Re-Order Buffer

Introduction

1. Exceptions, like divide by zero, require that programs be executed in program order
   - This lesson explains how to recover in the event of an exception

Exceptions in Out of Order Execution

1. Consider the following program:
   - F6 == 0, so a divide by zero exception occurs in cycle 40
   - Can also happen if a page fault occurs
   - This causes issues when subsequent instructions used different values from what would have been used had the exception handler run first
2. Lack of exception handling was considered to be the major drawback of Tomasulo's algorithm as it was implemented

Number	Instruction	IS	DIS	WR
1	DIV F10,F0,F6	1	2	42
2	LD F2,45(R3)	2	3	13
3	MUL F0,F2,F4	3	14	19
4	SUB F8,F2,F6	4	14	15

Branch Misprediction

1. If the branch predictor mis-predicts, we need to undo the effects of the subsequent instructions that may have processed out of order
   - DIV R1, R3, R4
   - BEQ R1, R2, Label
   - ADD R3, R4, R5 <- This should never have completed if branch taken
2. Phantom exceptions: An exception that occurs in a branch that should never have been taken

Correct Out of Order Execution

1. Execute out of order
2. Broadcast out of order
3. Deposit values to registers in order
   - This is needed because if we deposit register values out of order but an earlier instruction shouldn't have been executed (exception, branch misprediction) will lead to incorrect state
4. Reorder Buffer: Remembers program order
   - Keep result until safe to write

Re-Order Buffer Part 1

1. Table of entries with at least three fields
   - Value: Value produced by instruction
     - Instruction writes value here at broadcast instead of the register
   - Valid bit: Designates if this entry contains actual data or not
   - Register: Designates which register the value should be written to
2. Two pointers into data structure
   - Issue: Points to the entry for the next instruction
   - Commit: Points to the next entry to be written to register in program order
   - Entries between the issue and commit pointers are in program order

Re-Order Buffer Part 2

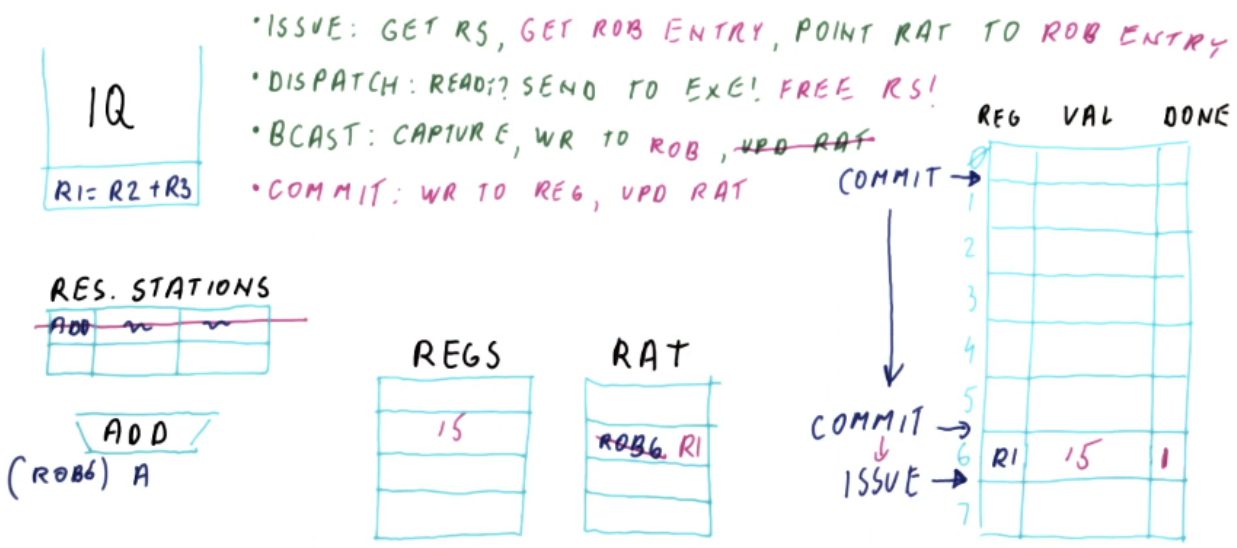
1. Issue:
   - Get RS
   - Get ROB entry
     - ROB entry must be the next one to maintain program order
     - Increments the issue pointer
   - Point RAT to ROB entry
     - Previously would point to the RS
2. Capturing results is identical to Tomasulo's algorithm
3. Dispatch is very similar to Tomasulo's algorithm
   - Except, as soon as the instruction is ready, it can be sent to the execution unit
     - The reservation station can be freed at this time since nothing is pointing at the RS anymore (ROB entry instead)

Free Reservation Stations Quiz

1. Instruction cannot issue because there is no available RS for it. When is this more likely to occur? Assume 2 ADD/2 MUL RSs
   - No ROB (Tomasulo)
   - With ROB
2. Lacking available reservation stations is more likely in Tomasulo's algorithm since RSs are immediately freed with a ROB

Re-Order Buffer Part 3

1. Broadcast:
   - Almost exactly as designed by Tomasulo, but the tag we carry for the result is the ROB entry
   - Tag and value are given to reservation stations and captured if needed
   - However, with the ROB, we don't write to registers immediately
   - Don't update the RAT since it's already pointing to the ROB
2. Commit:
   - Additional step compared to Tomasulo's algorithm
   - Update the register file
   - Update the RAT to point to the register file instead of the ROB
   - Update the commit pointer in the ROB



ROB Basic Example

Hardware Organization with ROB

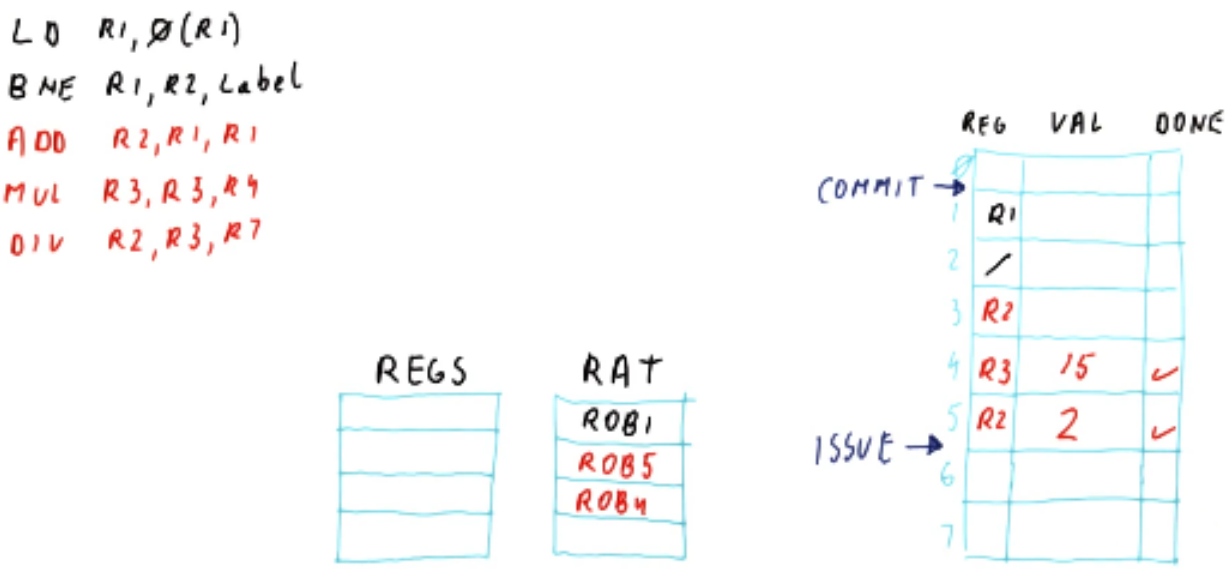
1. Still have all of the existing hardware:
   - Instruction Queue (IQ)
   - Reservation Station (RS)
   - Register Aliasing Table (RAT)
   - Register File (RF)
2. New hardware:
   - Reorder Buffer (ROB)
     - Head and tail pointers
     - RAT entries point here instead of the RF

ROB Quiz

1. We need the ROB to:
   - Remember the program order (true)
   - Temporarily store the instruction's result (true)
   - Serve as name (tag) for the result (true)
   - Store source operands until dispatch (false)
   - Determine which instruction goes to which execution unit (false)

Branch Misprediction and Recovery Part 1

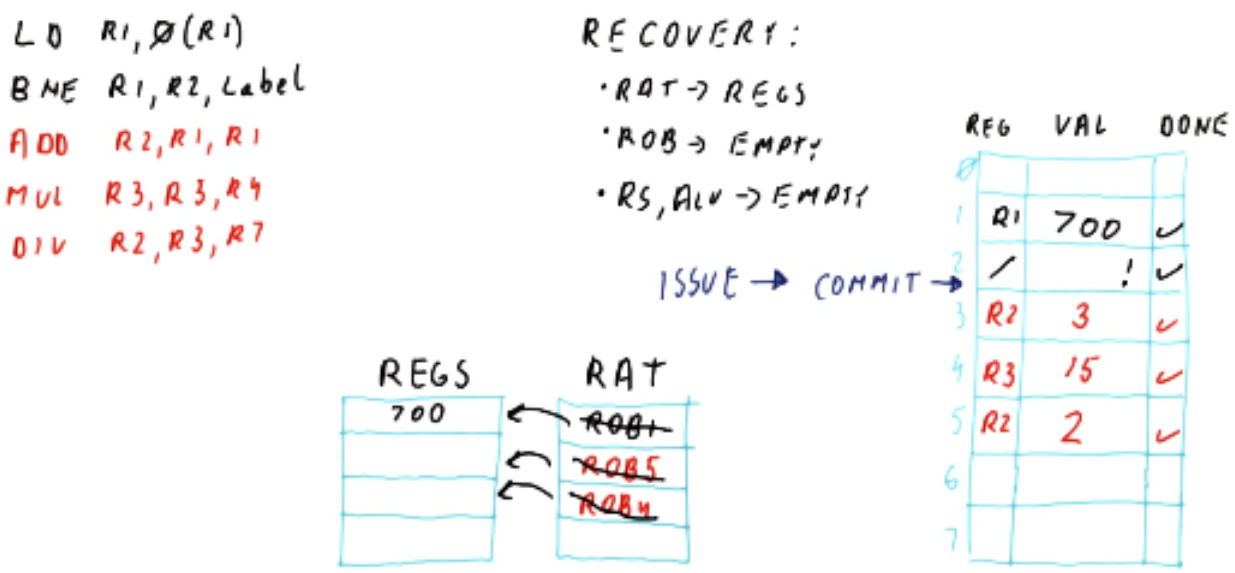
1. Suppose the load is slow in the following program, so the subsequent instructions finish
   - The branch was predicted not taken, but was actually taken
   - In a ROB processor, none of the results are written back to the registers, only to the ROB
2. This allows us to revert easily as no state has been updated



