HARDWARE SPECULATION

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

THE UNIVERSITY OF UTAH

CS/ECE 6810: Computer Architecture

Overview

- Announcement
  - Homework 3 is due tonight (11:59PM)
  - Midterm exam: Oct. 14th (right after Fall break)

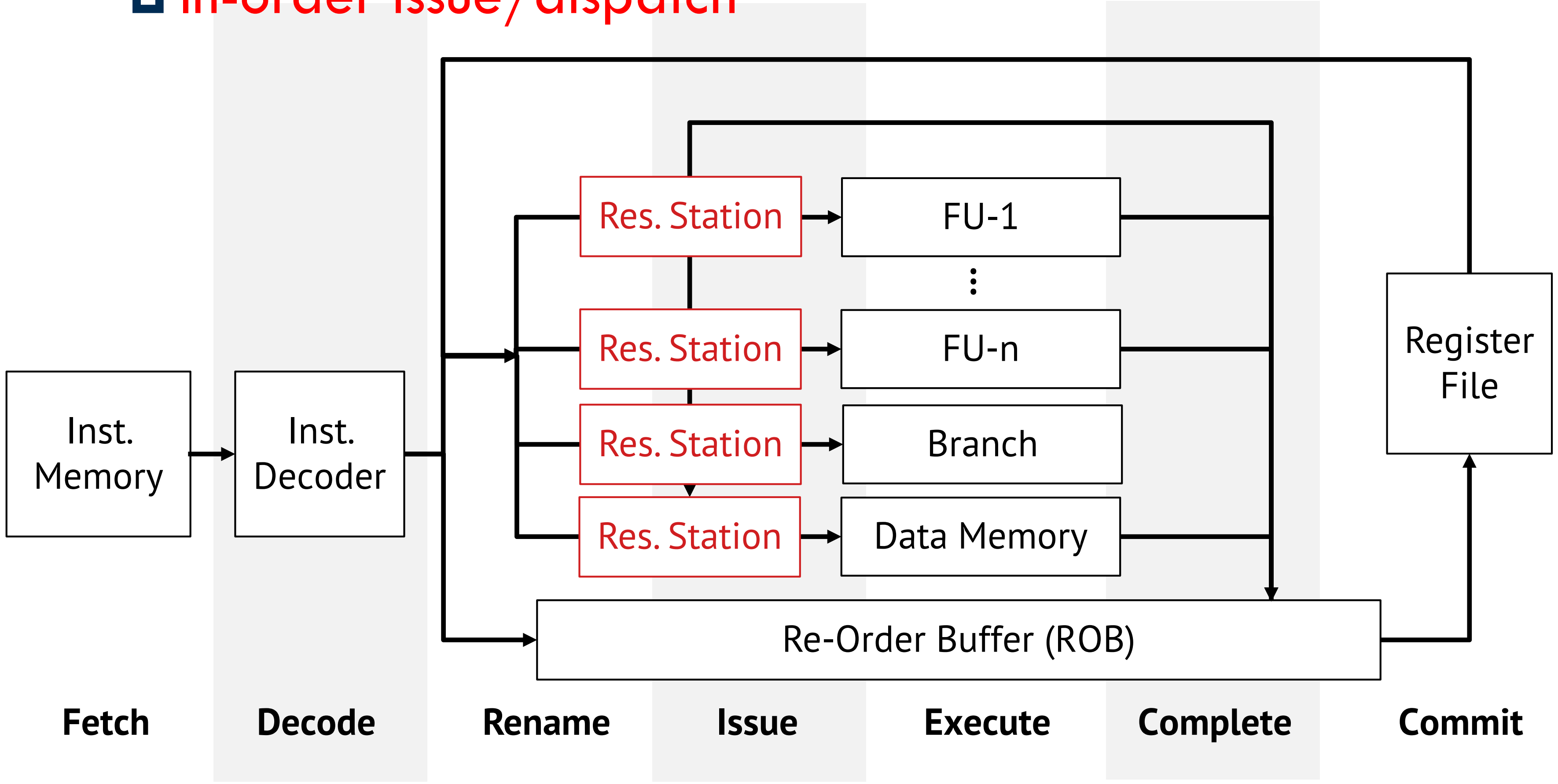
- This lecture
  - Out-of-order pipeline
    - Issue queue
    - Register renaming
    - Branch recovery
    - Speculated execution

Recall: Out-of-Order Execution

- Producer-consumer chains on the fly
  - Register renaming: remove anti-/output-dependences via register tags
  - Limited by the number of instructions in the instruction window (ROB)
- Out-of-order issue (dispatch)
  - Broadcast tags to waiting instructions
  - Wake up ready instructions and select among them
- Out-of-order execute/complete
- In-order fetch/decode and commit

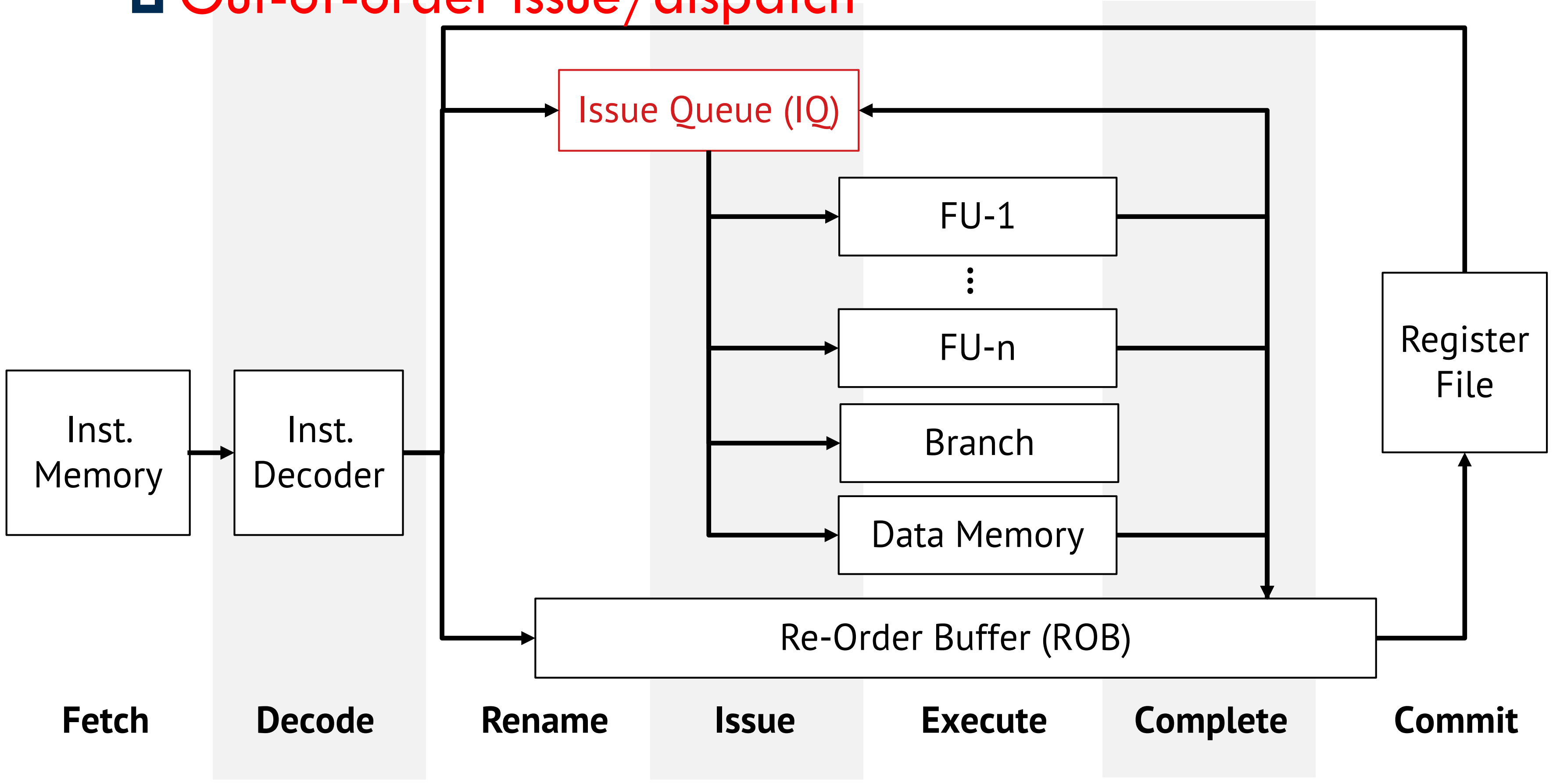
Out-of-Order Pipelines

- Distributed reservation stations
  - In-order issue/dispatch



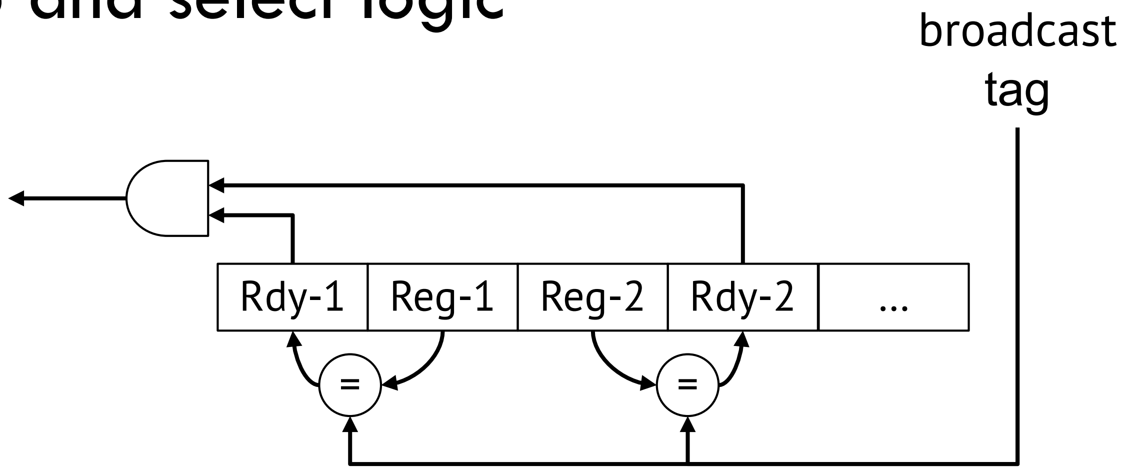
Out-of-Order Pipelines

- Out of order issue/dispatch to functional units
  - Out-of-order issue/dispatch



