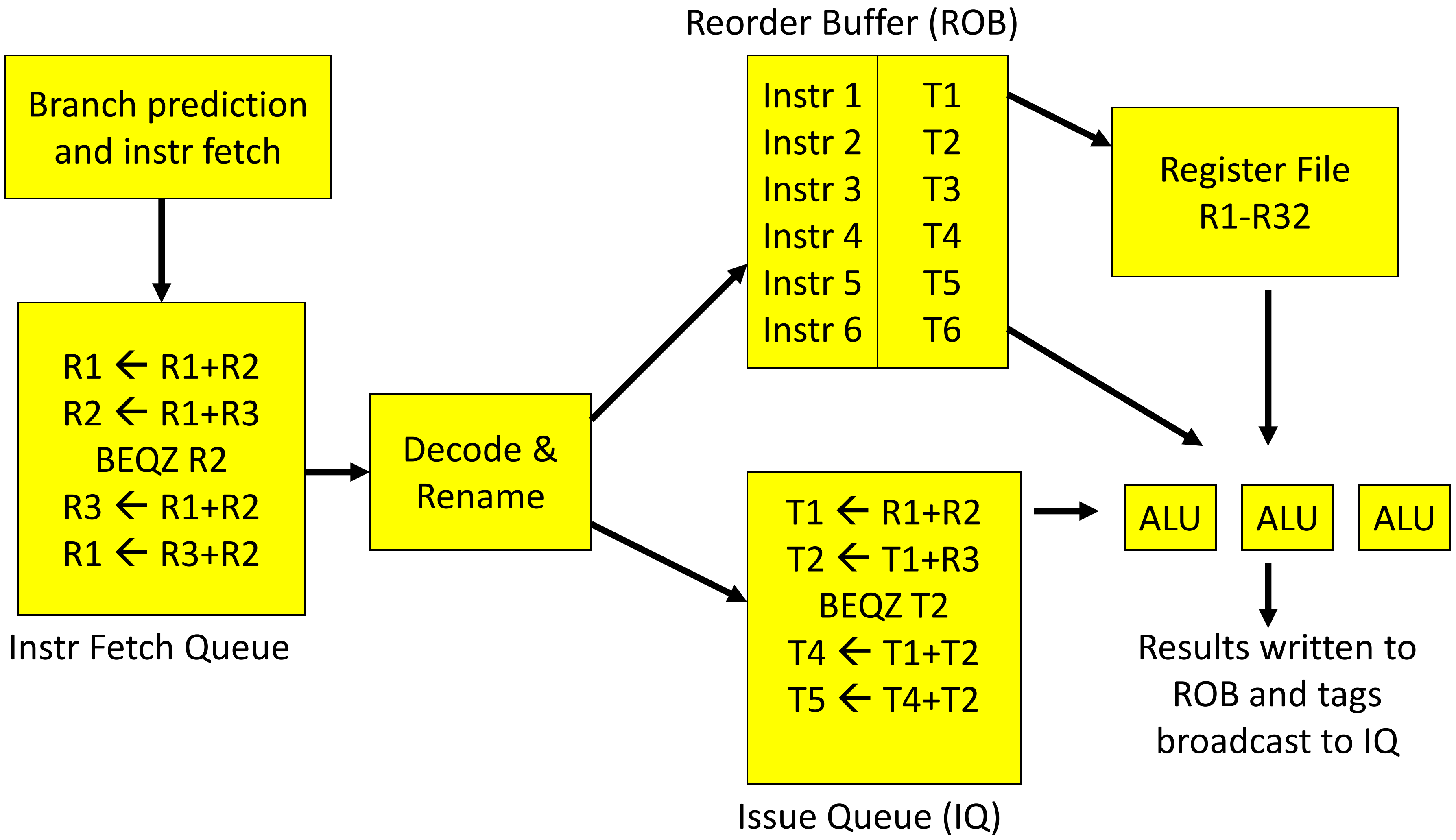
# Lecture: Out-of-order Processors

- Topics: out of order processor design details and examples

# An Out-of-Order Processor Implementation



## Problem 1

- Show the renamed version of the following code:
  Assume that you have 4 rename registers T1-T4

```
R1 ← R2+R3
R3 ← R4+R5
BEQZ  R1
R1 ← R1 + R3
R1 ← R1 + R3
R3 ← R1 + R3
```