Branch Misprediction Example 1

Branch Misprediction and Recovery Part 2

1. When an instruction finishes and is the next instruction, we can commit its result immediately
2. When a branch misprediction occurs, we mark its entry in the ROB accordingly
   - We continue fetching and executing other instructions until the commit pointer reaches that instruction
3. When a branch misprediction occurs, we complete the recovery before fetching the next instruction
   - Move the issue pointer to be the same as the commit pointer
   - Rewrite whatever is in the RAT to point to the register file
     - We guarantee that the register file is always correct because it isn't updated until the commit occurs, which is in order
     - This makes recovery simple
   - Clear out whatever is in the ALU and reservation stations



Branch Misprediction Example 2

ROB and Exceptions

1. Exceptions are handled similarly to branch mispredictions
   - When an exception occurs, its entry in the ROB is set to "exception" instead of the actual value
2. Phantom exceptions are trivial because the exception handler isn't actually called until the instruction commits
   - This means if a branch misprediction was the only reason this exception occurred, it will never be committed
3. "Treat the exception as any other result"
   - Delay exception handling until the instruction commits

Outside View of Executed

1. From the programmer's point of view, everything appears to be in order
   - Programmer can only observe the commits
   - All out-of-order execution is internal state to the processor

Exceptions with ROB Quiz

1. Consider the following program:

```
ADD R2, R2, R1 - Committed
LW R1, 0(R1) - Executing
ADD R3, R4, R5 - Done
DIV R3, R2, R3 - Executing
ADD R1, R4, R4 - Done
ADD R3, R2, R2 - Done
```

2. Suppose the divide triggers an exception. What is the status of each instruction when the exception handler is called?

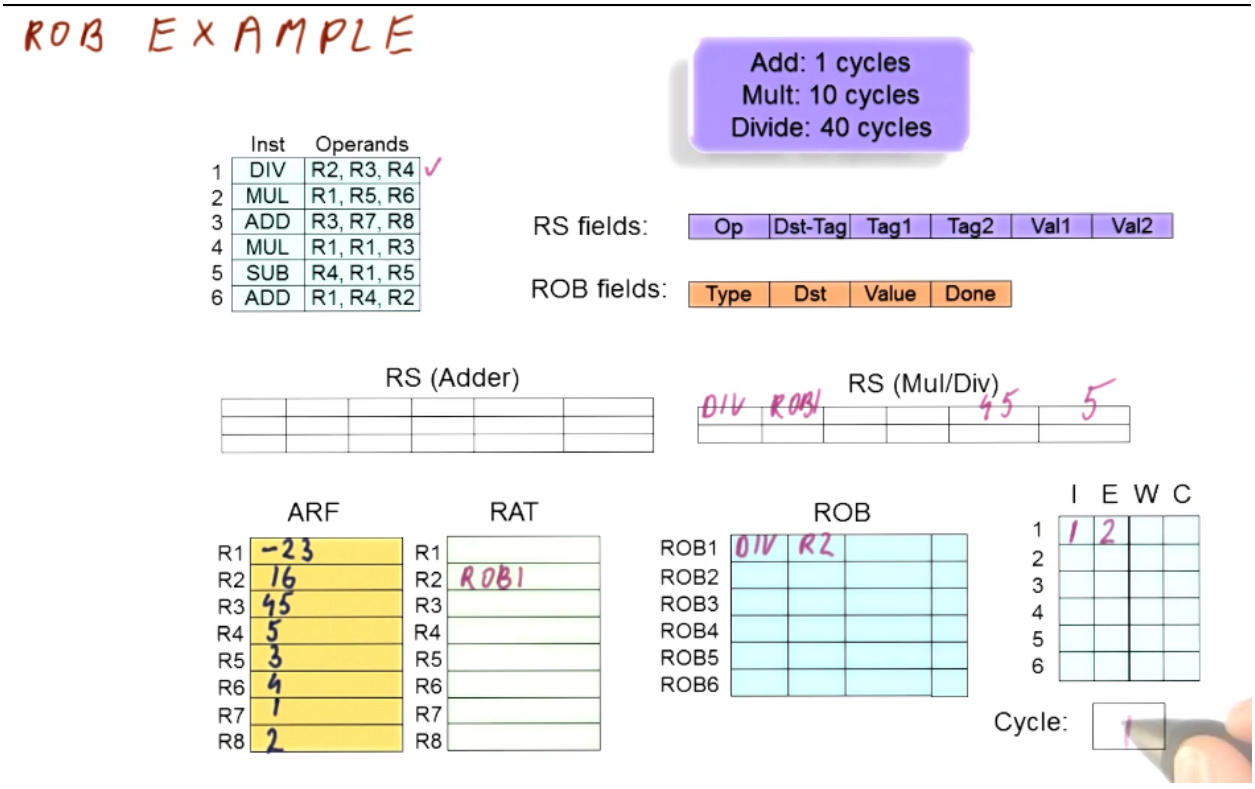
```
ADD R2, R2, R1 - Committed
LW R1, 0(R1) - Committed
```

```
ADD R3, R4, R5 - Committed
DIV R3, R2, R3 - Unexecuted
ADD R1, R4, R4 - Unexecuted
ADD R3, R2, R2 - Unexecuted
```

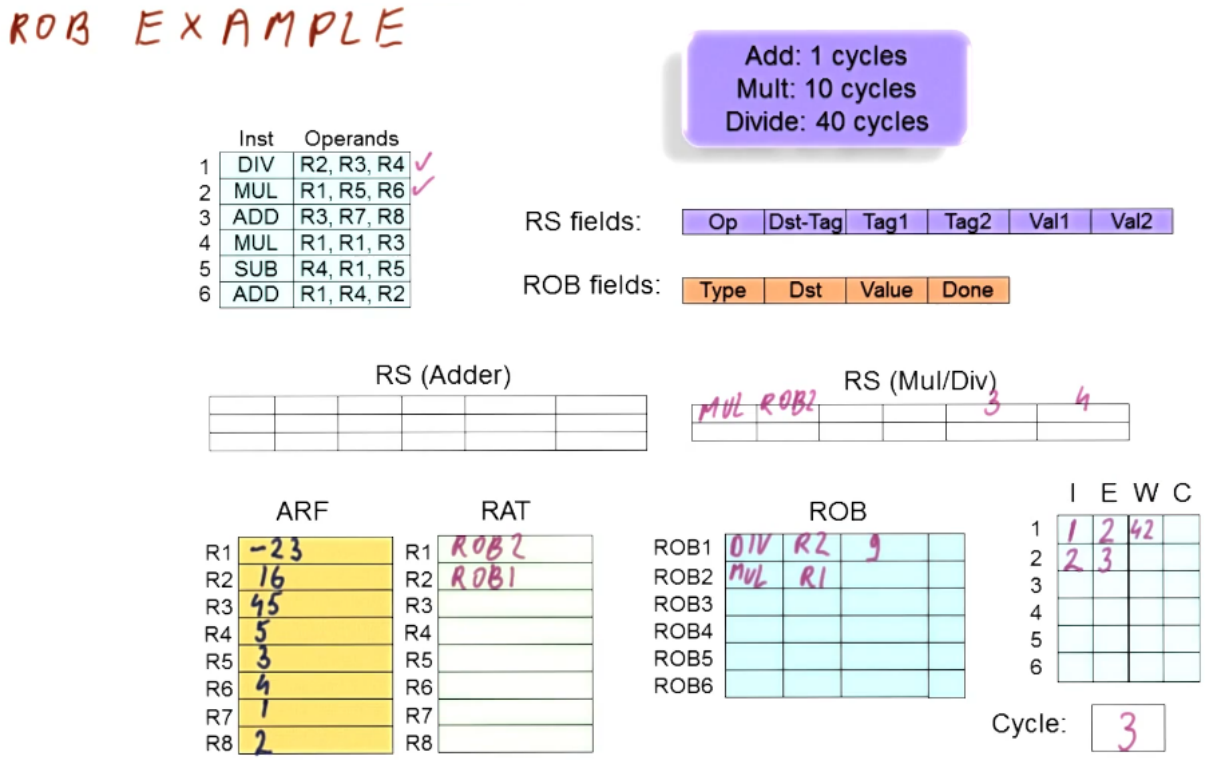
RAT Updates on Commit

1. Without a ROB, when an instruction finished, it checked the RAT to determine if it was the latest instruction to write to that register. If so, it updated the register file
2. With an ROB, this must be delayed until the commit phase
   - With a ROB, even if the instruction isn't the latest to update a register, its result must be written to the register file
     - This is necessary for the processor state to be correct
     - This also allows us to easily handle exceptions (just clear RAT/ROB)
   - If the RAT entry also points to the ROB entry, it must be updated to point to the register file again
   - After an instruction is freed, its ROB entry is cleared to allow for the next instruction
3. Be mindful of the fact that values are copied to registers regardless of what's in the RAT
   - However, the RAT should only be updated if it's pointing to this ROB entry