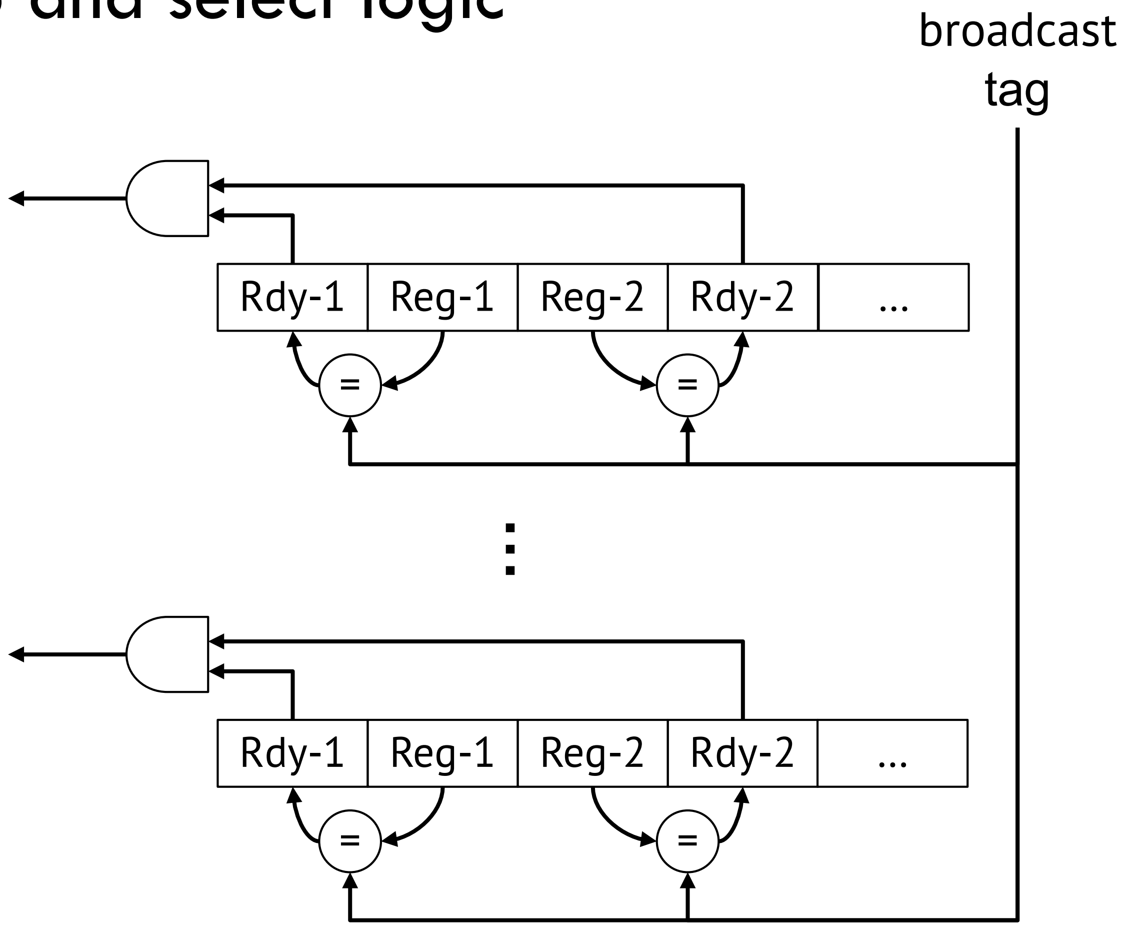
Out-of-Order Issue Queue

- Two step wakeup and select logic



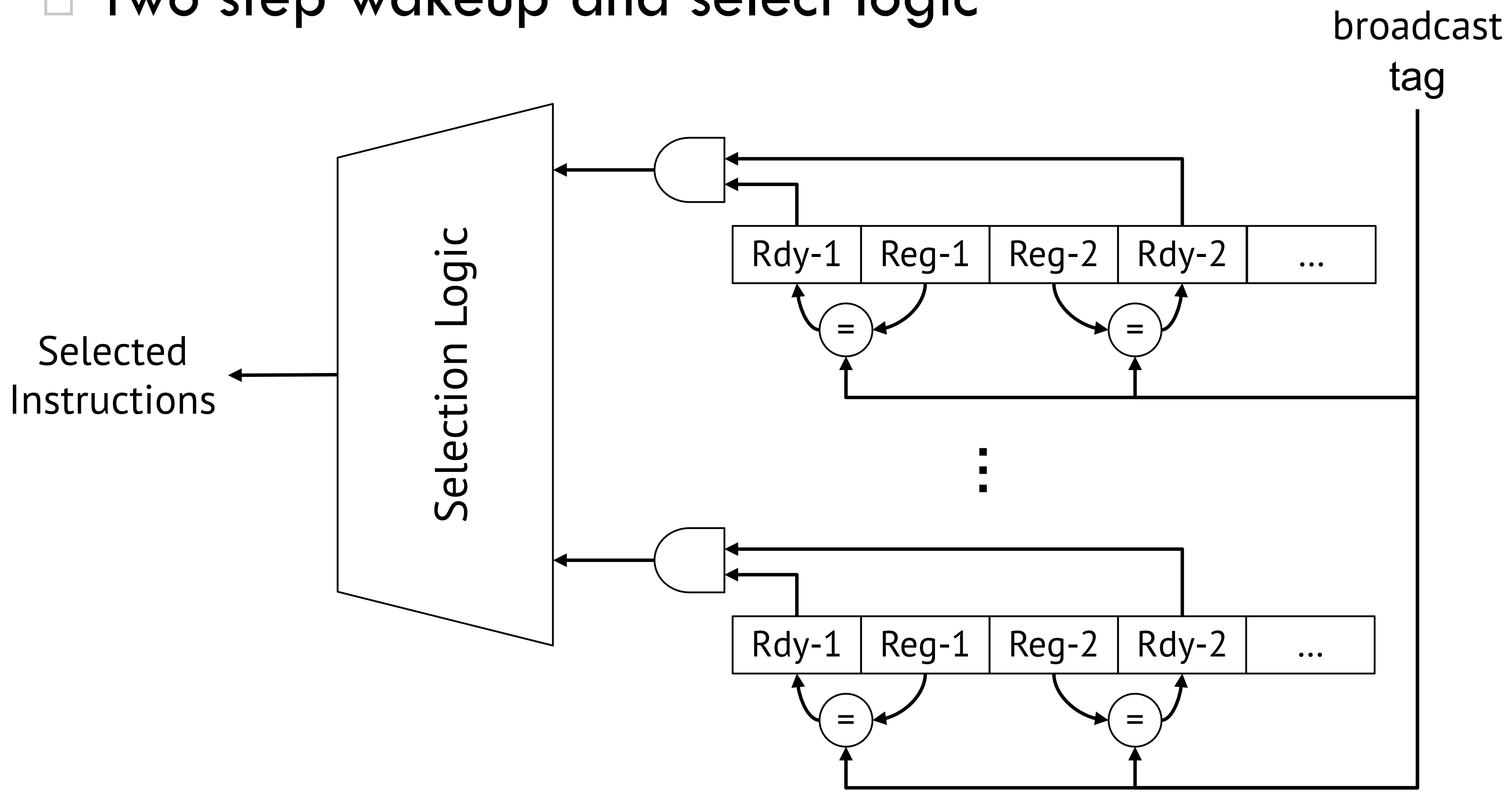
Out-of-Order Issue Queue

- Two step wakeup and select logic



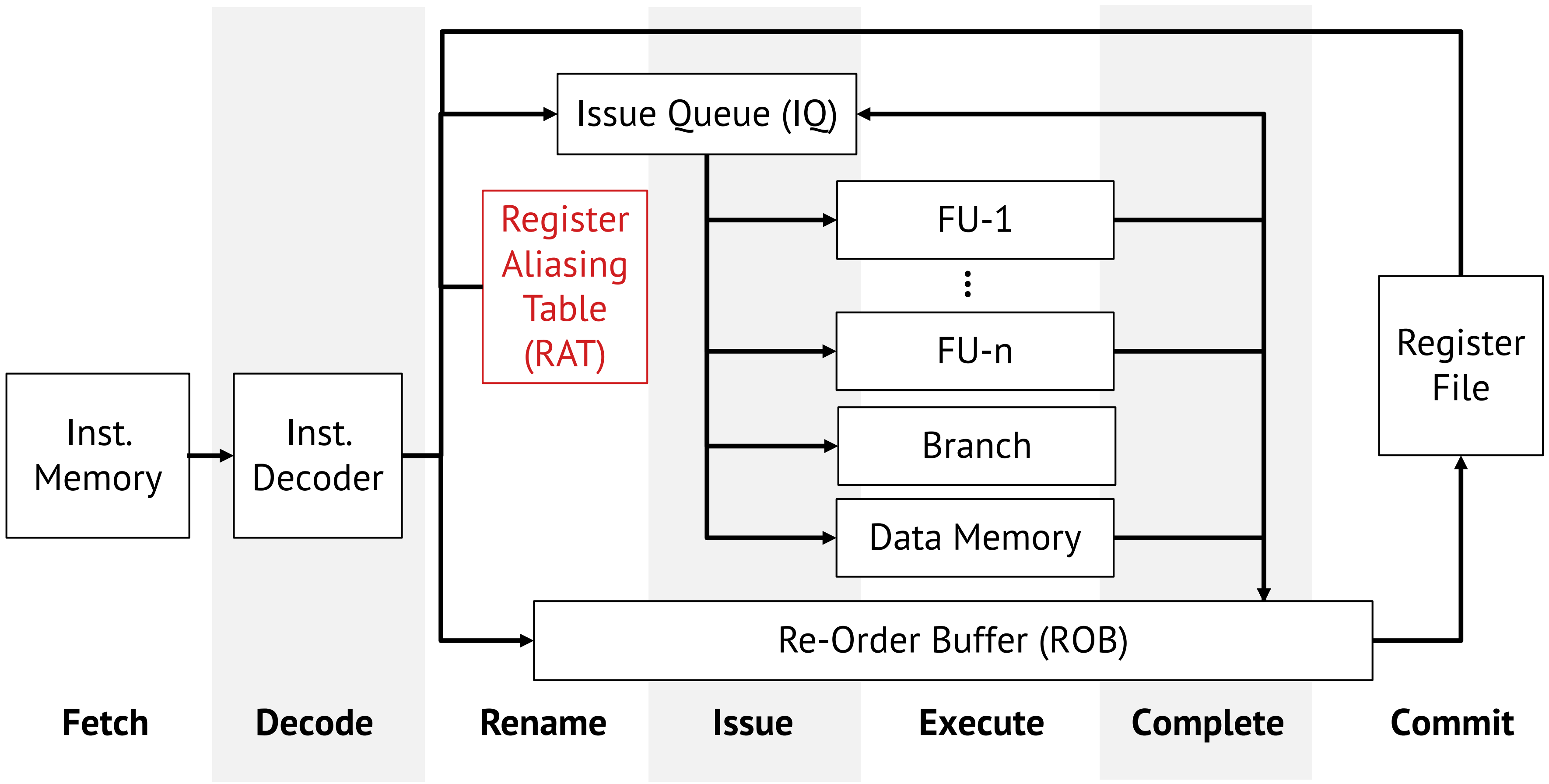
Out-of-Order Issue Queue

- Two step wakeup and select logic



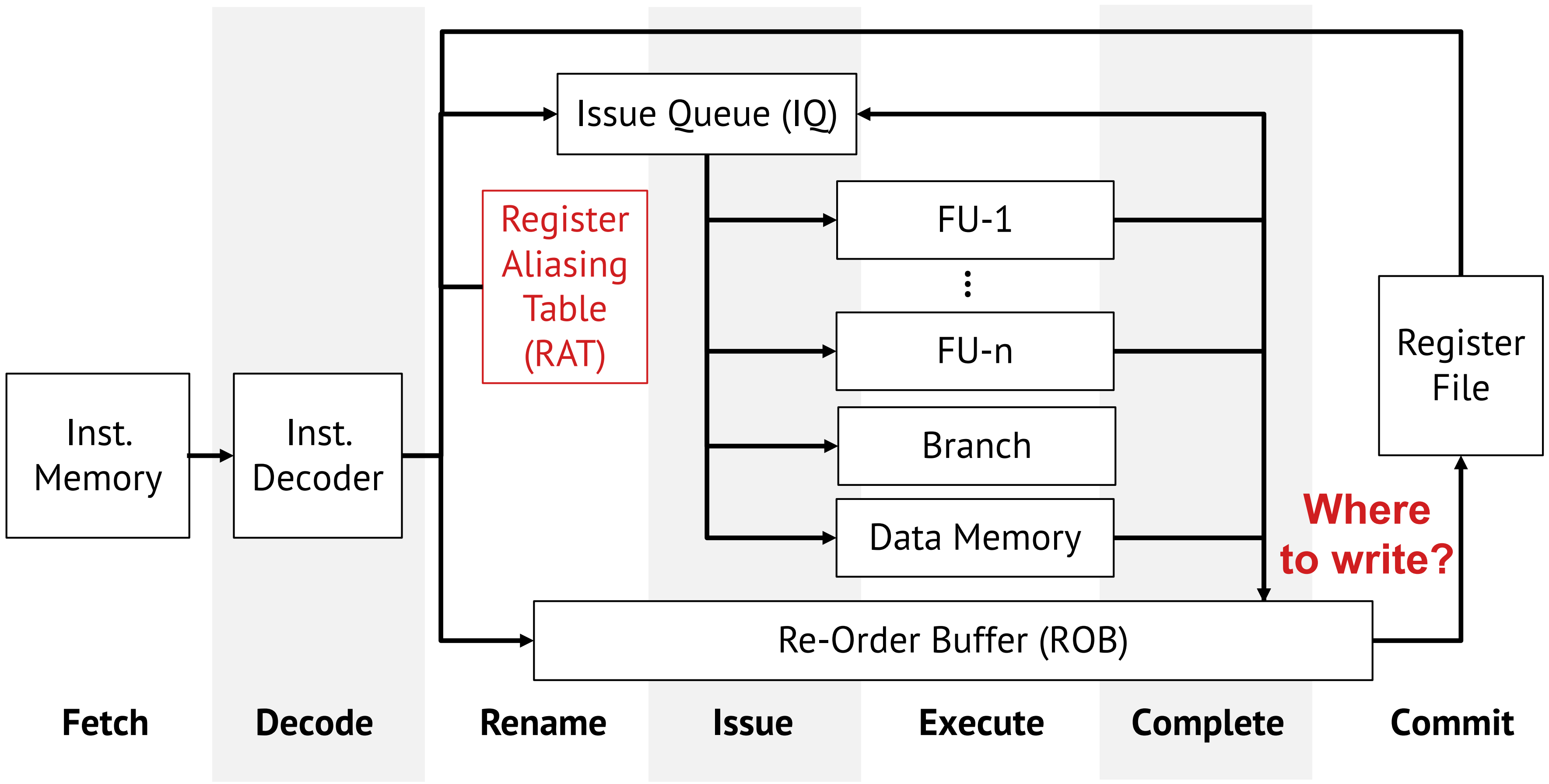
Register Renaming

- Register aliasing table for fast lookup



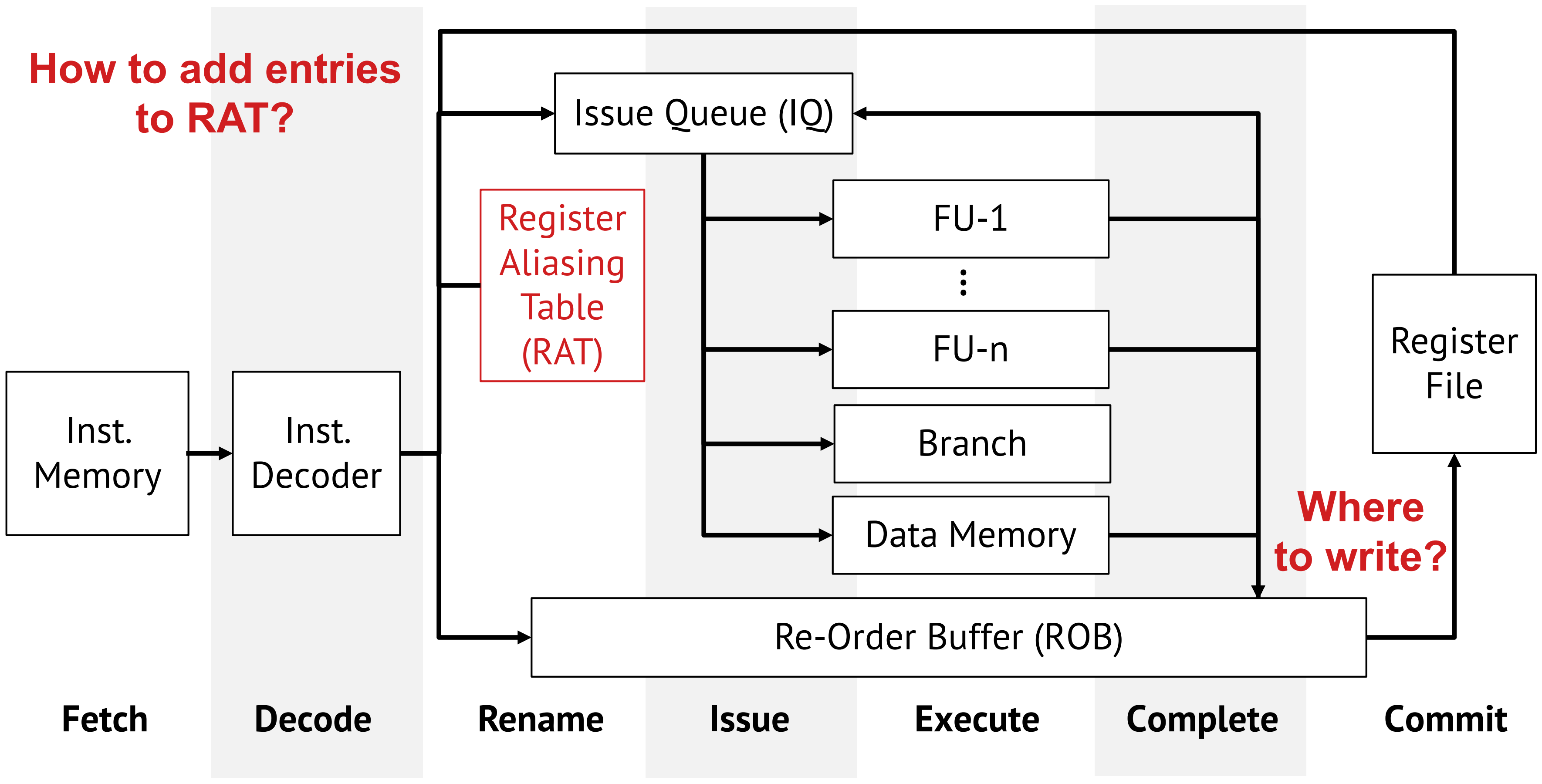
Register Renaming

- Register aliasing table for fast lookup



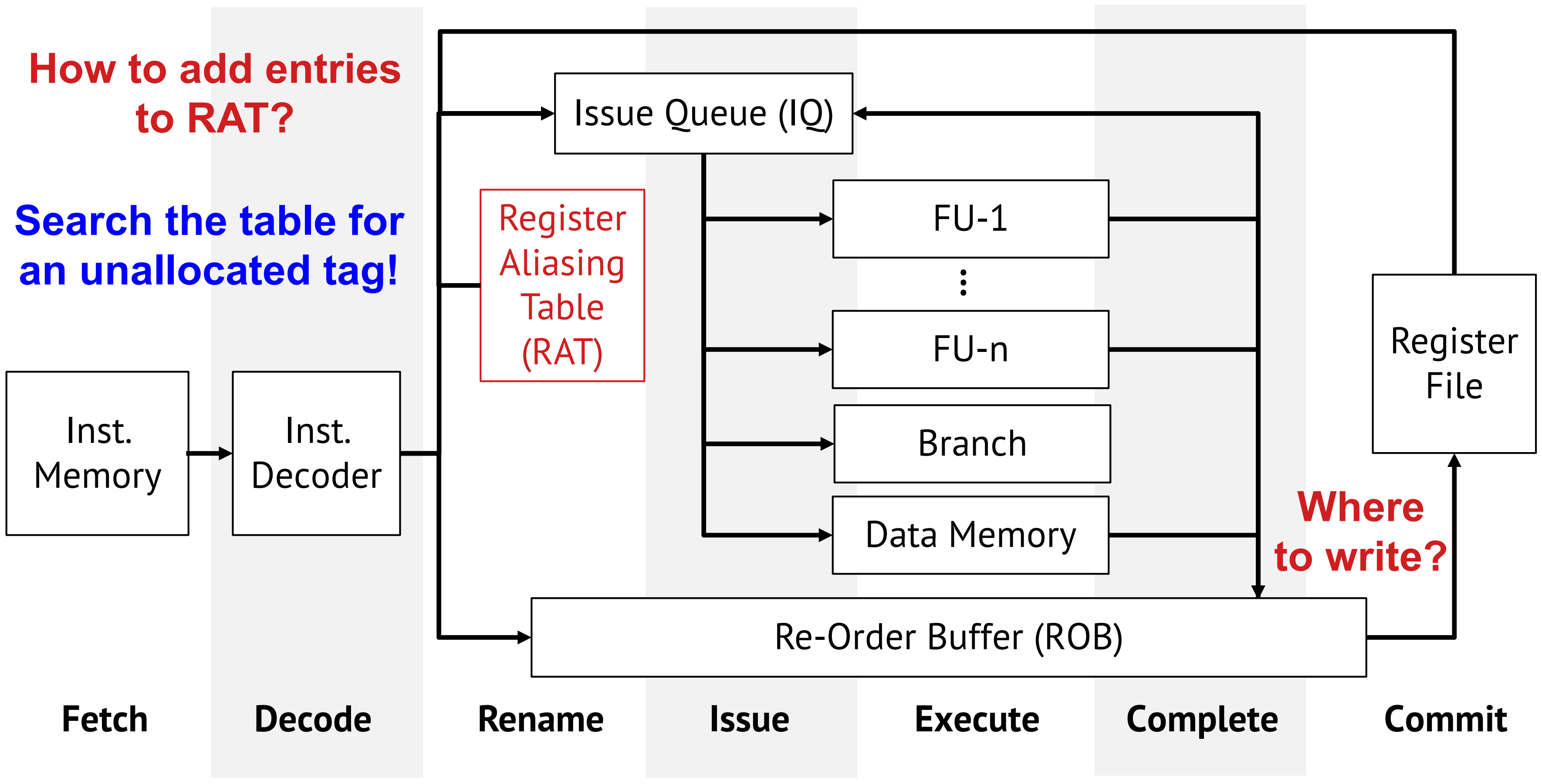
Register Renaming

- Register aliasing table for fast lookup



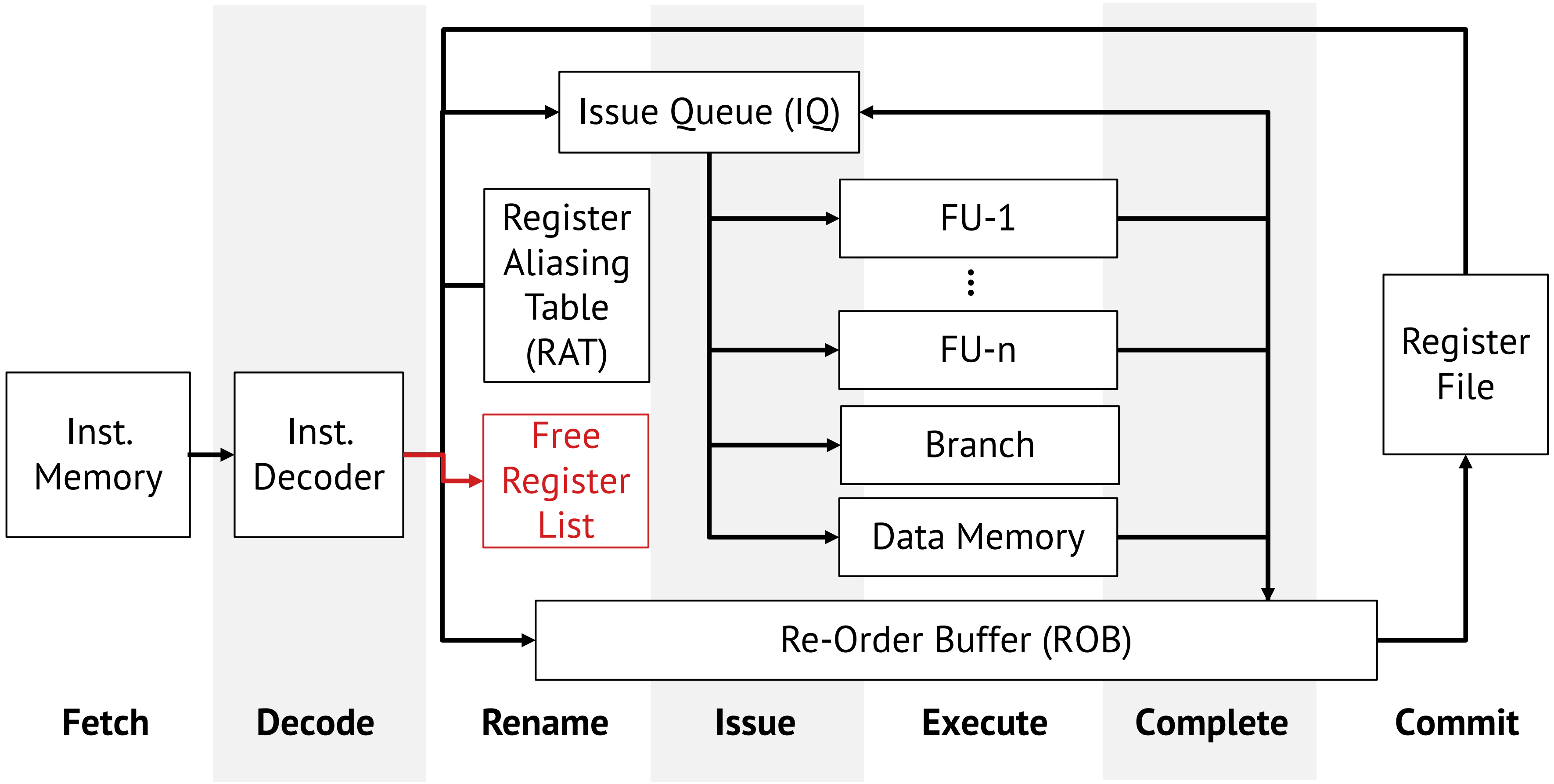
Register Renaming

- Register aliasing table for fast lookup



Register Renaming

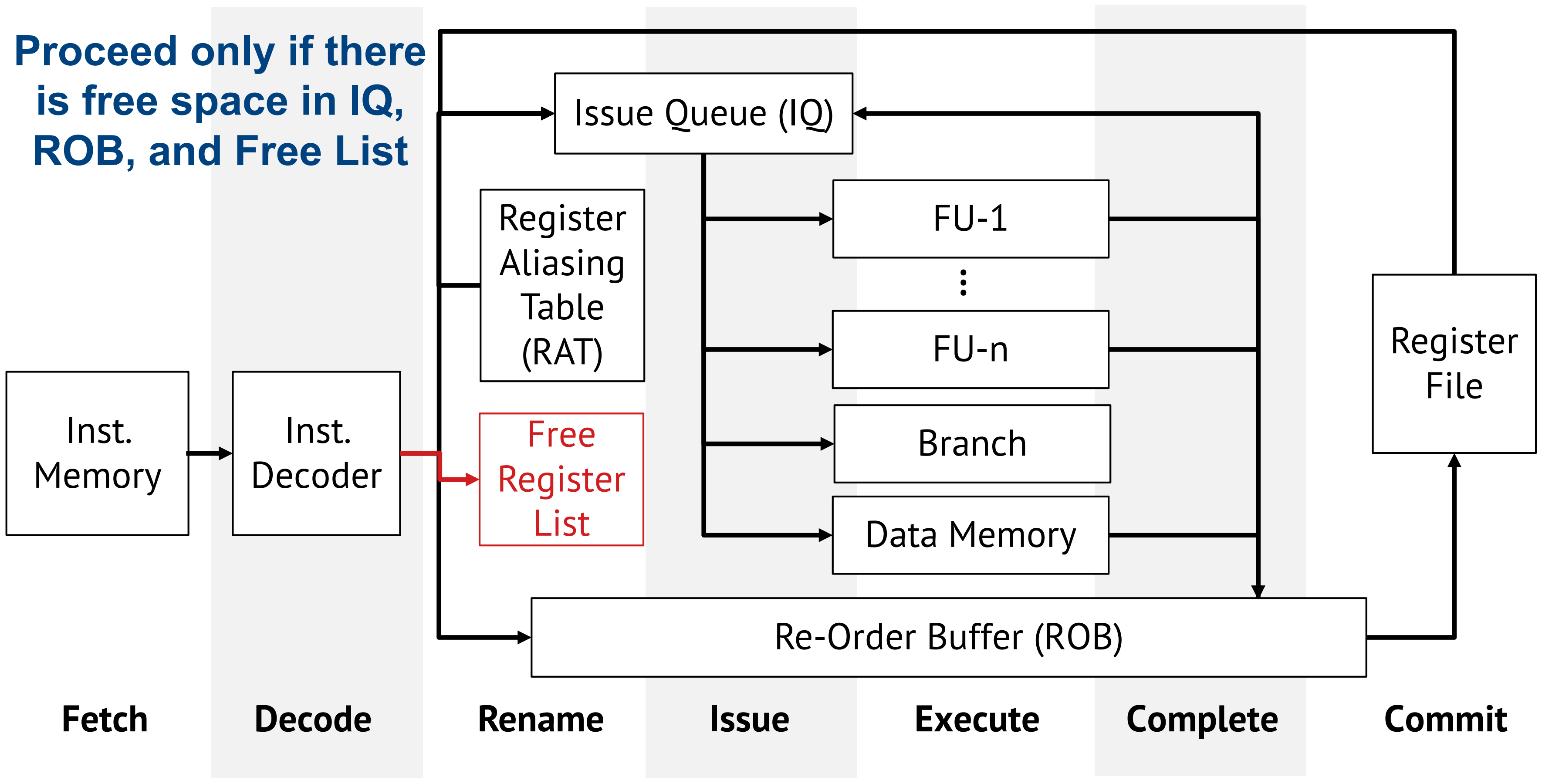
- Free register list for fast register renaming



