issue with Out-of-order Completion
  - Out-of-order Issue with Out-of-order Completion

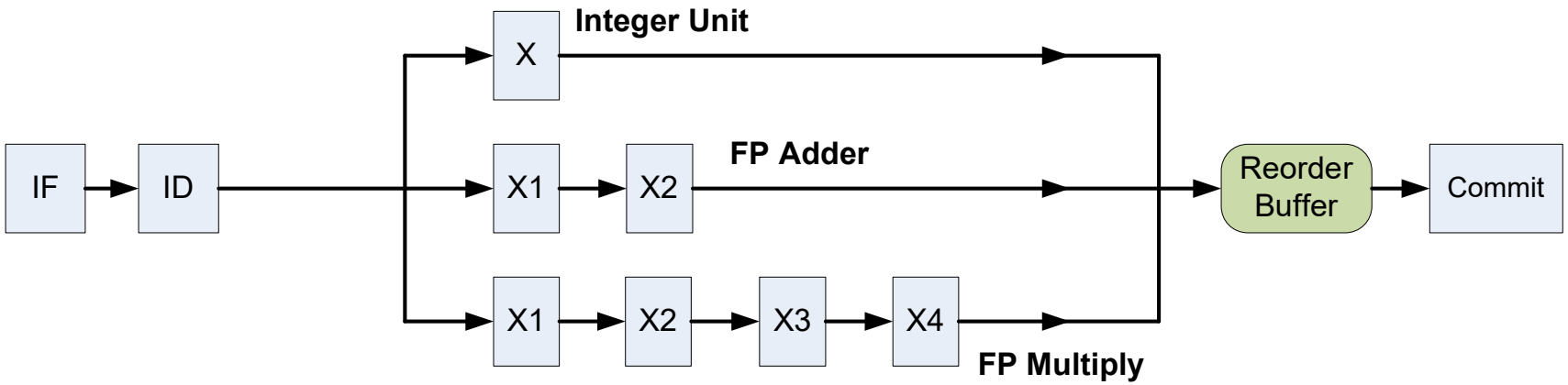
Instruction Execution Phases

- Four phases of instruction execution
- Any two phases have an intermediate buffer
- Each buffer is critical for multiple-issue architecture
- Instruction **fetch**, **decode**, **rename** and **dispatch** are in-order
- **Execute** and **commit** can be in-order or out-of-order



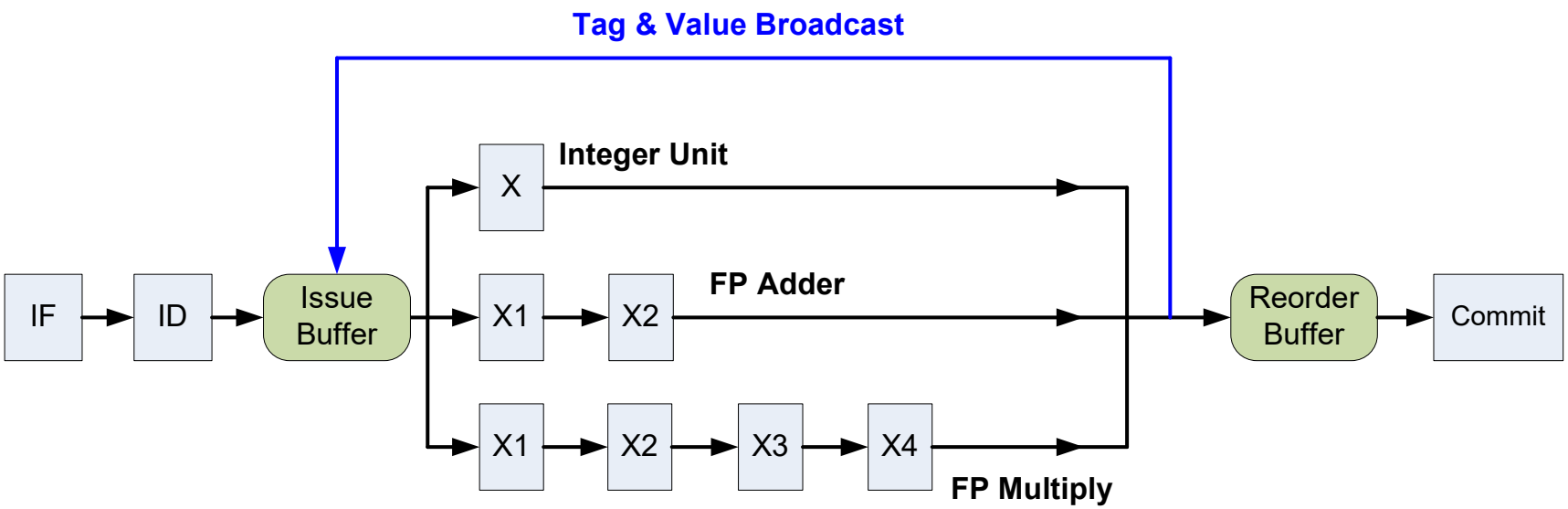
In-order Issue

- In-order issue and out-of-order completion
- Also called static scheduling
- Use reordering buffer for in-order commit



Out-of-Order Issue

- Out-of-order issue and out-of-order completion
- Also called dynamic scheduling
- Use issue buffer for out-of-order issue
- Use reordering buffer for in-order commit

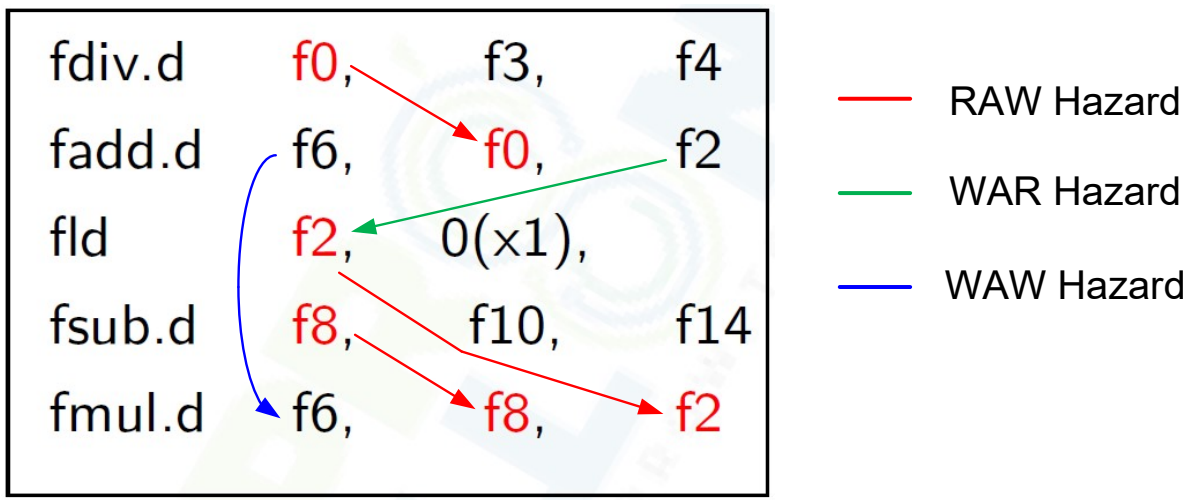


Problem with Out-of-Order Issue

- The WAW and WAR hazards
- False data dependencies

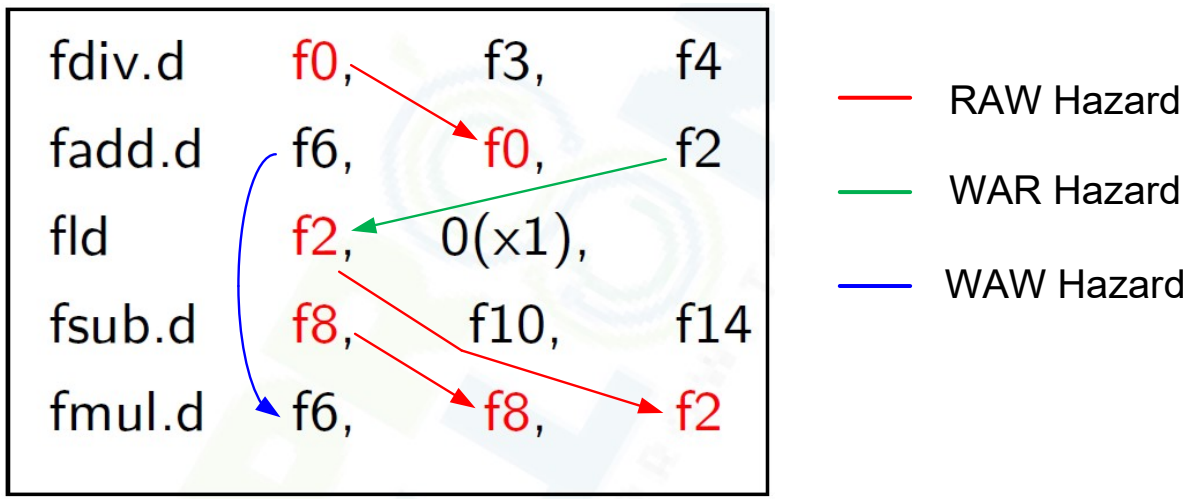
Problem with Out-of-Order Issue

- The WAW and WAR hazards
- False data dependencies
- Example program with floating point instructions



