High Performance Computer Architecture (CS60003)

Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Spring 2023

Tomasulo's Algorithm

Out of Order Processors

From Scalar to Superscalar

- Our 5-stage pipeline was based on the idea that the instruction execution cycle could be decomposed into non-overlapping stages with the instruction passing through every stage at every cycle.
  - This is called a scalar processor
  - Ideal throughput of 1, IPC=1
- With the same ISA, the possibility to improve the execution time, thus depends on:
  - Increasing the ideal IPC of 1 by radically modifying the structure of the pipeline to allow more than one instruction to be in each stage at a given time.
  - These are called as super-scalar processors.

Does this effect the pipeline?

- From the micro-architecture point of view, the pipeline is called wide pipeline.
- It brings several fundamental changes:
  - We need several functional units.
  - The pipeline also does not remain linear.
  - Each stage of the pipeline gets affected.
- Also hand-in-hand, to decrease execution time, by increasing clock frequency, the processor needs to do less job per cycle.
  - This is achieved by deeper pipelines.
- Modern processors are therefore both deeper and wider.

Viewing the pipelines in super-scalar processors

- To study the design decisions that are needed to implement concurrency caused by the superscalar effect, and the consequences of the deeper pipeline, we view the same in two parts:
  - Front end: Corresponds to the FETCH and DECODE stages
    - The front-end must now fetch and decode several instructions at once.
    - m-way superscalar: The number of instructions brought ideally into the pipeline at each cycle is m.
  - Back-end: Comprises of Execution, Memory, and Write Back stages.

In-order and Out-of-order Superscalar Processors

- In both cases, the instructions proceed in the front-end in program order.
- In-order or static processors: Instructions leave the front-end in program order and all data-dependencies are resolved before they go to the back-end.
- Out-of-order or dynamic processors: Instructions can leave the front-end and execute in the back-end before some of their program order predecessors.
  - The Write Back stage needs to be designed so that the semantics of the program are respected.
  - Such processors are more complex to build.
  - Require 30% more logic for the same number of functional units

Finer Points

- Theoretically, an m-way superscalar processor with k-times more clock frequency should provide mk-times speedup.
- However there are several bottlenecks:
  - We need to discover the amount of ILP in the program.
  - Very difficult in in-order processors as instructions that have RAW dependencies cannot leave the front-end until dependencies are resolved.
  - An out-of-order processor though can exploit the ILP better but is very complex to design.
  - Hardly more than 6-way.
  - Intel's 12th generation processors are 5-way decode.

Finer Points (Contd.)

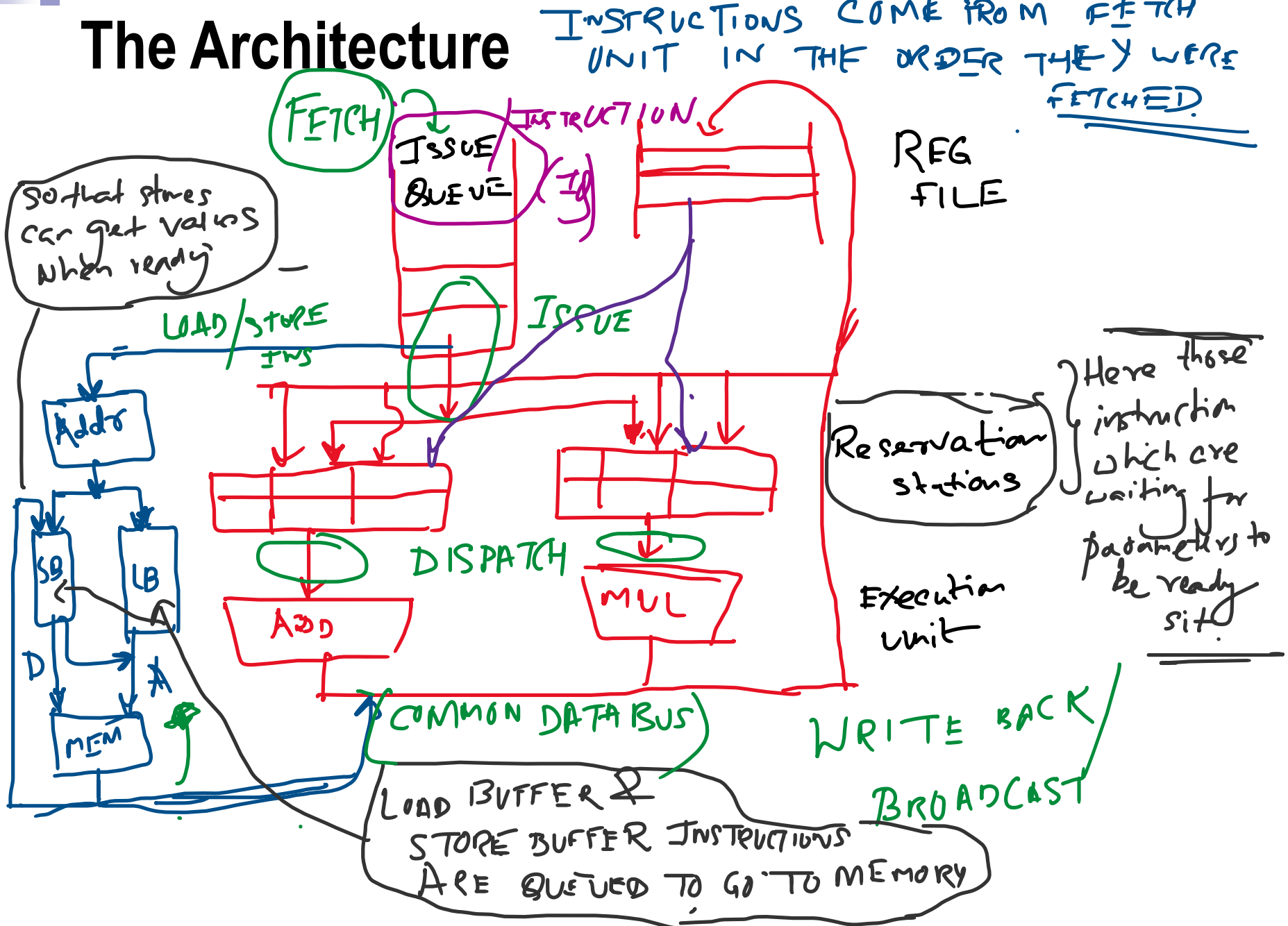
- Second factor that limits m is that the back-end requires m functional units, and the number of forwarding units increase quadratically.
- The increase in clock frequency is also capped due to power dissipations: hovers around 3-5 GHz.
- Second constraint is the pipelined registers, which needs to be written and read in every cycle.
  - This provides a lower bound to the cycle time.
- Deep pipelines also have problems due to mispredictions.
  - Intel core suggested 14 stage pipeline which reversed the trend of Pentium D with 31 stages vs 10 stages in Pentium III

Finer Points (Contd.)

- In-order processors were the first successors of scalar processors
- They were easier to implement compared to the dynamic superscalar processors then due to the higher clock speeds
  - Though out of order processors had better IPC.
- Today high performance single processor chips are out of order.
  - The speed advantages of static processors are not existent now due to cap for controlling power dissipations
- Because of the development of Moore's law, which allows more chip logic, single processors are replaced by multi-processors on a chip (CMP).
- The in-order to out-of-order transition requires to introduce concepts of Tomasulo's algorithm introduced for IBM Systems 360/91.

- Introduced in IBM 360 (More than 40 years ago)
- In the hardware register renaming is performed
- Which instructions have inputs ready are decided
- Strikingly very similar to modern processors
- Originally, Tomasulo's algorithm was developed only for floating point instructions
  - But today all instructions are executed in a similar manner
- There were fewer instructions in the window
  - 100s of instructions past the current instructions are in the window

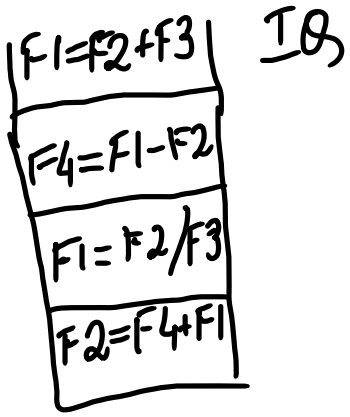
The Architecture

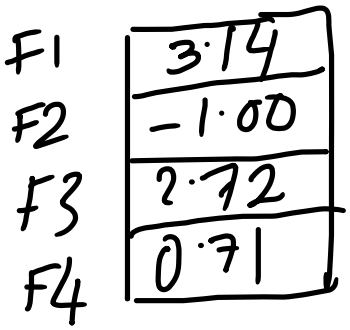


Issue Stage

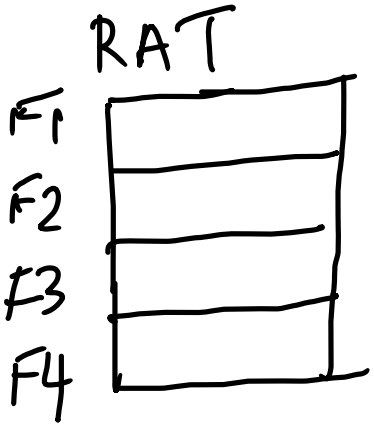
- Take next (in program order) instruction from instruction queue (IQ)
  - For register renaming to work
- Determine where inputs come from RAT table
- Get free reservation stations (RS) (of correct kind)
- Put instruction in RS
- Tag Destination Register

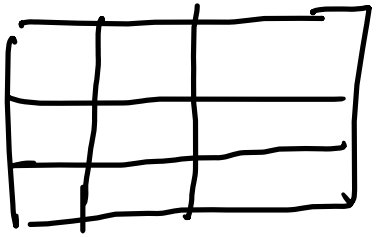
Example

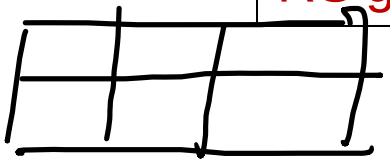


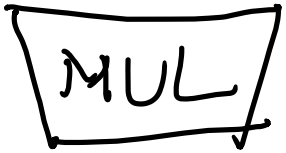


If RS is not empty, we have to wait till one of the instructions in the RS get executed.



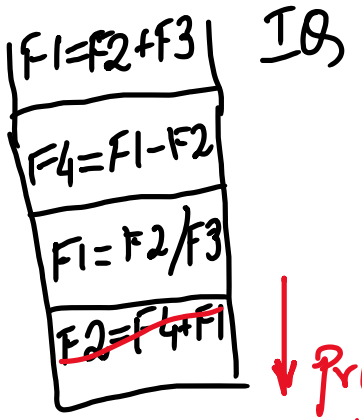


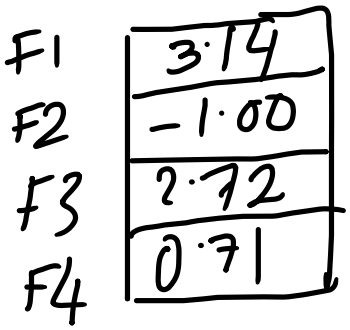




RAT entry stores which instruction is producing it. If it is empty see register-file.

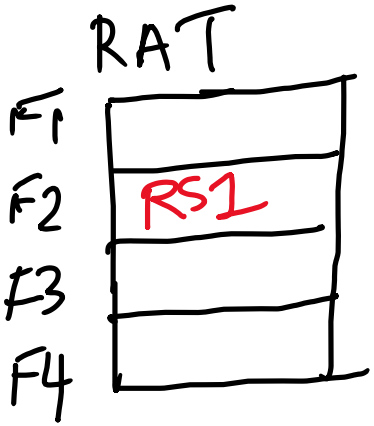
Example

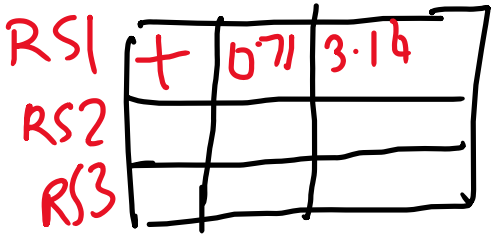


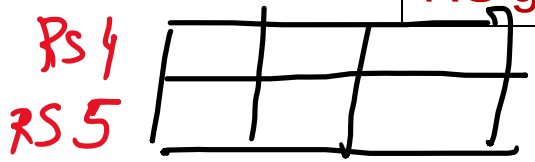


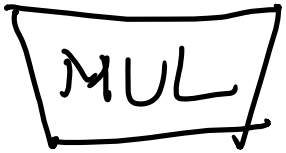
(RF)
Reg File

If RS is not empty, we have to wait till one of the instructions in the RS get executed.



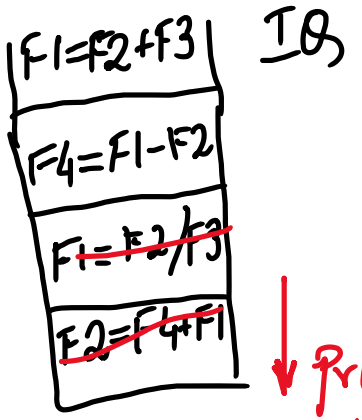


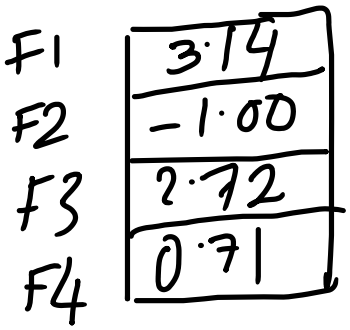




RAT entry stores which instruction is producing it. If it is empty see register-file.

Example



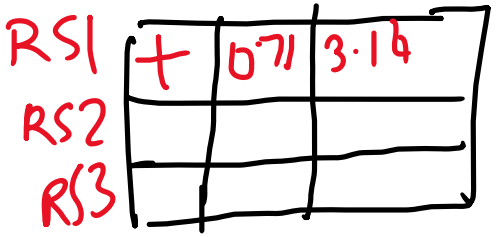


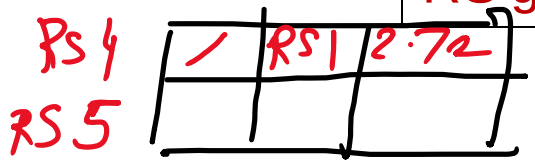
(RF)
Reg File

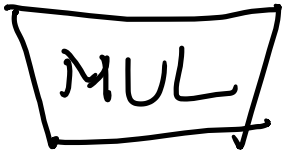
If RS is not empty, we have to wait till one of the instructions in the RS get executed.