Register Renaming

- Free register list for fast register renaming

Proceed only if there is free space in IQ, ROB, and Free List

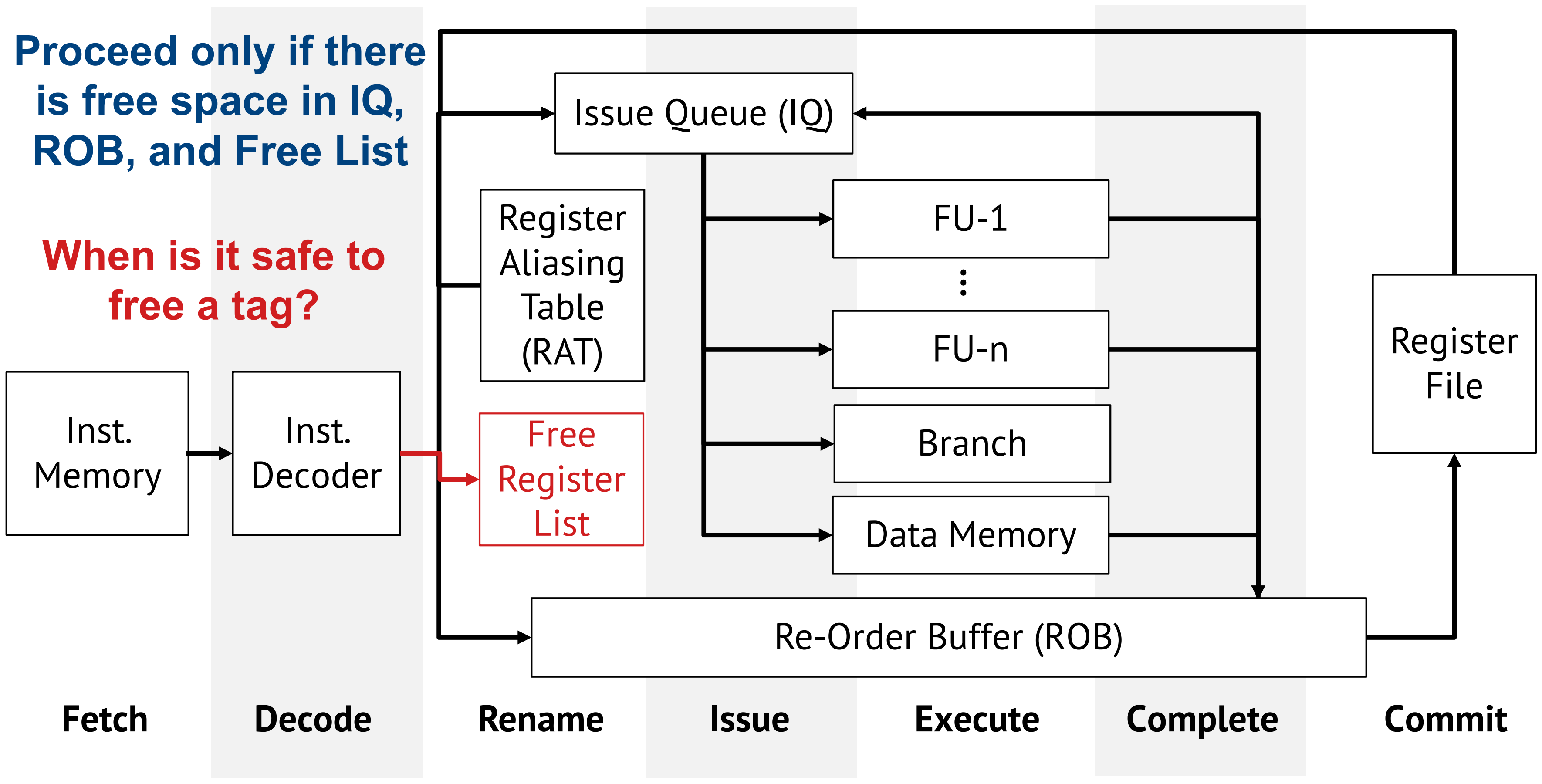


Register Renaming

- Free register list for fast register renaming

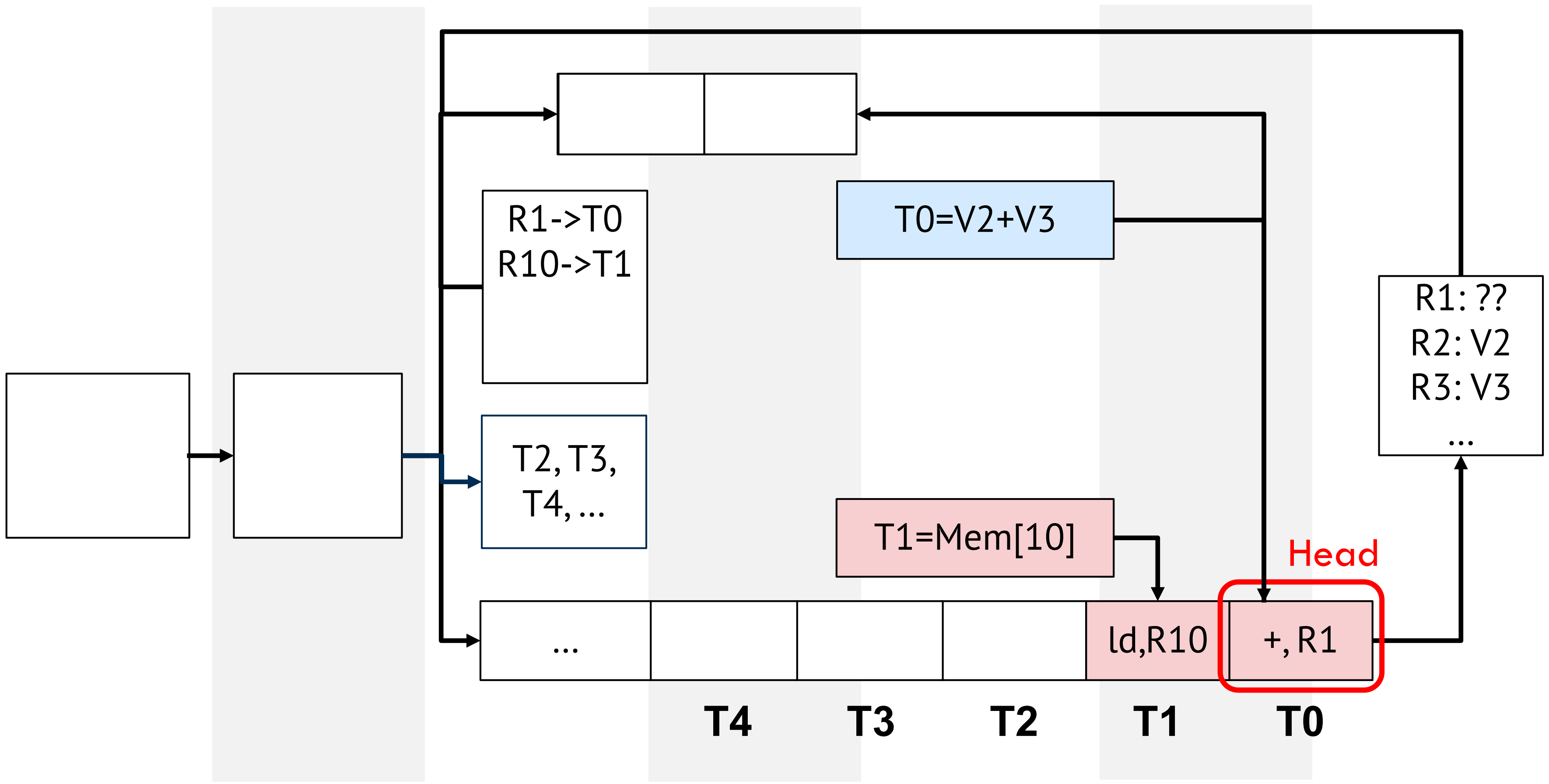
Proceed only if there is free space in IQ, ROB, and Free List

When is it safe to free a tag?