## Problem 1

- Show the renamed version of the following code:
  Assume that you have 4 rename registers T1-T4

```
R1 ← R2+R3                T1 ← R2+R3
R3 ← R4+R5                T2 ← R4+R5
BEQZ  R1                  BEQZ  T1
R1 ← R1 + R3              T4 ← T1+T2
R1 ← R1 + R3              T1 ← T4+T2
R3 ← R1 + R3              T2 ← T1 +R3
```

## Design Details - I

- Instructions enter the pipeline in order
- No need for branch delay slots if prediction happens in time
- Instructions leave the pipeline in order – all instructions that enter also get placed in the ROB – the process of an instruction leaving the ROB (in order) is called commit – an instruction commits only if it and all instructions before it have completed successfully (without an exception)
- To preserve precise exceptions, a result is written into the register file only when the instruction commits – until then, the result is saved in a temporary register in the ROB

## Design Details - II

- Instructions get renamed and placed in the issue queue – some operands are available (T1-T6; R1-R32), while others are being produced by instructions in flight (T1-T6)

- As instructions finish, they write results into the ROB (T1-T6) and broadcast the operand tag (T1-T6) to the issue queue – instructions now know if their operands are ready

- When a ready instruction issues, it reads its operands from T1-T6 and R1-R32 and executes (out-of-order execution)

- Can you have WAW or WAR hazards? By using more names (T1-T6), name dependences can be avoided

## Design Details - III

- If instr-3 raises an exception, wait until it reaches the top of the ROB – at this point, R1-R32 contain results for all instructions up to instr-3 – save registers, save PC of instr-3, and service the exception

- If branch is a mispredict, flush all instructions after the branch and start on the correct path – mispredicted instrs will not have updated registers (the branch cannot commit until it has completed and the flush happens as soon as the branch completes)

- Potential problems: ?

## Managing Register Names

Temporary values are stored in the register file and not the ROB

$$\begin{tikzcd}\boxed{\text{Logical Registers R1-R32}} \arrow[r] & \boxed{\text{Physical Registers P1-P64}}\end{tikzcd}$$

At the start, R1-R32 can be found in P1-P32

Instructions stop entering the pipeline when P64 is assigned

$$\begin{tikzcd}\boxed{\begin{array}{c}\text{R1} \leftarrow \text{R1+R2}\\ \text{R2} \leftarrow \text{R1+R3}\\ \text{BEQZ R2}\\ \text{R3} \leftarrow \text{R1+R2}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{P33} \leftarrow \text{P1+P2}\\ \text{P34} \leftarrow \text{P33+P3}\\ \text{BEQZ P34}\\ \text{P35} \leftarrow \text{P33+P34}\end{array}}\end{tikzcd}$$

What happens on commit?