ROB Example Cycles 1-2

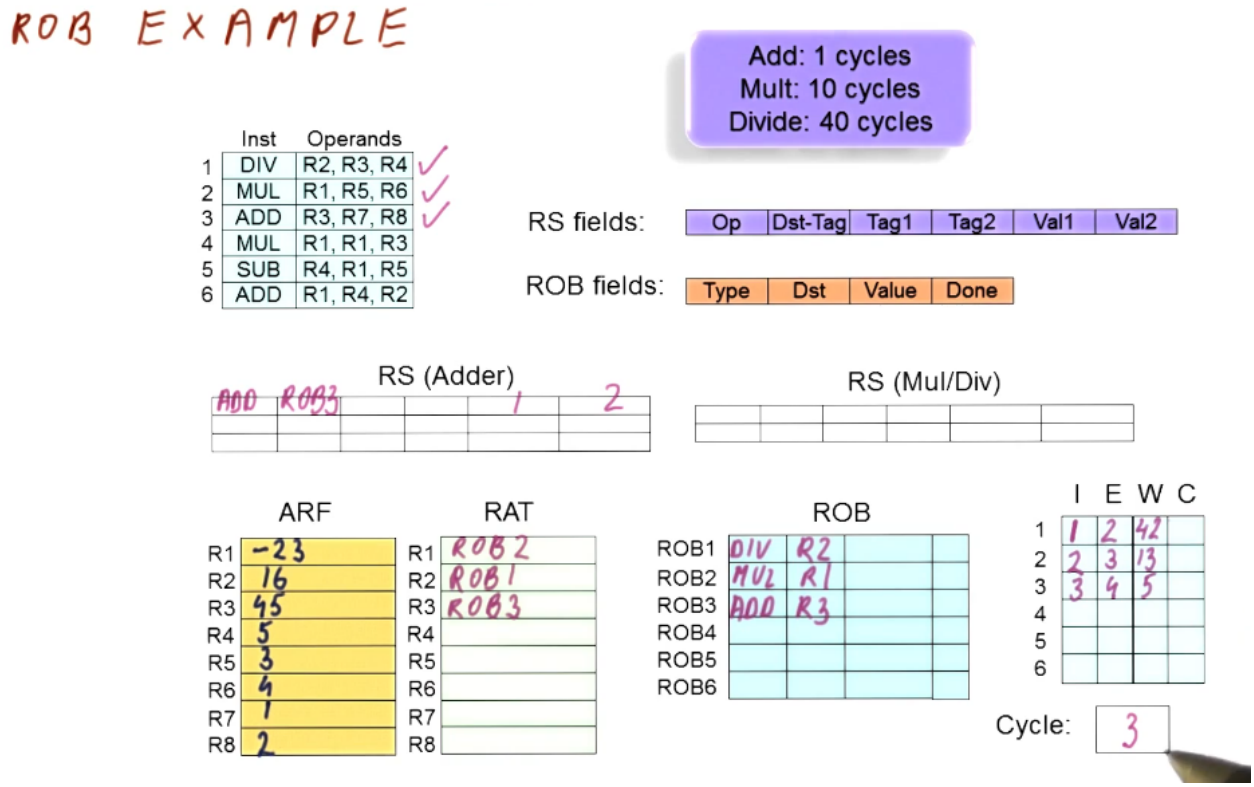


ROB Example Cycle 1

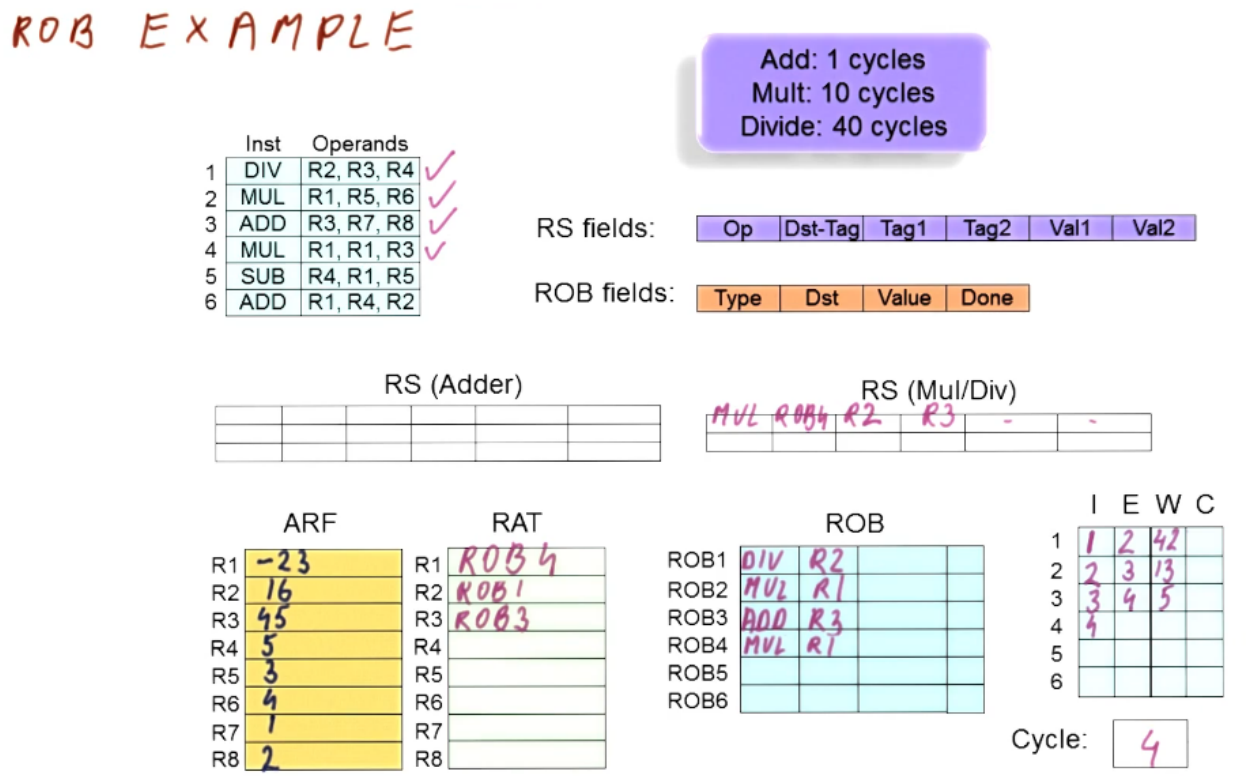


ROB Example Cycle 2

ROB Example Cycles 3-4



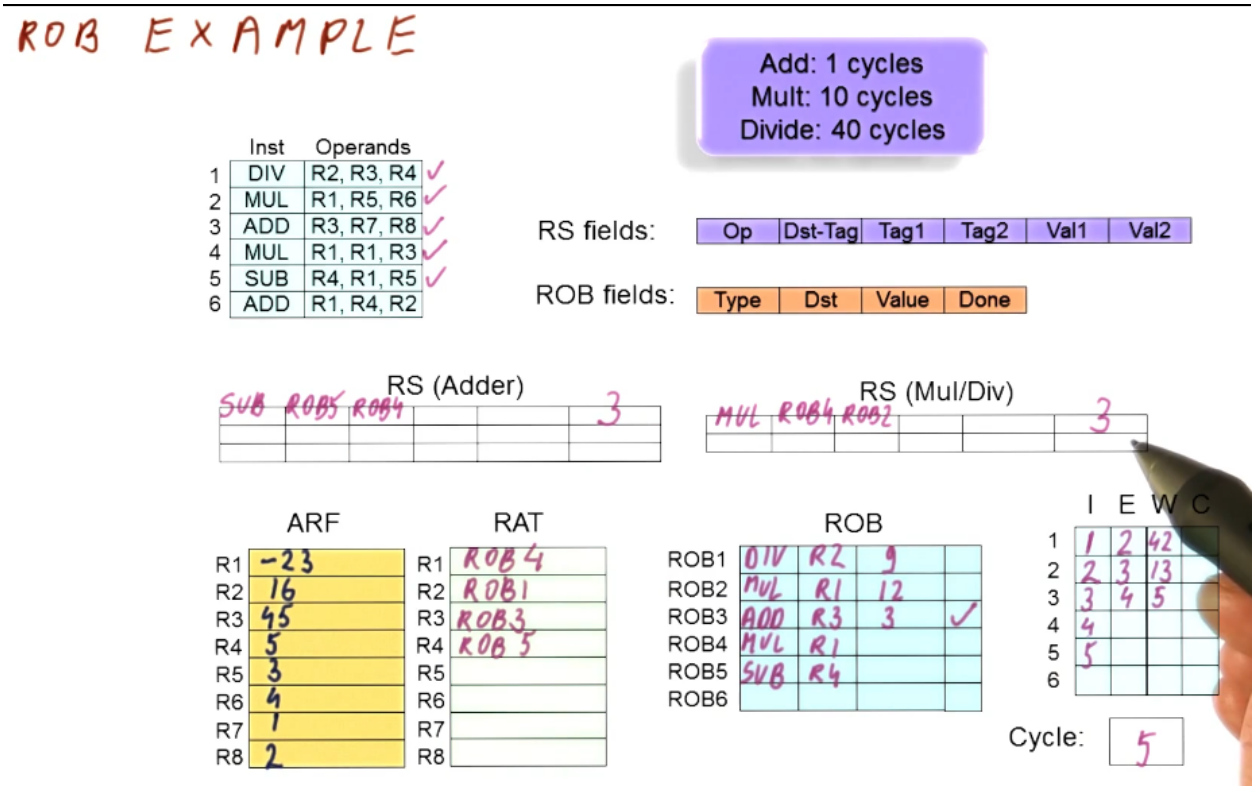
ROB Example Cycle 3



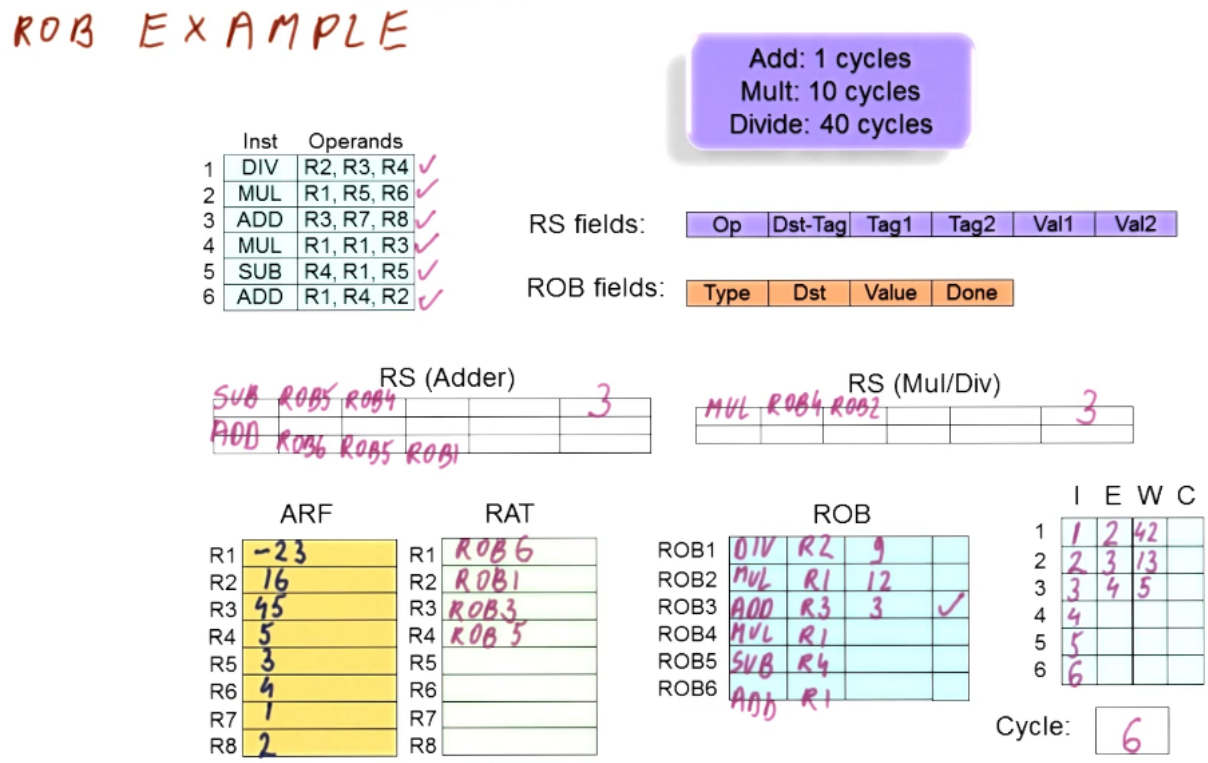
ROB Example Cycle 4

1. Make sure to first get the input registers, then update the RAT
   - If an instruction is overwriting an input register, it would be waiting for its result forever if we updated the RAT first