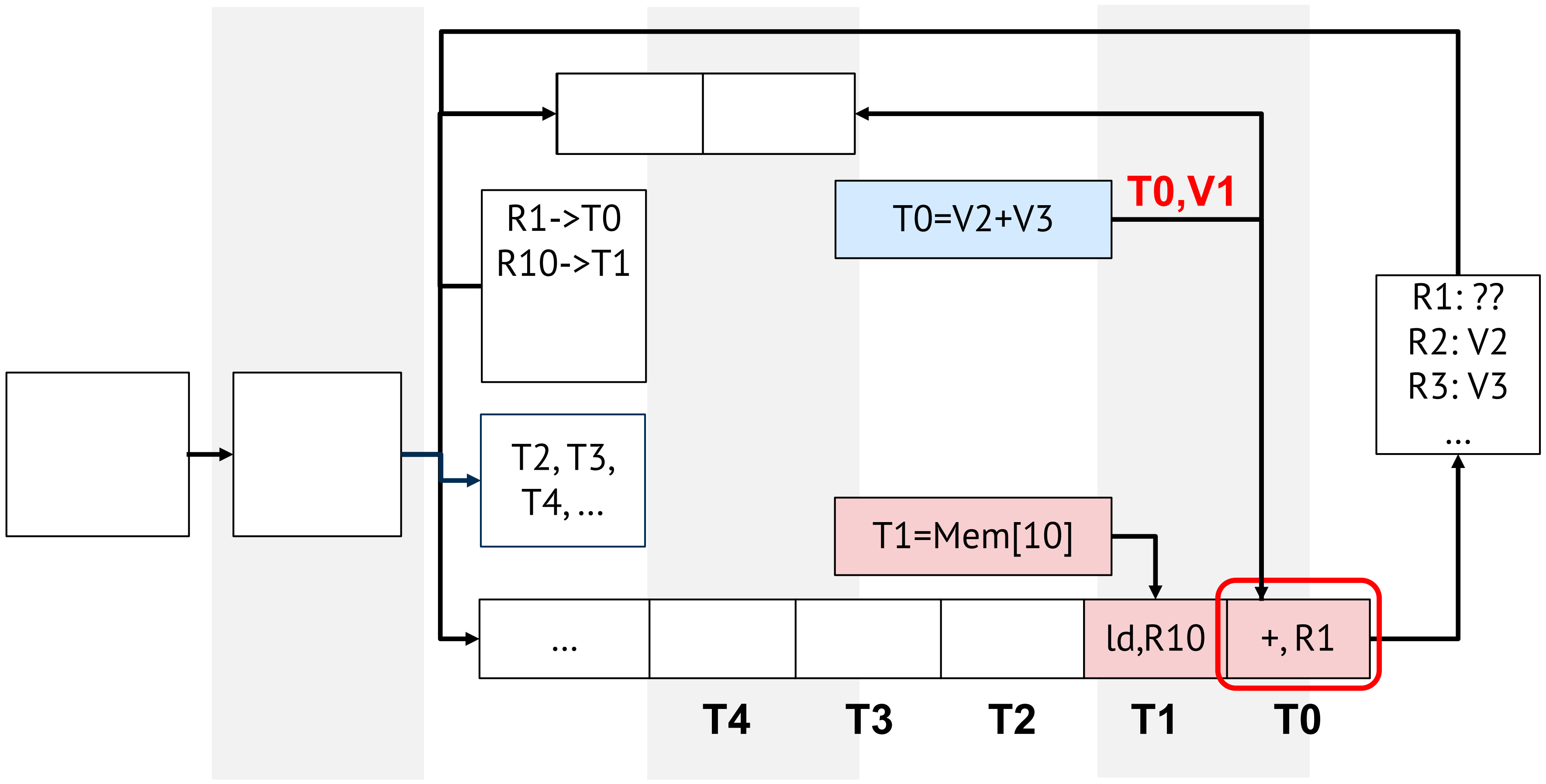
Example: Instruction Commit

- Update value in ROB



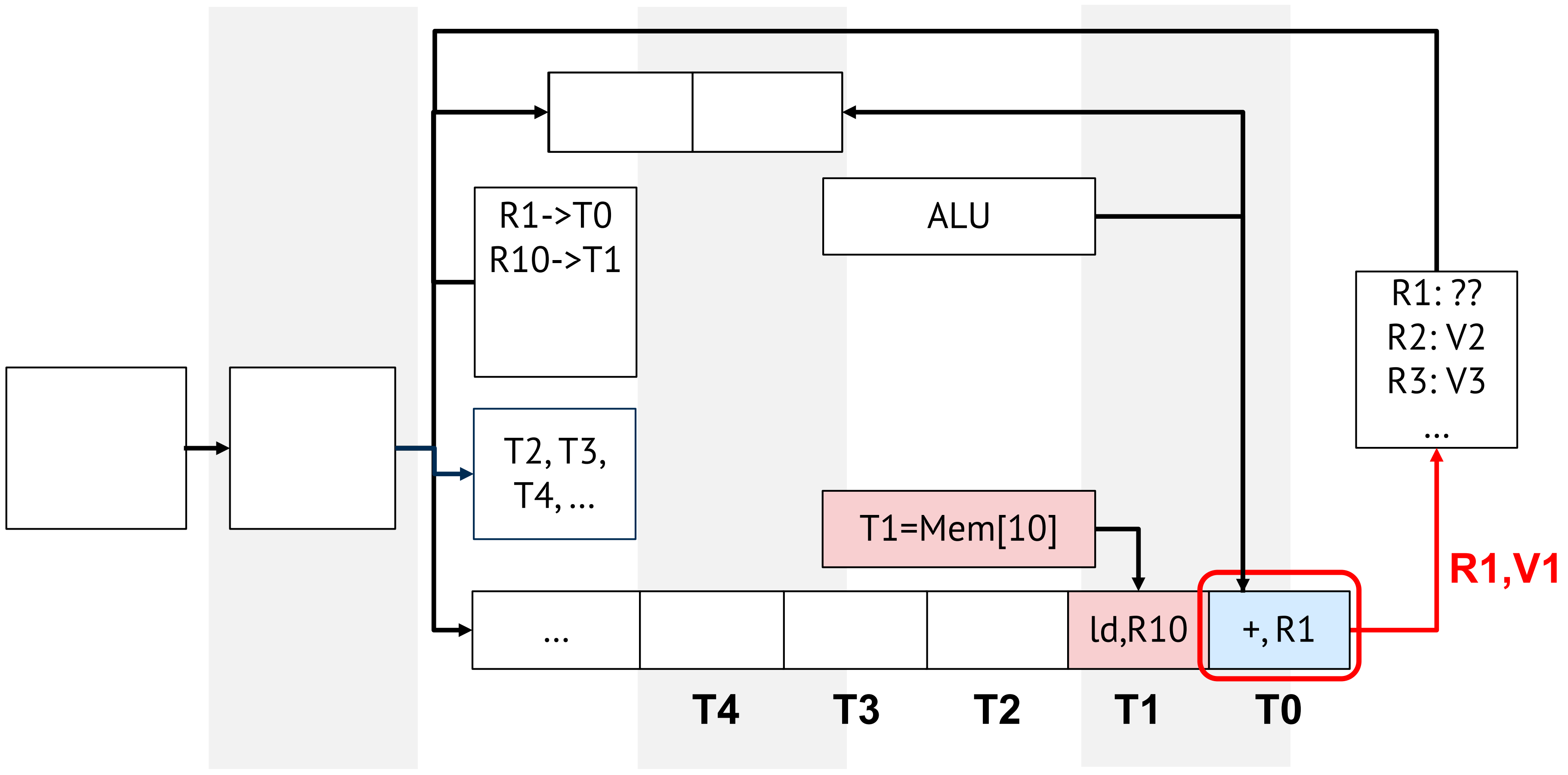
Example: Instruction Commit

- Update value in ROB



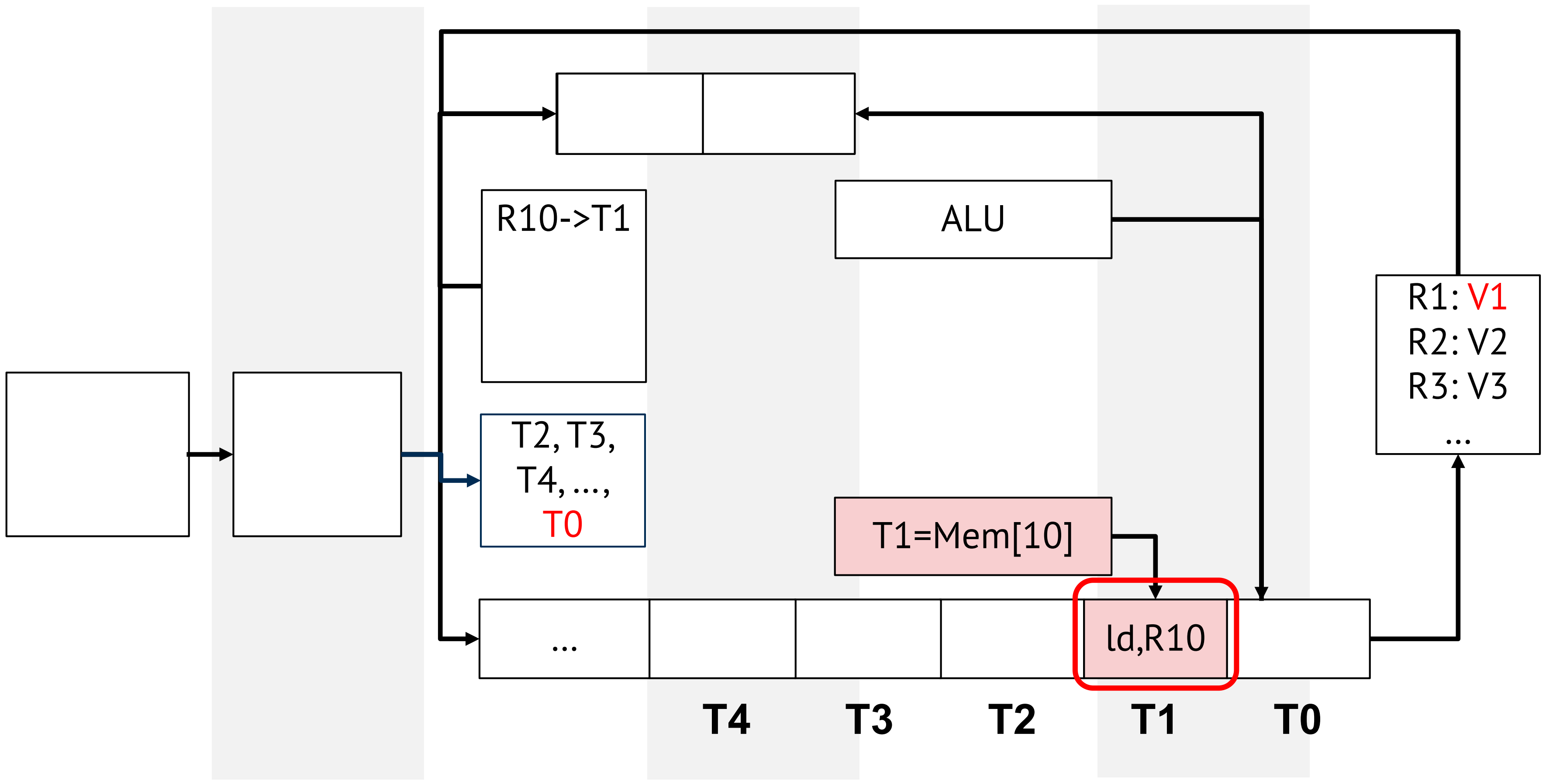
Example: Instruction Commit

- Register file write



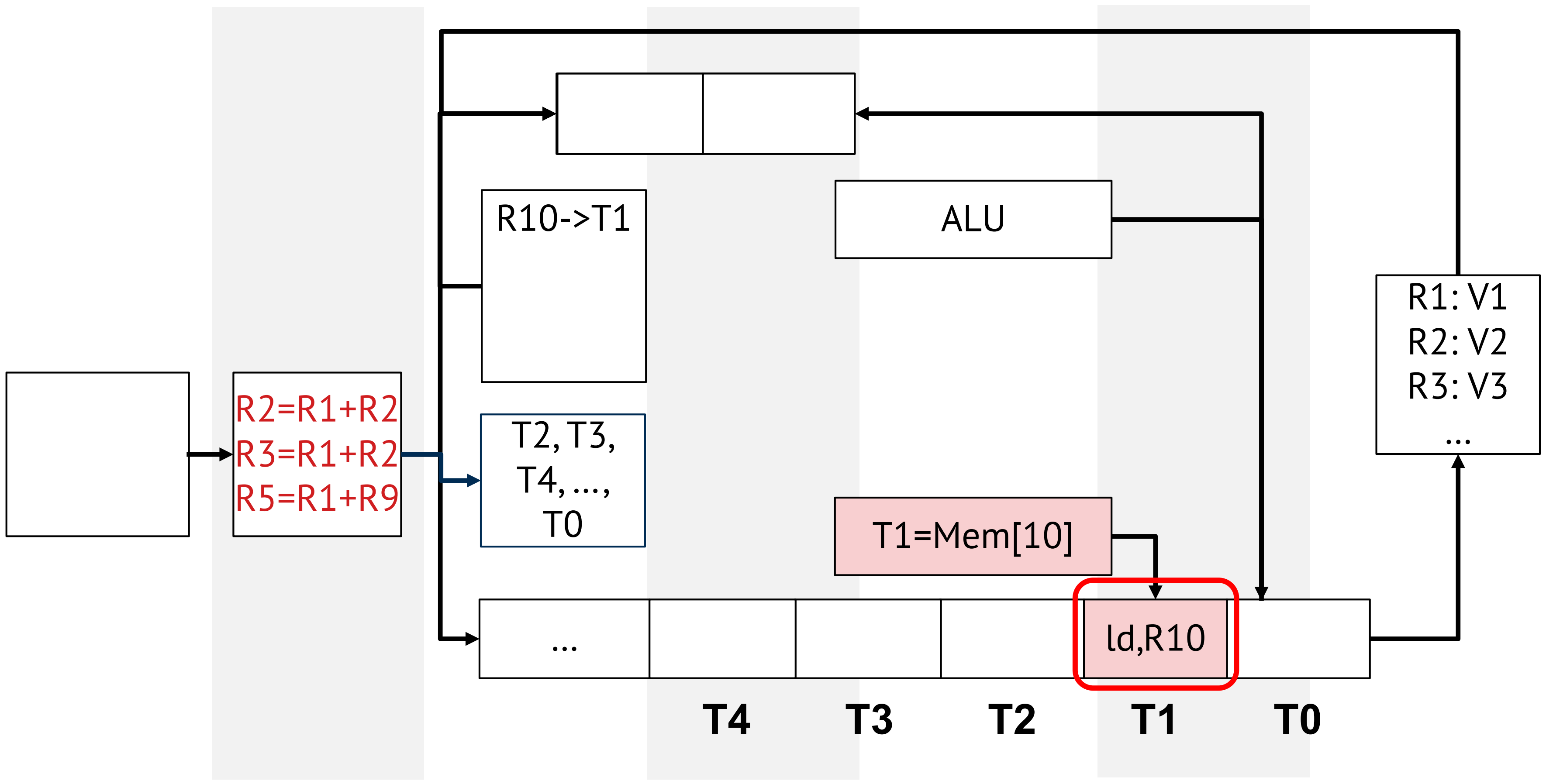
Example: Instruction Commit

- Update tables and ROB



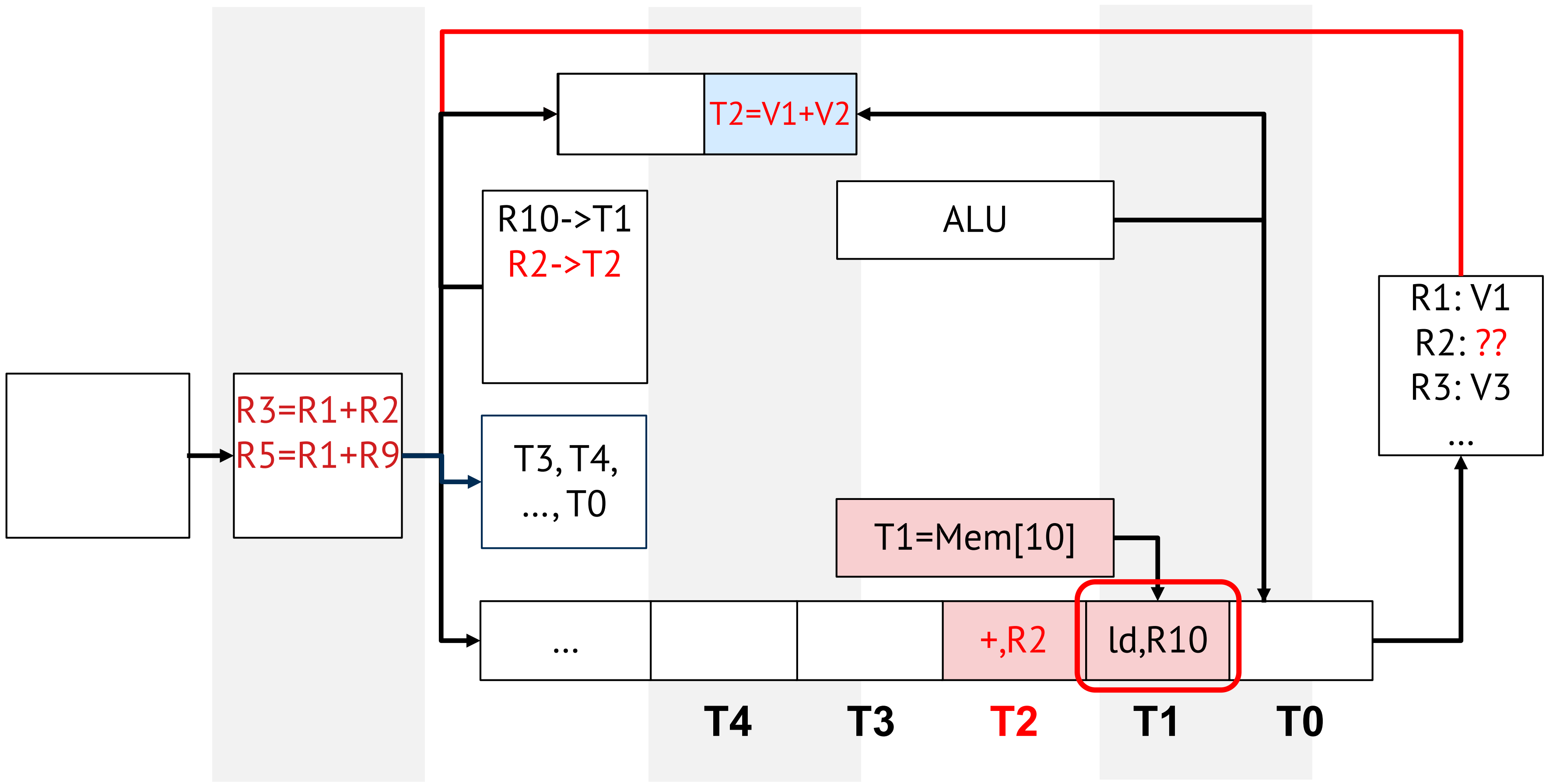
Example: Instruction Rename

- Allocate entries on ROB, IQ, and FL



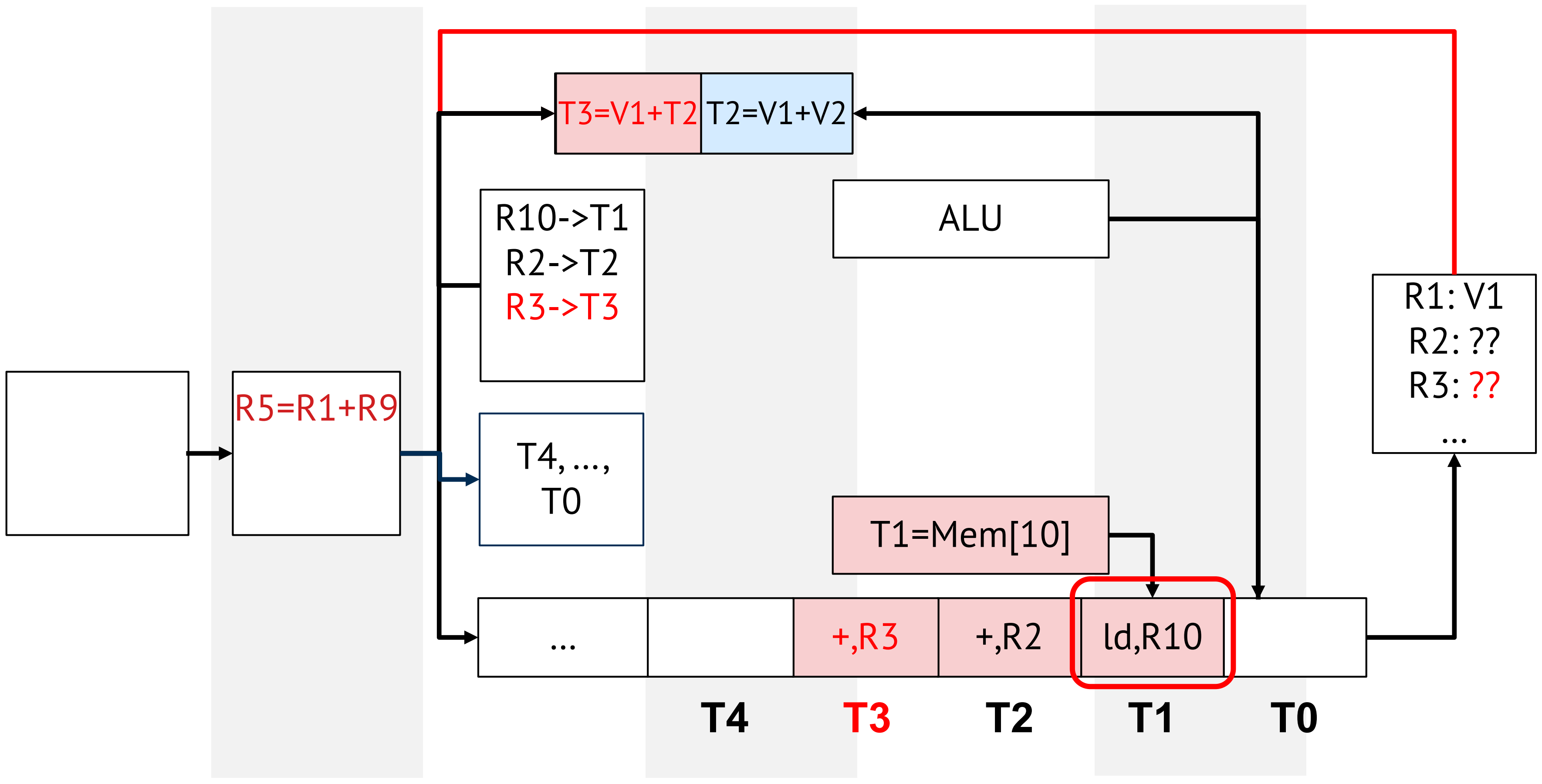
Example: Instruction Rename

- Allocate entries on ROB, IQ, and FL



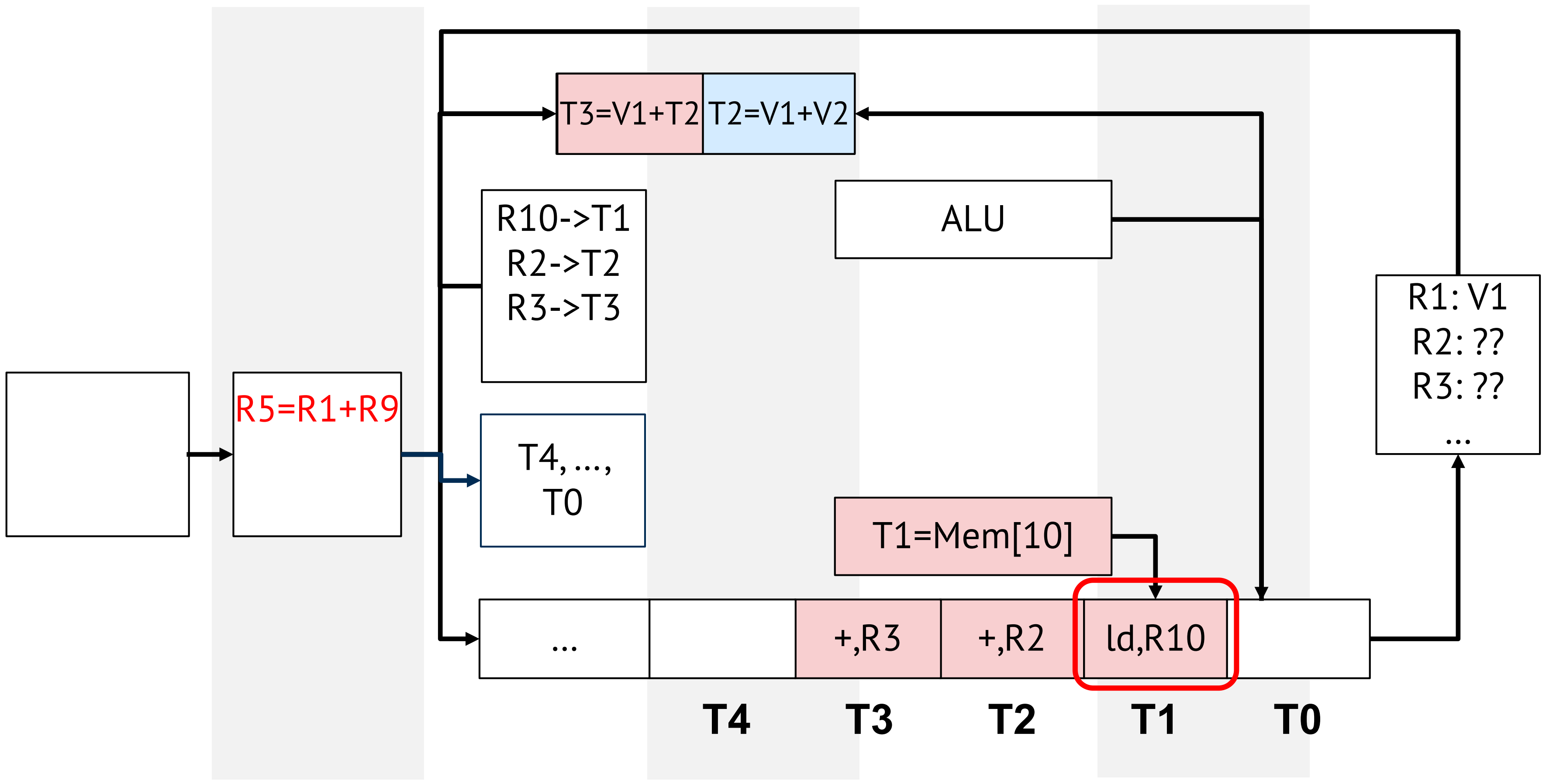
Example: Instruction Rename

- Allocate entries on ROB, IQ, and FL