Problem with Out-of-Order Issue

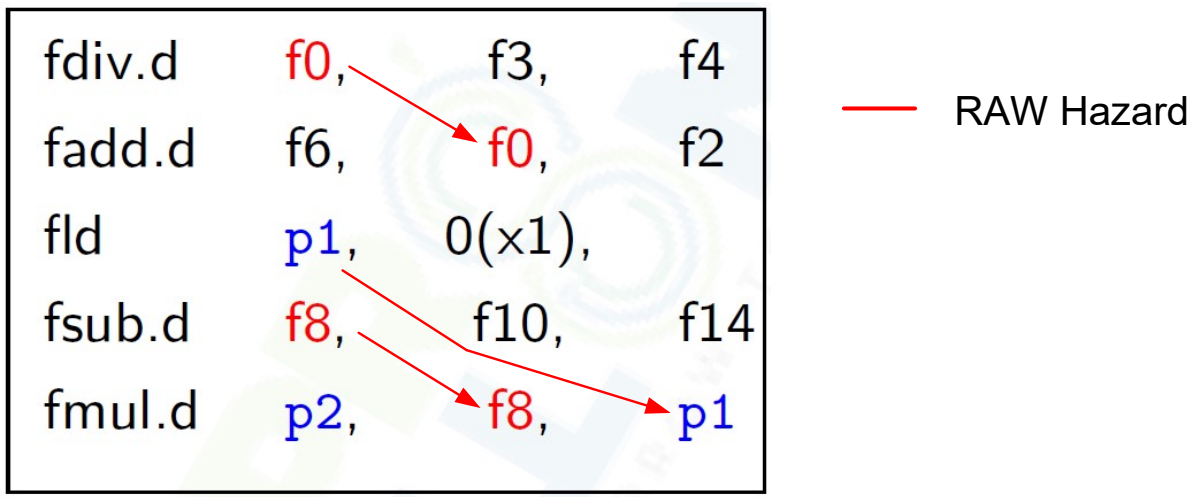
- The WAW and WAR hazards
- False data dependencies
- Example program with floating point instructions



- Assume `fdiv` takes 5 cycles, `fmul` takes 3 cycles, while other instructions take one cycle

Register Renaming

- The solution for WAW and WAR hazards
- Use register renaming



In-order Execution

- In-order issue
- No forwarding

	Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fdiv.d f0, f3, f4		F	D	X_1	X_2	X_3	X_4	X_5	W								

In-order Execution

- In-order issue
- No forwarding

				Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fdiv.d	f0,	f3,	f4		F	D	X_1	X_2	X_3	X_4	X_5	W								
fadd.d	f6,	f0,	f2			F	D	-	-	-	-	-	X_1	W						

In-order Execution

- In-order issue
- No forwarding

				Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fdiv.d	f0,	f3,	f4		F	D	X_1	X_2	X_3	X_4	X_5	W								
fadd.d	f6,	f0,	f2			F	D	-	-	-	-	-	X_1	W						
fld	f2,	0(x1),					F	-	-	-	-	-	D	X_1	W					

In-order Execution

- In-order issue
- No forwarding

				Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fdiv.d	f0,	f3,	f4		F	D	X_1	X_2	X_3	X_4	X_5	W								
fadd.d	f6,	f0,	f2			F	D	-	-	-	-	-	X_1	W						
fld	f2,	0(x1),					F	-	-	-	-	-	D	X_1	W					
fsub.d	f8,	f10,	f14										F	D	X_1	W				

In-order Execution

- In-order issue
- No forwarding

				Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fdiv.d	f0,	f3,	f4		F	D	X_1	X_2	X_3	X_4	X_5	W								
fadd.d	f6,	f0,	f2			F	D	-	-	-	-	-	X_1	W						
fld	f2,	0(x1),					F	-	-	-	-	-	D	X_1	W					
fsub.d	f8,	f10,	f14										F	D	X_1	W				
fmul.d	f6,	f8,	f2											F	D	-	X_1	X_2	X_3	W

Out-of-order Execution

- Out-of-order issue
- With forwarding and imprecise exceptions

	Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
fdiv.d f0, f3, f4		F	D	X_1	X_2	X_3	X_4	X_5	W					

Out-of-order Execution

- Out-of-order issue
- With forwarding and imprecise exceptions

				Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
fdiv.d	f0,	f3,	f4		F	D	X_1	X_2	X_3	X_4	X_5	W					
fadd.d	f6,	f0,	f2			F	D	-	-	-	-	X_1	W				

Out-of-order Execution

- Out-of-order issue
- With forwarding and imprecise exceptions

				Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
fdiv.d	f0,	f3,	f4		F	D	X_1	X_2	X_3	X_4	X_5	W					
fadd.d	f6,	f0,	f2			F	D	-	-	-	-	X_1	W				
fld	f2,	0(x1),					F	D	-	-	-	-	X_1	W			

Out-of-order Execution

- Out-of-order issue
- With forwarding and imprecise exceptions

				Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
fdiv.d	f0,	f3,	f4		F	D	X_1	X_2	X_3	X_4	X_5	W					
fadd.d	f6,	f0,	f2			F	D	-	-	-	-	X_1	W				
fld	f2,	0(x1),					F	D	-	-	-	-	X_1	W			
fsub.d	f8,	f10,	f14					F	D	X_1	W						

Out-of-order Execution

- Out-of-order issue
- With forwarding and imprecise exceptions

		Cycles		1	2	3	4	5	6	7	8	9	10	11	12	13
fdiv.d	f0,	f3,	f4	F	D	X_1	X_2	X_3	X_4	X_5	W					
fadd.d	f6,	f0,	f2		F	D	-	-	-	-	X_1	W				
fld	f2,	0(x1),				F	D	-	-	-	-	X_1	W			
fsub.d	f8,	f10,	f14				F	D	X_1	W						
fmul.d	f6,	f8,	f2					F	D	-	-	-	X_1	X_2	X_3	W

Out-of-order Execution with Register Renaming

- Out-of-order issue with register renaming
- With forwarding, precise exceptions and multiple commits

		Cycles		1	2	3	4	5	6	7	8	9	10	11
fdiv.d	f0,	f3,	f4	F	D	X_1	X_2	X_3	X_4	X_5	R	W		
fadd.d	f6,	f0,	f2		F	D	-	-	-	-	X_1	R	W	
fld	p1,	0(x1),				F	D	X_1	R	-	-	-	W	

Out-of-order Execution with Register Renaming

- Out-of-order issue with register renaming
- With forwarding, precise exceptions and multiple commits

		Cycles		1	2	3	4	5	6	7	8	9	10	11
fdiv.d	f0,	f3,	f4	F	D	X_1	X_2	X_3	X_4	X_5	R	W		
fadd.d	f6,	f0,	f2		F	D	-	-	-	-	X_1	R	W	
fld	p1,	0(x1),				F	D	X_1	R	-	-	-	W	
fsub.d	f8,	f10,	f14				F	D	X_1	R	-	-	W	
fmul.d	p2,	f8,	p1					F	D	X_1	X_2	X_3	R	W

Dynamic Scheduling Techniques

- Two key approaches for Dynamic Scheduling
  - Scorboarding
  - Tomasulo's Algorithm

Tomasulo's Algorithm: Key Features

- Reservation Stations
  - A **Buffer** for a functional unit to hold instructions and their **operands**
  - An **operand** can hold an actual *value* or the *tag* of a reservation station entry. It can also hold a load buffer entry that will provide the value
- Common Data Bus
  - Connects *functional units* and *load buffer* to reservations stations, registers and store buffer
  - Responsible to transfer data *simultaneously* to *all* the stations/registers that require it

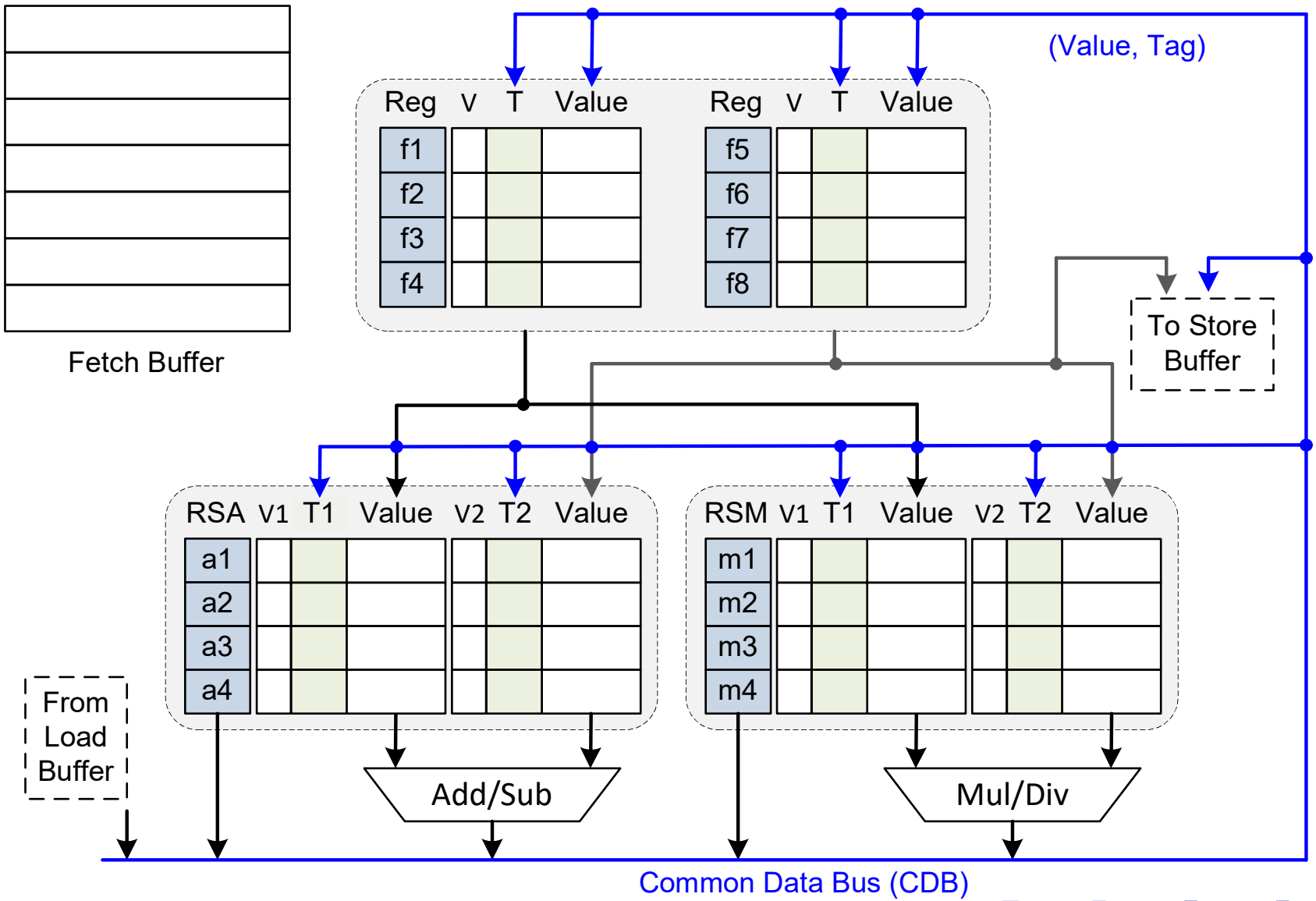
Tomasulo's Algorithm: Key Features Cont'd