ROB Example Cycles 5-6



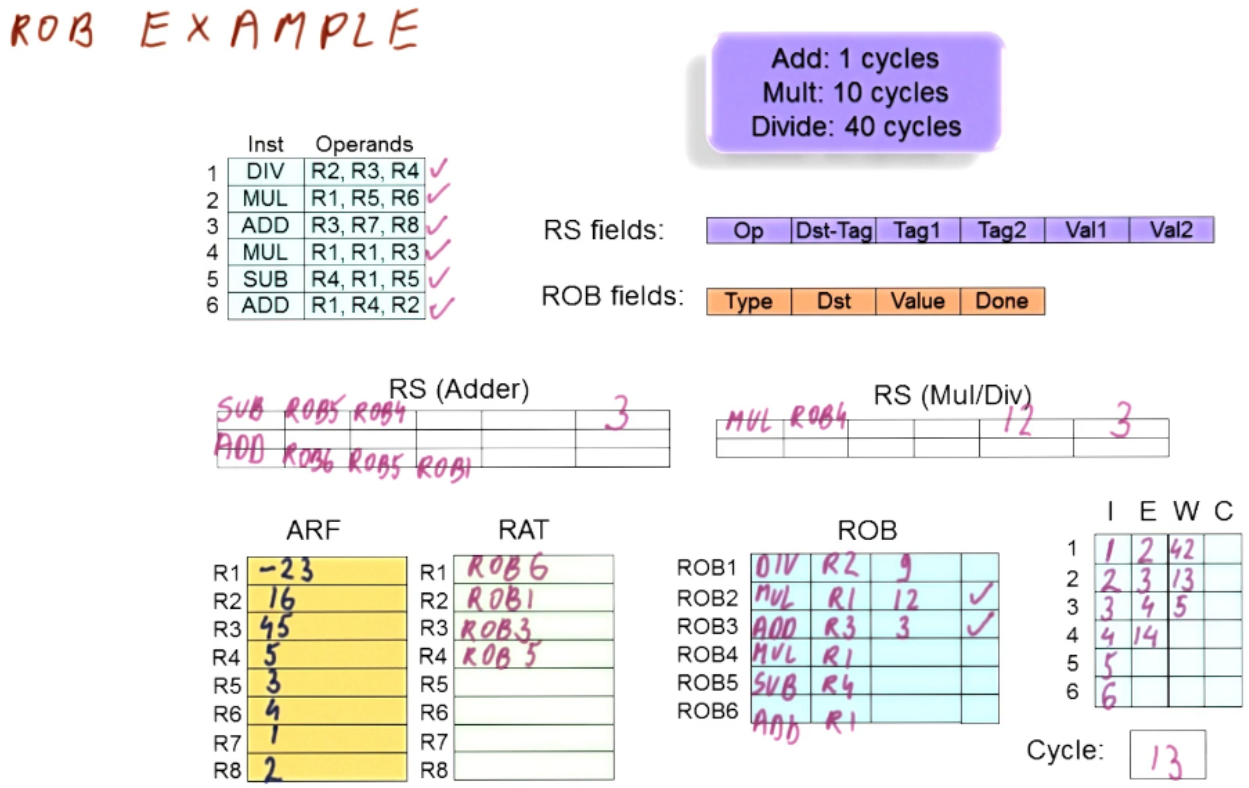
ROB Example Cycle 5



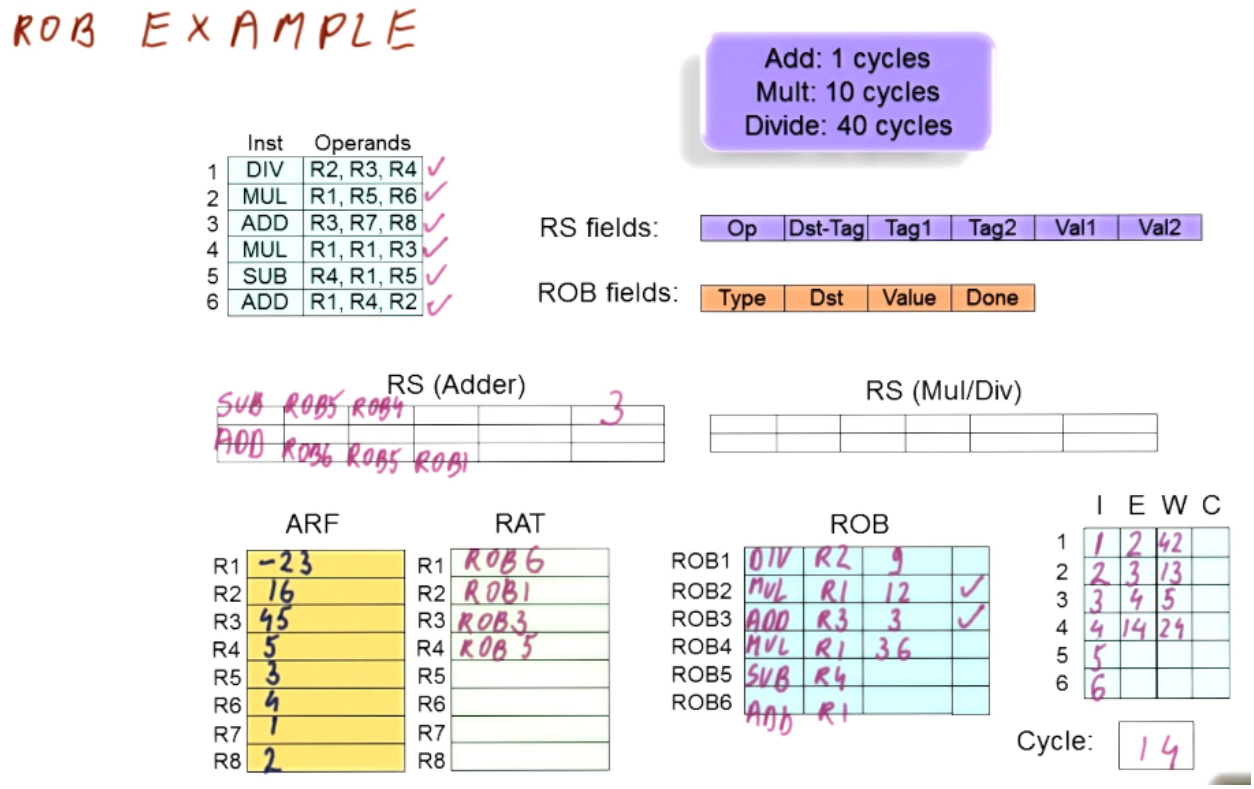
ROB Example Cycle 6

1. Instruction 3 is done, but cannot be committed until all previous instructions have been committed
2. After cycle 6, nothing interesting can happen until cycle 13 since all instructions are waiting on results in registers