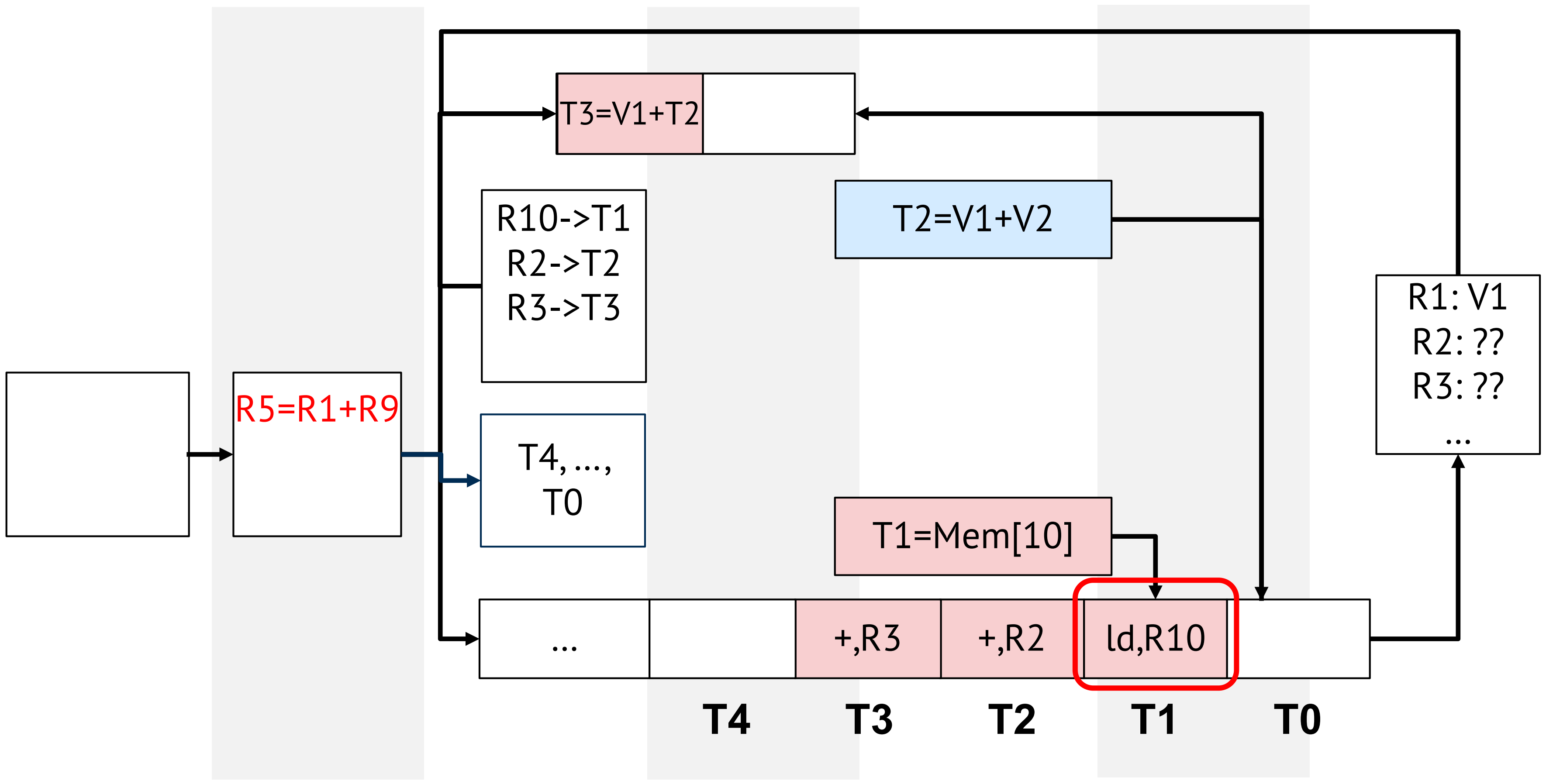
Example: Instruction Rename

- Instruction has to wait for free resources



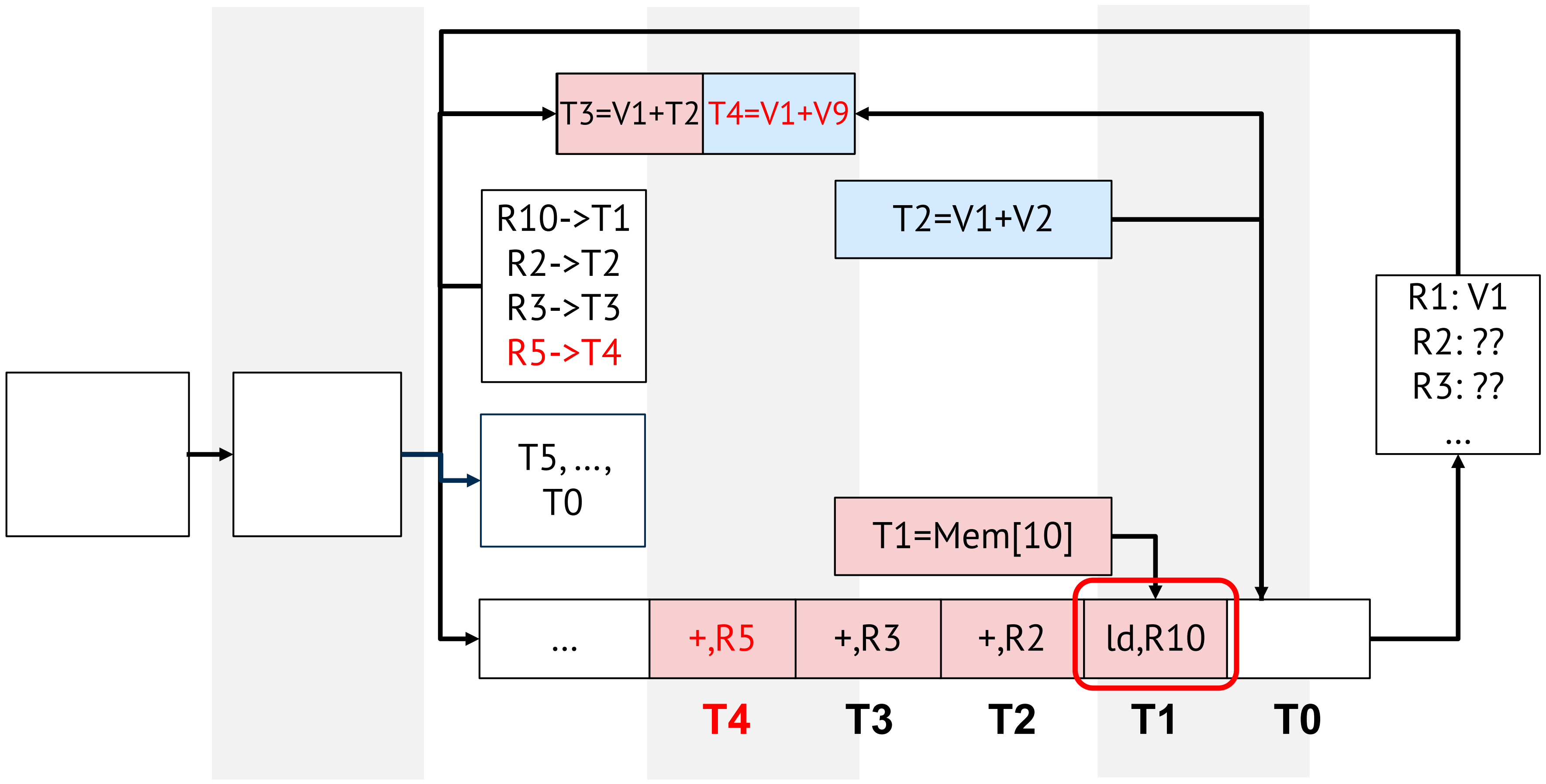
Example: Instruction Issue

- Issue ready instruction if free FU exists



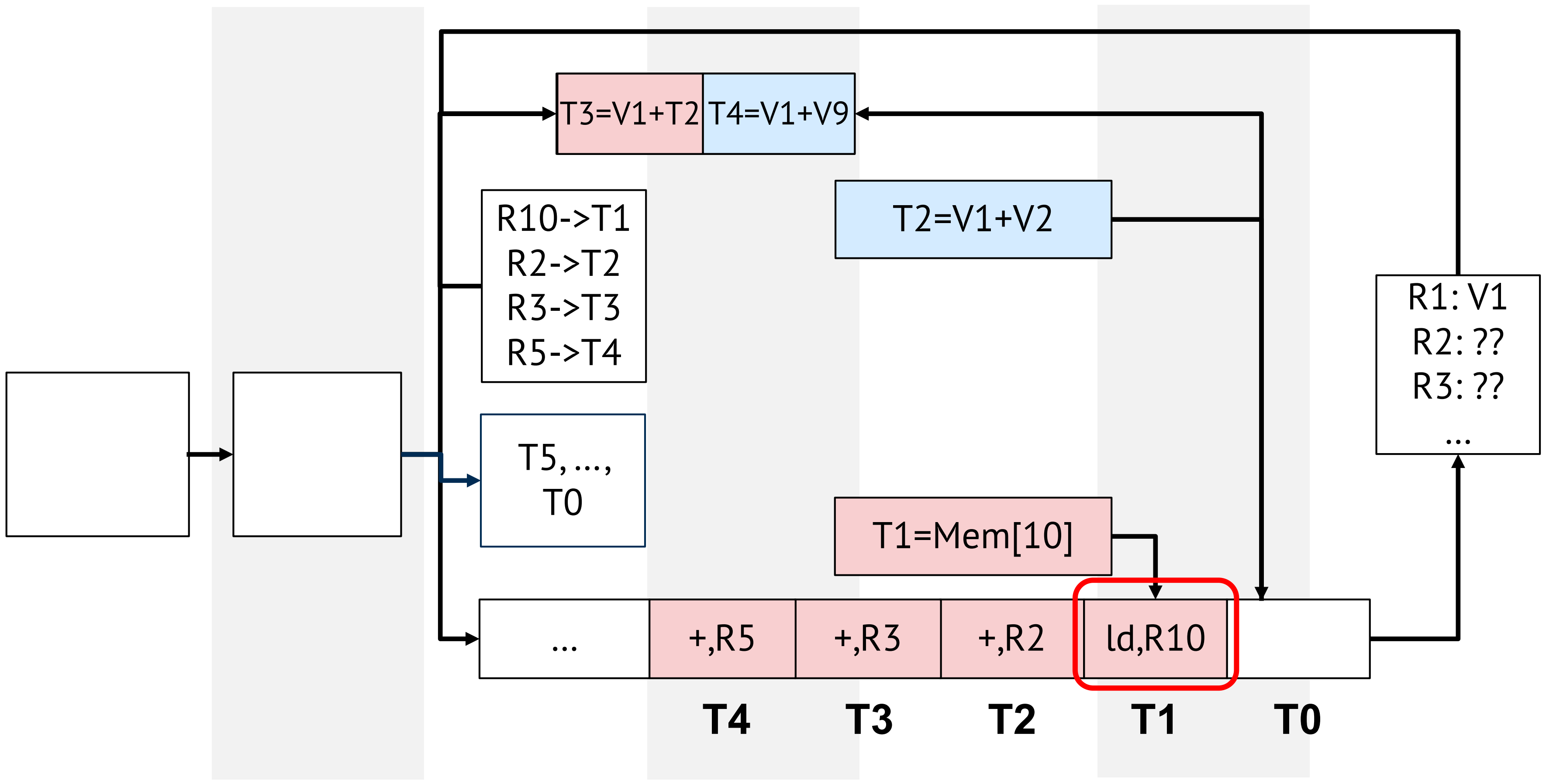
Example: Instruction Issue

- Out-of-order issue is now possible



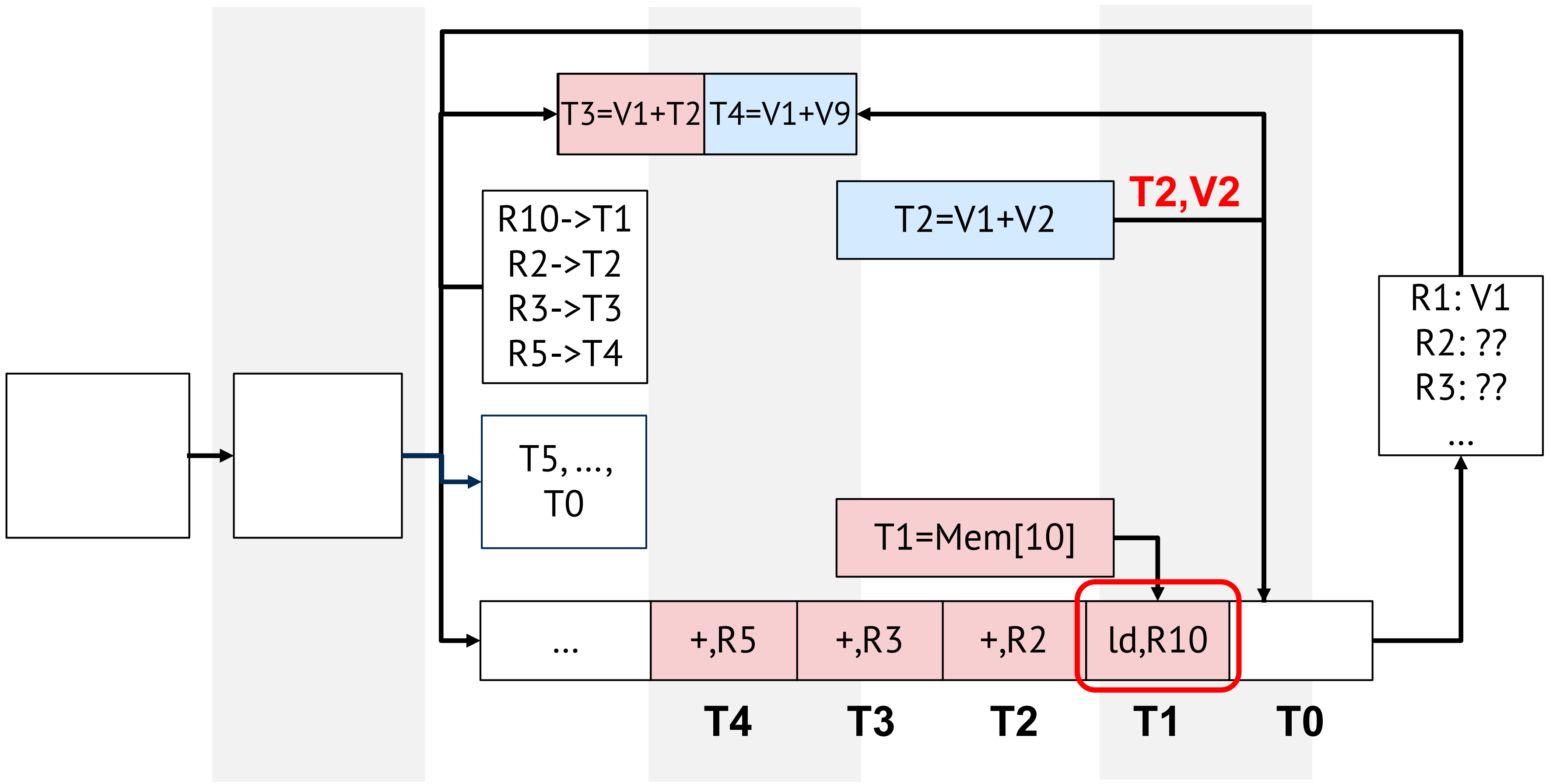
Example: Instruction Issue

- Out-of-order issue is now possible



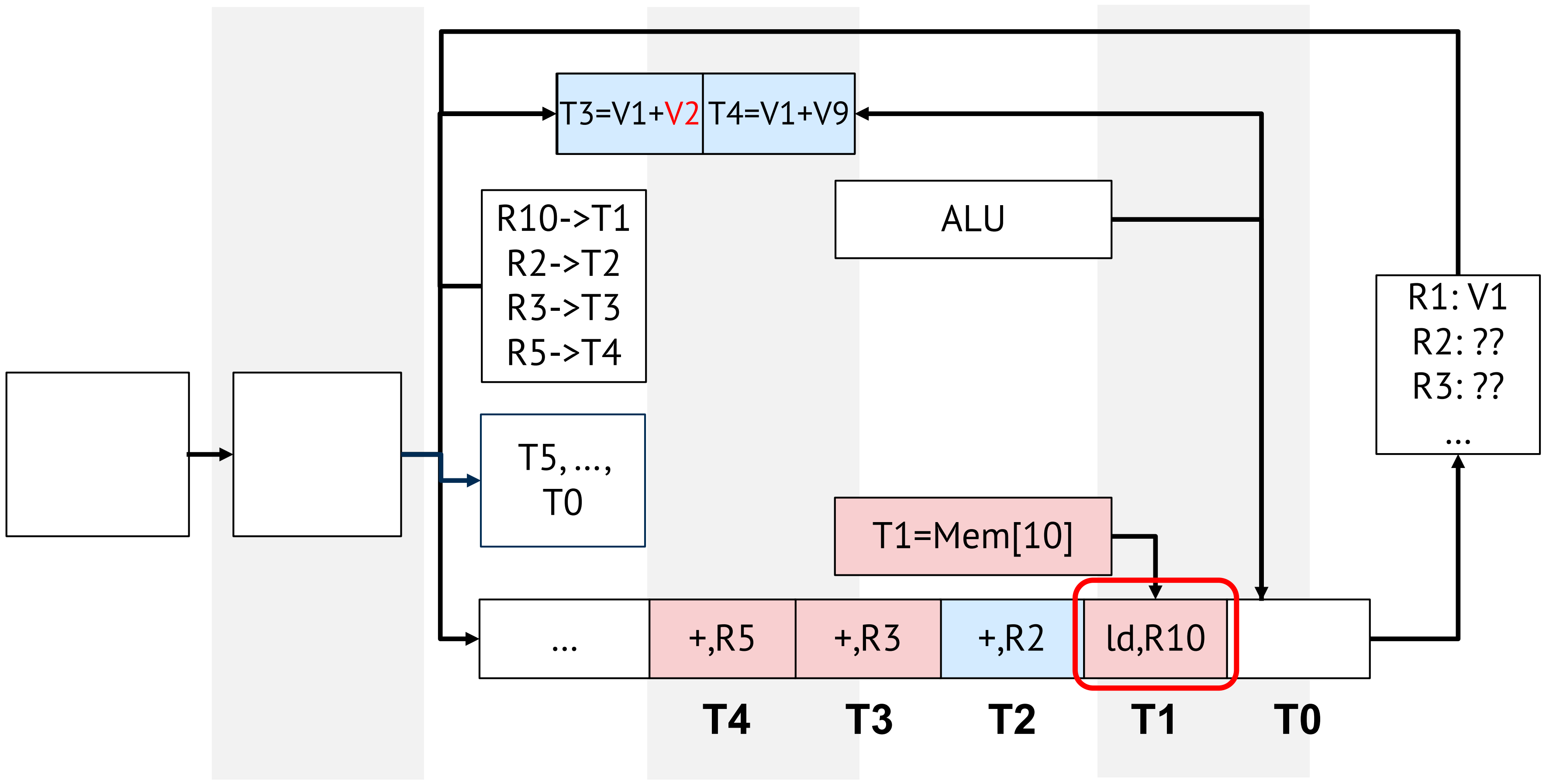
Example: Instruction Issue

- Out-of-order issue is now possible



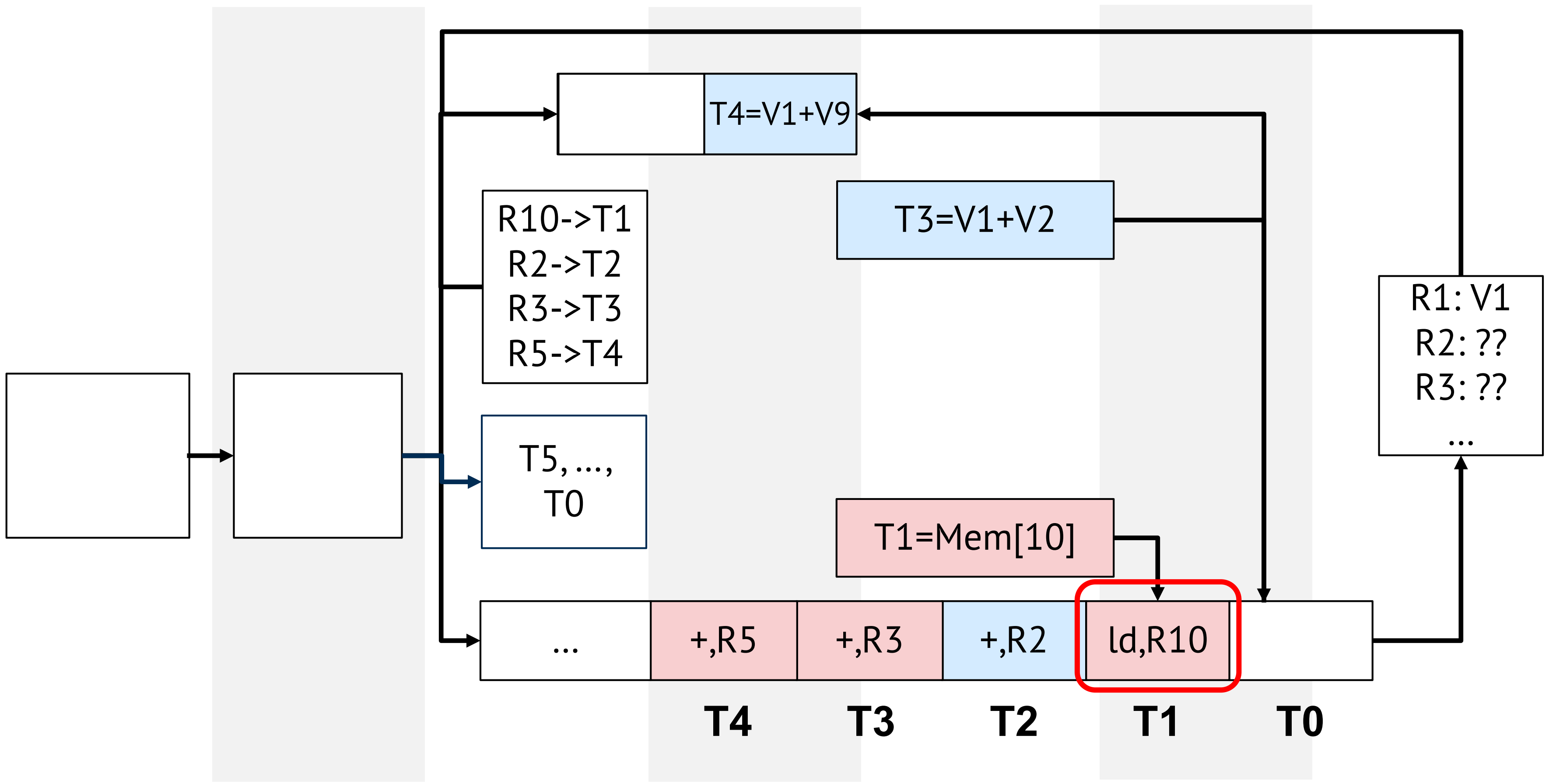
Example: Instruction Issue

- Wakeup and select



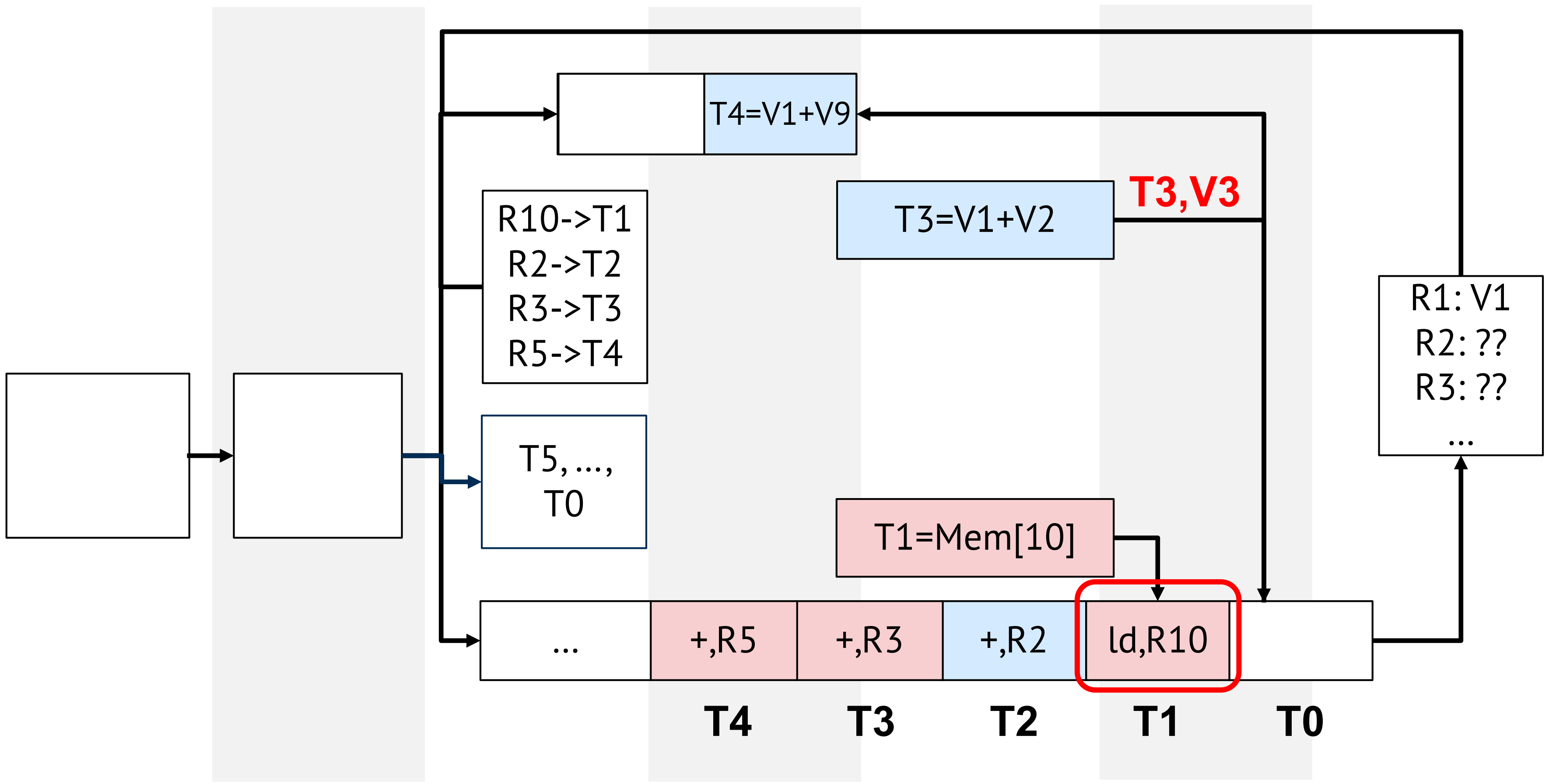
Example: Instruction Issue