Program Order

RAT

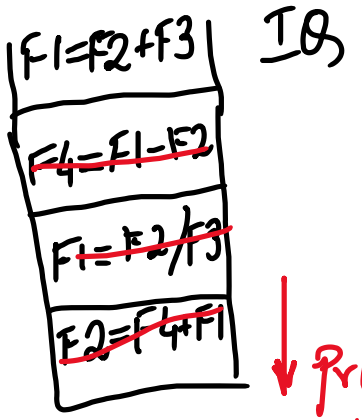


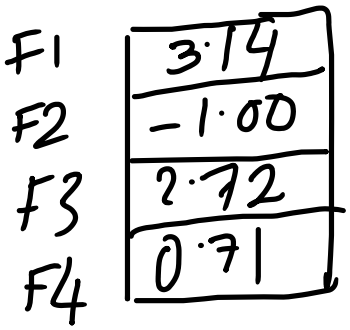




RAT entry stores which instruction is producing it. If it is empty see register-file.

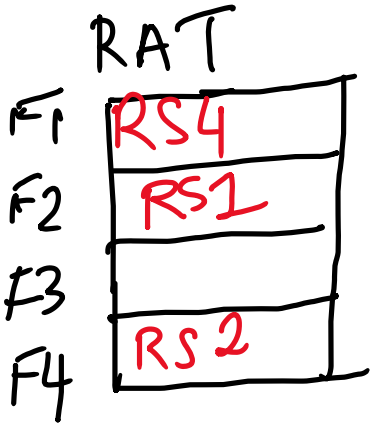
Example

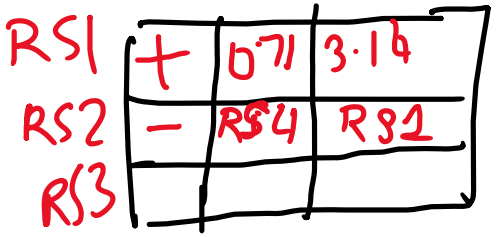


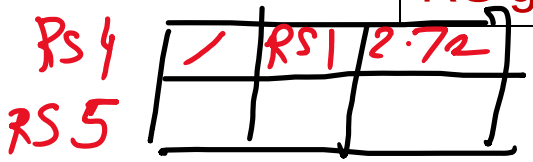


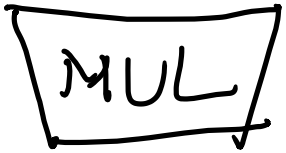
(RF)
Reg File

If RS is not empty, we have to wait till one of the instructions in the RS get executed.



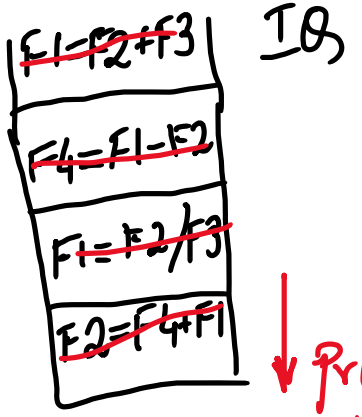


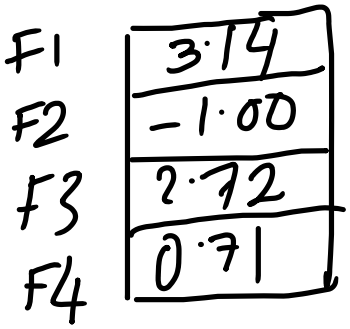




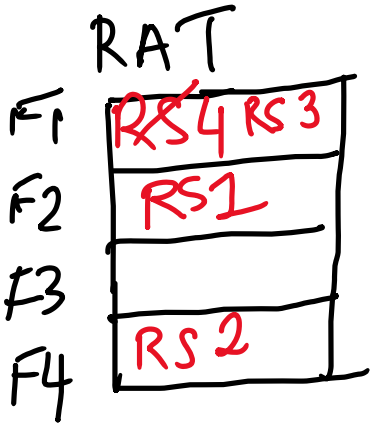
RAT entry stores which instruction is producing it. If it is empty see register-file.

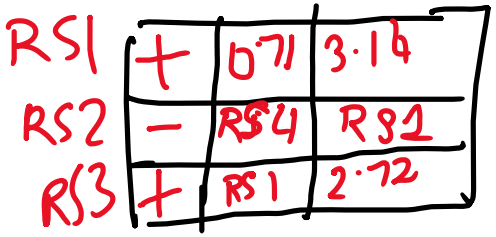
Example

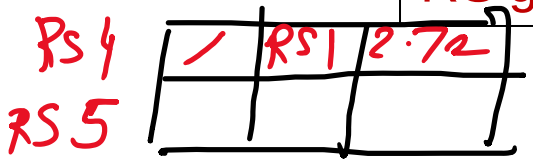


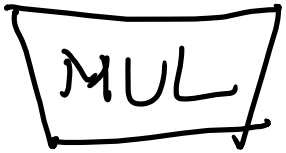


If RS is not empty, we have to wait till one of the instructions in the RS get executed.





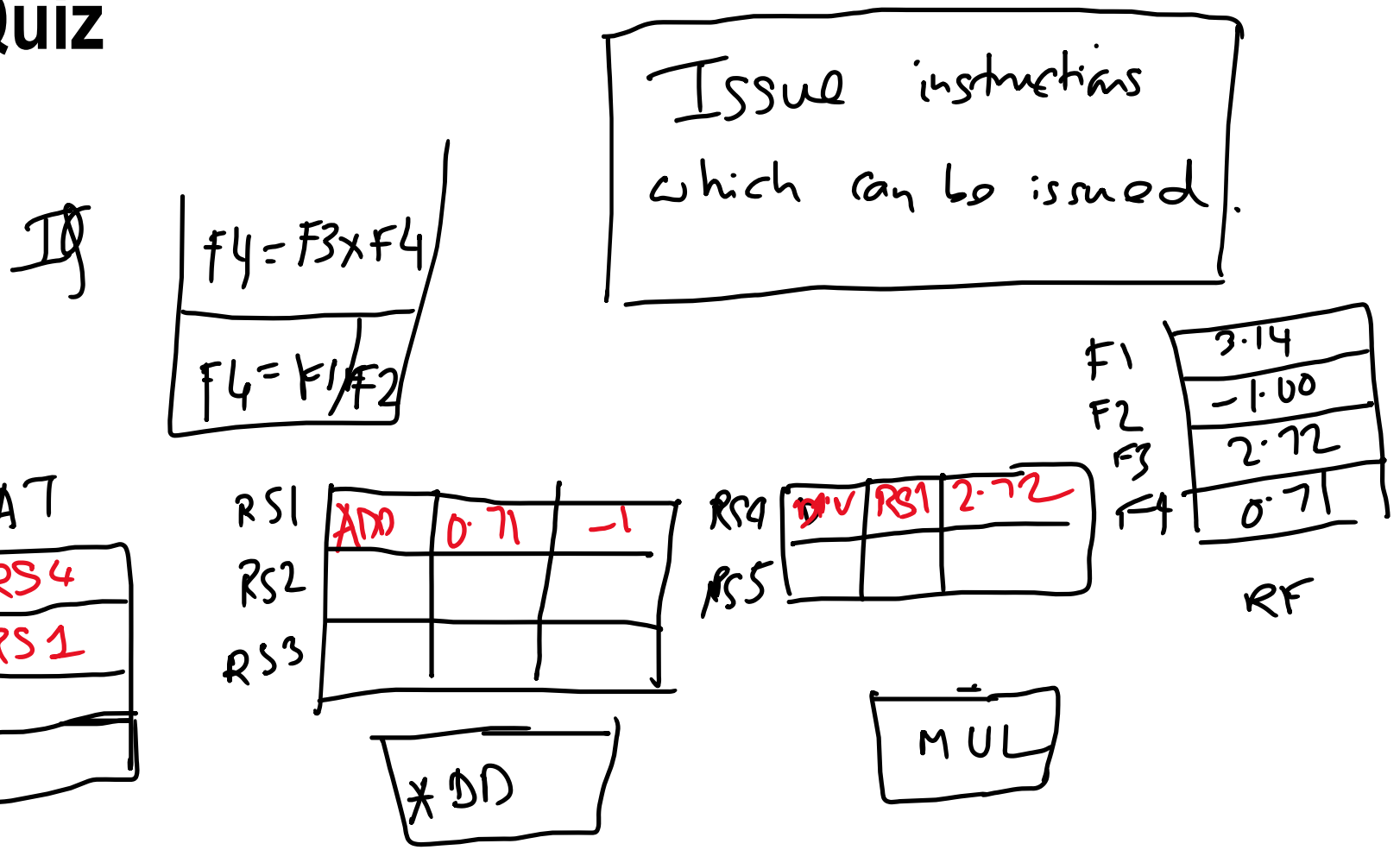




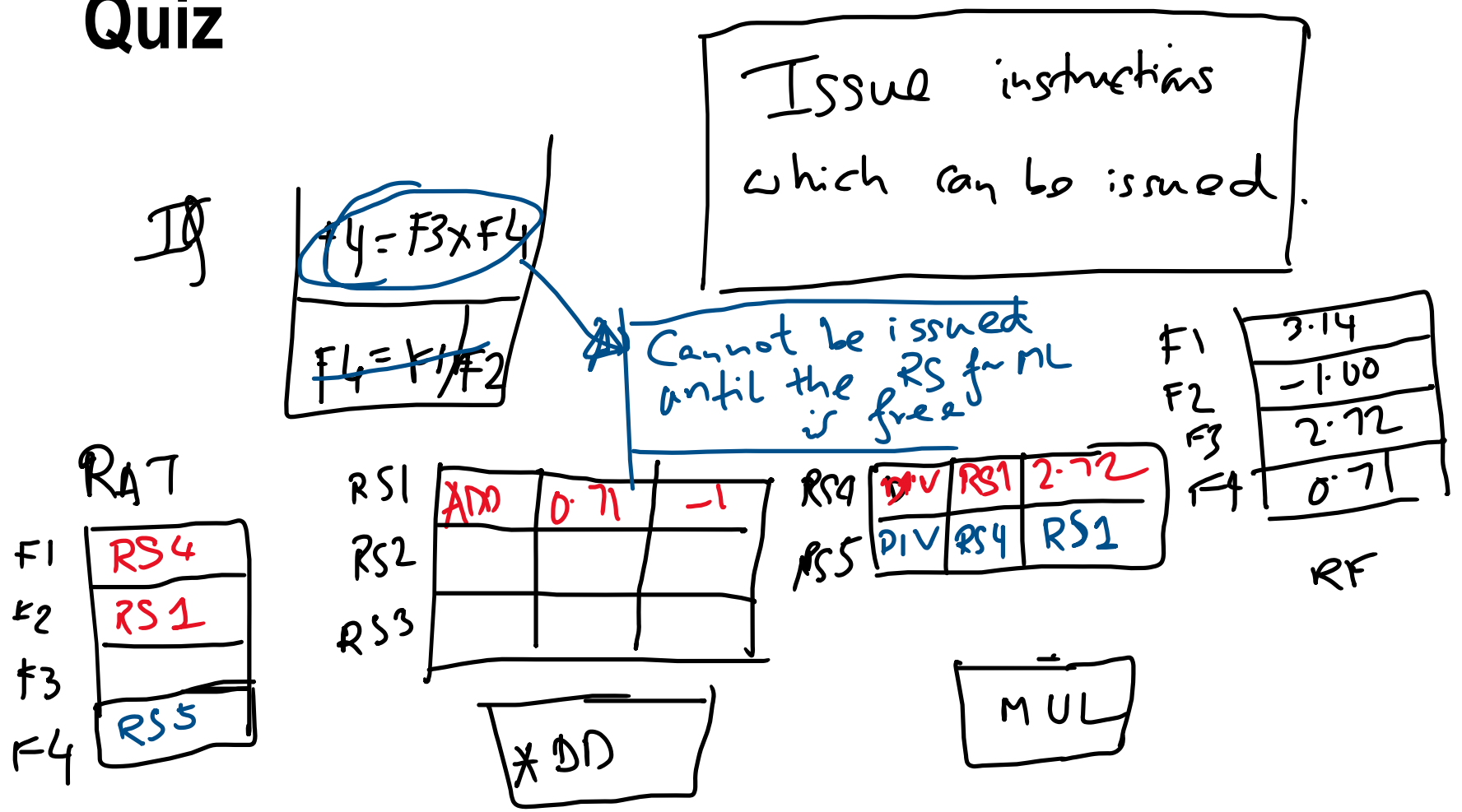
RAT entry stores which instruction is producing it. If it is empty see register-file.

In real life processor, one instruction would be issued per cycle, and some of the instructions will also execute by that time. Here we saw how issue works if nothing executes due to some reason!