- Addressing Hazards
  - Uses (hardware) register renaming to eliminate WAR and WAW hazards
  - Operands refer to reservation stations/load buffer (data producer) for the values
  - Usually have more reservation stations than registers (for effective register renaming)

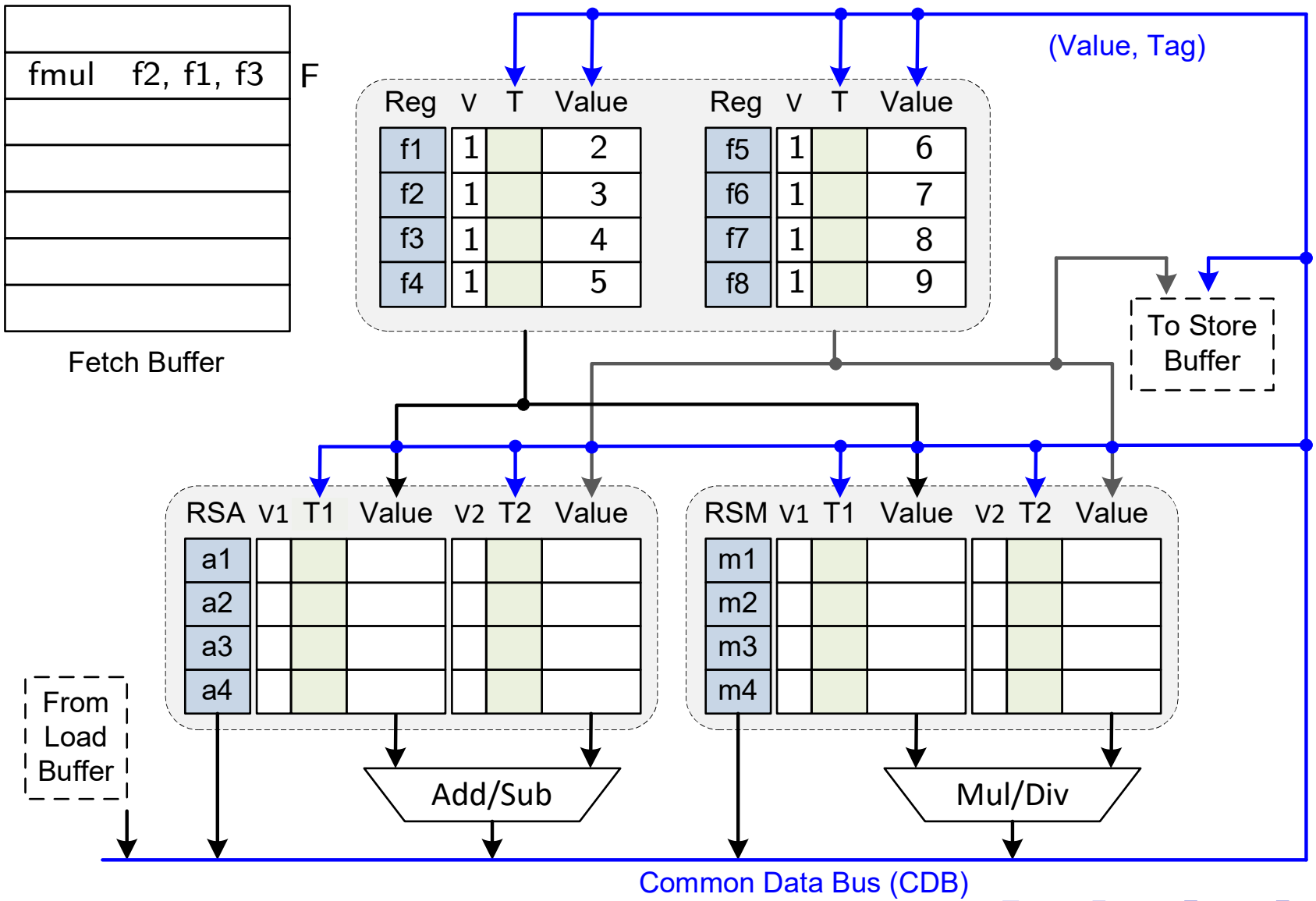
Tomasulo's Algorithm: Pipeline Stages

- Fetch: Get the instructions to **fetch buffer**
- Dispatch: Decode instruction, perform register renaming and allocate **reservation station** (RS) (or **issue buffer**), stall in case of structural hazard (no free space in RS)
- Execute: Issue the instruction in RS to **functional unit** (FU), in case of RAW hazard wait and monitor **common data bus** (CDB) for the value
- Write: Write the result to register (store buffer), free the RS entry