- Keep the program order to avoid starvation



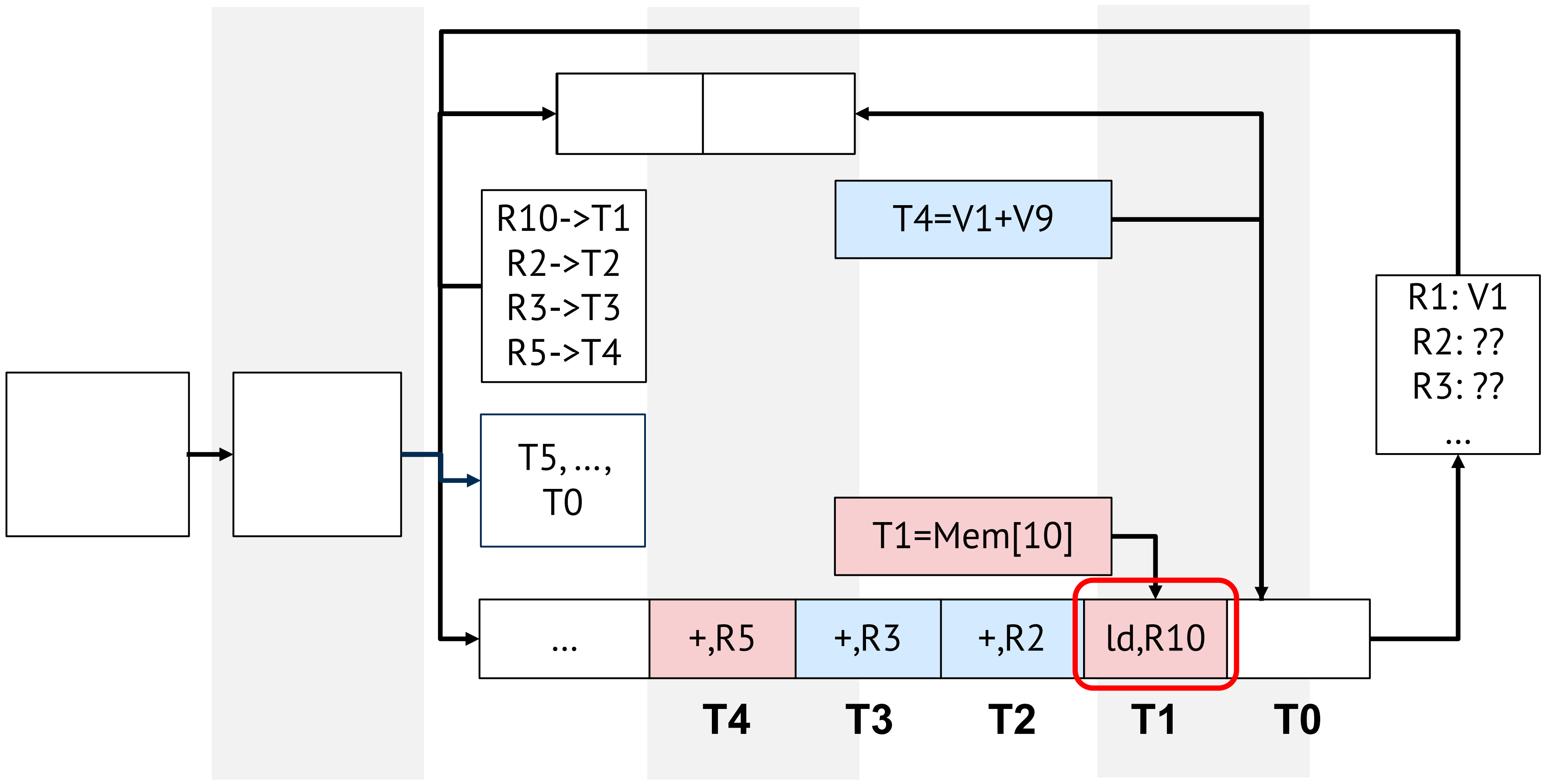
Example: Instruction Issue

- Keep the program order to avoid starvation



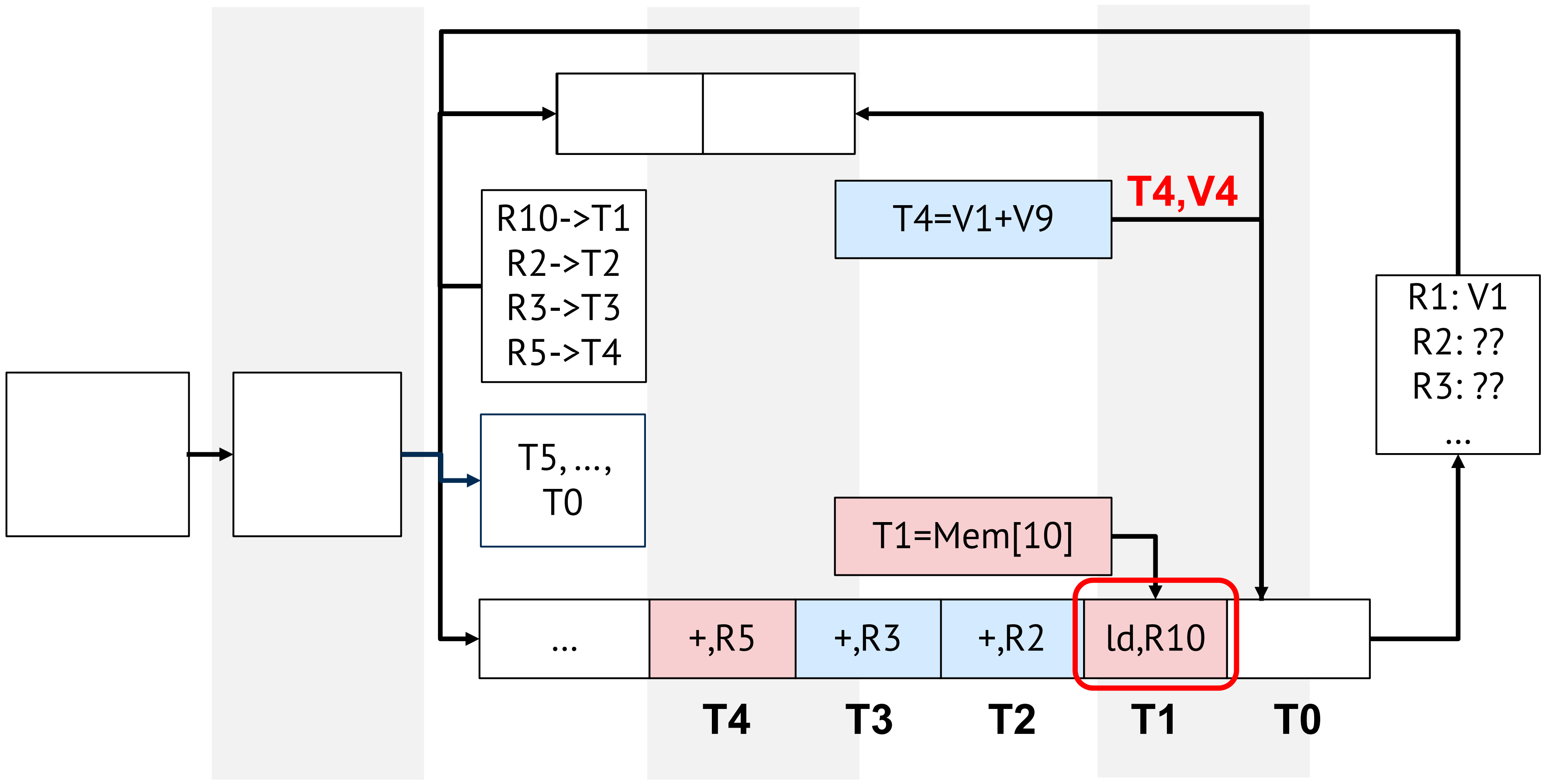
Example: Instruction Execute

- Issue ready instructions



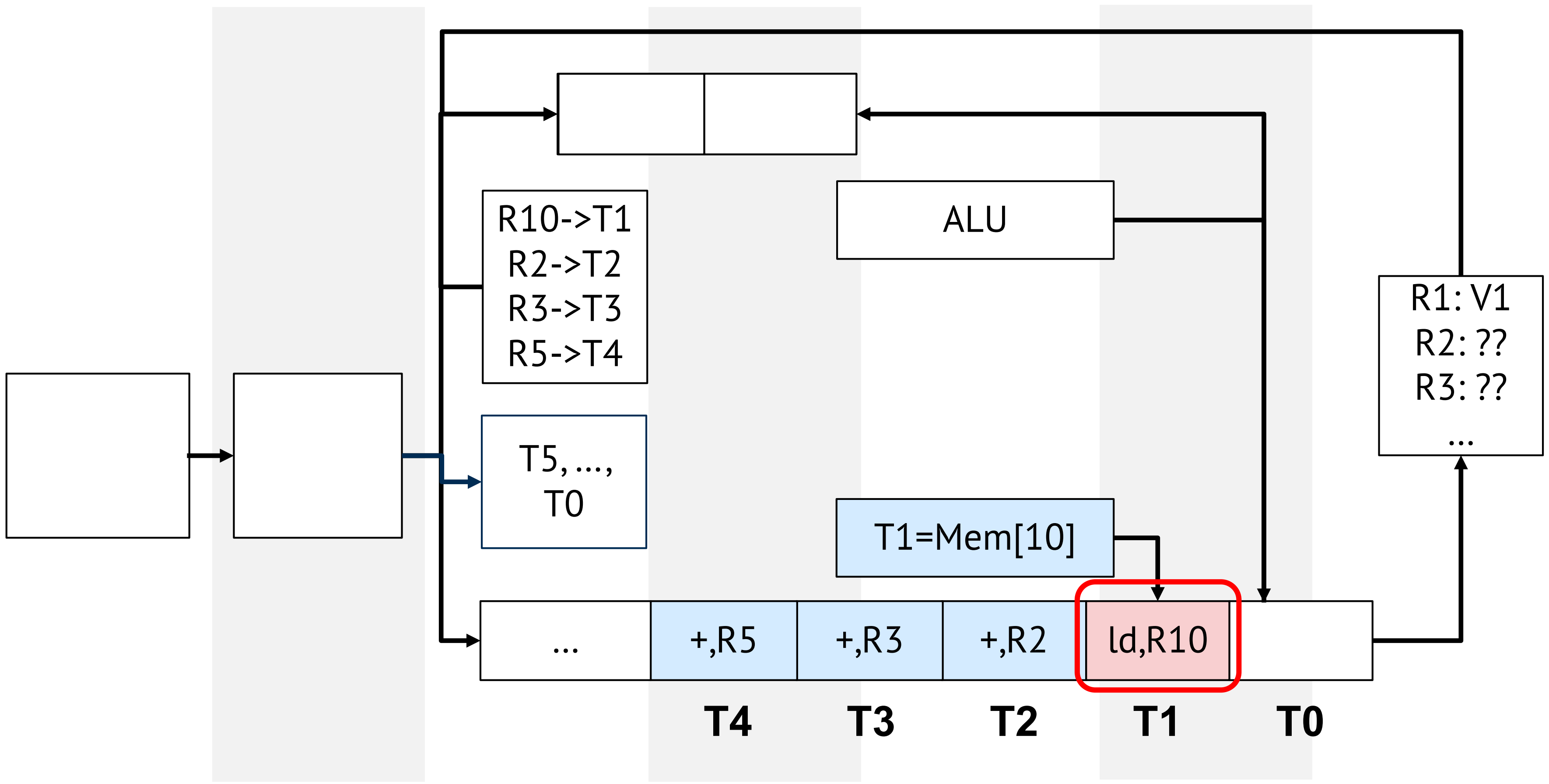
Example: Instruction Execute

- Issue ready instructions



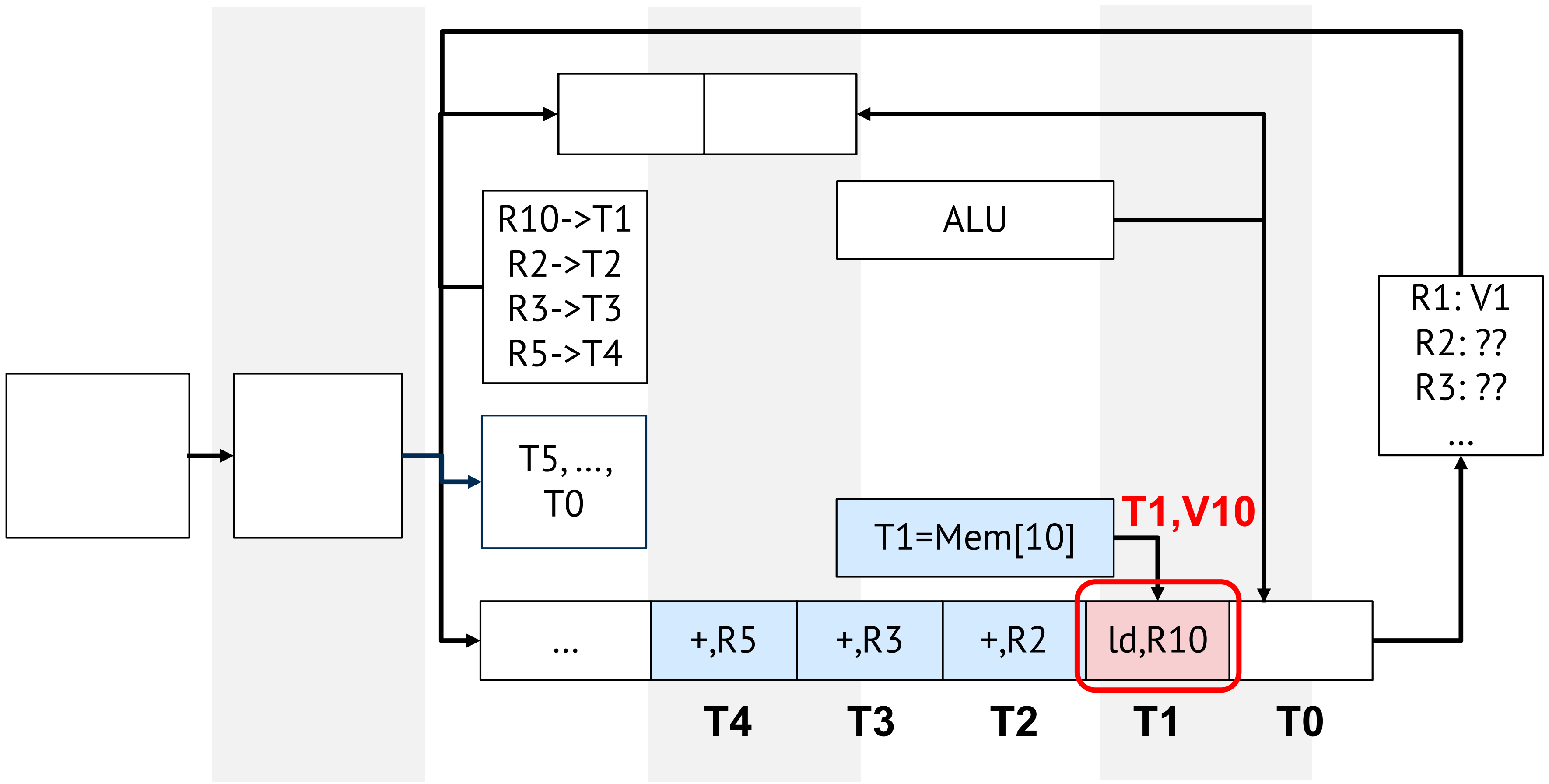
Example: Instruction Execute

- Update ROB



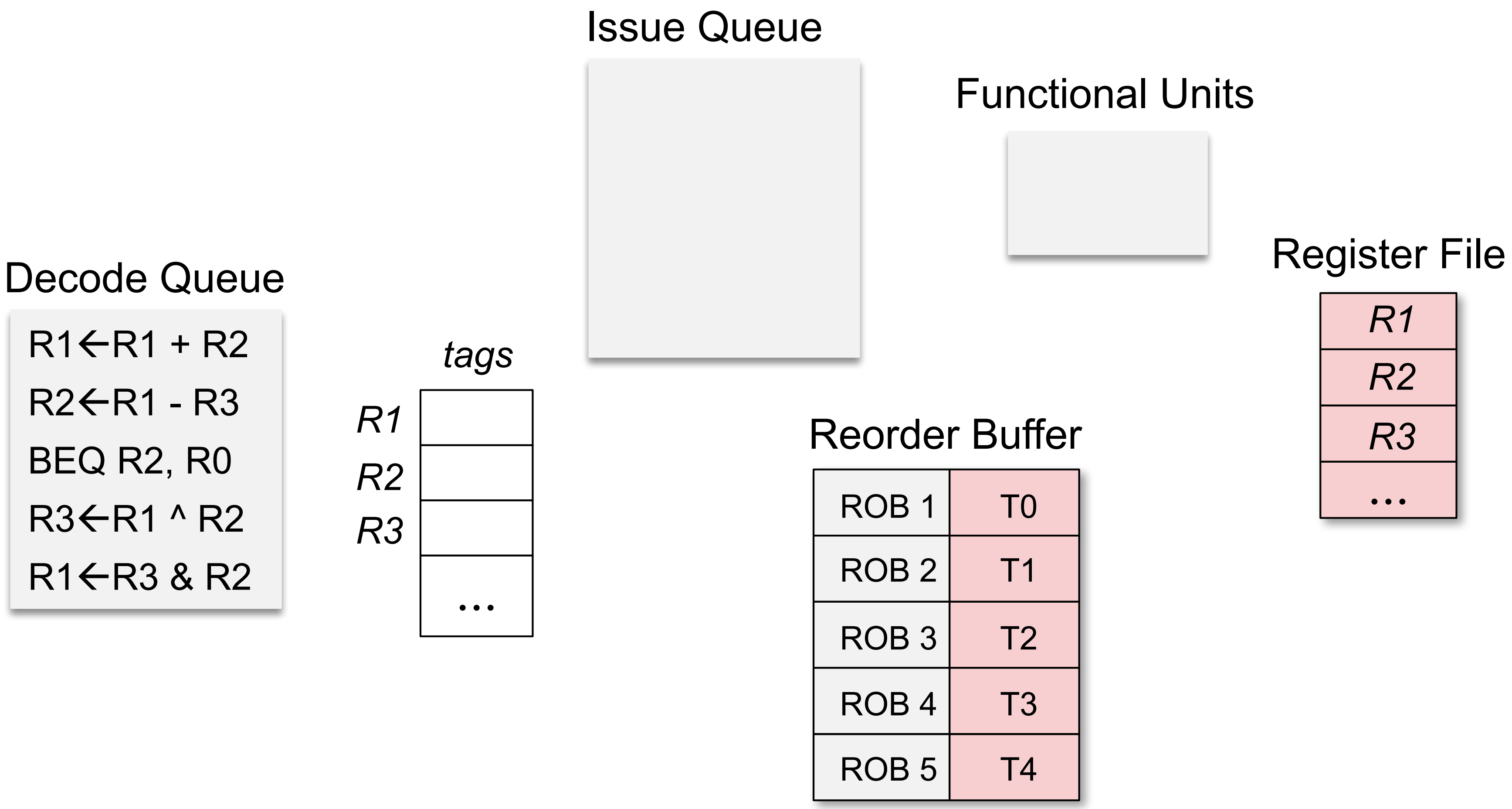
Example: Instruction Execute

- Update ROB



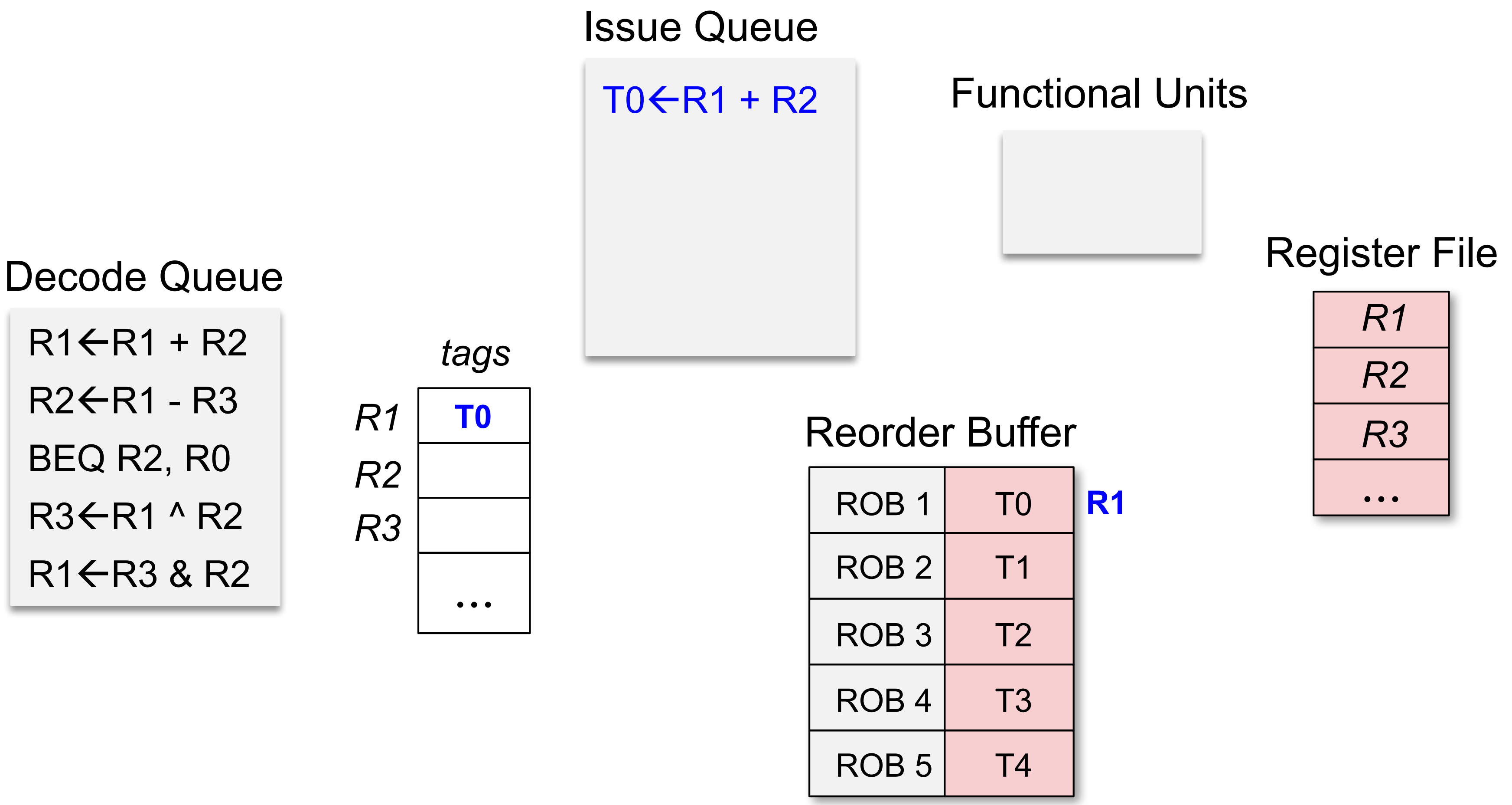
Register Renaming Example

- Where values are stored?



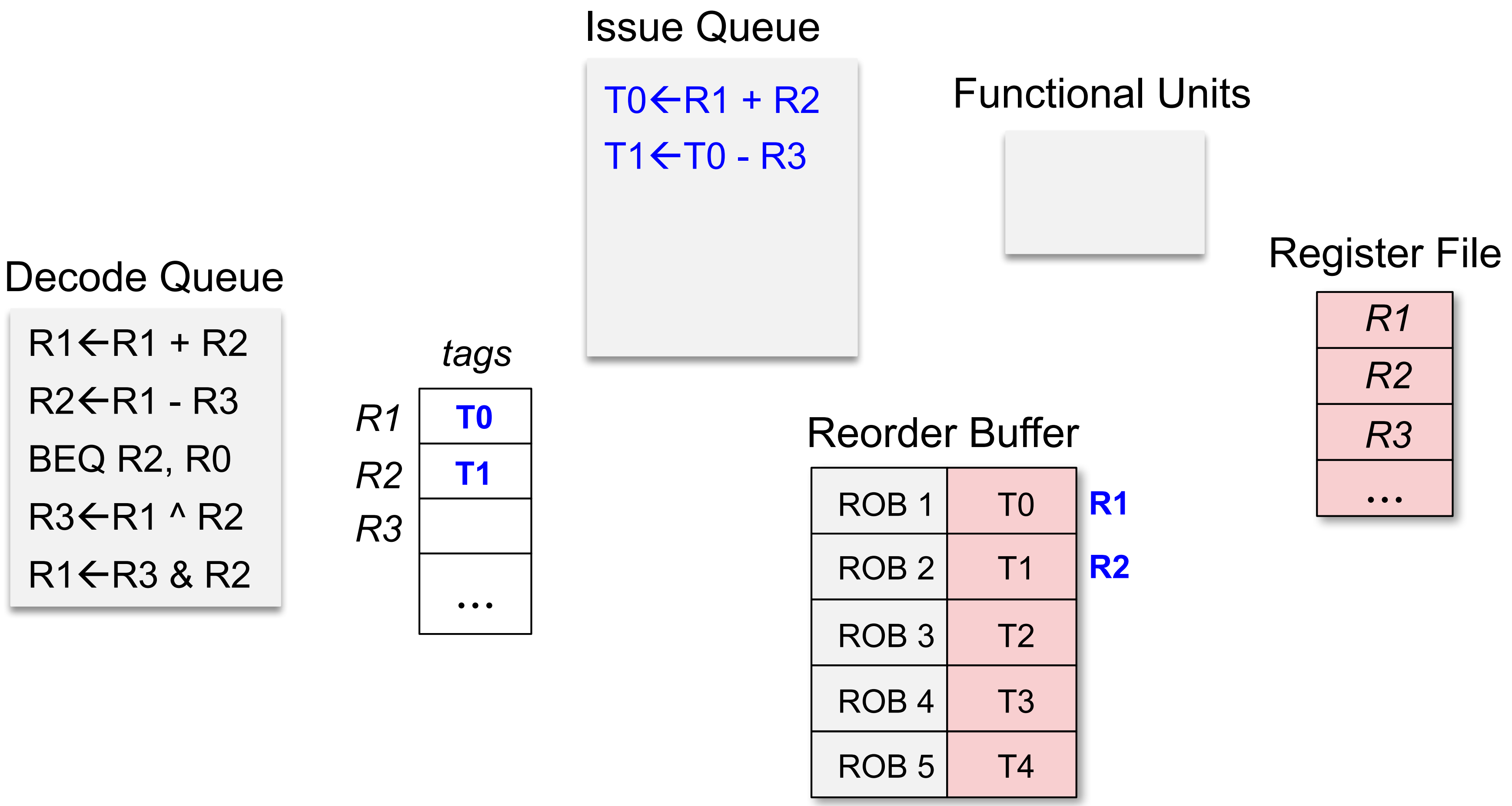
Register Renaming Example

- Where values are stored?