Quiz


Issue instructions which can be issued.
IQ
F4 = F3 x F4
F4 = F1 / F2
F1
F2
F3
F4
3.14
-1.00
2.72
0.7
RF
RAT
F1
F2
F3
F4
RS4
RS1
RS1
RS2
RS3
ADD
0.71
-1
RS4
RS5
DIV
RS1
2.72
ADD
MUL

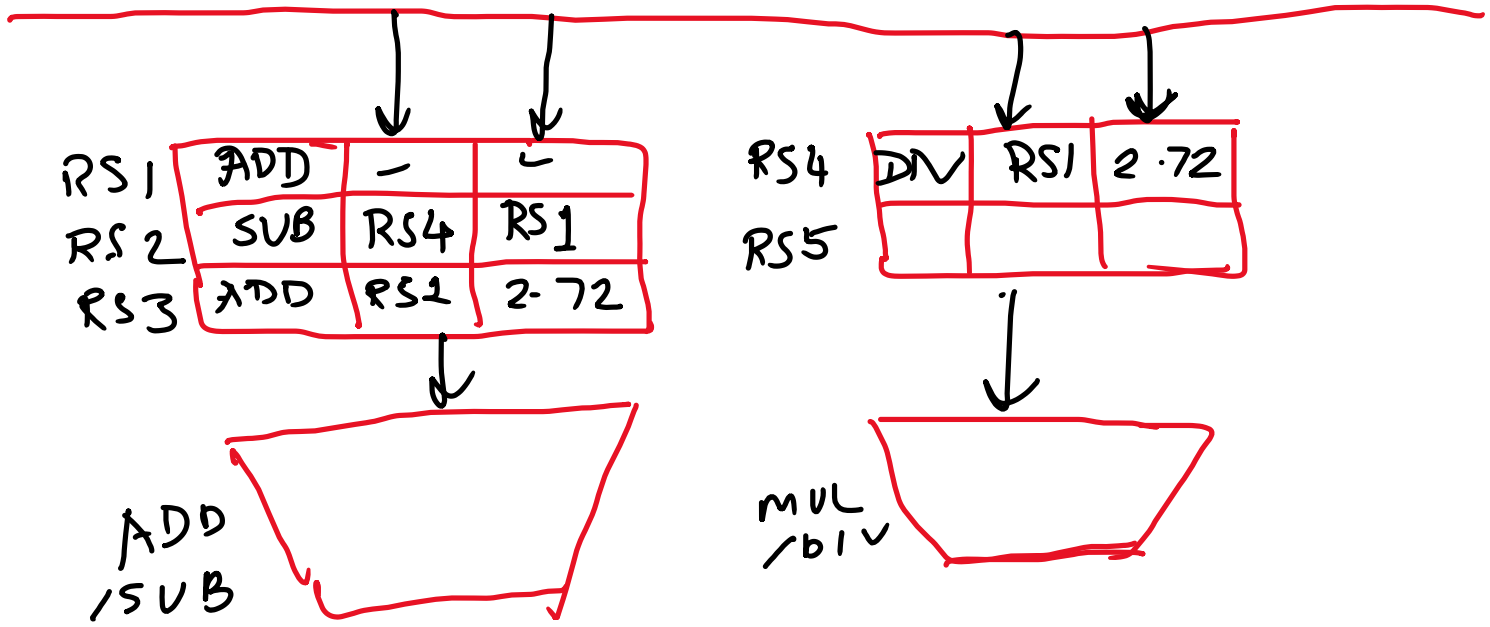
Quiz


Issue instructions which can be issued.
IQ
F4 = F3 x F4
F4 = F1 / F2
Cannot be issued until the RS for MUL is free
RAT
F1 RS4
F2 RS1
F3
F4 RS5
RS1 ADD 0.71 -1
RS2
RS3
RS4 DIV RS1 2.72
RS5 DIV RS4 RS1
F1 3.14
F2 -1.00
F3 2.72
F4 0.71
RF
ADD
MUL

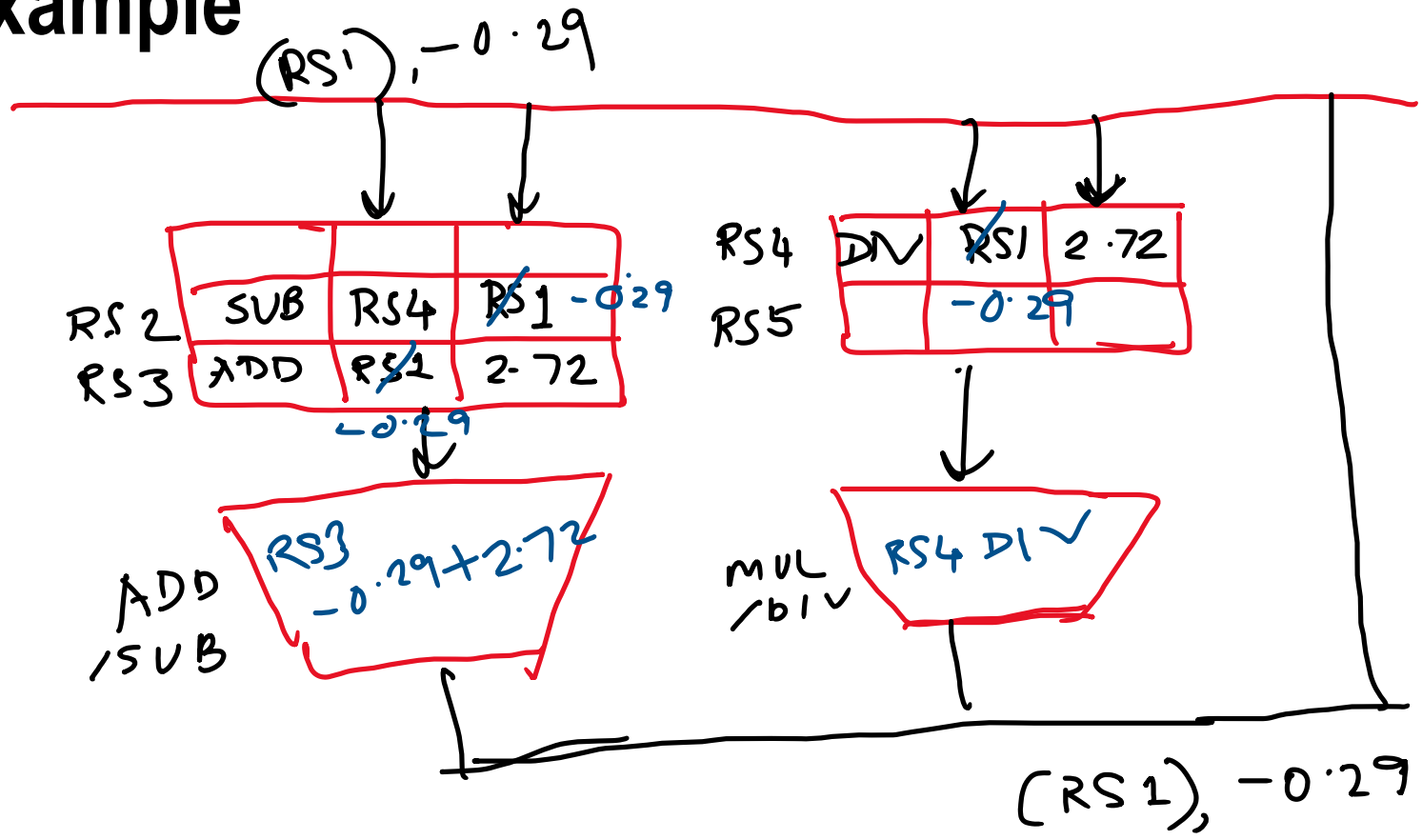
Dispatch

- Needs to consider latching of operation results that are produced.
- Also which instructions are ready to execute.
- We usually perform both in the same cycle.
  - At the beginning of the cycle we latch/update the broadcasted result (capture)
  - At the end of the cycle, we decide which instructions are ready to execute.
- Once these results are ready we would broadcast the results.

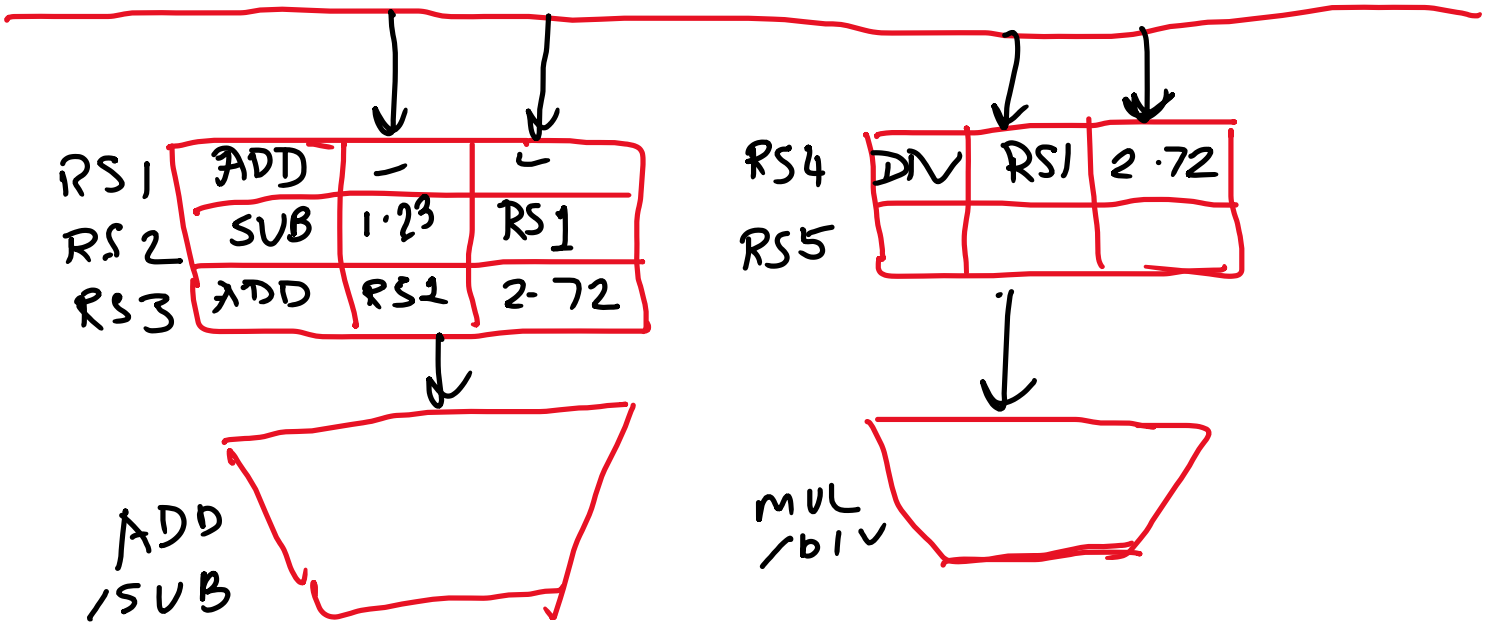
Example


RS1
RS2
RS3
ADD
SUB
ADD
–
RS4
RS1
–
RS1
2·72
ADD
/SUB
RS4
RS5
DIV
RS1
2·72
MUL
/DIV

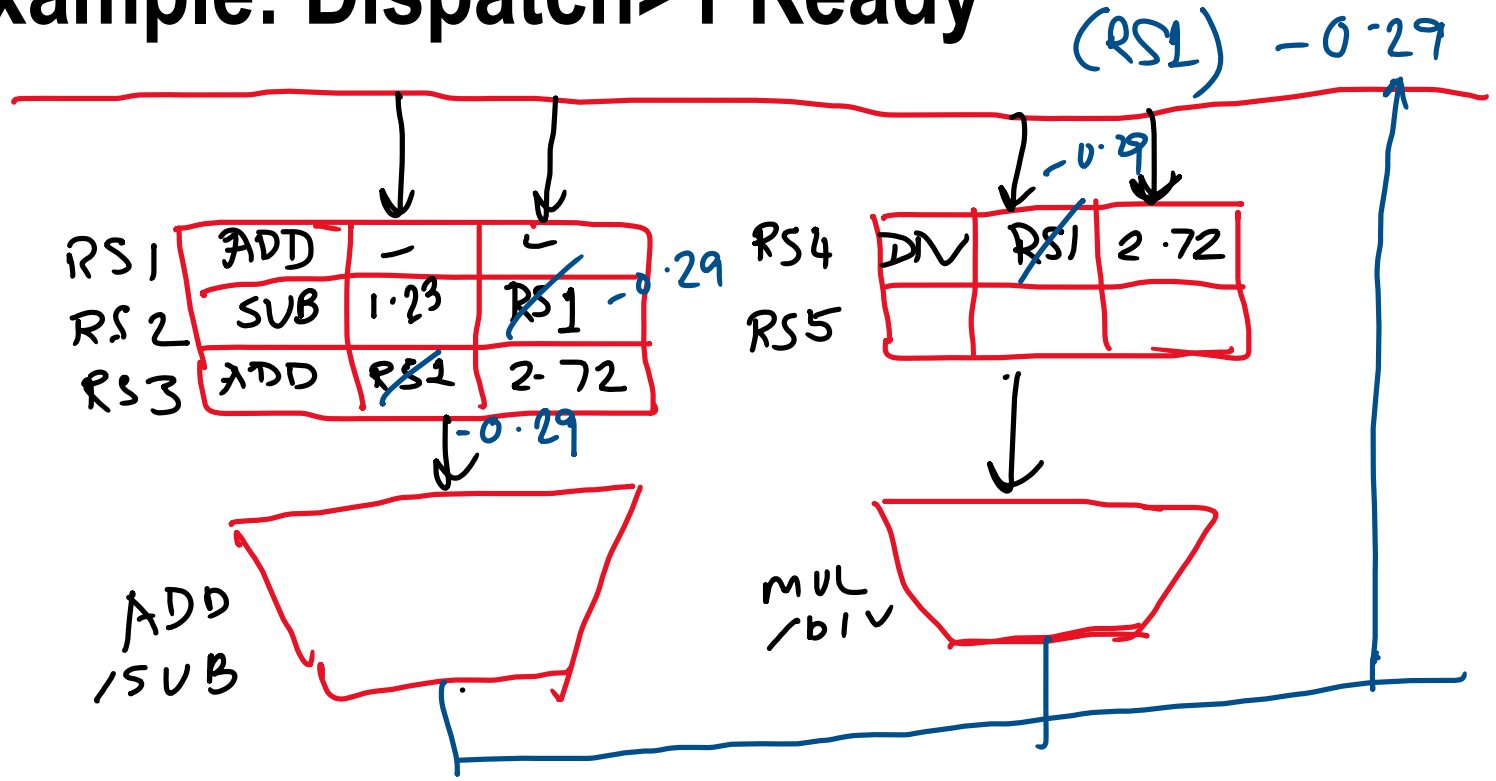
Example


(RS1), −0.29
RS2
RS3
SUB
RS4
RS1 −0.29
ADD
RS1
2.72
−0.29
RS3
−0.29+2.72
ADD
/SUB
RS4
DIV
RS1
2.72
RS5
−0.29
MUL
/DIV
RS4 DIV
(RS1), −0.29

Example: Dispatch>1 Ready



Example: Dispatch>1 Ready

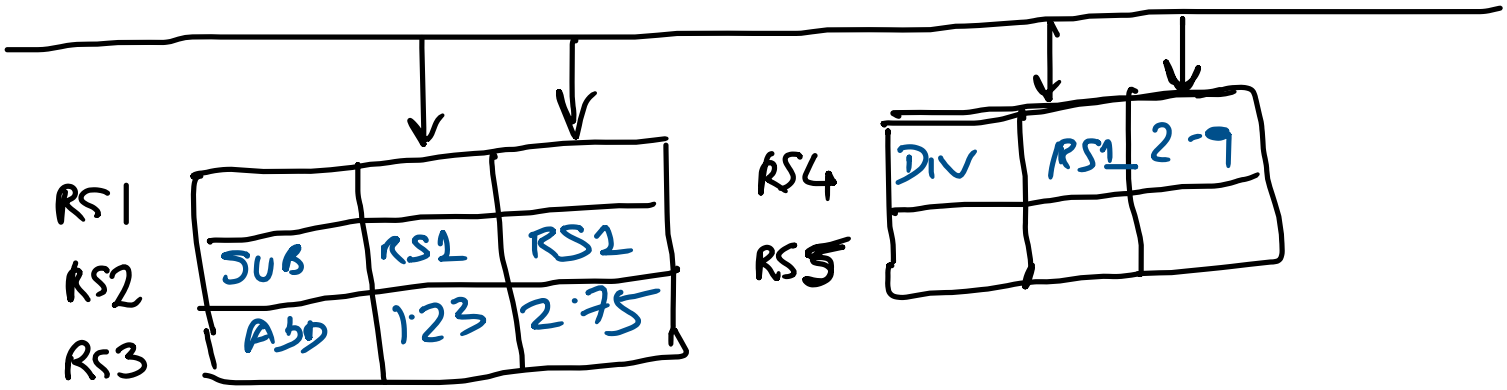


We need to choose which one will be dispatched?