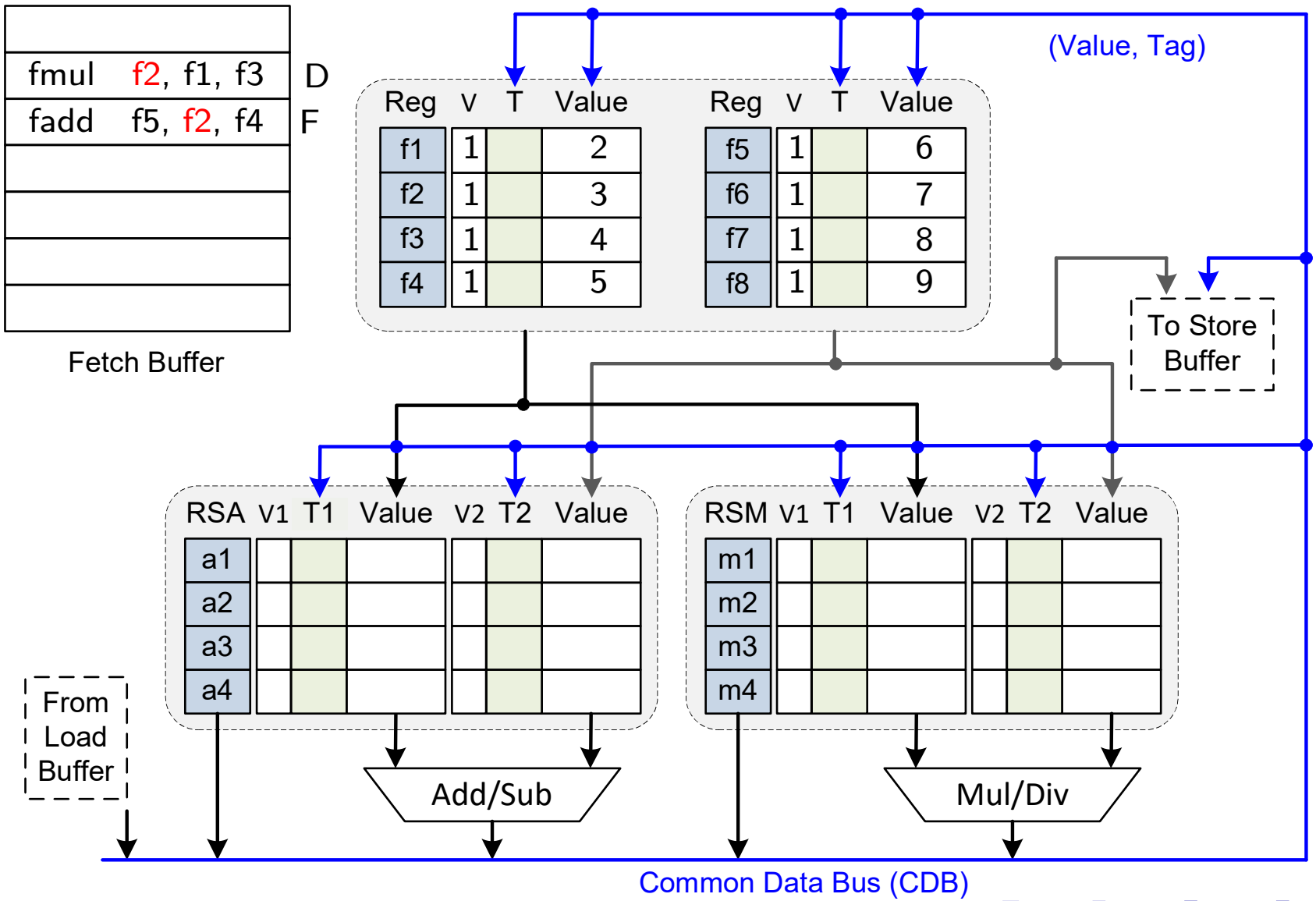
Tomasulo's Algorithm: Block Diagram



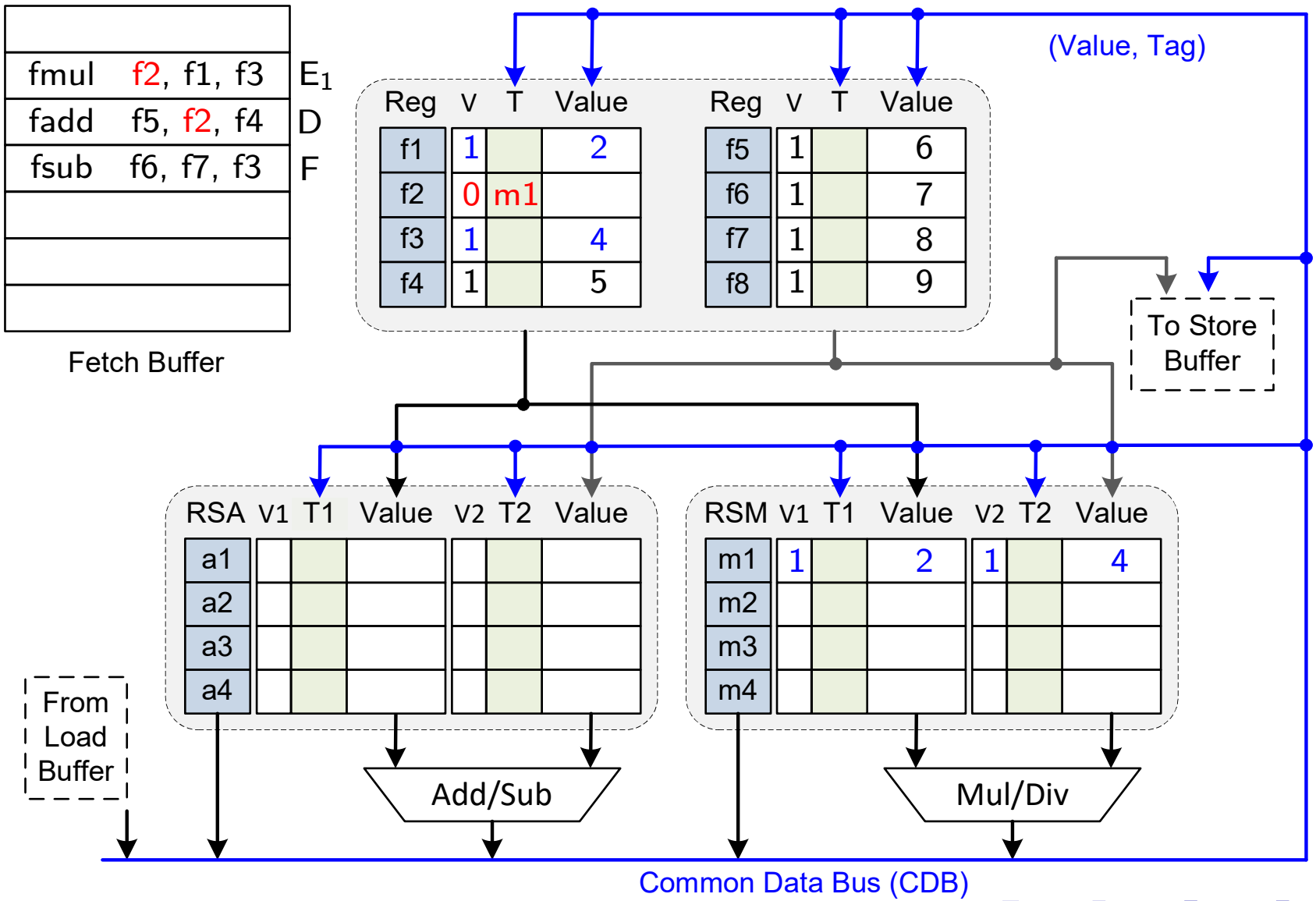
Out-of-Order Implementation: Cycle 1



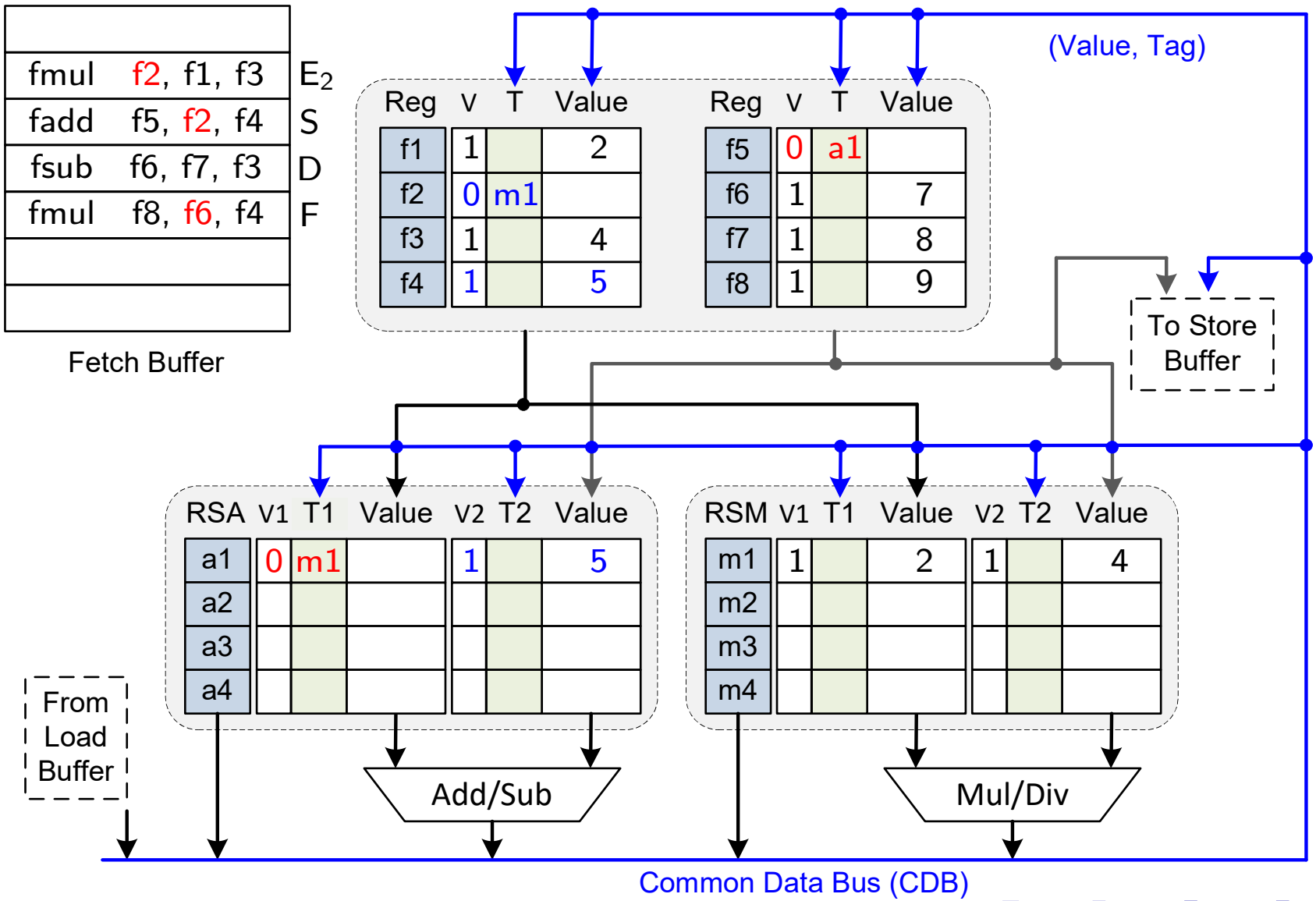
Out-of-Order Implementation: Cycle 2



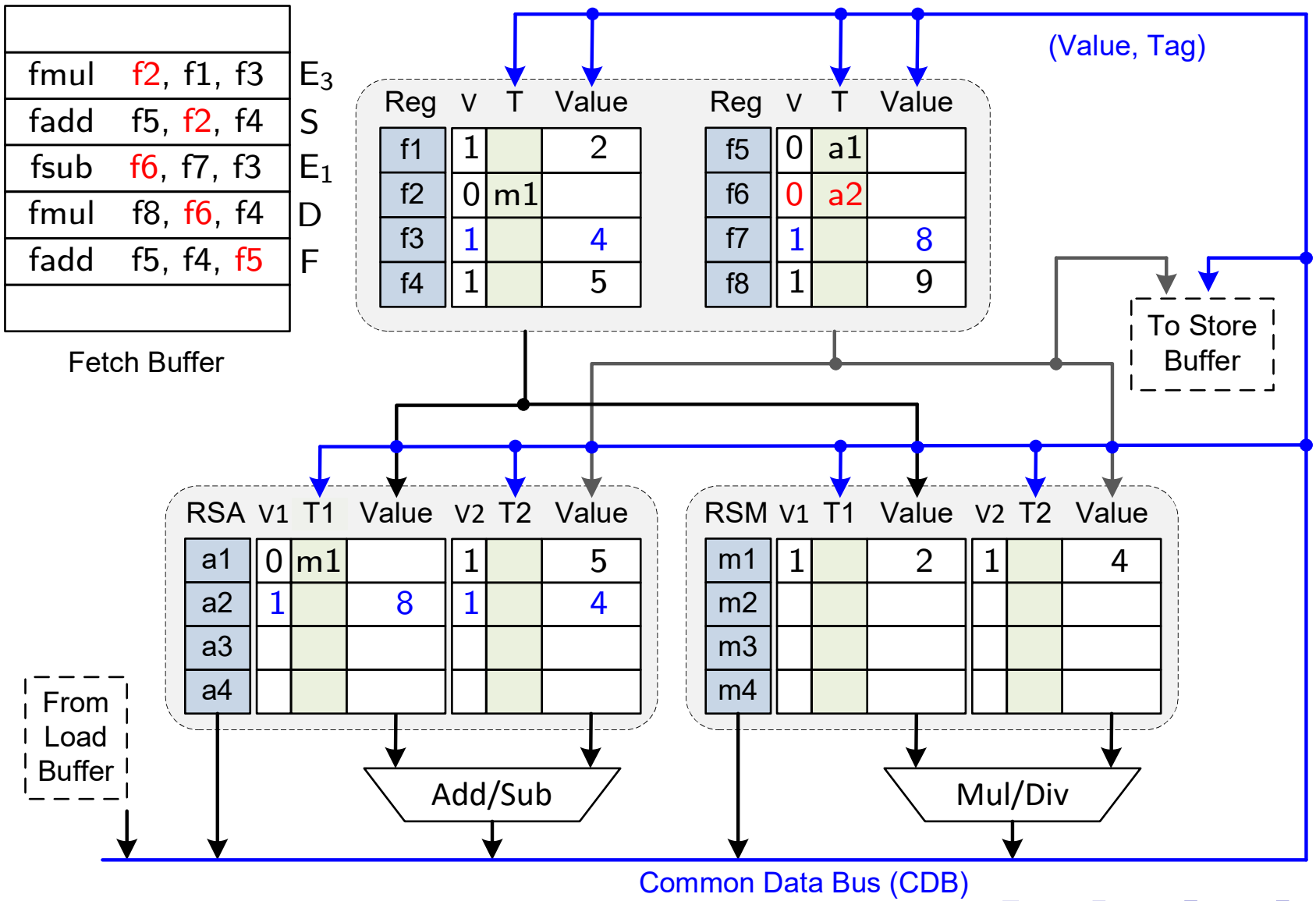