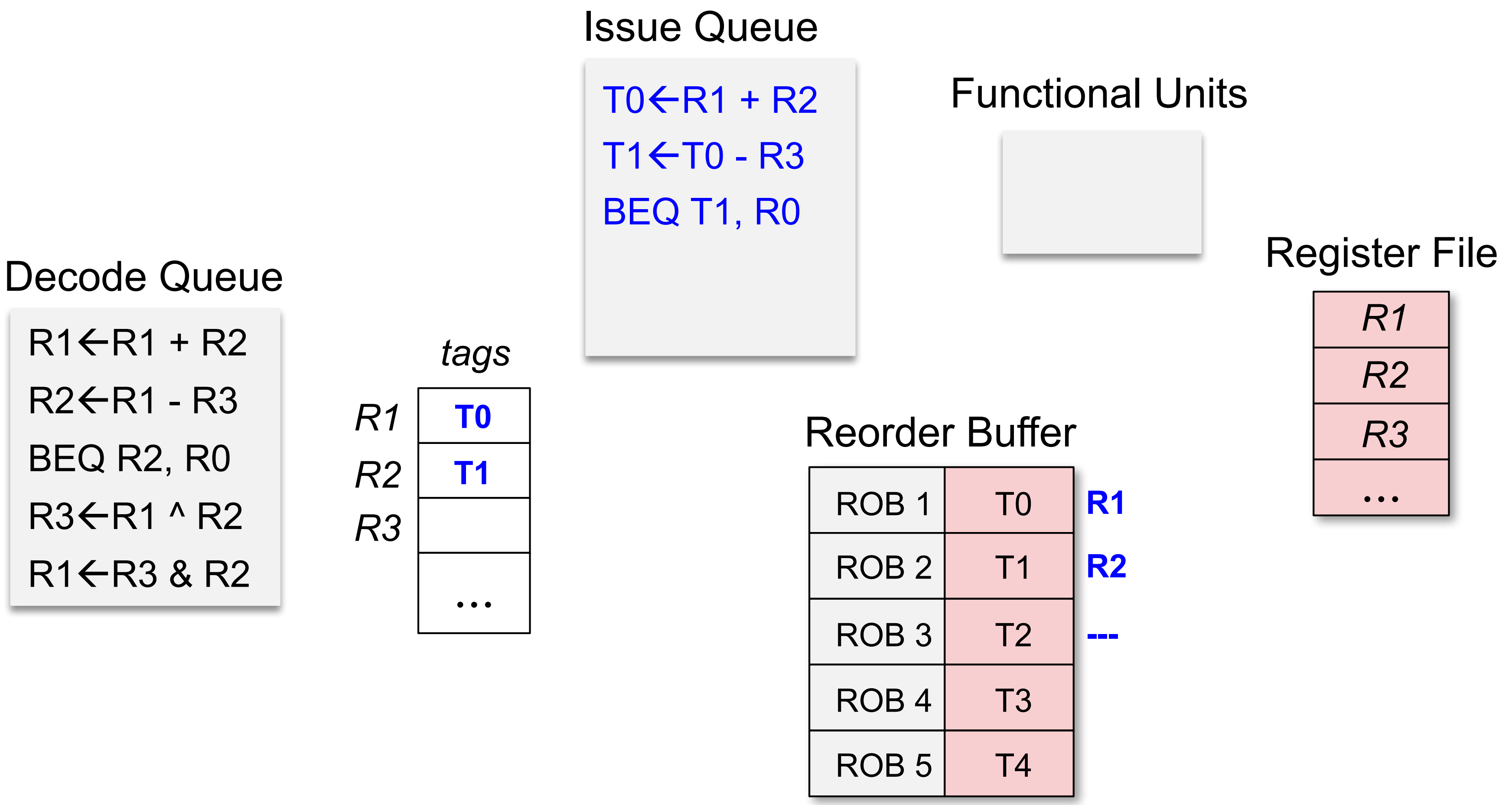
Register Renaming Example

- Where values are stored?



Register Renaming Example

- Where values are stored?



Decode Queue

```
R1←R1 + R2
R2←R1 - R3
BEQ R2, R0
R3←R1 ^ R2
R1←R3 & R2
```

Issue Queue

```
T0←R1 + R2
T1←T0 - R3
BEQ T1, R0
```

	tags
R1	**T0**
R2	**T1**
R3	
	…

Reorder Buffer

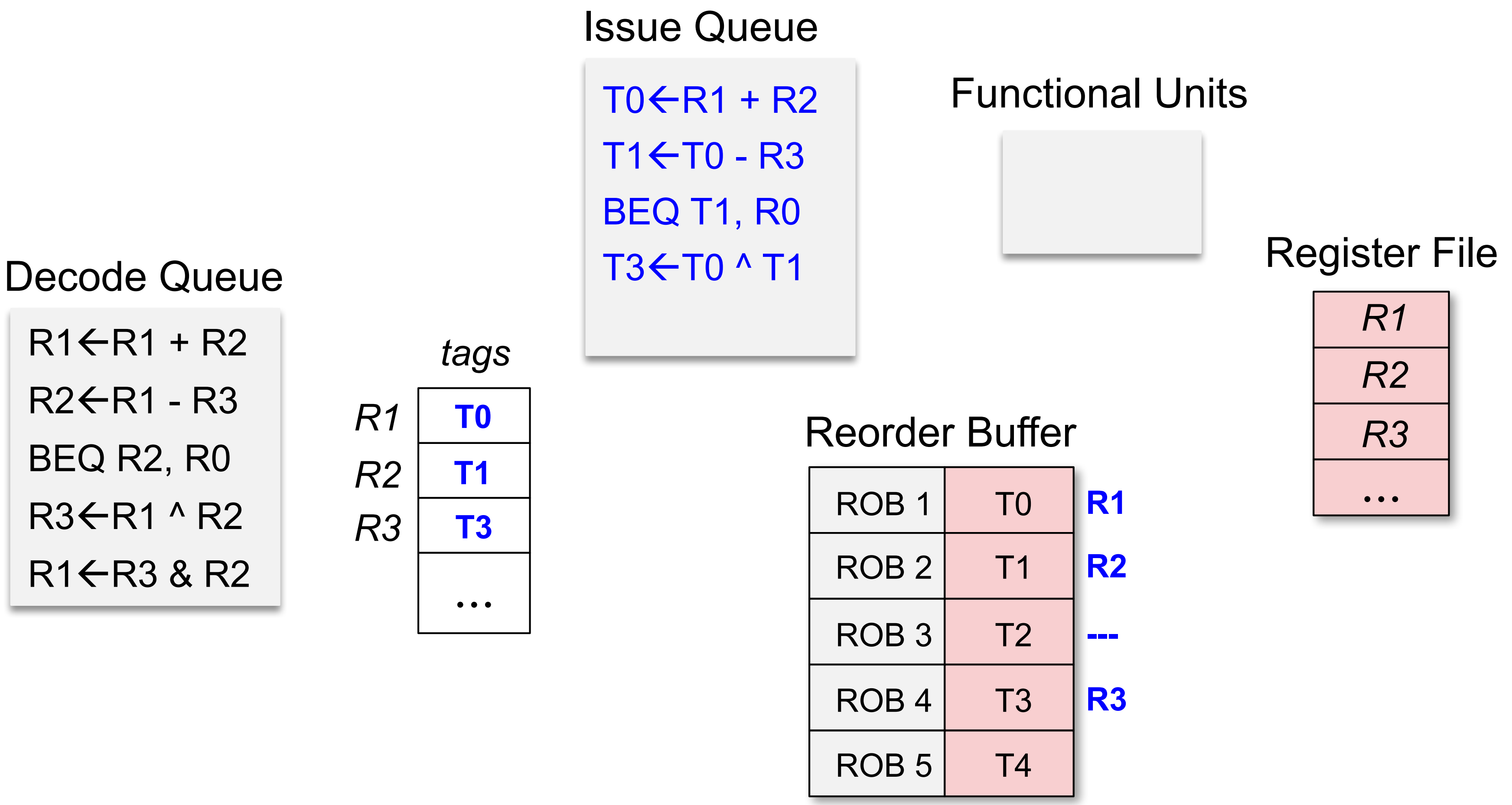
ROB 1	T0	**R1**
ROB 2	T1	**R2**
ROB 3	T2	**---**
ROB 4	T3	
ROB 5	T4	

Register File

R1
R2
R3
…

Register Renaming Example

- Where values are stored?



Register Renaming Example

- Where values are stored?

Issue Queue

- T0←R1 + R2
- T1←T0 - R3
- BEQ T1, R0
- T3←T0 ^ T1
- T4←T3 & T1

Functional Units

Decode Queue

- R1←R1 + R2
- R2←R1 - R3
- BEQ R2, R0
- R3←R1 ^ R2
- R1←R3 & R2

	tags
R1	**T4**
R2	**T1**
R3	**T3**
	…

Reorder Buffer

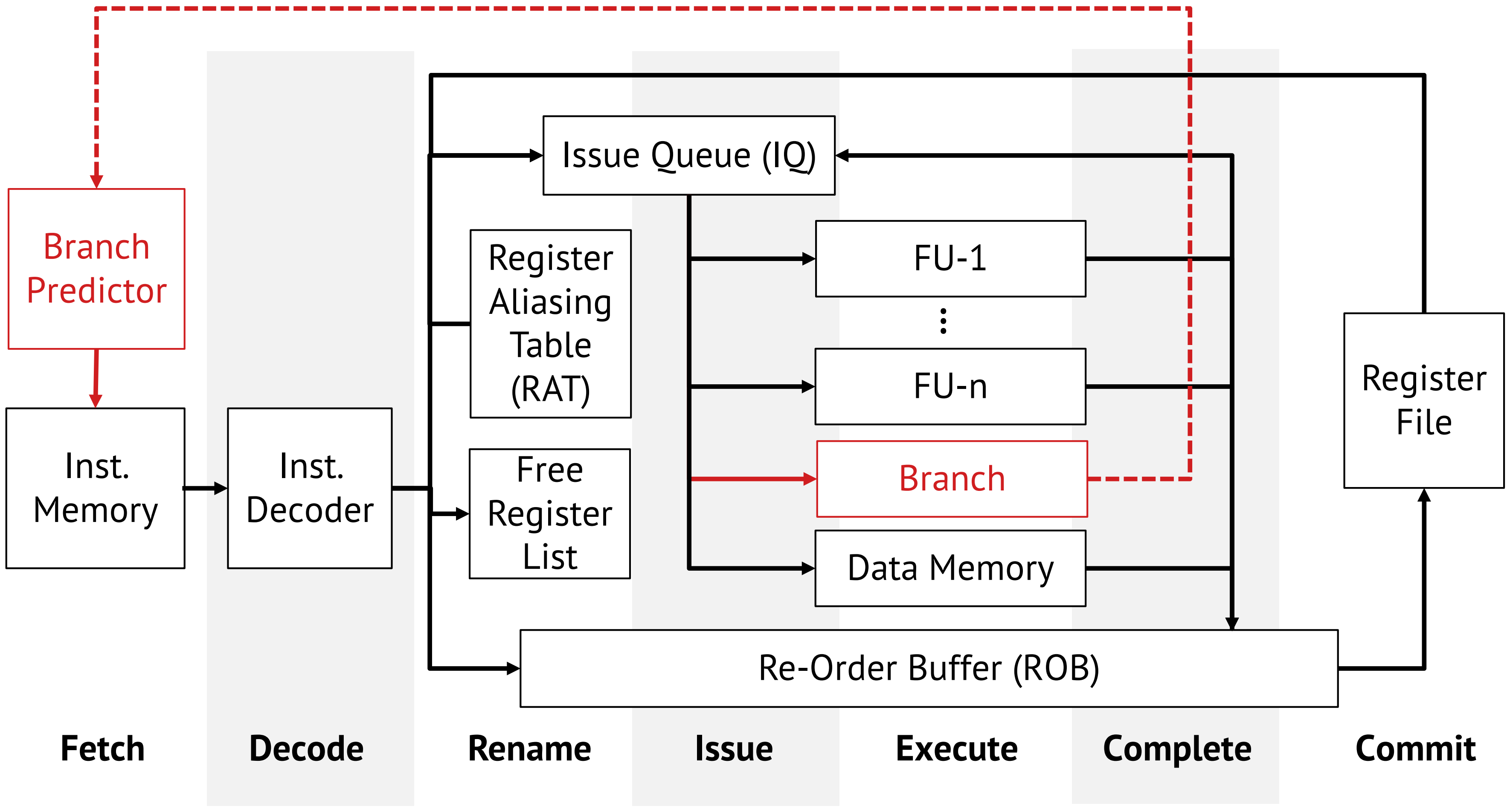
ROB 1	T0	**R1**
ROB 2	T1	**R2**
ROB 3	T2	**---**
ROB 4	T3	**R3**
ROB 5	T4	**R1**

Register File

R1
R2
R3
…

Branch Recovery

- How to handle branches?



Revisit Branch Prediction

- Problem: find the average number of stall cycles caused by branches in a pipeline, where branch misprediction penalty is 20 cycles, branch predictor accuracy is 90%, and branch target buffer hit rate is 80%. Every fifth instruction is a branch; 30% of branches are actually taken.

Revisit Branch Prediction

- Problem: find the average number of stall cycles caused by branches in a pipeline, where branch misprediction penalty is 20 cycles, branch predictor accuracy is 90%, and branch target buffer hit rate is 80%. Every fifth instruction is a branch; 30% of branches are actually taken.

  - Average misses = 1- (0.3x0.9x0.8 + 0.7x0.9) = 0.151
  - Average stalls = 20x0.2x0.151 = 0.6

Speculated Execution

- Problem: branch may significantly limit performance
  - consumer of a load or long latency instructions
- Solution: speculative instruction execution
  - Fetch and decode instructions speculatively
  - Issue and execute speculative instructions
  - Branch resolution
    - Nullify the impact of speculative instructions if mispredicted
    - Commit speculative instructions (writes to register file/memory) only if prediction was correct