## The Commit Process

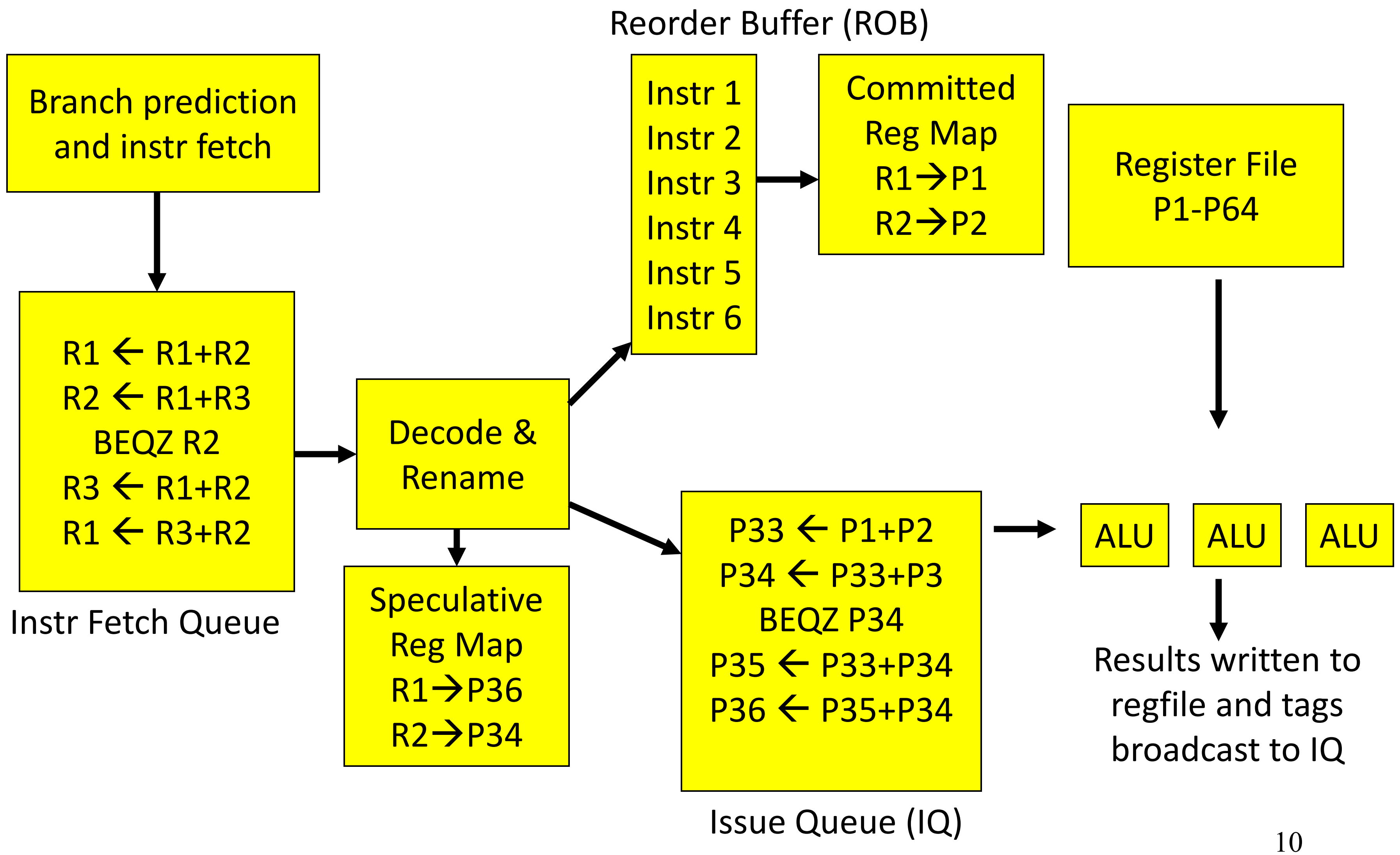
- On commit, no copy is required

- The register map table is updated – the "committed" value of R1 is now in P33 and not P1 – on an exception, P33 is copied to memory and not P1

- An instruction in the issue queue need not modify its input operand when the producer commits

- When instruction-1 commits, we no longer have any use for P1 – it is put in a free pool and a new instruction can now enter the pipeline → for every instr that commits, a new instr can enter the pipeline → number of in-flight instrs is a constant = number of extra (rename) registers

# The Alpha 21264 Out-of-Order Implementation



## Problem 3

- Show the renamed version of the following code: Assume that you have 36 physical registers and 32 architected registers. When does each instr leave the IQ?

```
R1 ← R2+R3
R1 ← R1+R5
BEQZ  R1
R1 ← R4 + R5
R4 ← R1 + R7
R1 ← R6 + R8
R4 ← R3 + R1
R1 ← R5 + R9
```

## Problem 3

- Show the renamed version of the following code:
  Assume that you have 36 physical registers and 32 architected registers. When does each instr leave the IQ?

|  |  |
|---|---|
| R1 ← R2+R3 | P33 ← P2+P3 |
| R1 ← R1+R5 | P34 ← P33+P5 |
| BEQZ R1 | BEQZ P34 |
| R1 ← R4 + R5 | P35 ← P4+P5 |
| R4 ← R1 + R7 | P36 ← P35+P7 |
| R1 ← R6 + R8 | P1 ← P6+P8 |
| R4 ← R3 + R1 | P33 ← P3+P1 |
| R1 ← R5 + R9 | P34 ← P5+P9 |