Out-of-Order Implementation: Cycle 3



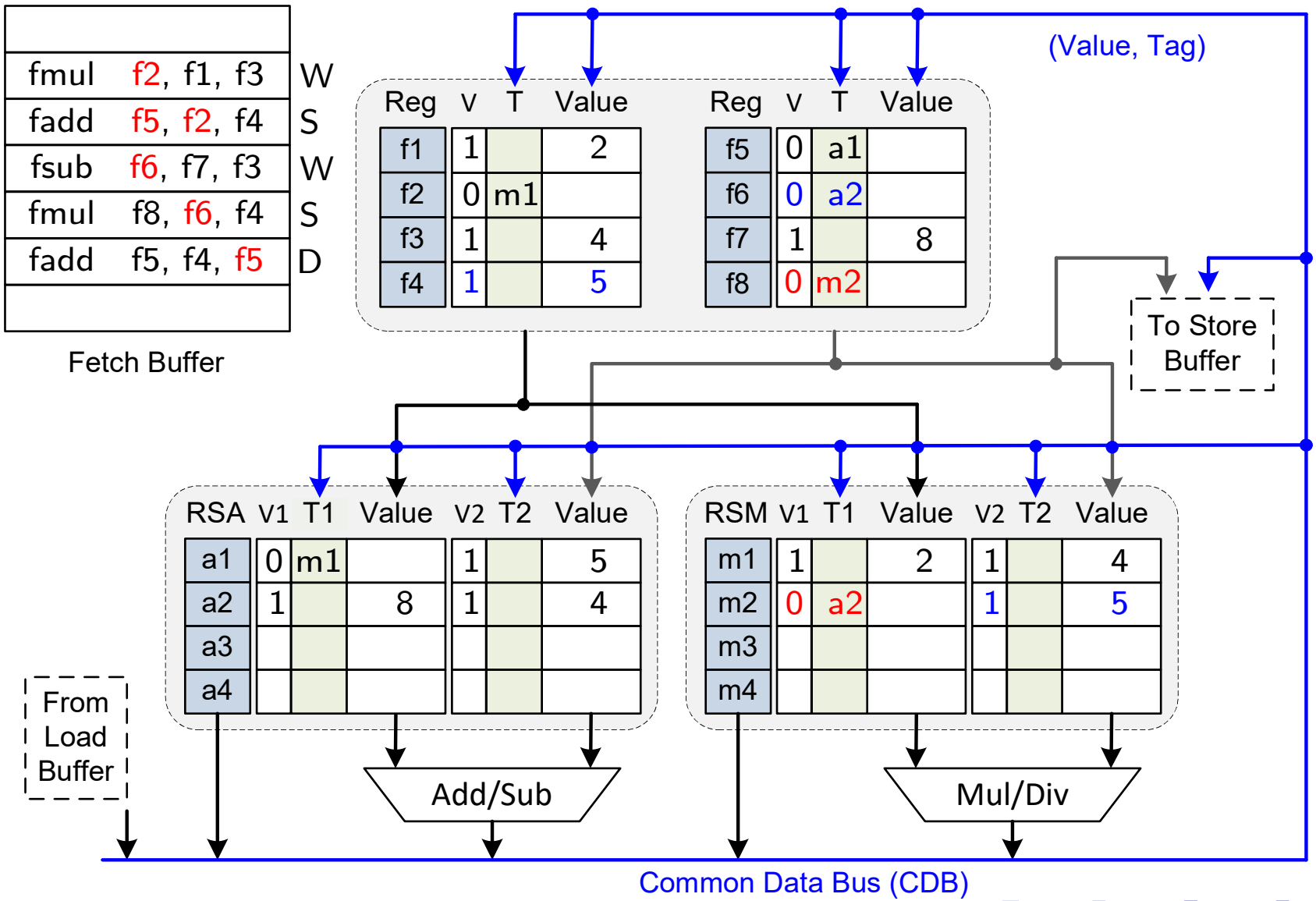
Out-of-Order Implementation: Cycle 4



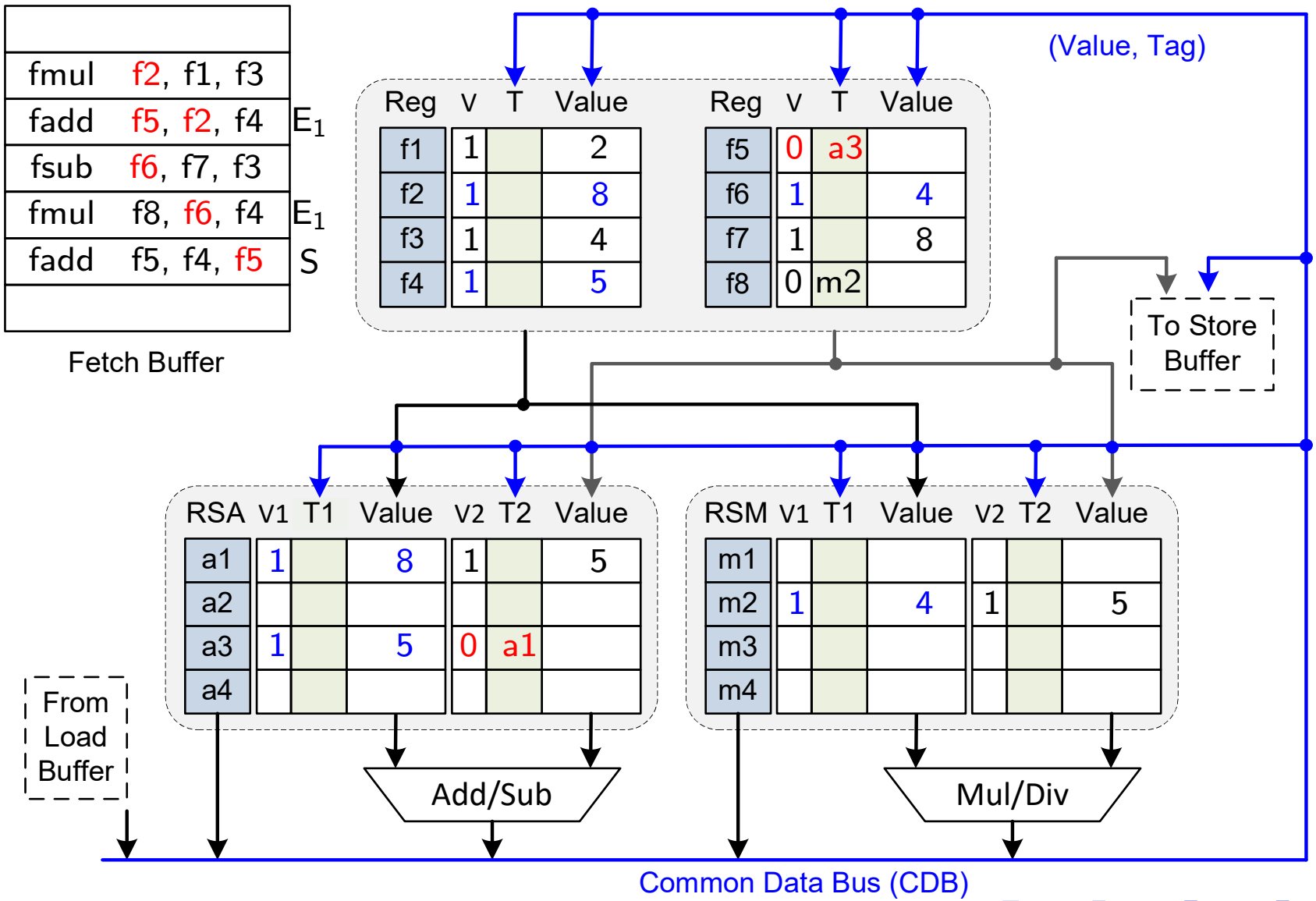
Out-of-Order Implementation: Cycle 5



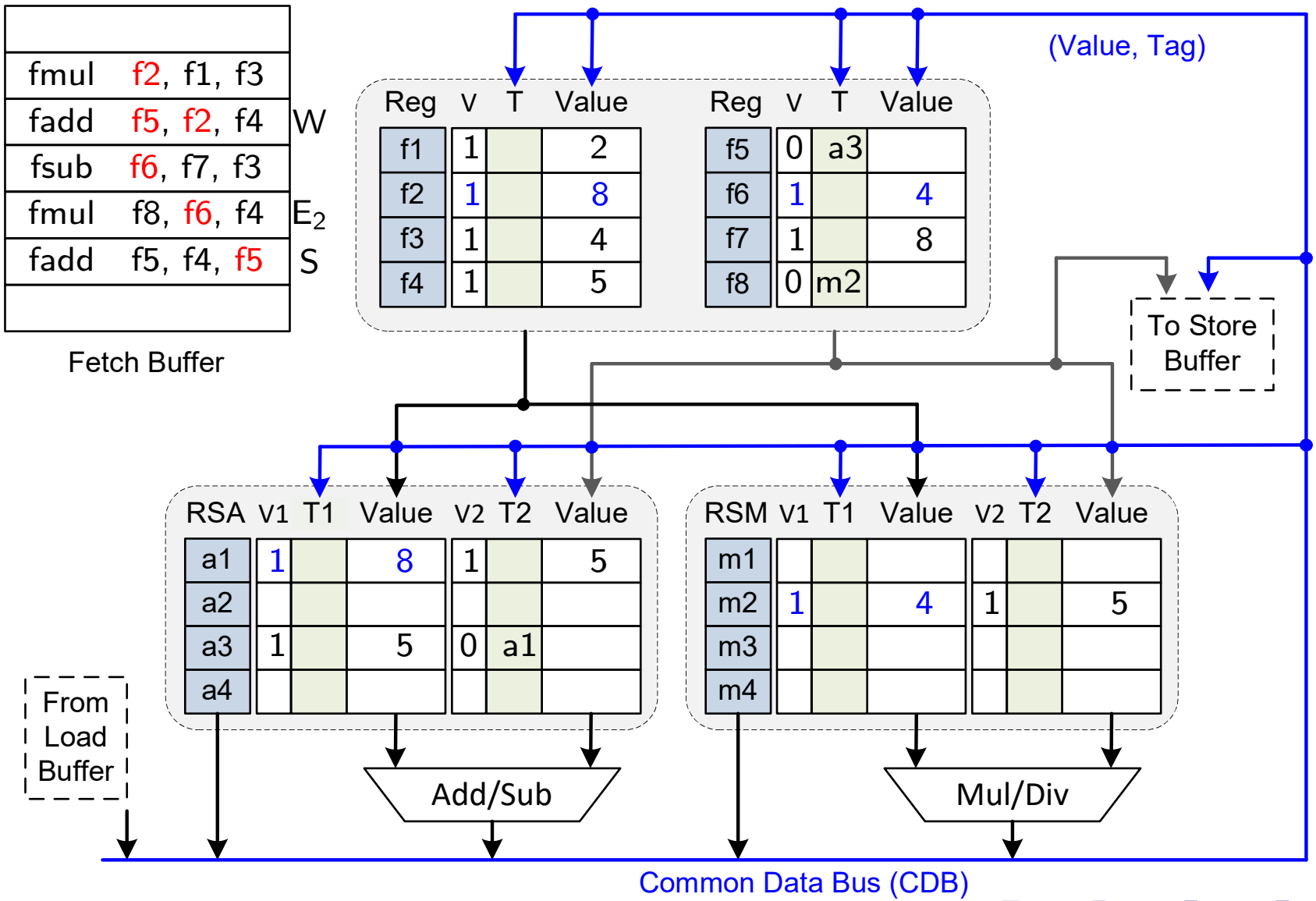