Branch Recovery

- Squash all mispredicted entries

Issue Queue

T0←R1 + R2
T1←T0 + R3
BEQ T1, R0
T3←T0 + T1
T4←T3 + T1

Functional Units

Decode Queue

R1←R1 + R2
R2←R1 - R3
BEQ R2, R0
R3←R1 ^ R2
R1←R3 & R2

RAT

R1	**T4**
R2	**T1**
R3	**T3**
	…

Register File

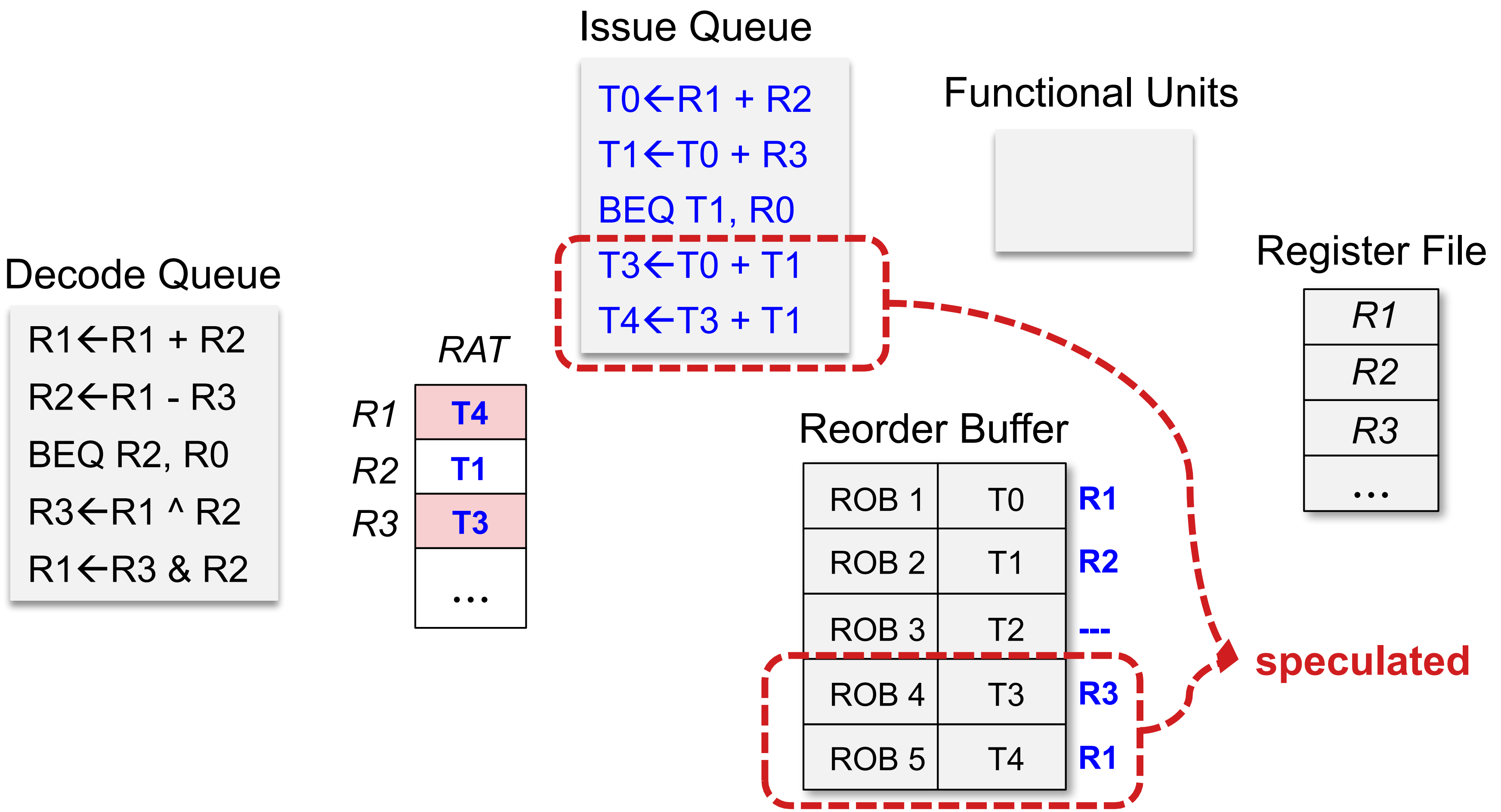
R1
R2
R3
…

Reorder Buffer

ROB 1	T0	**R1**
ROB 2	T1	**R2**
ROB 3	T2	**---**
ROB 4	T3	**R3**
ROB 5	T4	**R1**

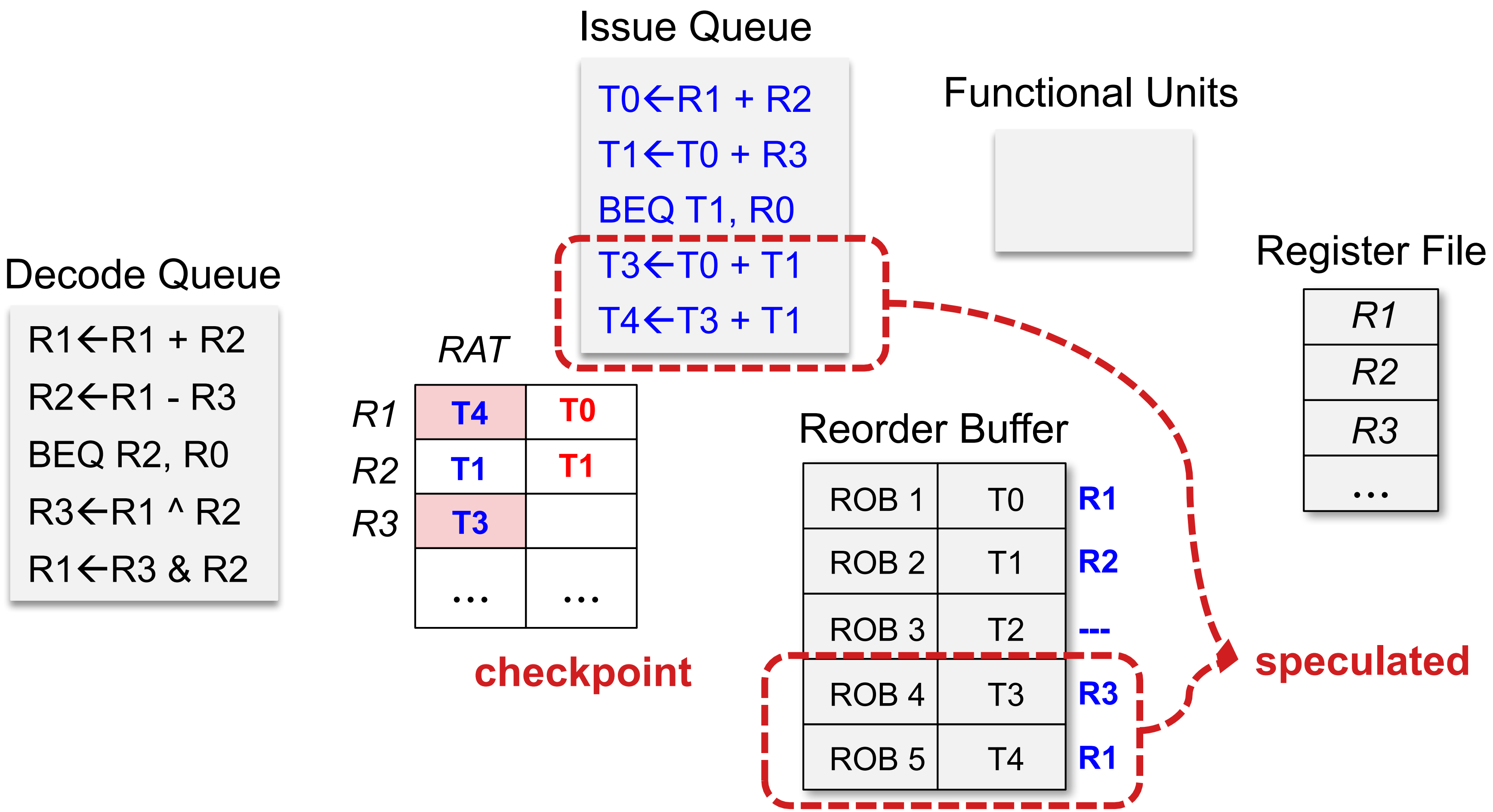
Branch Recovery

- Squash all mispredicted entries



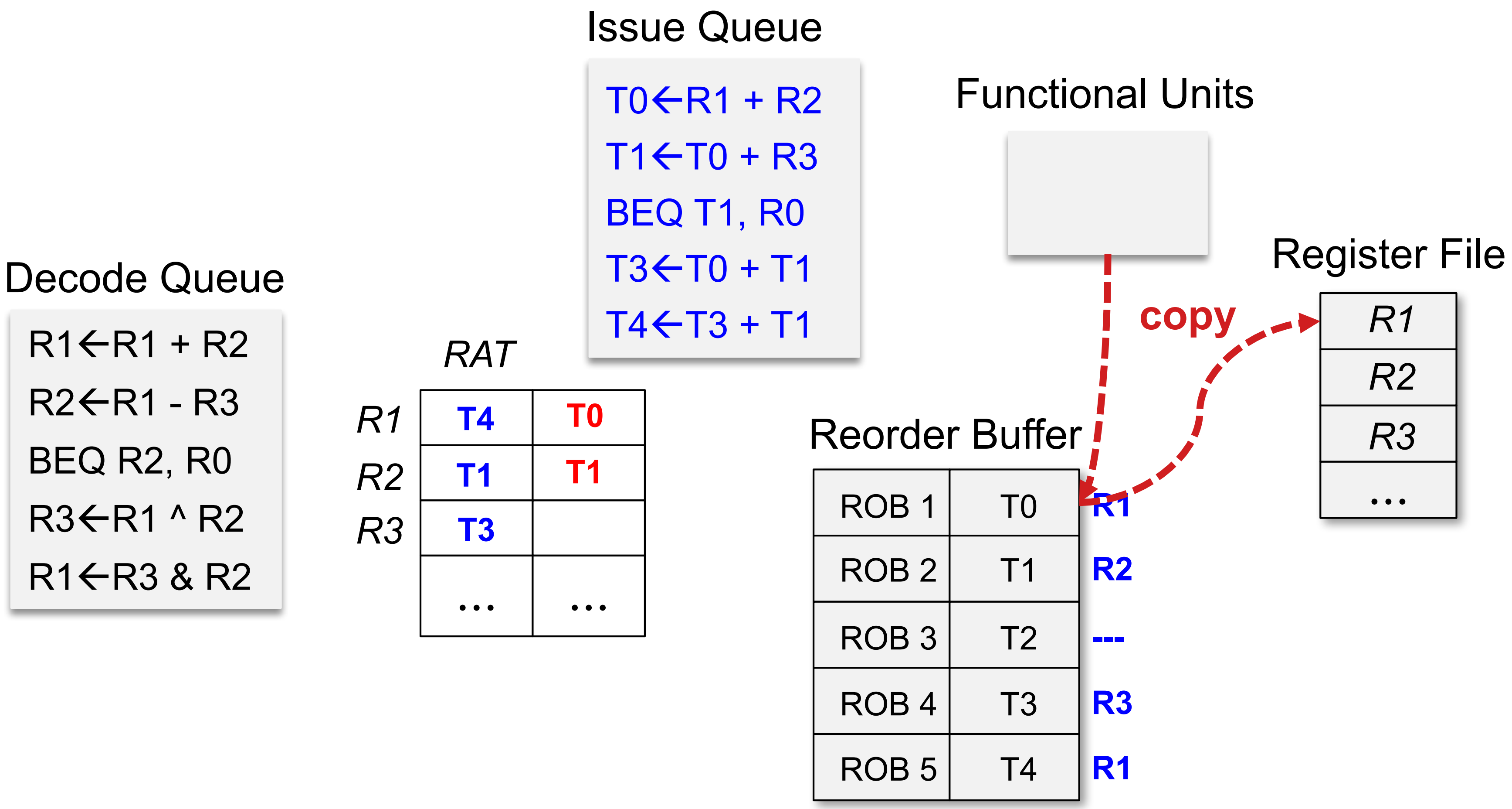
Branch Recovery

- Squash all mispredicted entries



Physical Register File

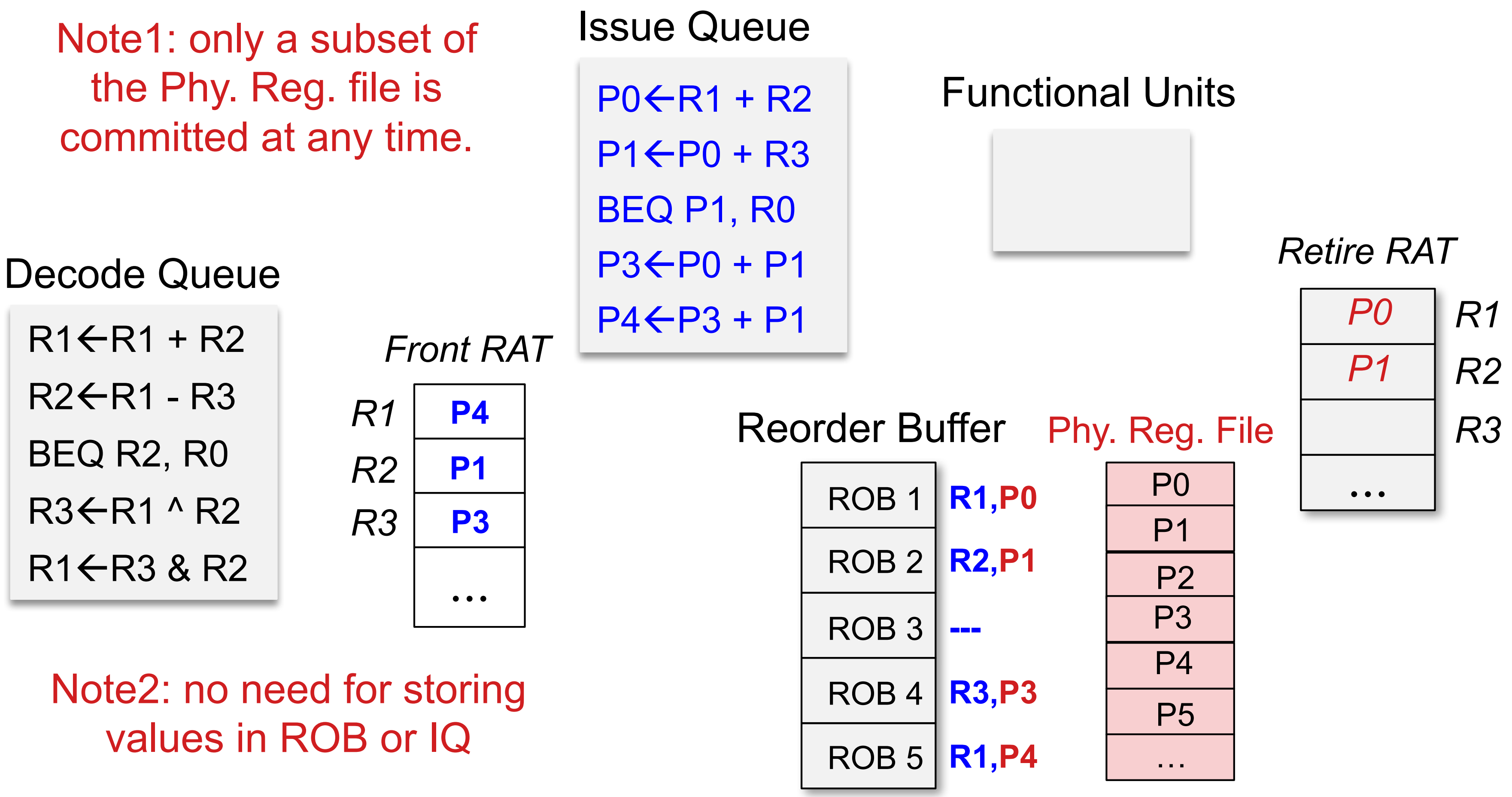
- Avoid copying register values multiple times



Physical Register File

- Avoid copying register values multiple times

Note1: only a subset of the Phy. Reg. file is committed at any time.



Issue Queue

P0←R1 + R2
P1←P0 + R3
BEQ P1, R0
P3←P0 + P1
P4←P3 + P1

Decode Queue

R1←R1 + R2
R2←R1 - R3
BEQ R2, R0
R3←R1 ^ R2
R1←R3 & R2

Front RAT

R1	**P4**
R2	**P1**
R3	**P3**
	…

Retire RAT

P0	*R1*
P1	*R2*
	R3
…	

Reorder Buffer

ROB 1	**R1,P0**
ROB 2	**R2,P1**
ROB 3	**---**
ROB 4	**R3,P3**
ROB 5	**R1,P4**

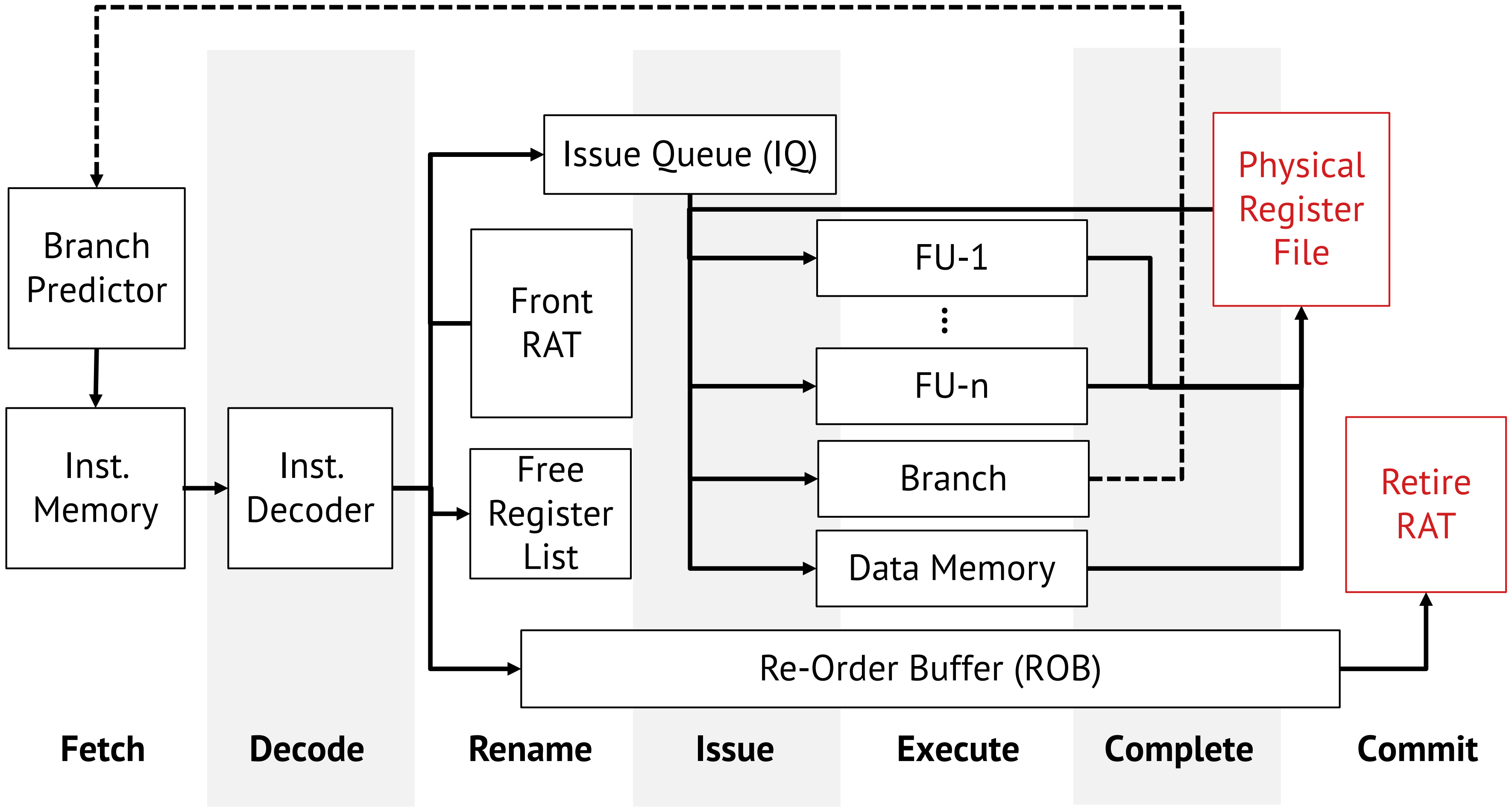
Phy. Reg. File

P0
P1
P2
P3
P4
P5
…

Note2: no need for storing values in ROB or IQ

Double RAT Architecture

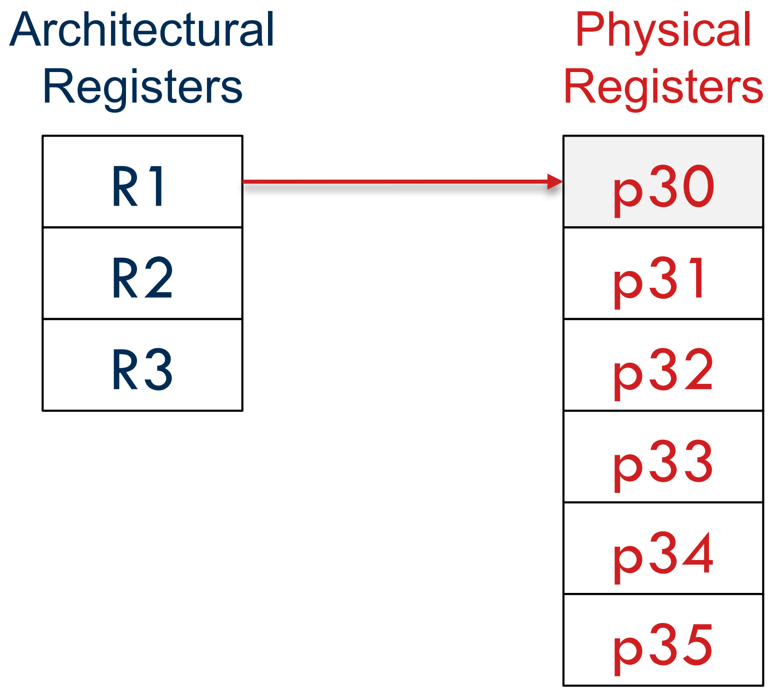
- What is the size of ROB?



Physical Register Release

- Example: when is it safe to free p30 (R1)?

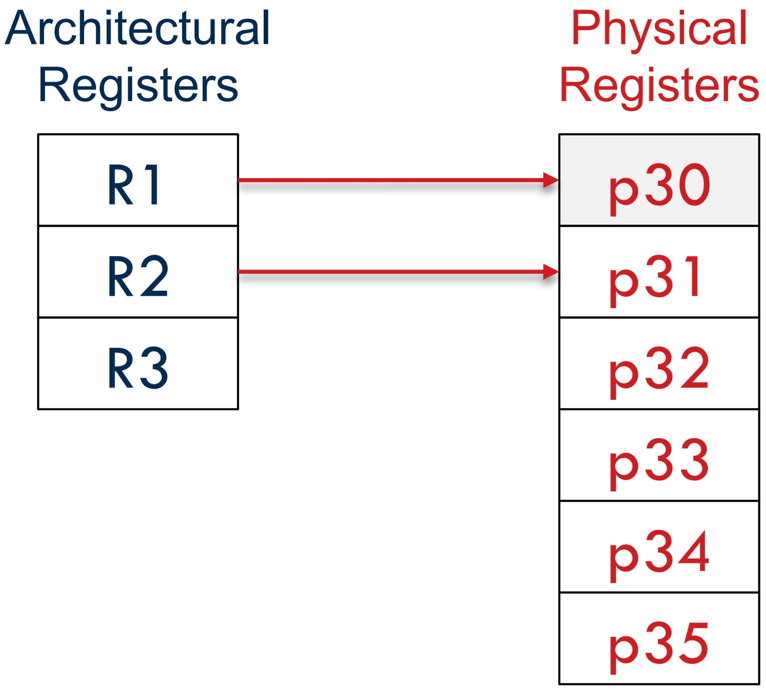
```
ADD  R1, R2, R3
SUB   R2, R1, R3
…
ADD  R3, R1, R2
…
SUB  R1, R3, R2
ADD R2, R1, R3
```



Physical Register Release

- Example: when is it safe to free p30 (R1)?

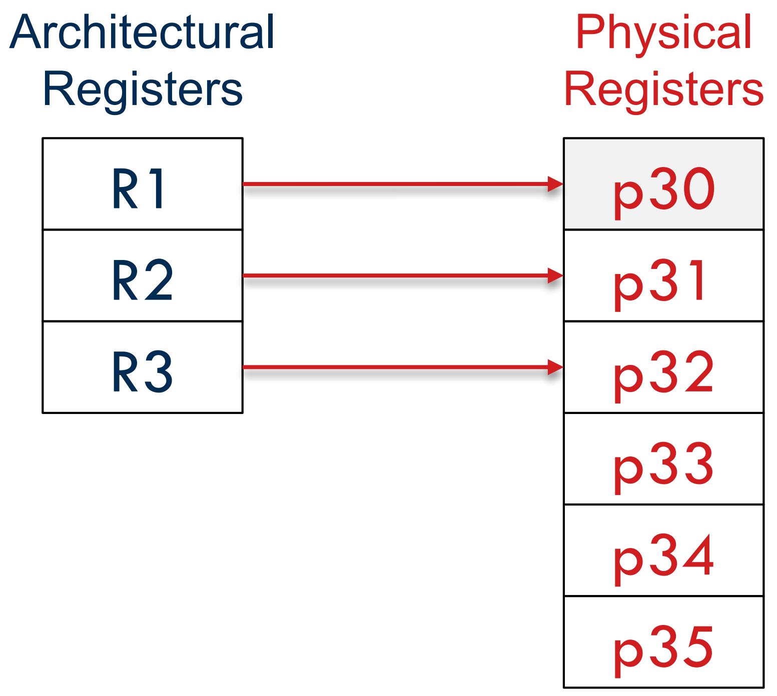
```
ADD  R1, R2, R3
SUB  R2, R1, R3
...
ADD  R3, R1, R2
...
SUB  R1, R3, R2
ADD R2, R1, R3
```



Physical Register Release

- Example: when is it safe to free p30 (R1)?

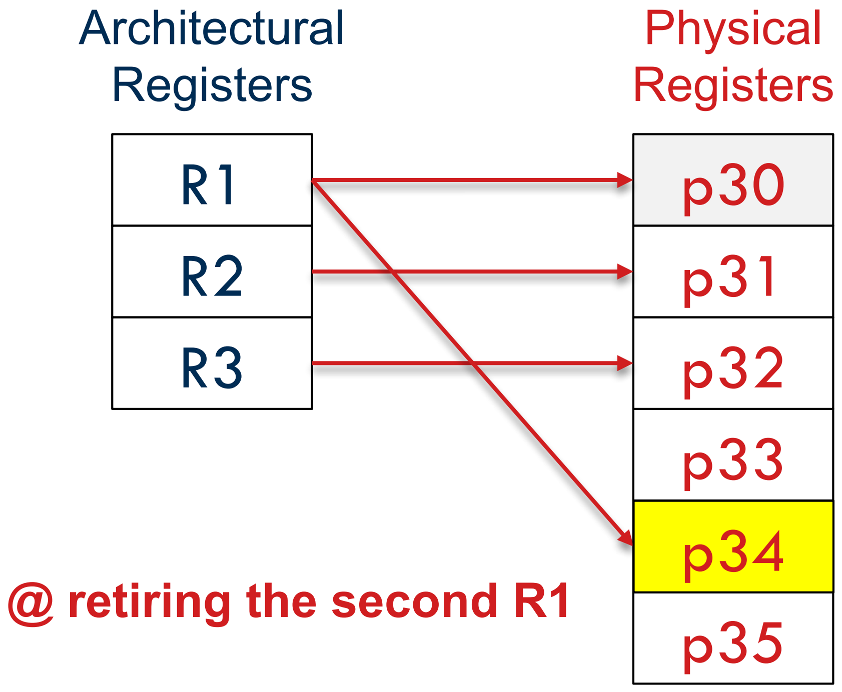
```
ADD  R1, R2, R3
SUB  R2, R1, R3
...
ADD  R3, R1, R2
...
SUB  R1, R3, R2
ADD R2, R1, R3
```



Physical Register Release

- Example: when is it safe to free p30 (R1)?

```
ADD  R1, R2, R3
SUB  R2, R1, R3
…
ADD  R3, R1, R2
…
SUB  R1, R3, R2
ADD R2, R1, R3
```


Architectural Registers
R1
R2
R3
Physical Registers
p30
p31
p32
p33
p34
p35
@ retiring the second R1