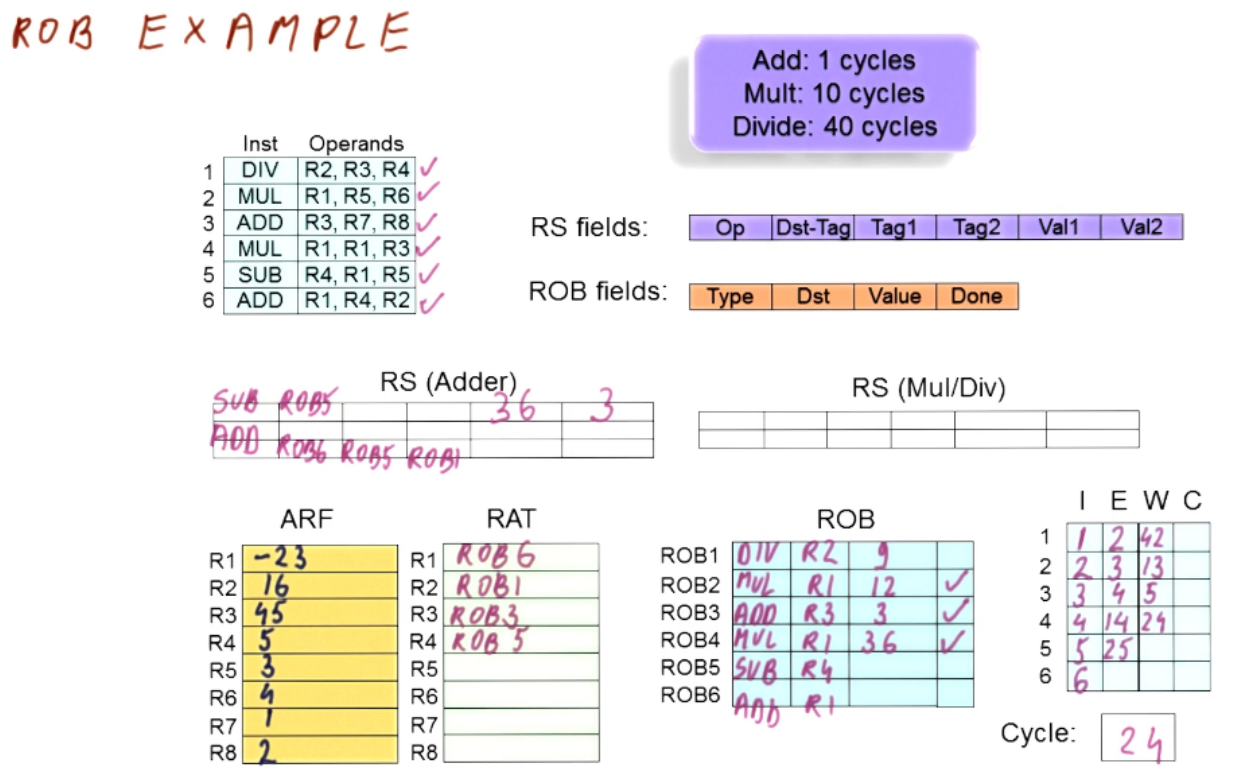
ROB Example Cycles 13-24



ROB Example Cycle 13

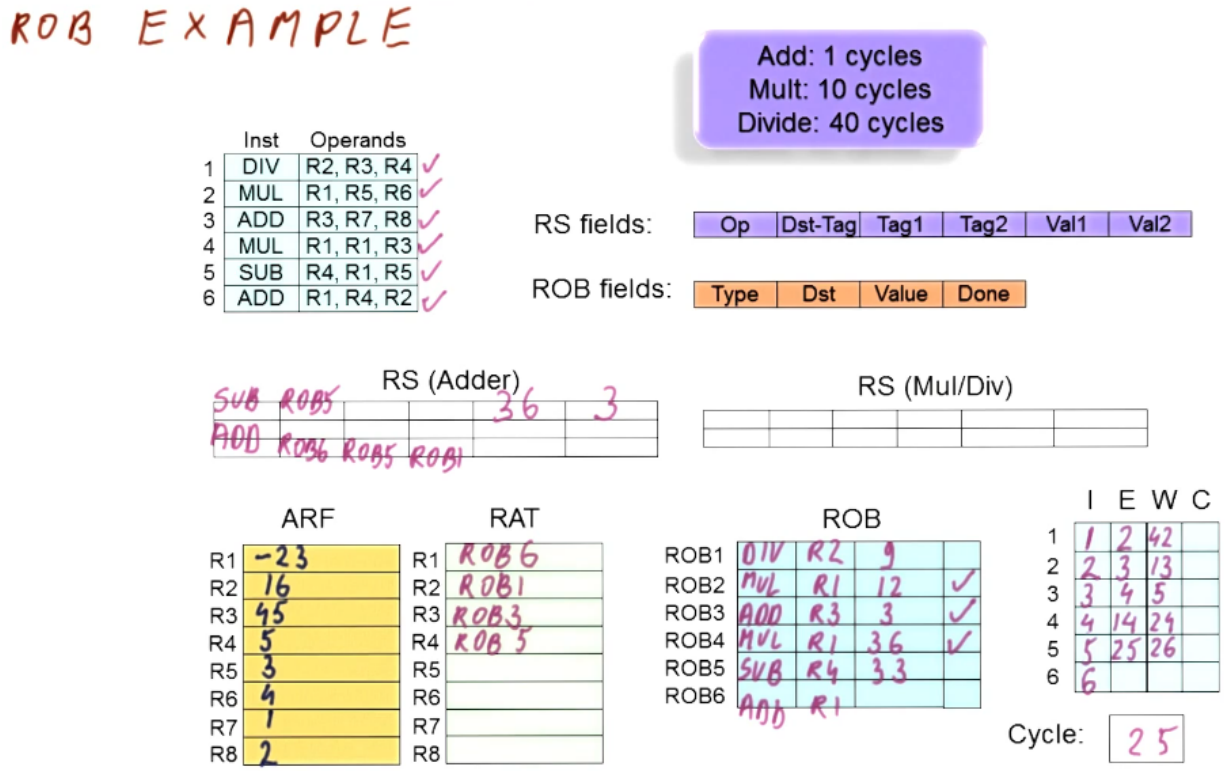


ROB Example Cycle 14

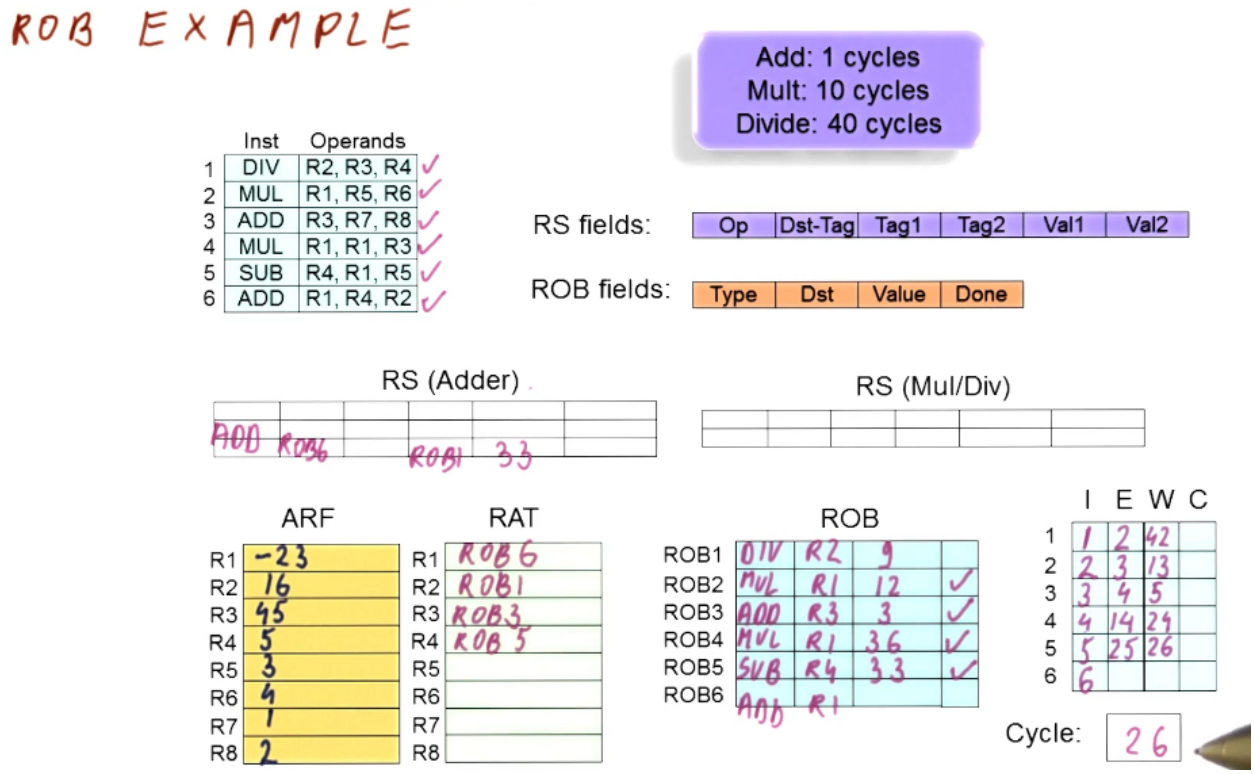


ROB Example Cycle 24

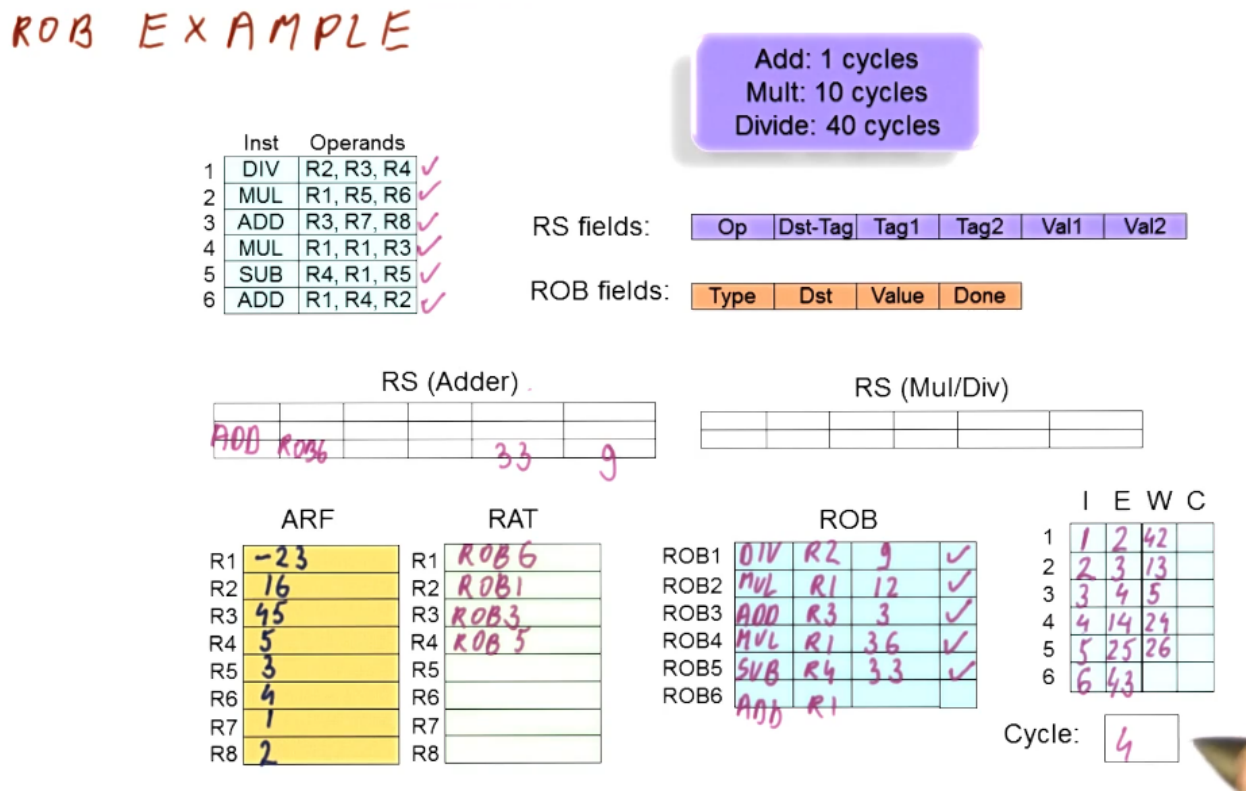
ROB Example Cycles 25-43



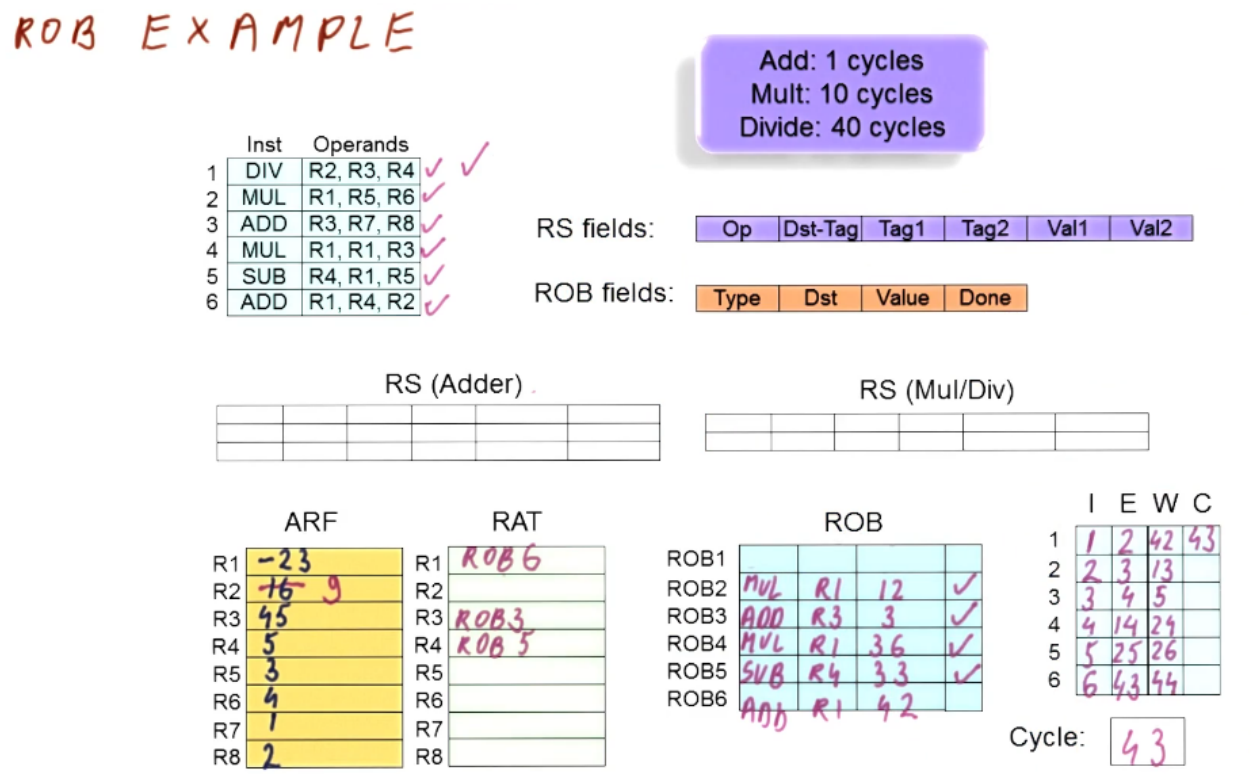
ROB Example Cycle 25



ROB Example Cycle 26



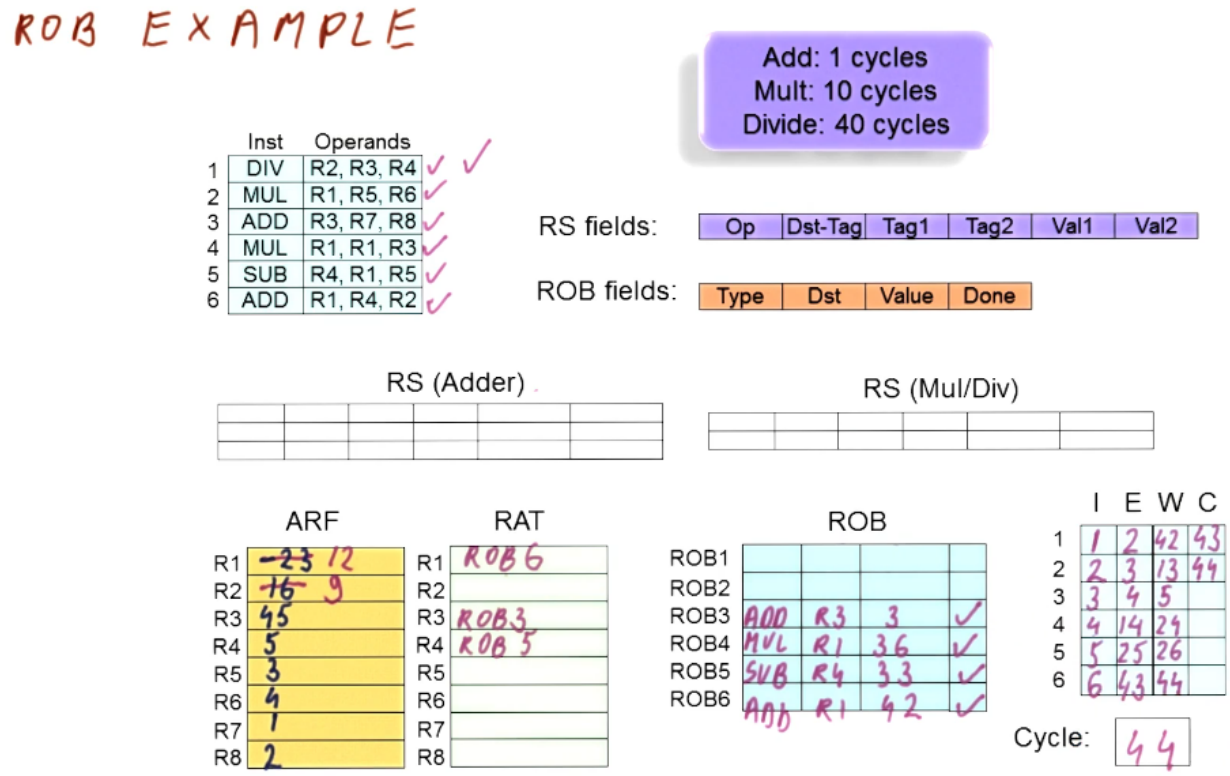
ROB Example Cycle 42



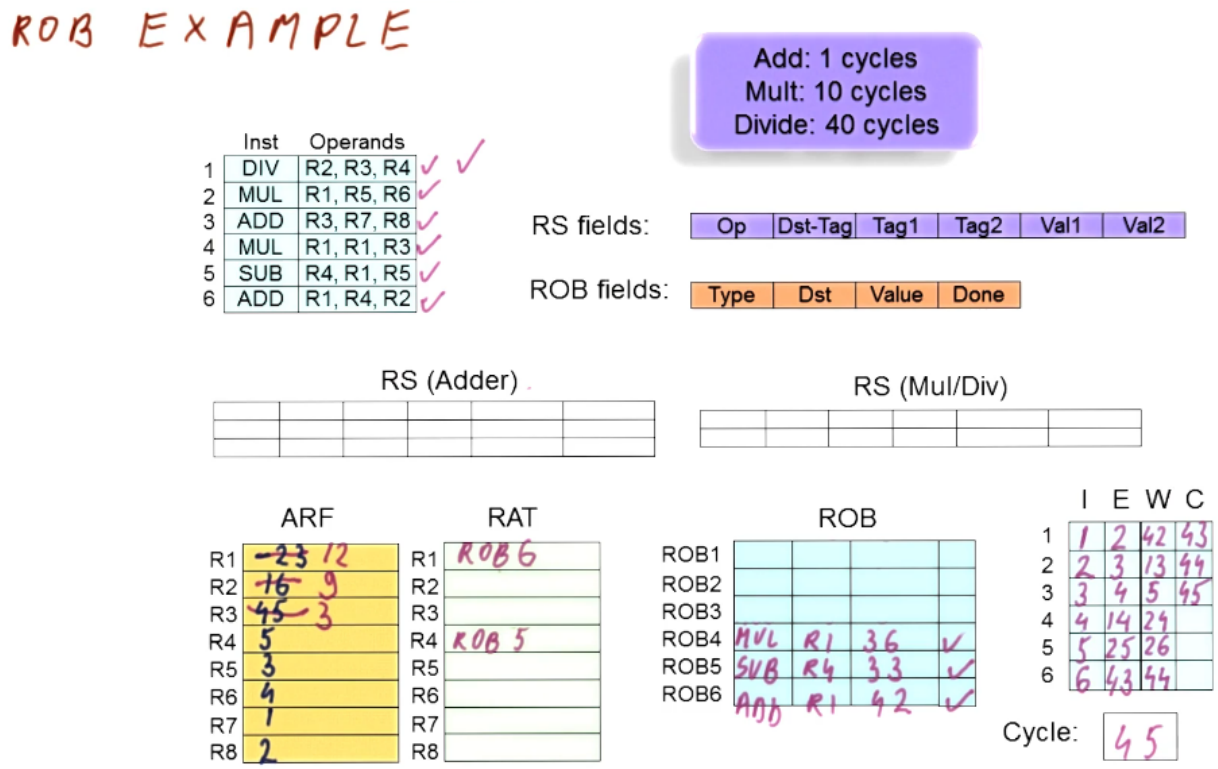
ROB Example Cycle 43

1. When an instruction commits, remember to update the ARF and RAT

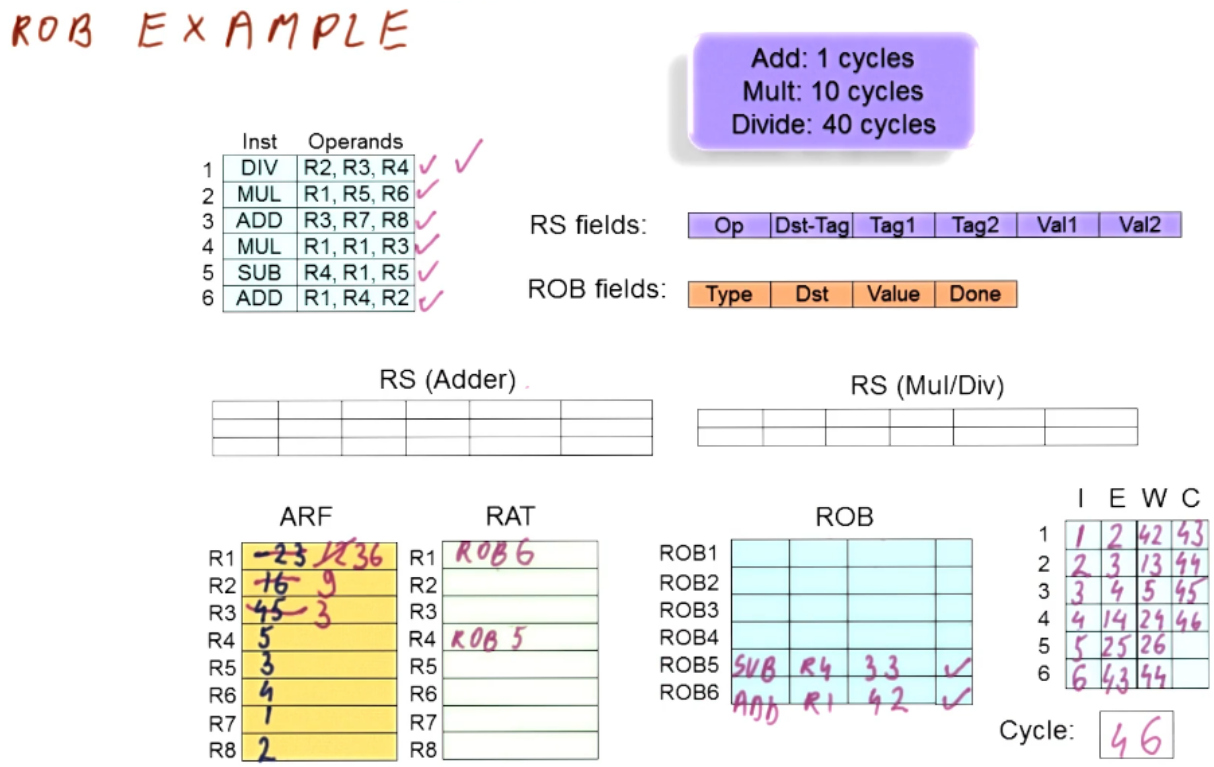
ROB Example Cycles 44-48



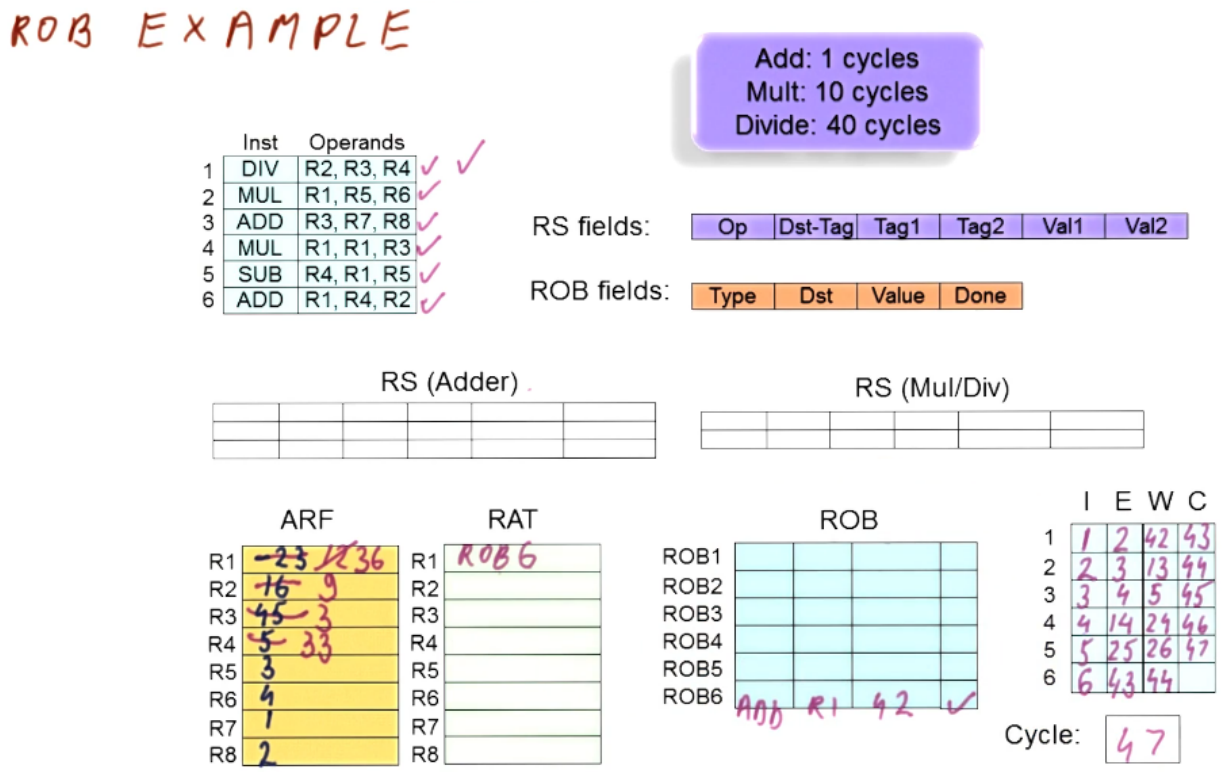
ROB Example Cycle 44



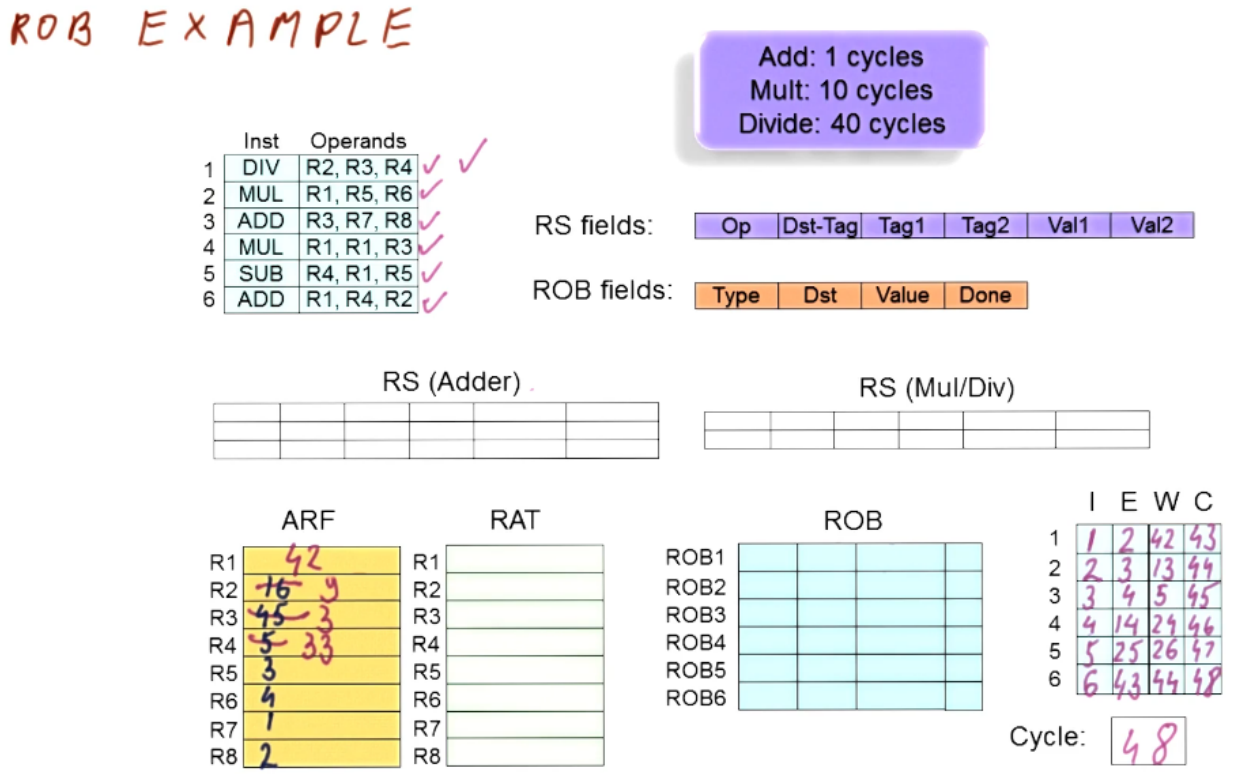
ROB Example Cycle 45



ROB Example Cycle 46



ROB Example Cycle 47



ROB Example Cycle 48

ROB Quiz 1

1. Consider the following program:

```
DIV R2, R3, R4
MUL R1, R5, R6
ADD R3, R7, R8
MUL R1, R1, R2
SUB R4, R3, R5
ADD R1, R4, R2
```

2. What is contained in the RAT for R2 at the end of cycle 1?
   - ROB1

ROB Quiz 2

1. What is contained in the multiplication reservation station at the end of cycle 4?

Operation	Dst-Tag	Tag1	Tag2	Val1	Val 2
MUL	ROB4	ROB2	ROB1		

2. Which RAT entry is modified as a result of this instruction?
   - R1

ROB Quiz 3

1. What instruction is dispatched in cycle 5?

- The result of the add that is written in cycle 5 allows the SUB R4, R3, R5 instruction to be dispatched
- Important: An instruction can be issued and dispatched in the same cycle, but fetch/execute/write/commit cannot