Common Heuristics

- Oldest first: It is more likely that larger number of instructions are waiting.
- Most dependencies first: difficult to implement
- Random:
  - Could be based on whichever is first in the reservation station
  - Does not hamper correctness
  - If we execute the instruction ahead on which less instructions depend, very soon there will be no instruction to dispatch.
  - So, then we will dispatch the one (may be the oldest) on which more instructions are waiting
  - So, correctness wise we are still ok, but performance may be affected
- So, we tradeoff by selecting the oldest first approach.

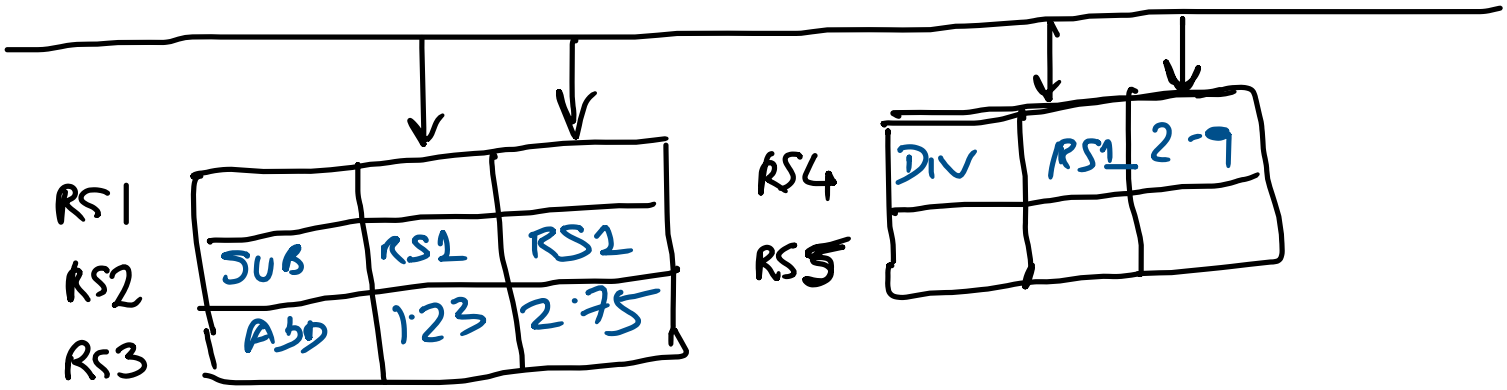
Dispatch Quiz



Why has not RS3 dispatched when its arguments are ready?

1. Another instruction was dispatched.
2. It was just issued, dispatch will take place at the end of the cycle.
3. RS2 is older than RS3, so RS3 cannot dispatch until RS2 does.

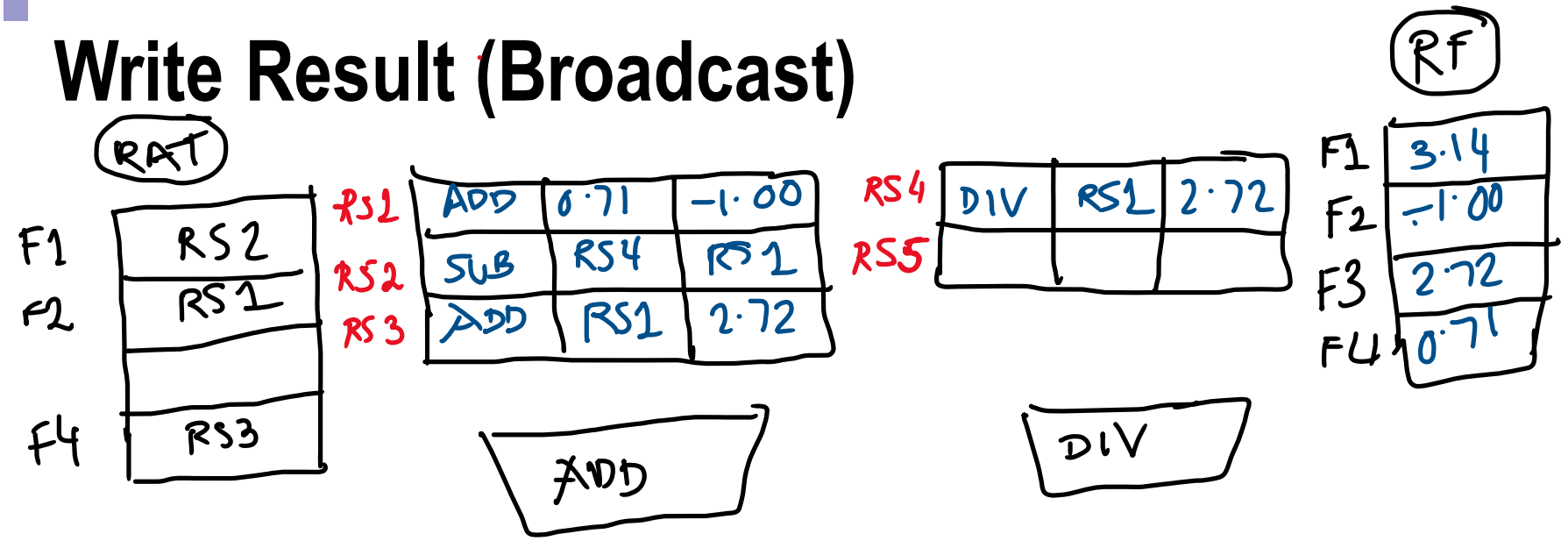
Dispatch Quiz



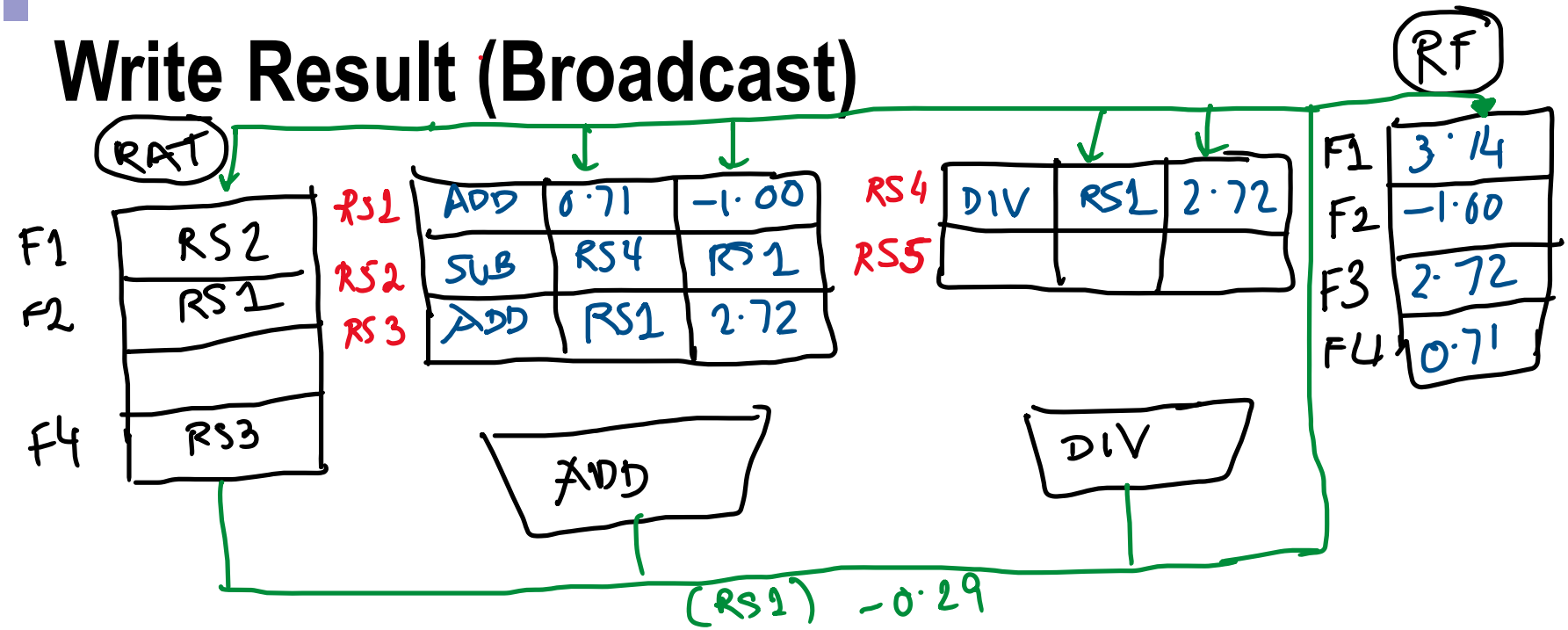
Why has not RS3 dispatched when its arguments are ready?

1. Another instruction was dispatched. ✓
2. It was just issued, dispatch will take place at the end of the cycle. ✓
3. RS2 is older than RS3, so RS3 cannot dispatch until RS2 does. ✗ (then it would be in-order).

Write Result (Broadcast)

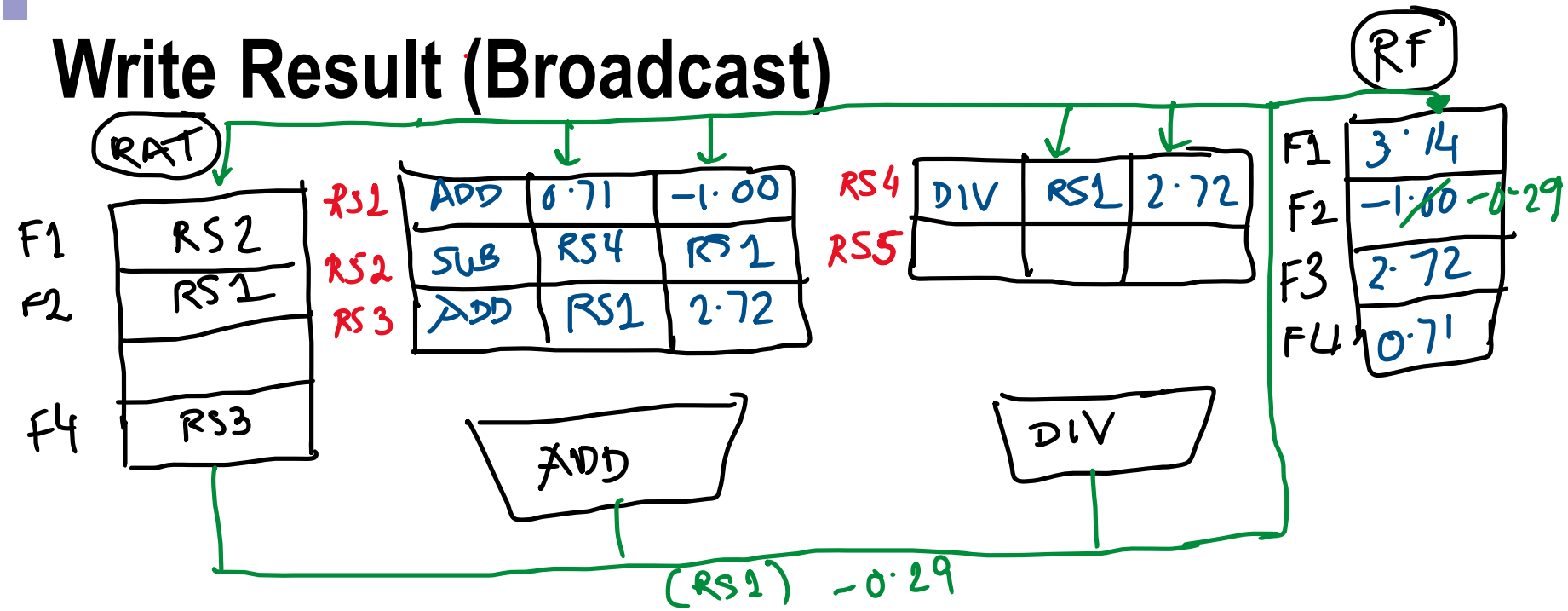


Write Result (Broadcast)