Out-of-Order Implementation: Cycle 6



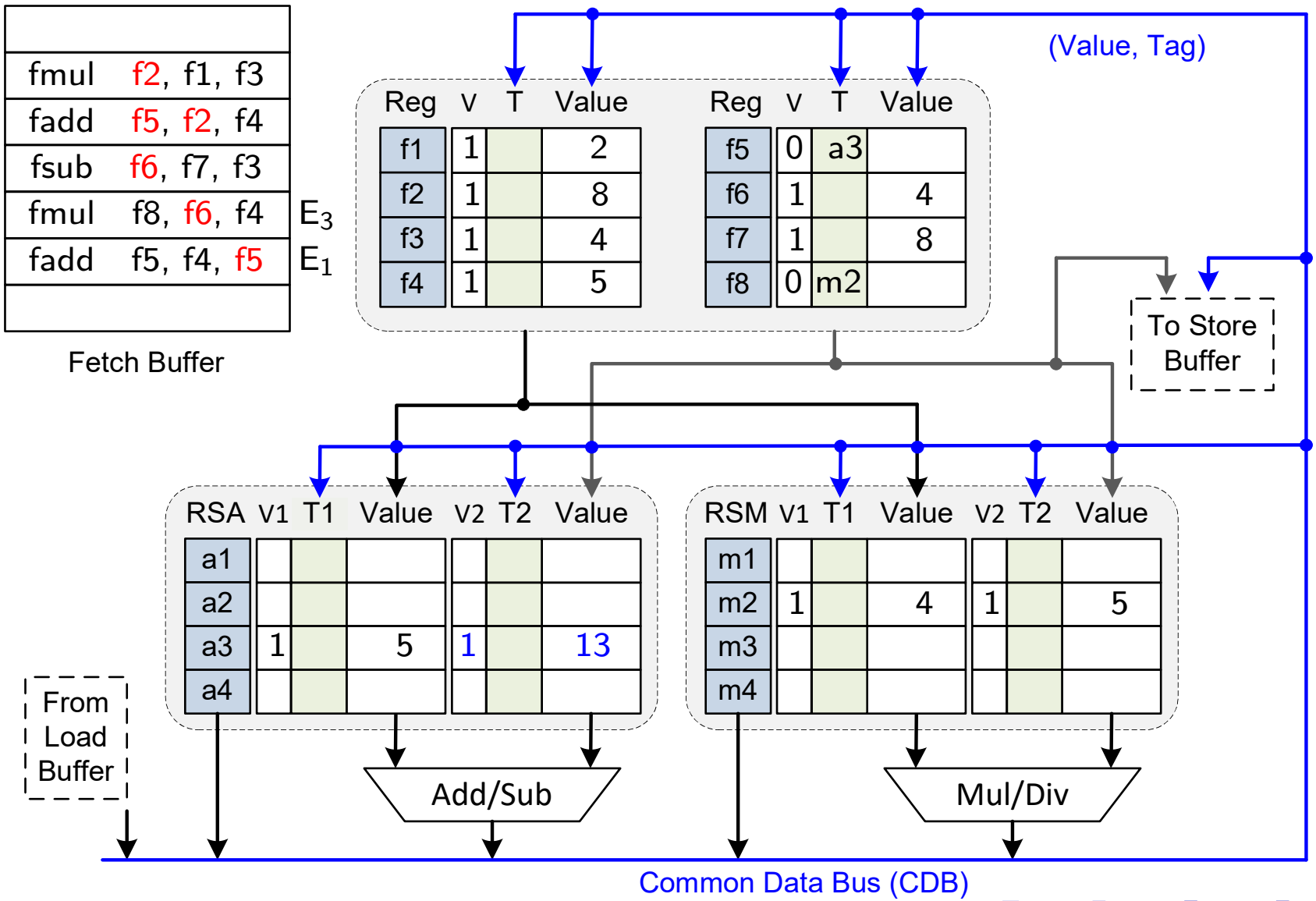
Out-of-Order Implementation: Cycle 7



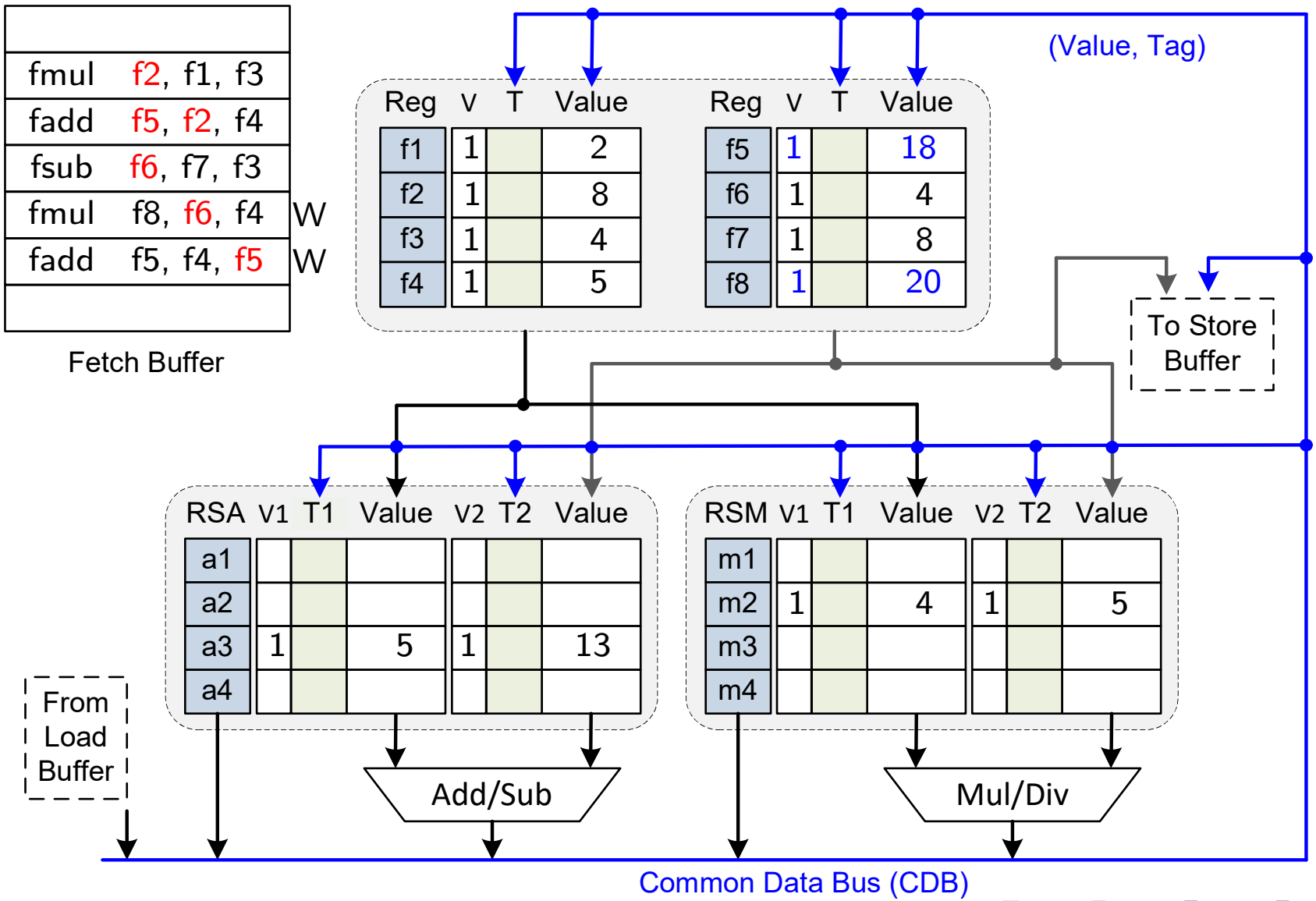
Out-of-Order Implementation: Cycle 8



Out-of-Order Implementation: Cycle 9



Out-of-Order Implementation: Cycle 10