  - Dispatch isn't its own stage in the pipeline

2. What results are written in cycle 5?
   - ADD R3, R7, R8 (ROB3) is written

ROB Quiz 4

1. In what cycle is the MUL R1, R1, R2 in ROB4 dispatched?
   - Must wait for the result of ROB1 (DIV R2, R3, R4), so it will be dispatched once that value is broadcast in cycle 12

ROB Quiz 5

1. In addition to instructions 4 and 6 being dispatched, what else happens in cycle 13?
   - ROB1 commits

ROB Quiz 6

1. Which RAT entry changes in cycle 14?
   - Nothing changes. Instruction 2 commits, but the RAT isn't pointing at ROB2 anymore since ROB6 also writes to R1. RAT updates only occur on commits.

ROB Quiz 7

1. Which entry in the architecture register file changes at the end of cycle 14?
   - R1 becomes 8 when ROB2 commits

ROB Quiz 8

1. When does the last instruction commit?
   - Cycle 19

ROB Timing Example

1. Assumptions:
   - RS is freed on broadcast, not dispatch (some processors actually do this)
   - Can Issue, Capture, and Dispatch in same cycle
   - Not limited by ROB entries
   - 3 add/subtract reservation stations
   - 2 divide/multiply reservation stations
   - Add: 1 cycle
   - Multiply: 10 cycles
   - Divide: 40 cycles

Inst	Operands	Is	Exec	Wr	Commit	Comments
DIV	R2,R3,R4	1	2	42	43	
MUL	R1,R5,R6	2	3	13	44	
ADD	R3,R7,R8	3	4	5	45	
MUL	R1,R1,R3	14	15	25	46	Need RS -> Issue
SUB	R4,R1,R5	15	26	27	47	Need R1 -> Exec
ADD	R1,R4,R2	16	43	44	48	Need R2 -> Exec

ROB Timing Quiz

1. Assumptions:
   - RS is freed on dispatch
   - Can Issue, Capture, and Dispatch in same cycle
   - Not limited by ROB entries
   - 3 add/subtract reservation stations
   - 2 divide/multiply reservation stations
   - Broadcast one add/sub AND one mul/div per cycle
   - Add: 1 cycle
   - Multiply: 2 cycles
   - Divide: 4 cycles

Inst	Operands	Is	Exec	Wr	Commit	Comments
DIV	R2,R3,R4	1	2	6	7	
MUL	R1,R5,R6	2	3	5	7	
ADD	R3,R7,R8	3	4	5	8	
MUL	R1,R1,R2	4	7	9	10	Need R1 -> Exec
SUB	R4,R2,R5	5	7	8	10	
ADD	R1,R4,R2	6	9	10	11	Need R4 -> Exec

2. Question: For the last instruction, why can't it begin executing in cycle 8? The assumptions state we can issue, capture, and dispatch in the same cycle and nothing else is being executed during this cycle.

Unified Reservation Stations

1. In our examples so far, there are three RS for add/sub and 2 for mul/div
   - It's possible that we run out of one type of RS and must wait, even if there are available RS for the other unit
   - The logic of the RS are identical, just dispatch to different units
2. Instead of separate RS, we can make one array of reservation stations
   - Downside: Logic for dispatch to add or mul unit is more complicated
   - Every cycle, we need to select one unit for add and one for multiply
3. Reservation stations are very expensive, so modern processors typically take this approach

Superscalar

1. Fetch more than 1 instruction/cycle
2. Decode more than 1 instruction/cycle (multiple decoders)
3. Issue more than 1 instruction/cycle