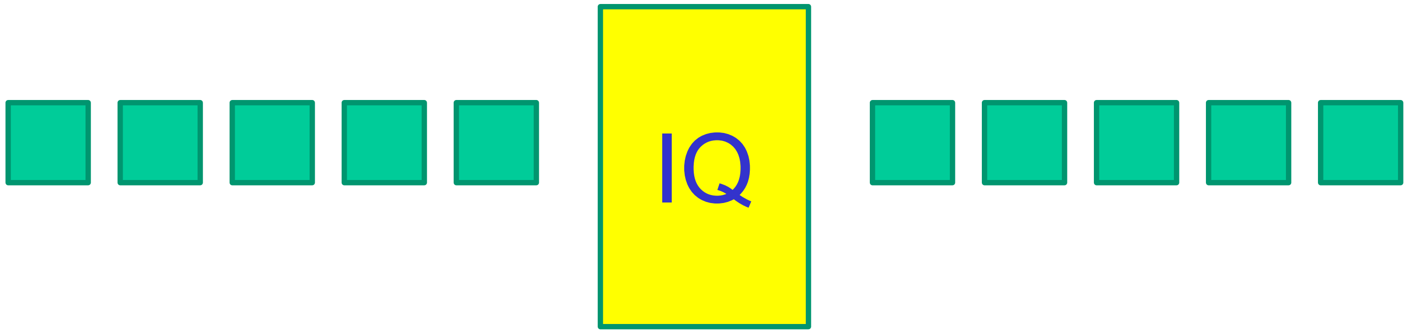
## Problem 3

- Show the renamed version of the following code:
  Assume that you have 36 physical registers and 32 architected registers. When does each instr leave the IQ?

| | | |
|---|---|---|
| R1 ← R2+R3 | P33 ← P2+P3 | cycle i |
| R1 ← R1+R5 | P34 ← P33+P5 | i+1 |
| BEQZ R1 | BEQZ P34 | i+2 |
| R1 ← R4 + R5 | P35 ← P4+P5 | i |
| R4 ← R1 + R7 | P36 ← P35+P7 | i+1 |
| R1 ← R6 + R8 | P1 ← P6+P8 | j |
| R4 ← R3 + R1 | P33 ← P3+P1 | j+1 |
| R1 ← R5 + R9 | P34 ← P5+P9 | j+2 |

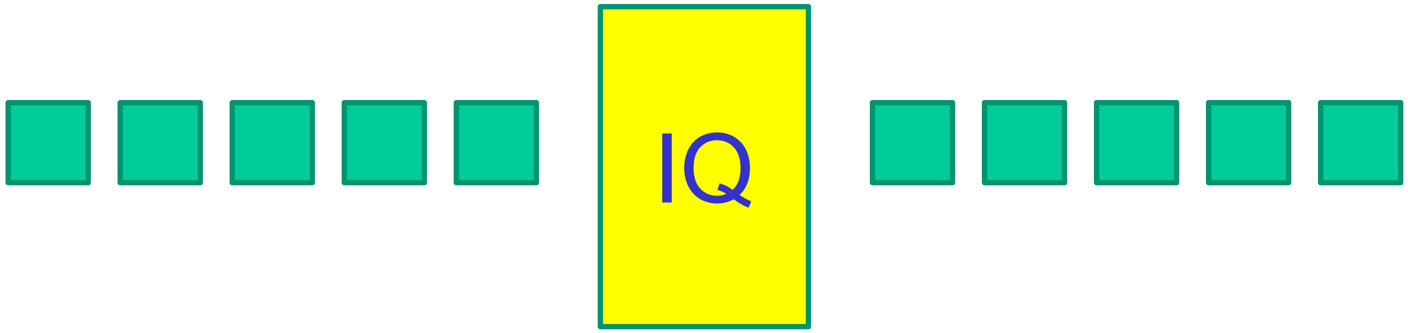
Width is assumed to be 4.
j depends on the #stages between issue and commit.

## OOO Example



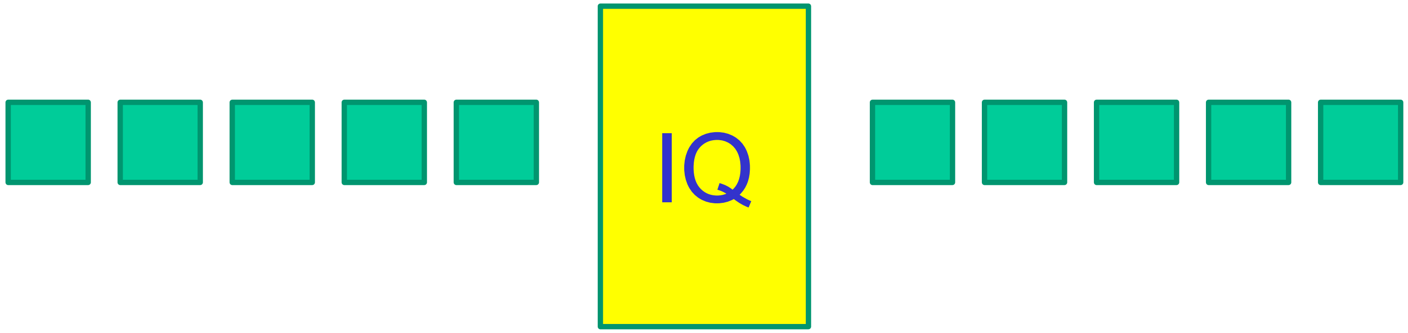
- Assume there are 36 physical registers and 32 logical registers, and width is 4
- Estimate the issue time, completion time, and commit time for the sample code