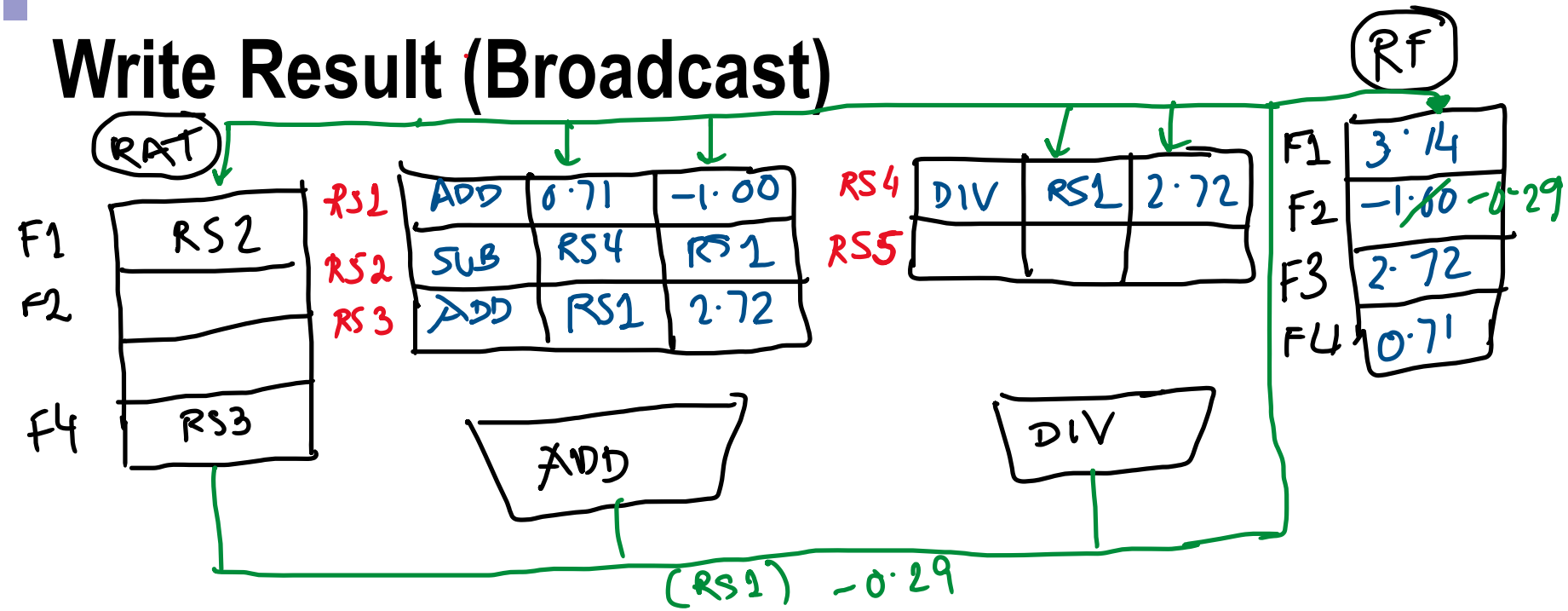
1. Put TAG and Result on Bus

Write Result (Broadcast)



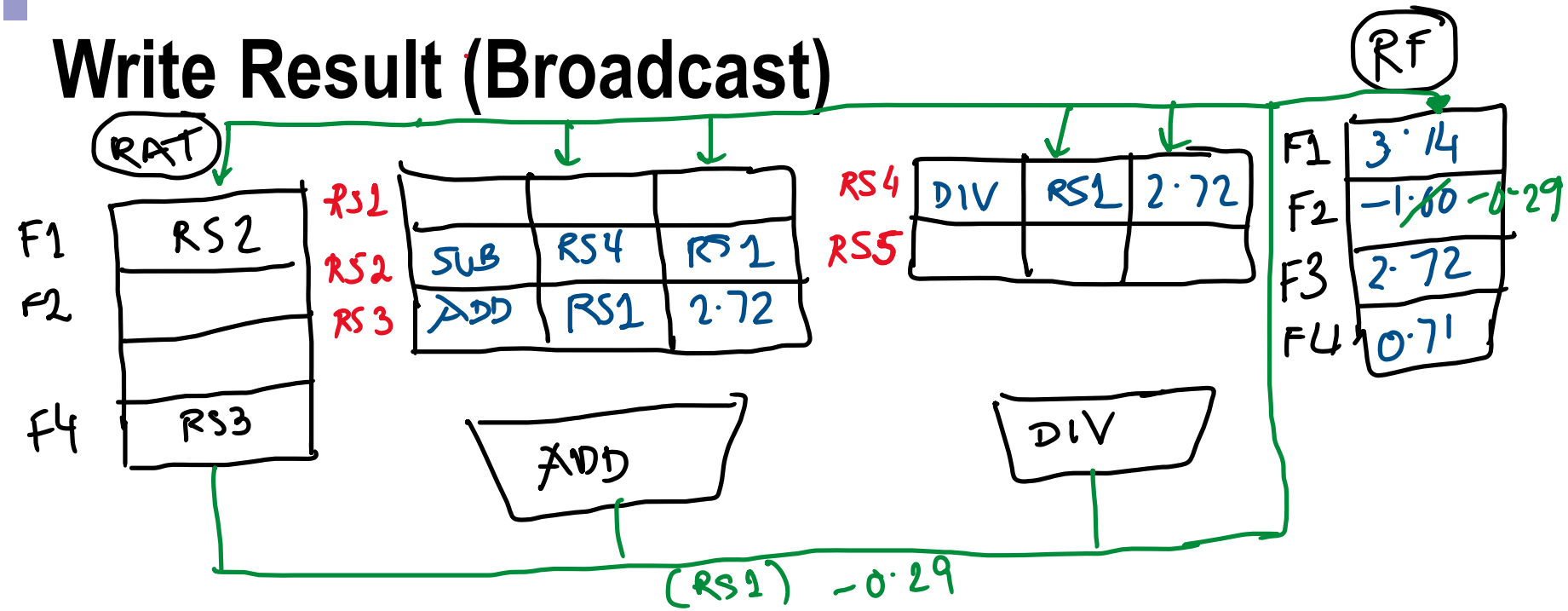
1. Put TAG and Result on Bus
2. Write to Register File: We do not need to remember F2, it can be obtained from RAT

Write Result (Broadcast)



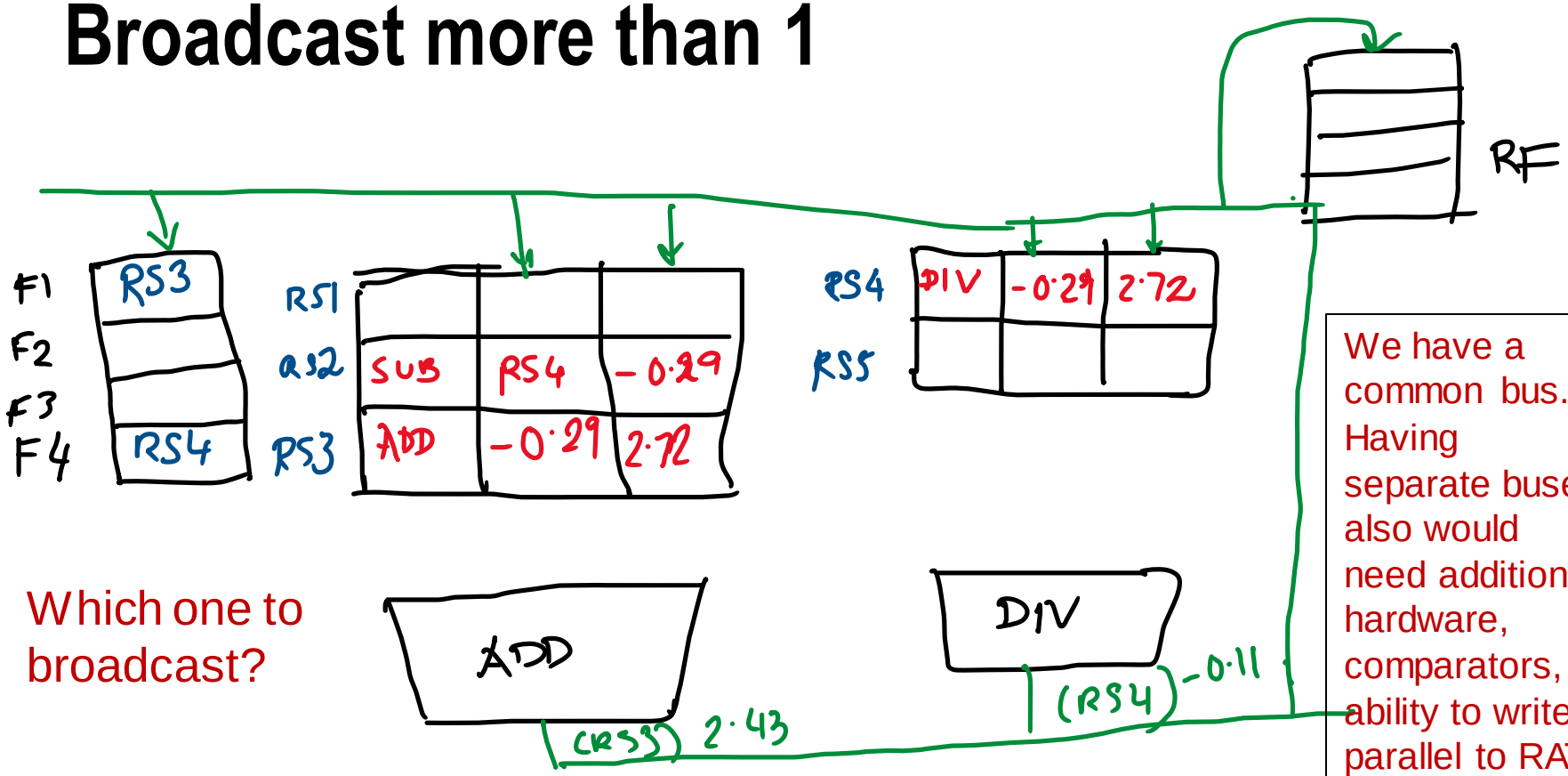
1. Put TAG and Result on Bus
2. Write to Register File: We do not need to remember F2, it can be obtained from RAT
3. Update RAT: Empty the entry of F2 by using a valid/invalid bit

Write Result (Broadcast)



1. Put TAG and Result on Bus
2. Write to Register File: We do not need to remember F2, it can be obtained from RAT
3. Update RAT: Empty the entry of F2 by using a valid/invalid bit
4. Free RS (in real hardware we just invalidate a valid bit)

Broadcast more than 1

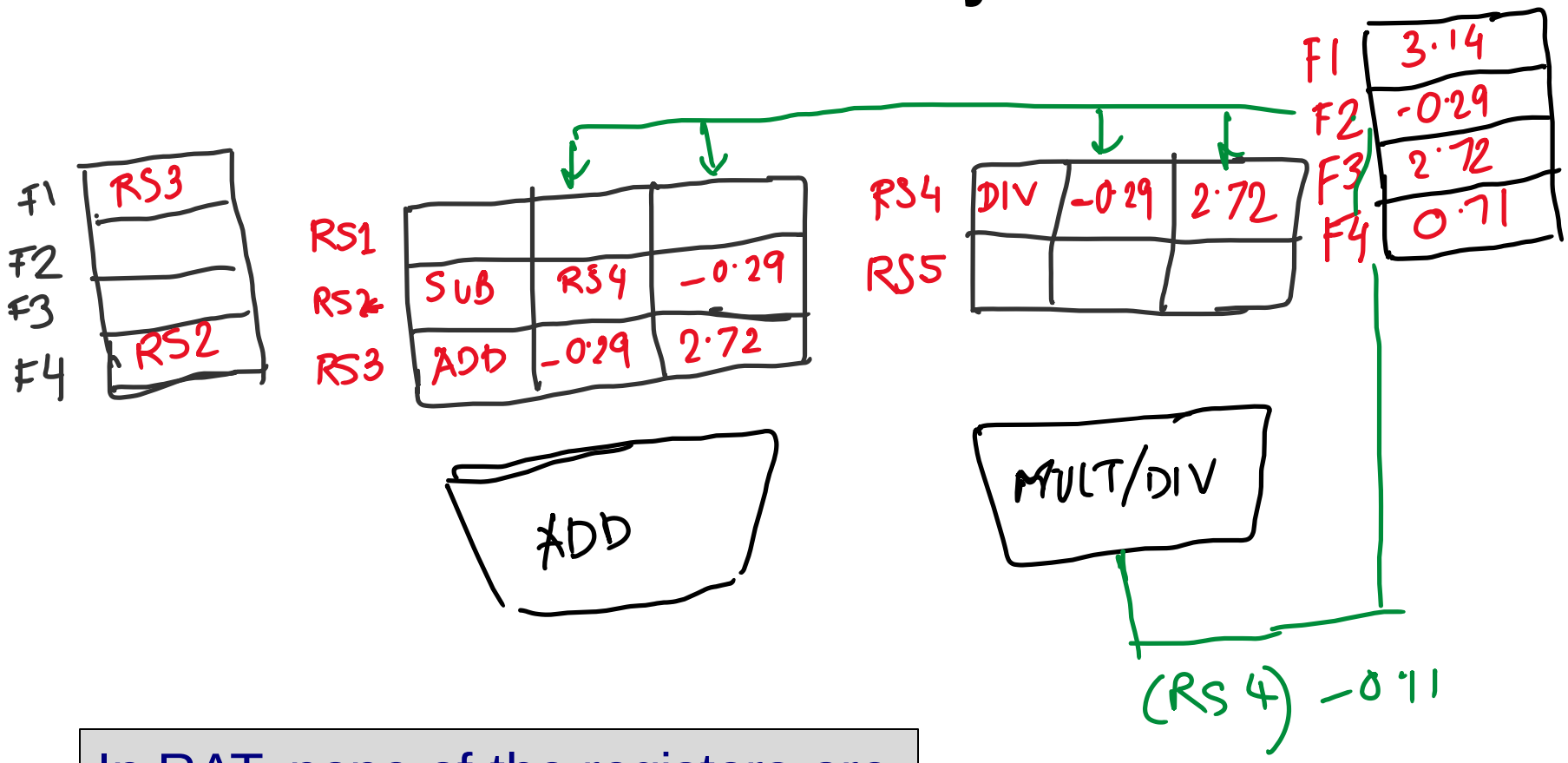


Which one to broadcast?

We have a common bus. Having separate buses also would need additional hardware, comparators, ability to write in parallel to RAT, RF.