   - Must be done in order; if the next instruction can issue, also consider the next N-1 instructions
4. Dispatch more than 1 instruction/cycle
   - This was possible in our ROB examples, simply need multiple units
5. Broadcast more than 1 instruction/cycle
   - Increases the cost of each RS; cost of RS is proportional to the number of broadcasts we have to monitor (must compare tags)
6. Commit more than 1 instruction/cycle
7. Need to be concerned with the "weakest link" (bottleneck)

Terminology Confusion

1. Issue: Obtained RS and ROB entry
   - Also referred to as "allocate" or "dispatch"
2. Dispatch: Instruction selected for execution

- Also referred to as “issue” or “dispatch”

3. Commit: Instruction is committed
   - Also referred to as “complete” or “retire” or “graduate”
4. Original Tomasulo paper and first paper on ROBs used the issue, dispatch, commit terminology

Out of Order

1. Example pipeline: Fetch -> Decode -> Issue -> Exec -> Broadcast -> Commit
   - Fetch, decode, and issue must all happen in order to ensure dependencies are compatible with original program order
   - Execution and broadcast happen in order of data dependencies
   - Commit must happen in order for the output to appear in program order

In Order vs Out of Order Quiz

1. Which stages are in order and which are out of order?

Stage	In-Order	Out-of-Order
Fetch	X	
Decode	X	
Issue	X	
Dispatch		X
Execute 1		X
Execute 2		X
Broadcast		X
Commit	X	
Release ROB	X	

Conclusion

1. All modern high-performance processors include a reorder buffer