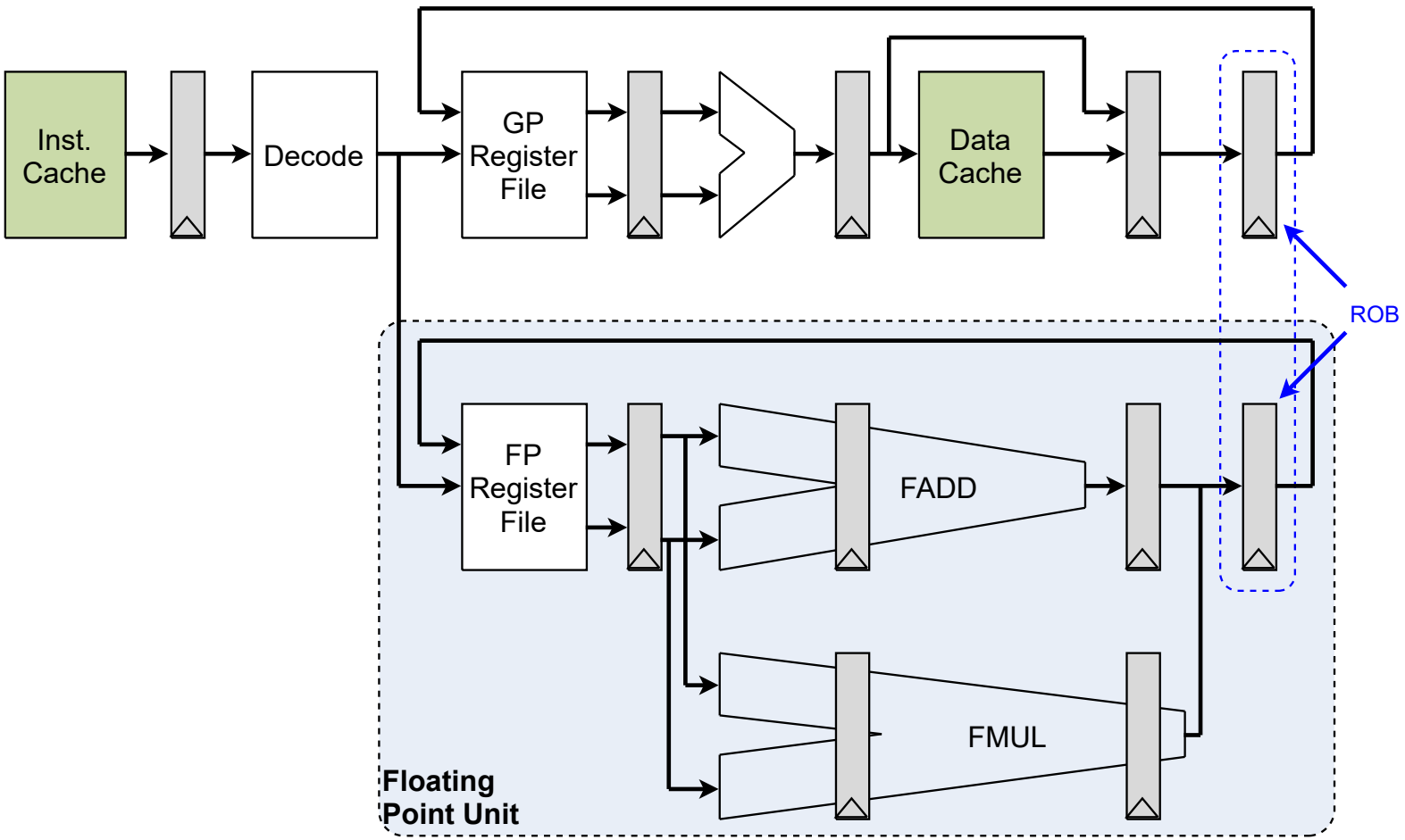
Reorder Buffer

Using reorder buffer (ROB) allows:

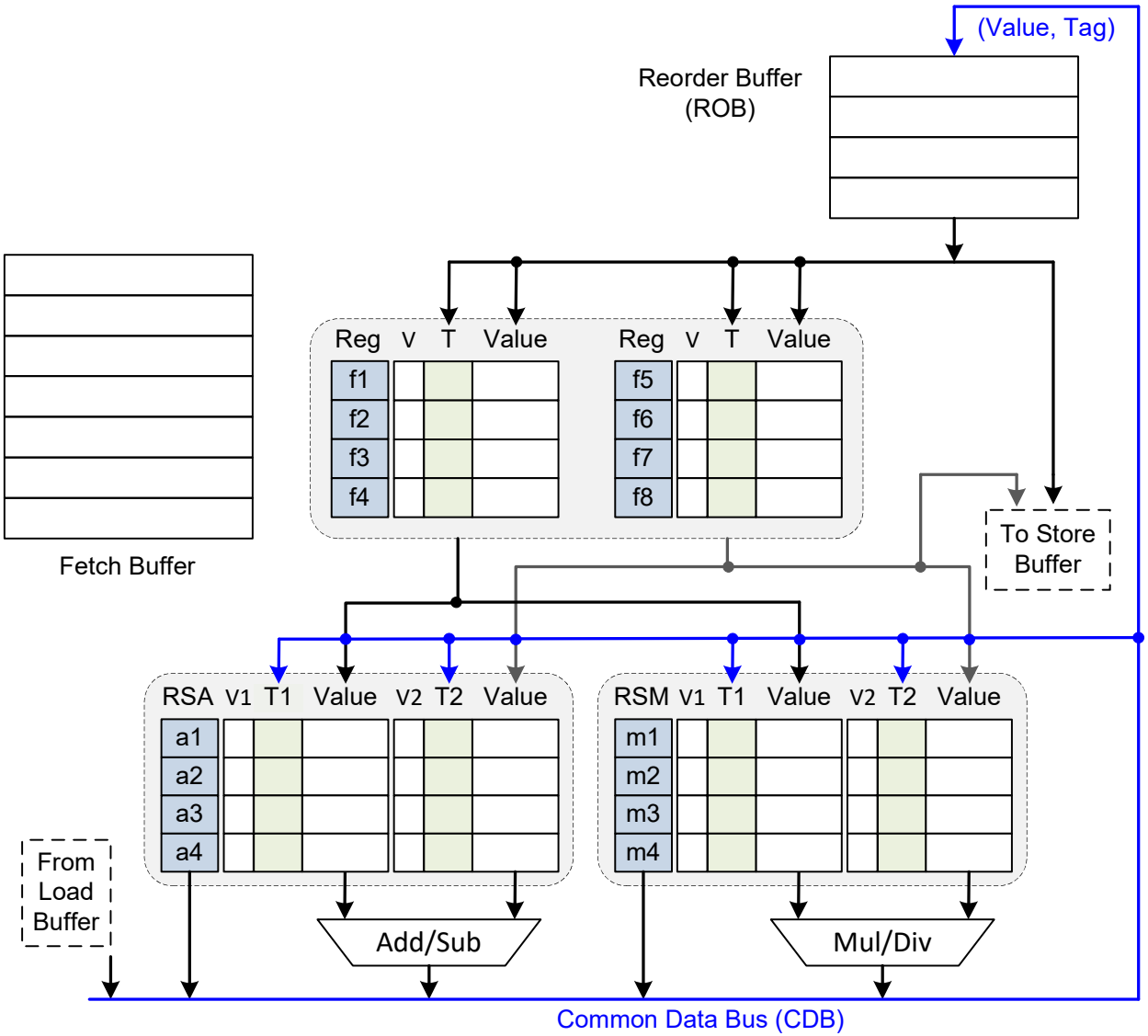
- In-order commit to maintain program order state
- Precise exception handling
- Speculative execution (branch prediction is implied)