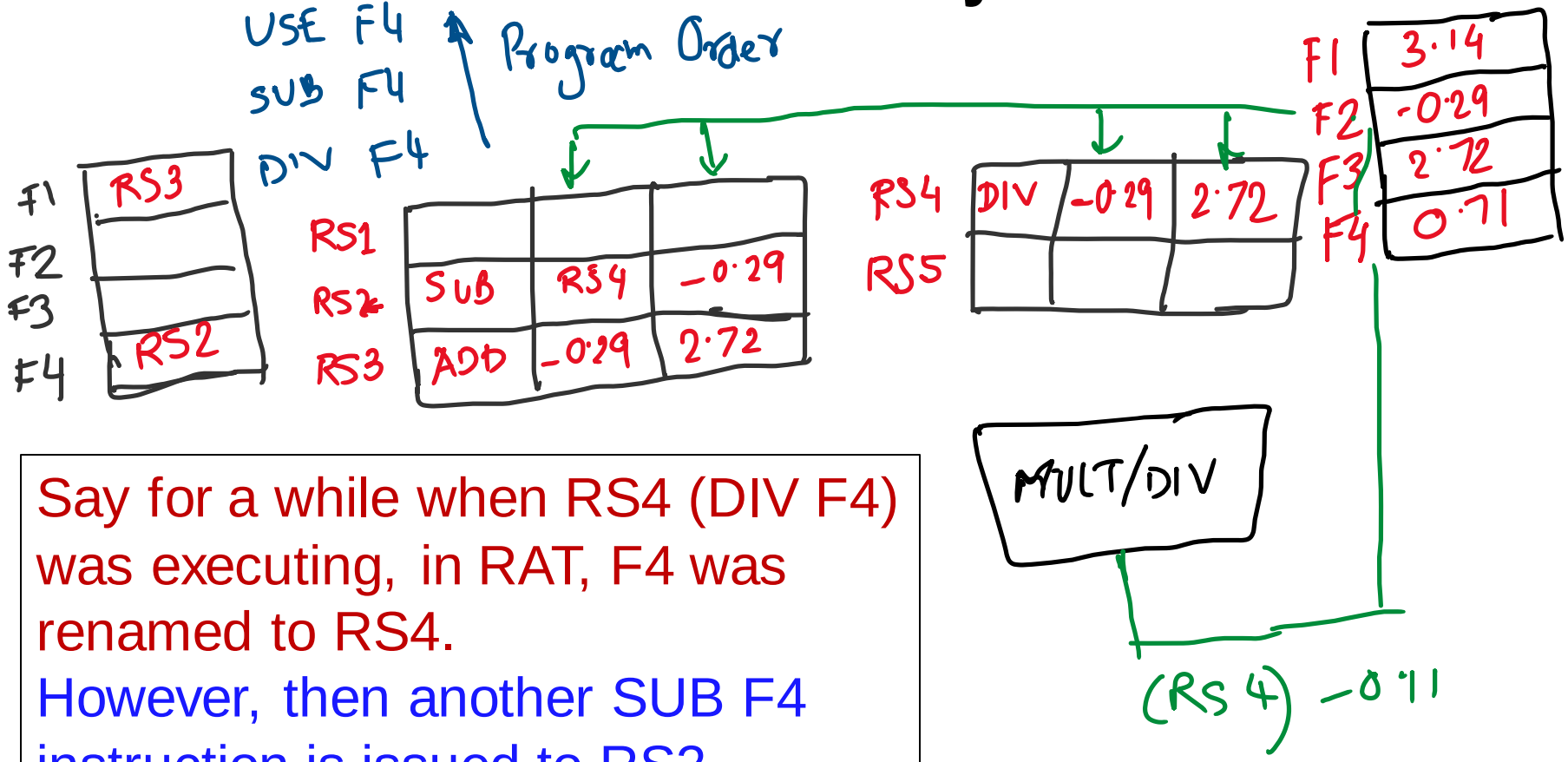
Give priority to slower units. Like Div/Mult, as they being slower it is expected more instructions are waiting for their results.

Broadcast TAG has no entry in RAT



In RAT, none of the registers are renamed to RS4! How can this happen?

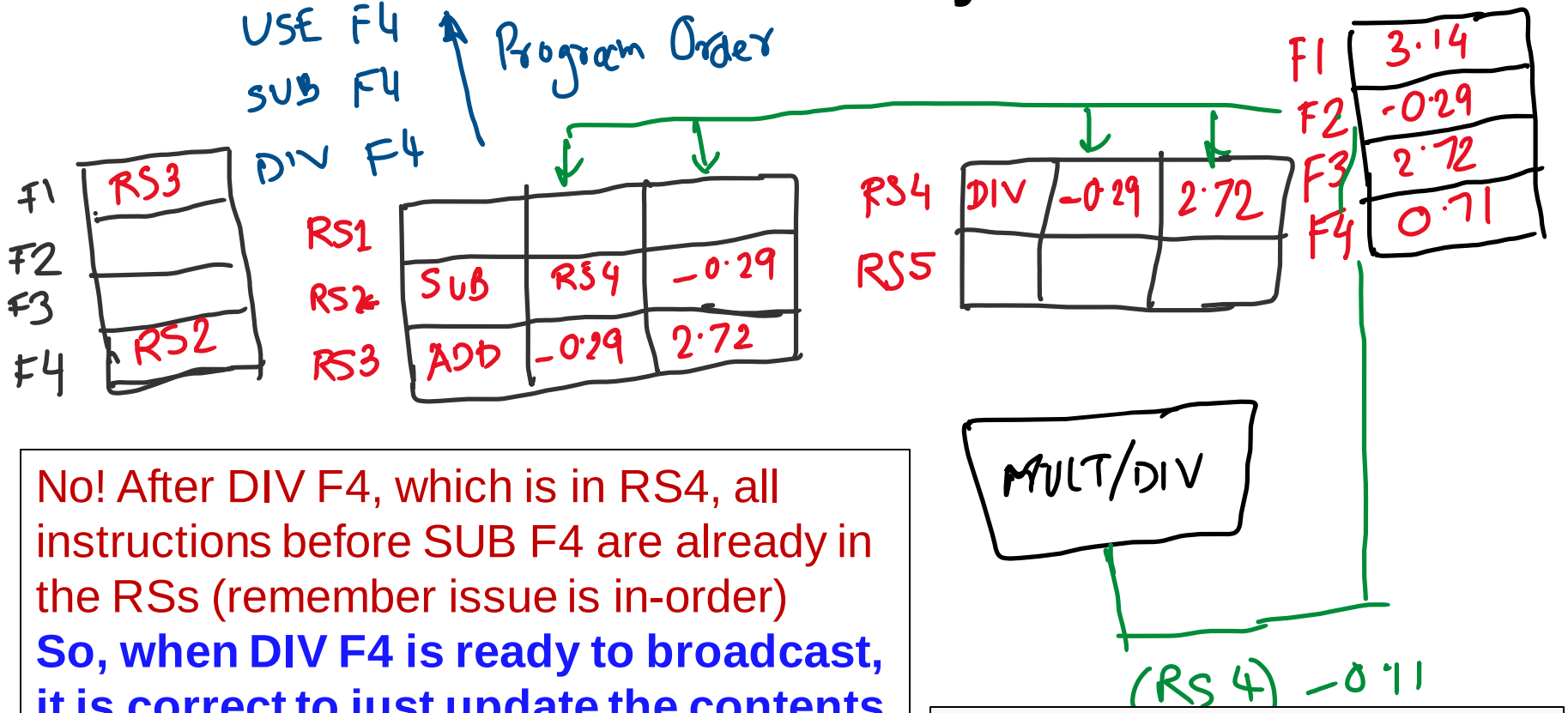
Broadcast TAG has no entry in RAT



Say for a while when RS4 (DIV F4) was executing, in RAT, F4 was renamed to RS4.
However, then another SUB F4 instruction is issued to RS2.
Thus, in RAT F4 gets renamed to RS2!

Is this a problem?

Broadcast TAG has no entry in RAT



No! After DIV F4, which is in RS4, all instructions before SUB F4 are already in the RSs (remember issue is in-order)

So, when DIV F4 is ready to broadcast, it is correct to just update the contents of the RSs.

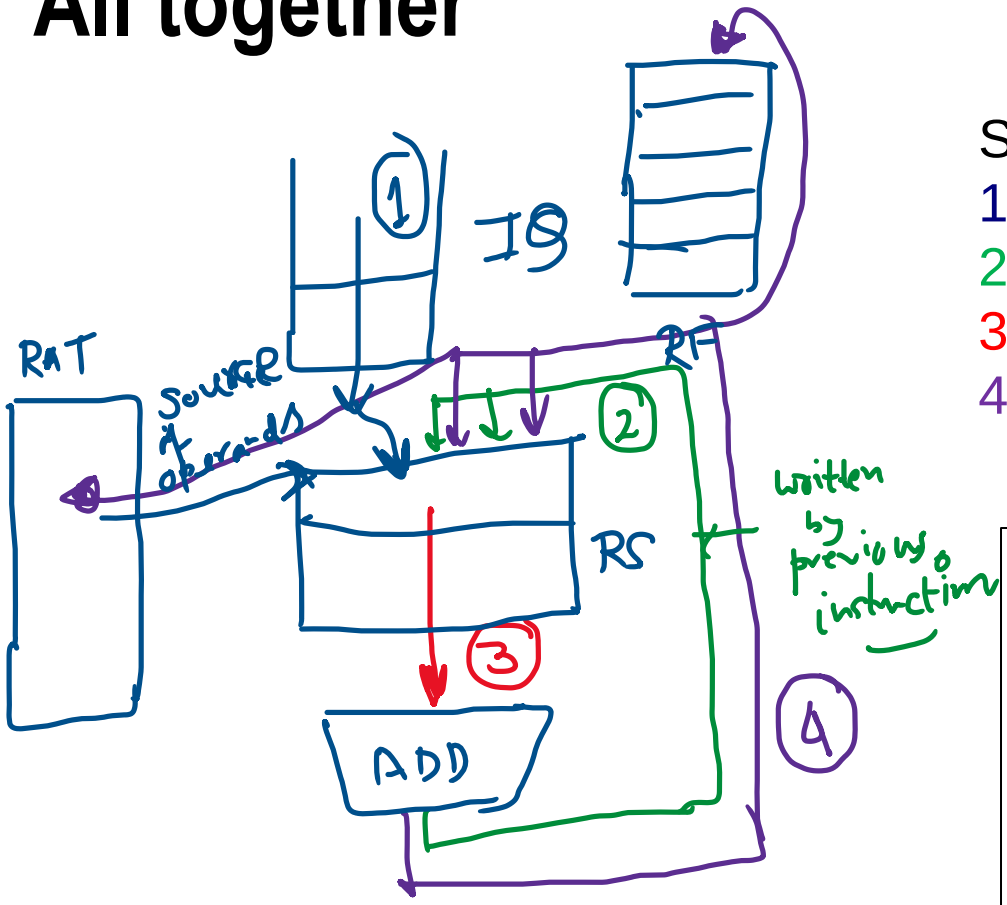
The RAT and RF should not be deleted or updated, as they should now contain the output of SUB F4, which is produced in RS2.

Summary:

Update the RS4 parameters in the RS.

However, if the entries in RAT do not match the tag RS4, do not update RAT & RF.

All together

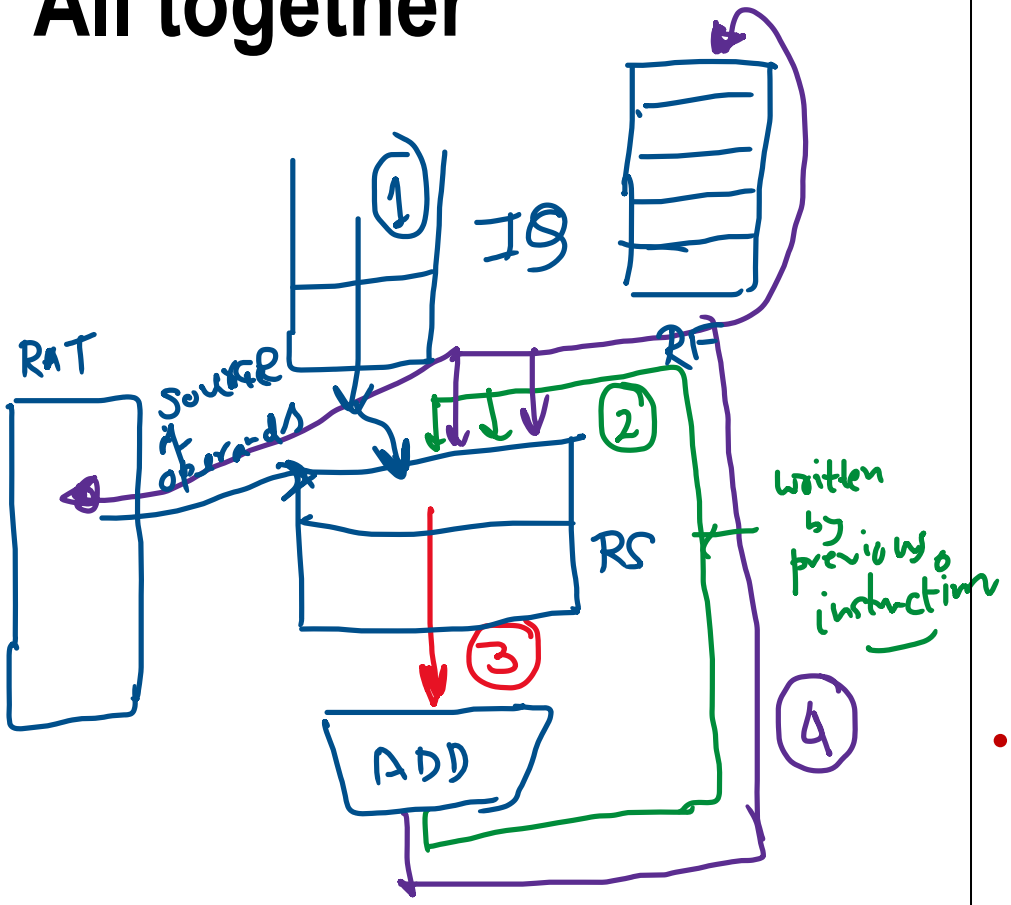


Steps:

1. Issue
2. Capture
3. Dispatch
4. Write Result

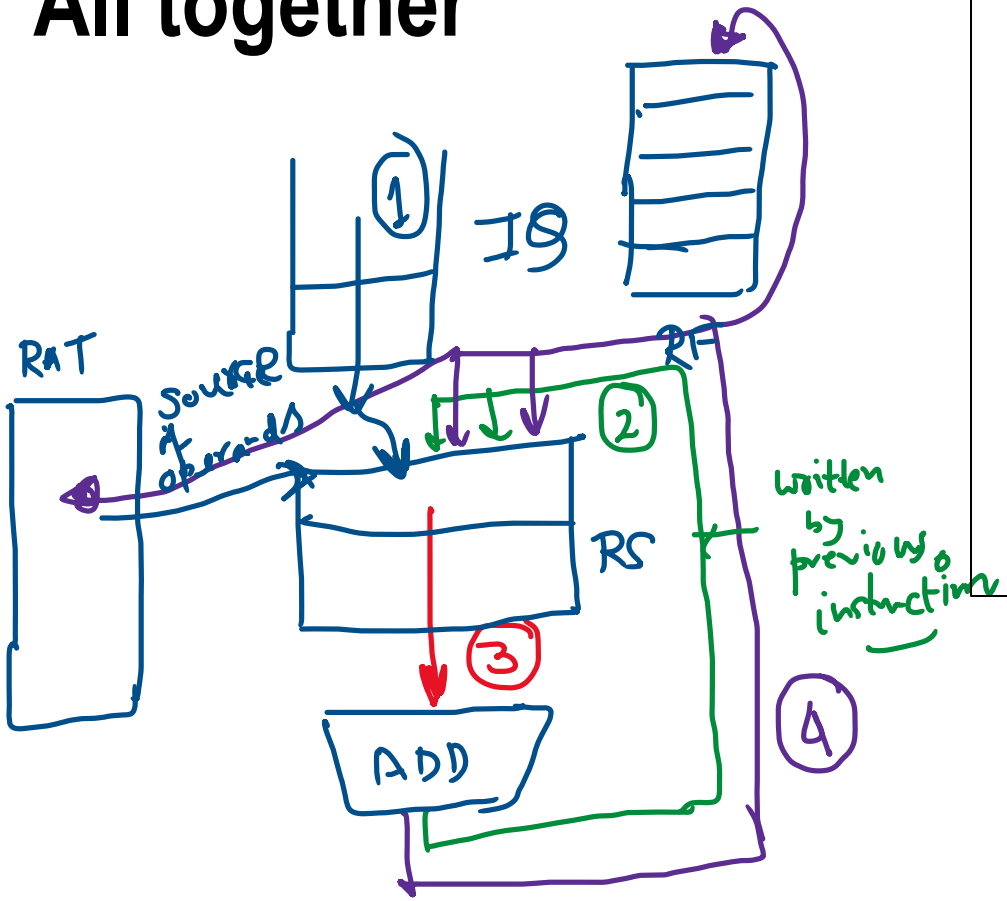
It is important to note that all of these steps are happening concurrently: **Some instructions are in issue, some in capture, some in dispatch, some in write results, and so on.**

All together



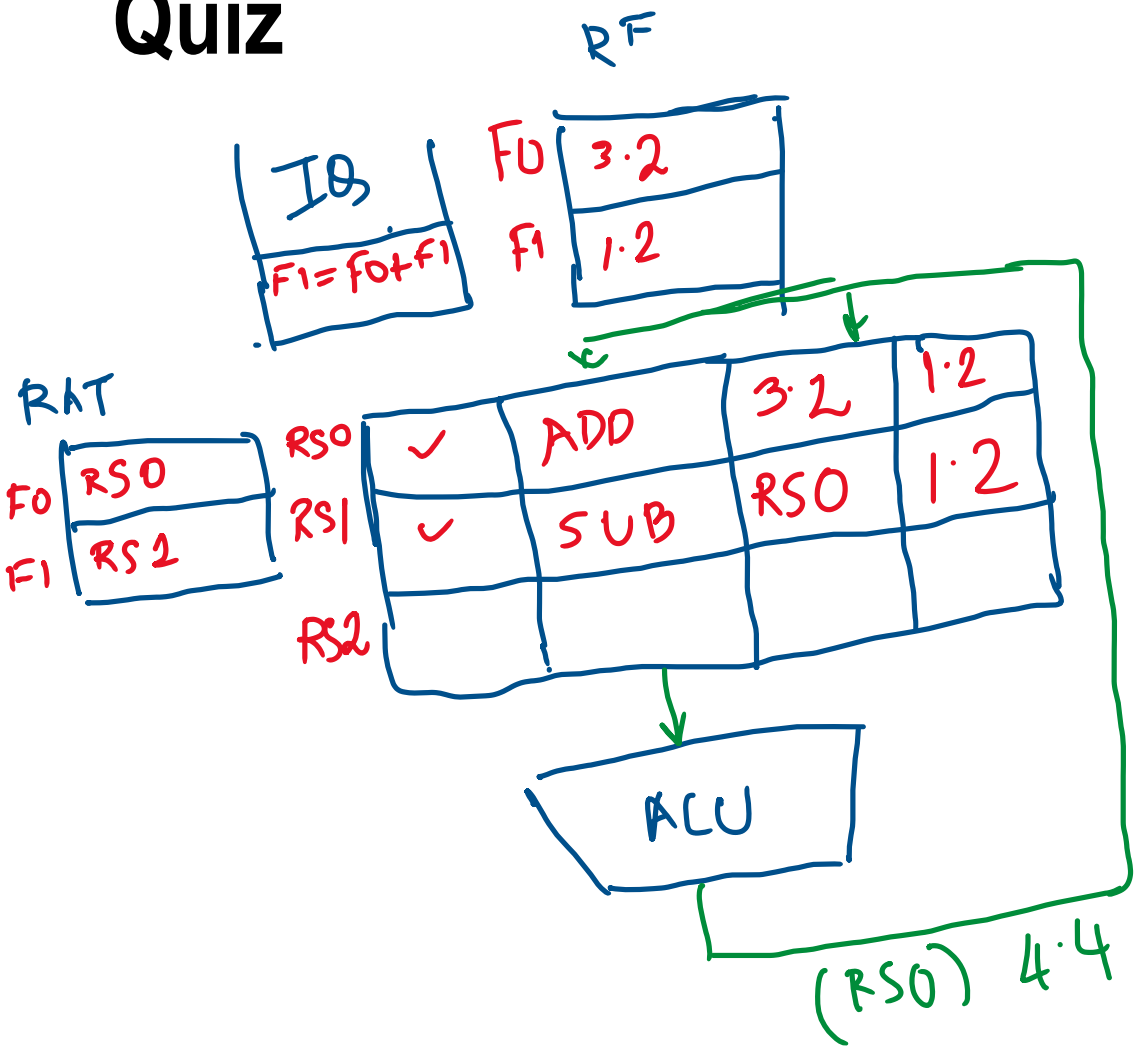
- Can we do same cycle Issue, and Dispatch of an instruction if it is not waiting to capture any operand value?
  - During issue we are writing into the RS. But for dispatch, we need to test the RSs. So we usually allow dispatch for the same instruction in the next clock cycle.
- Can we do same cycle capture to dispatch?
  - The RS updates it status from operands missing to operands available in capture. In next cycle, it is ready for dispatch.

All together



- Can we update entry for RAT for issue and WR result in same cycle?
  - Yes
  - If both refer to the same register, we write the one for issue and not the one for Write Result (broadcast)
  - Why?

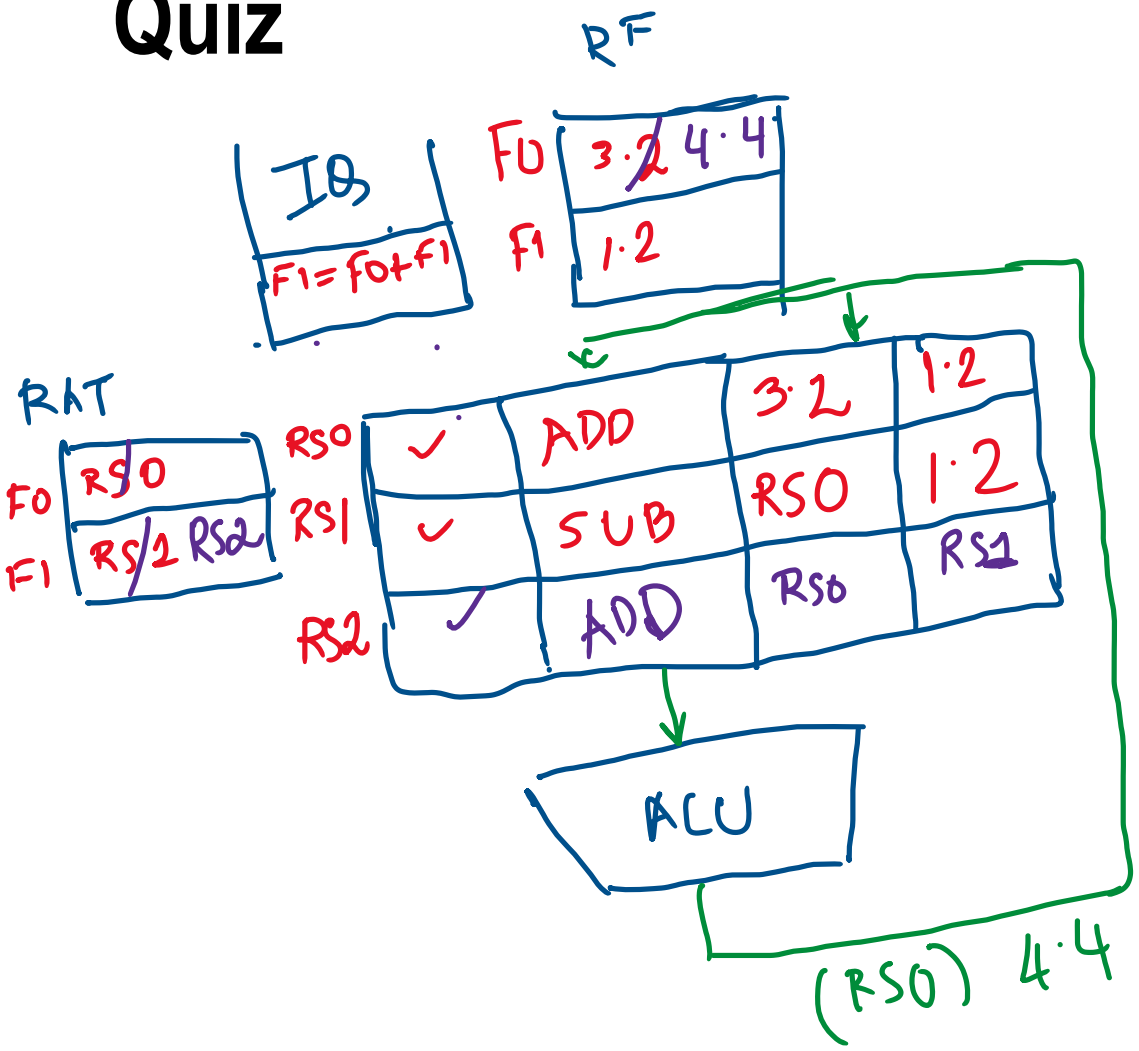
Quiz



Same cycle:
Issue → Dispatch: No
Capture →Dispatch: Yes
Issue→Broadcast: Yes

At the end of the cycle what is the content of RAT and RF?

Quiz

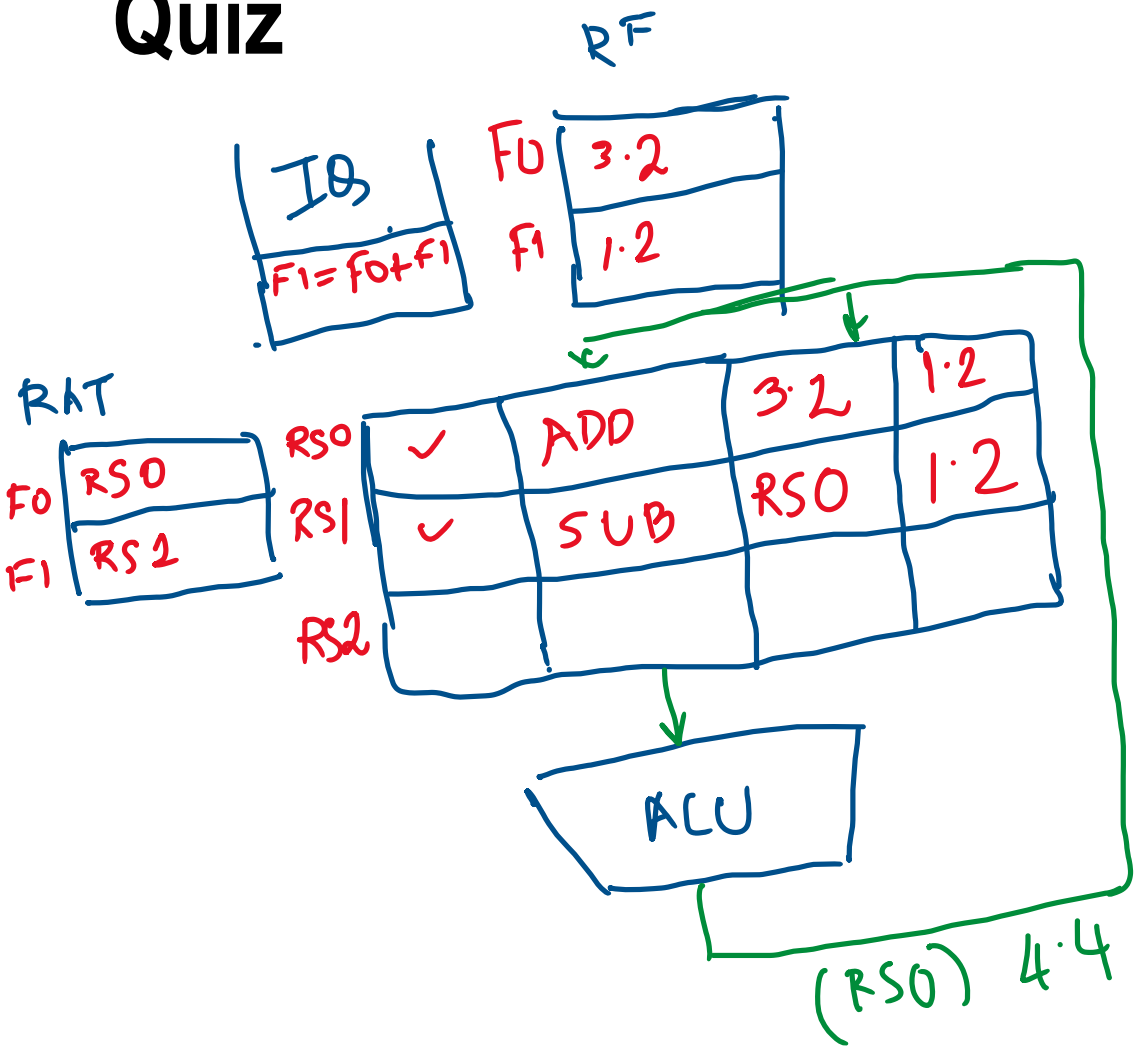


Same cycle:
Issue → Dispatch: No
Capture → Dispatch: Yes
Issue → Broadcast: Yes

At the end of the cycle what is the content of RAT and RF?

Both issue and broadcast attempts to write to RAT. During issue we don't write to RF.

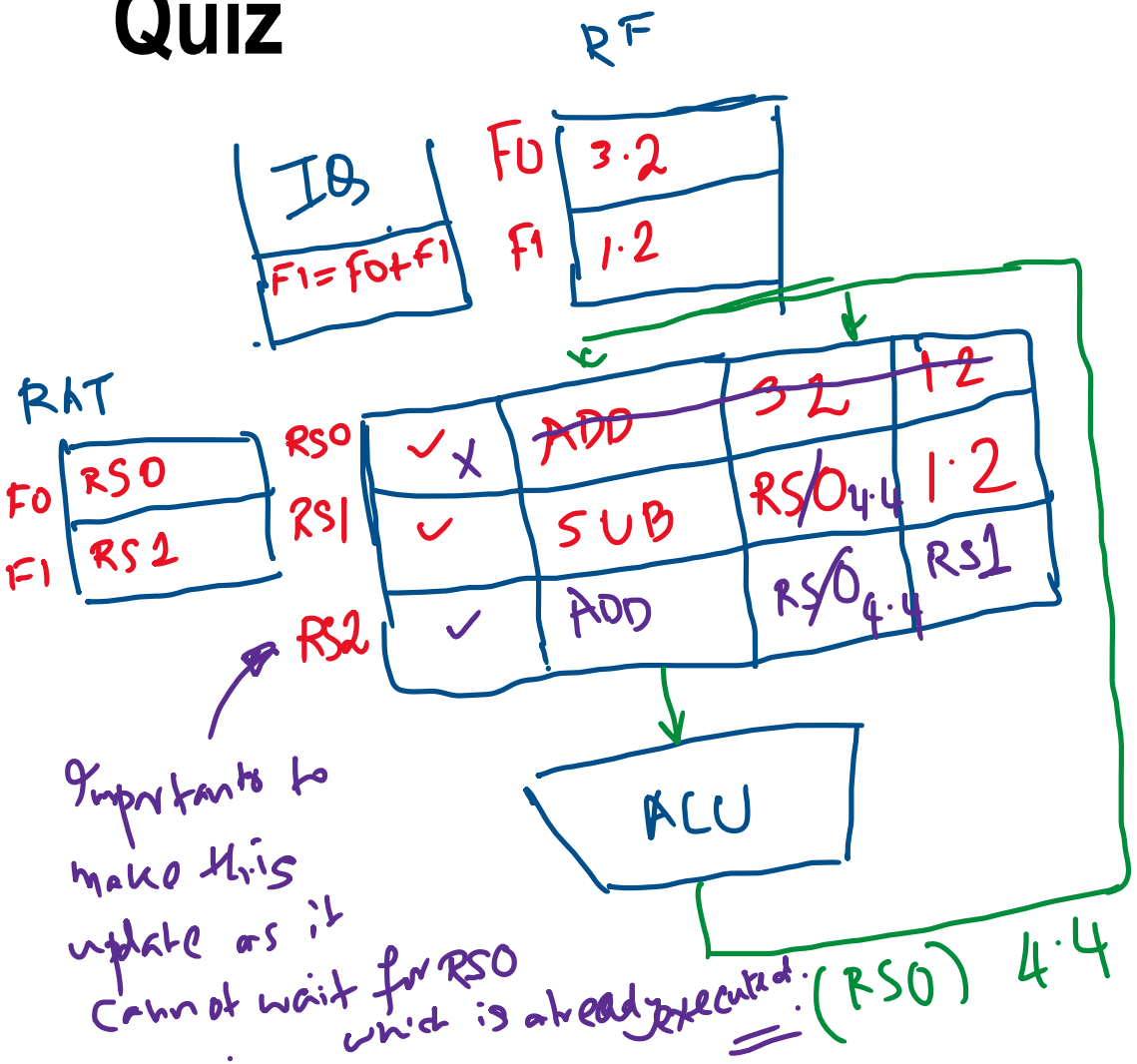
Quiz



Same cycle:
Issue → Dispatch: No
Capture →Dispatch: Yes
Issue→Broadcast: Yes

At the end of the cycle what is the content of the RS?

Quiz

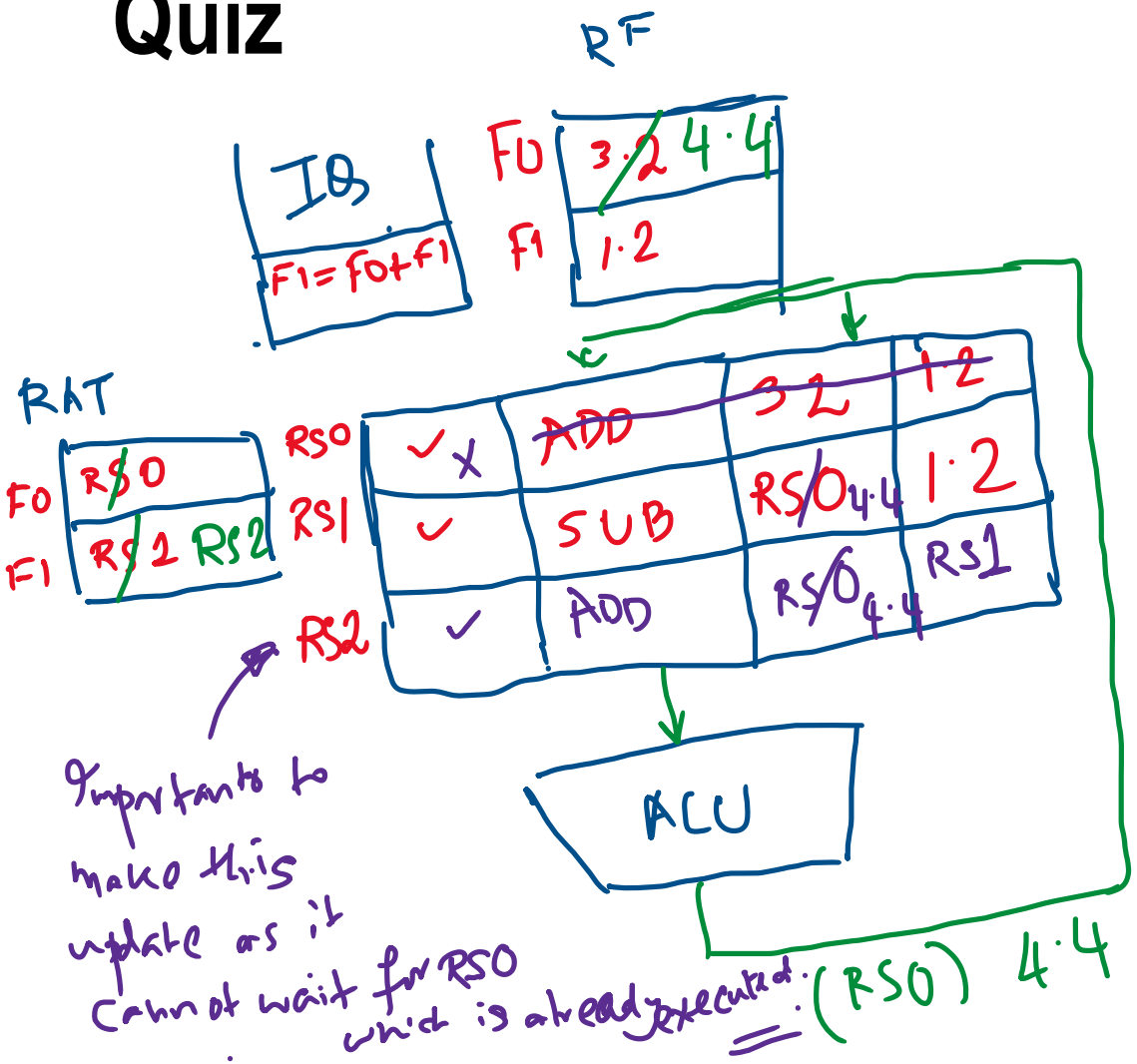


Same cycle:
Issue → Dispatch: No
Capture → Dispatch: Yes
Issue → Broadcast: Yes

At the end of the cycle what is the content of RS?

We need to consider issue & broadcast.

Quiz

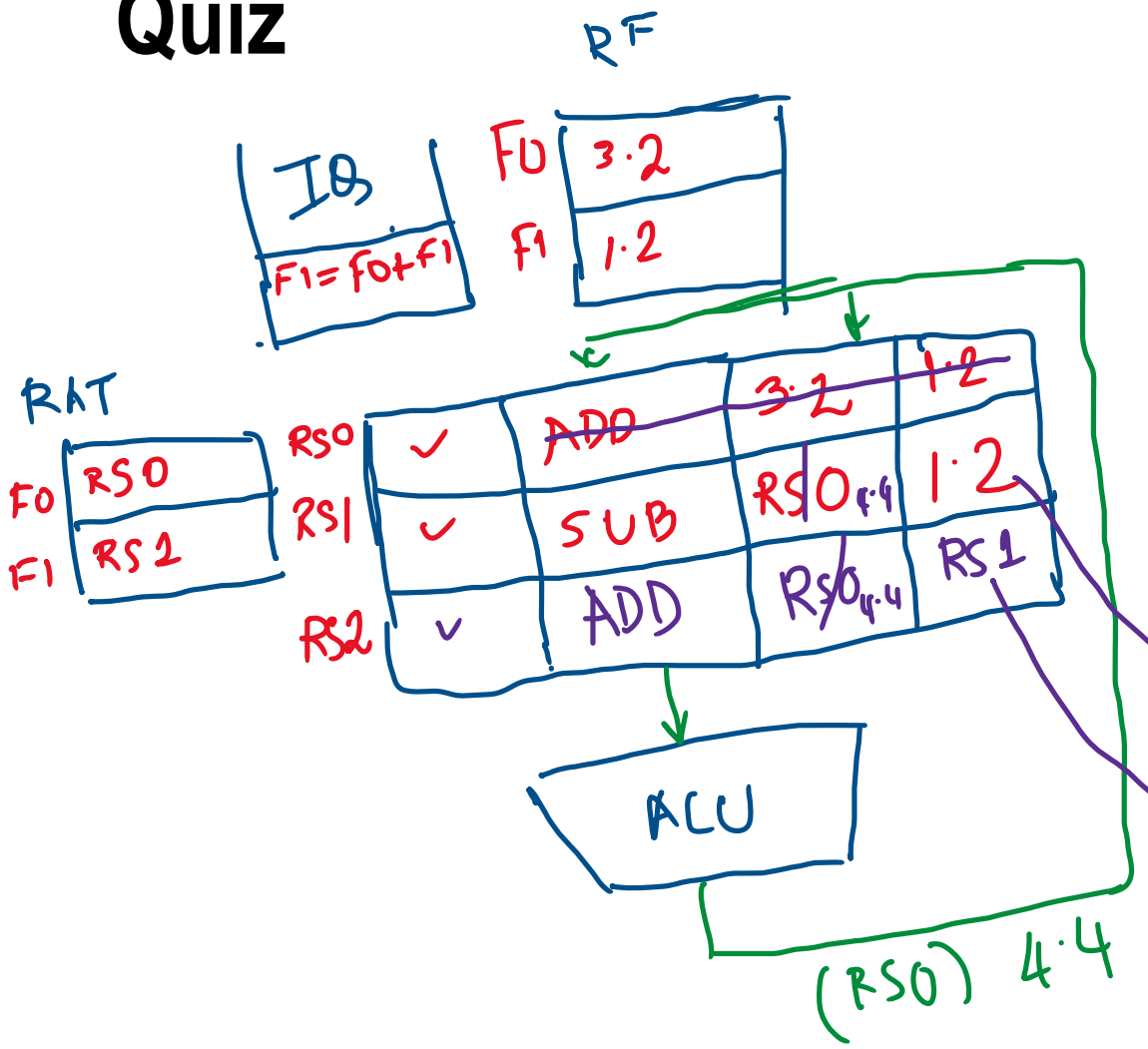


Same cycle:
Issue → Dispatch: No
Capture → Dispatch: Yes
Issue → Broadcast: Yes

At the end of the cycle what is the content of RS?

We need to consider issue & broadcast.

Quiz



Same cycle:
Issue → Dispatch: No
Capture → Dispatch: Yes
Issue → Broadcast: Yes

At the end of the cycle what will dispatch if any?

As we allow same cycle capture, dispatch this instruction is eligible.

This instruction is not eligible as same cycle issue, dispatch not allowed

Quiz

- In Tomasulo's algorithm which of the following are not correct:
  - It issues instructions in program order.
  - It dispatches instructions in program order.
  - It writes results in program order.

Quiz

- In Tomasulo's algorithm which of the following are not correct:
  - ☐ It issues instructions in program order.
  - ☒ It dispatches instructions in program order.
  - ☒ It writes results in program order.

Thank You!