Reorder Buffer: Out-of-order Completion

- Managing out-of-order completion using reorder buffer



Tomasulo's Algorithm with ROB



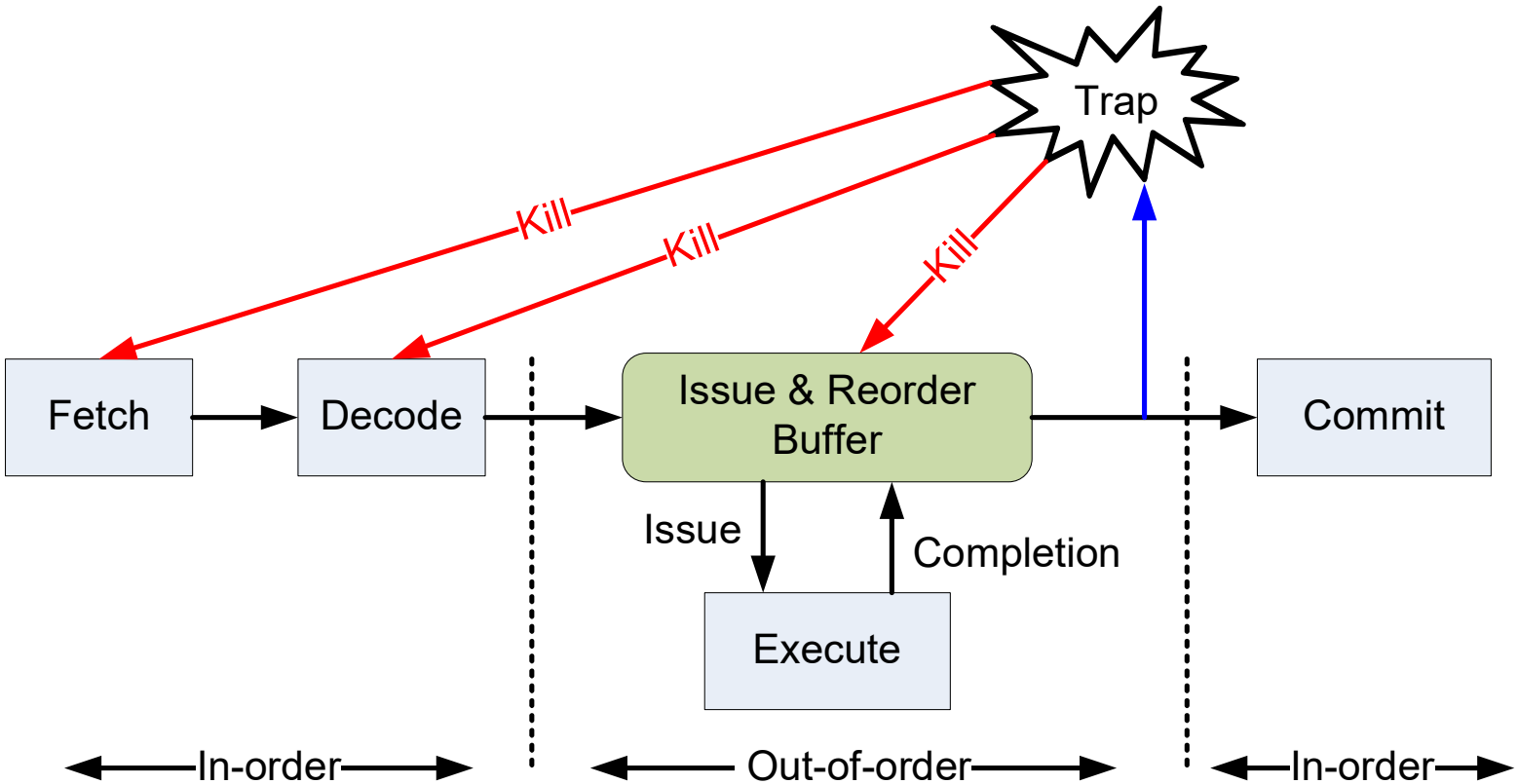
Reorder Buffer & Speculation

- Branch is issued with prediction marking in ROB
- Speculatively fetch, decode and issue instructions based on prediction
- Branch must resolve before leaving ROB
  - **Resolved** as predicted: Commit the following instructions after branch
  - **Not Resolved** as predicted: Mark the following instructions in ROB as invalid

Handling Exceptions and Interrupts

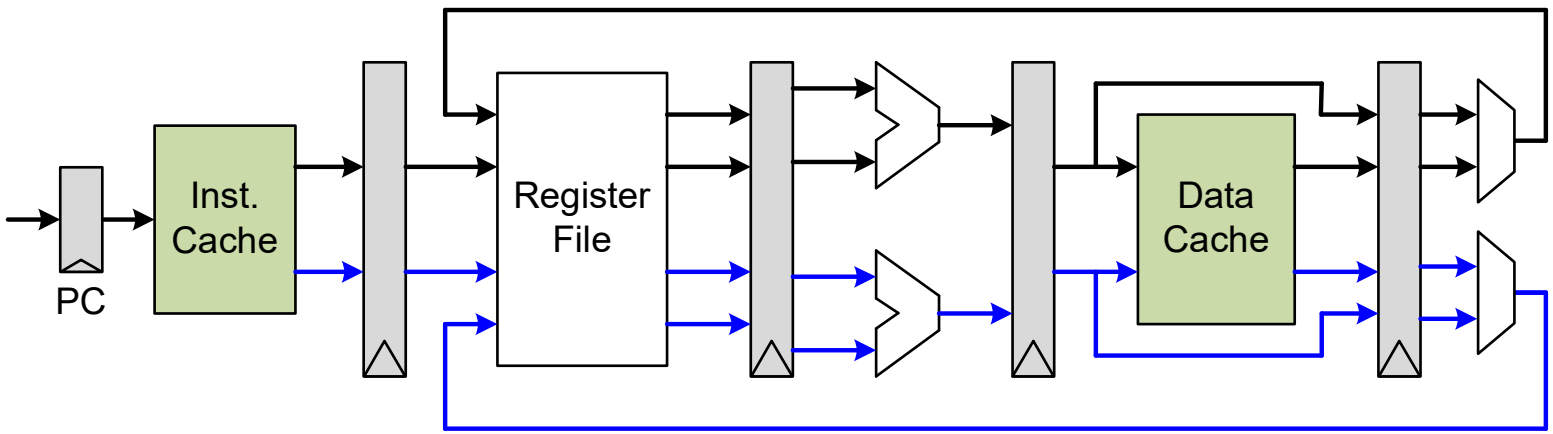
- The exceptions and interrupts are handled at commit stage
- Out-of-order commit leads to **imprecise traps**
- In-order commit leads to **precise traps**
- In-order commit requires **Reorder Buffer** or **Queue**

Handling Exceptions and Interrupts Cont'd

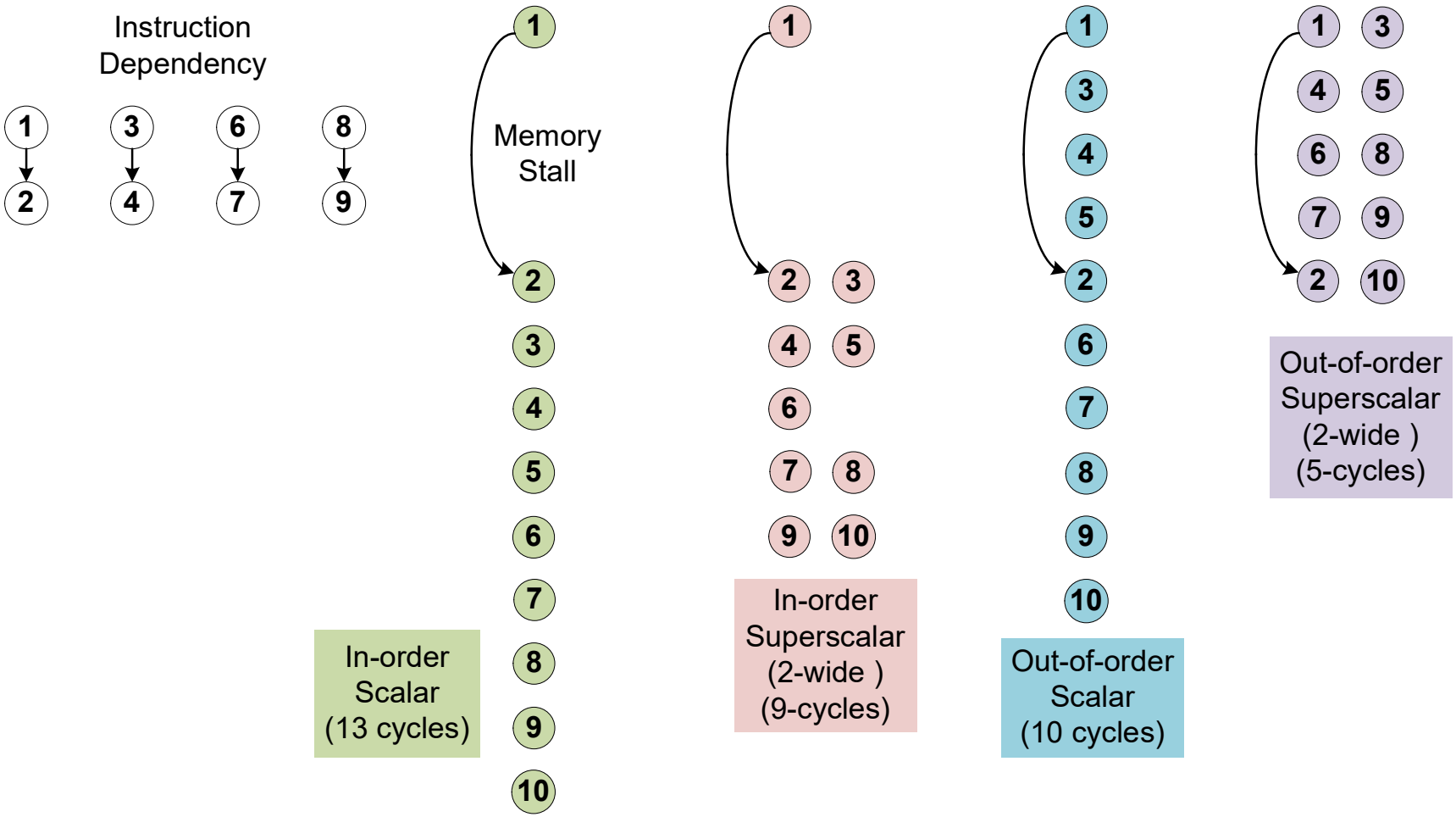


The Superscalar: Implementation

- In-order, 2-wide pipeline



The Superscalar IS NOT OOO



VLIW Processor

- **VLIW processor**: Statically scheduled by the compiler with multiple operations packed into *very large instruction word* (VLIW)
- **VLIW compiler**:
  - Schedule operations to maximize parallelism (loop unrolling, trace scheduling etc.)
  - Avoid data hazards
- **Example VLIW processors**: TMS320C6x DSPs (embedded DSP with partial success), Intel Itanium (discontinued)
- **Performance limitation**: Limitations of static scheduling

Suggested Reading

- Read relevant sections of Chapter 3 of [Patterson and Hennessy, 2019].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Patterson, D. and Hennessy, J. (6th Edition, 2019).
Computer Architecture: A Quantitative Approach.
Morgan Kaufmann.