## Assumptions



- Perfect branch prediction, instruction fetch, caches
- ADD → dep has no stall; LD → dep has one stall
- An instr is placed in the IQ at the end of its 5th stage, an instr takes 5 more stages after leaving the IQ (ld/st instrs take 6 more stages after leaving the IQ)

## OOO Example

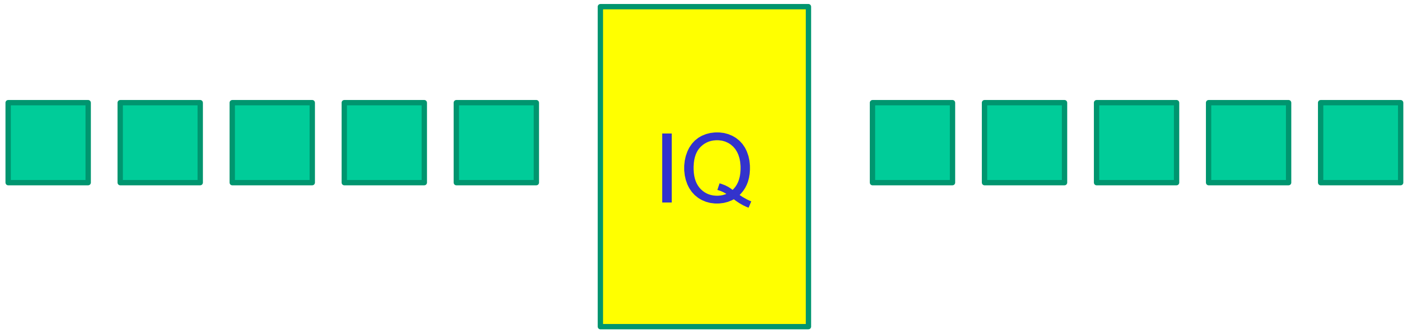


Original code

```
ADD   R1, R2, R3
LD    R2, 8(R1)
ADD   R2, R2, 8
ST    R1, (R3)
SUB   R1, R1, R5
LD    R1, 8(R2)
ADD   R1, R1, R2
```

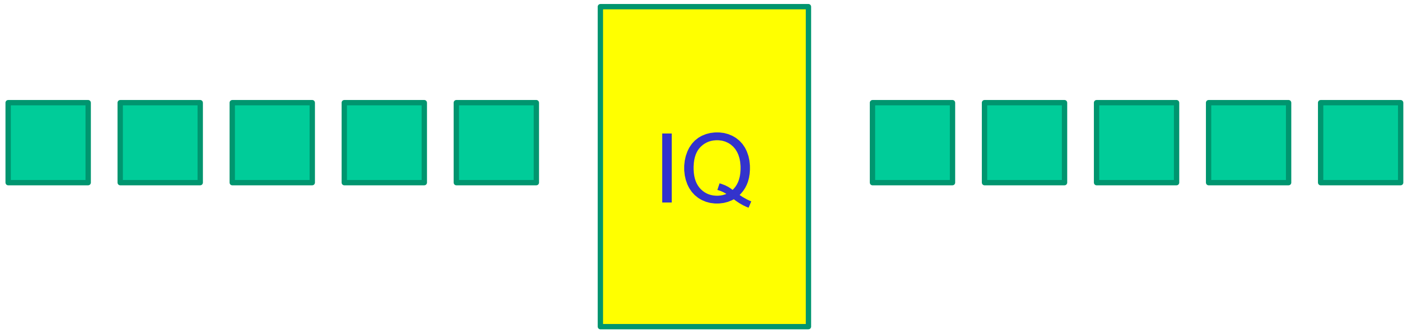
Renamed code

## OOO Example



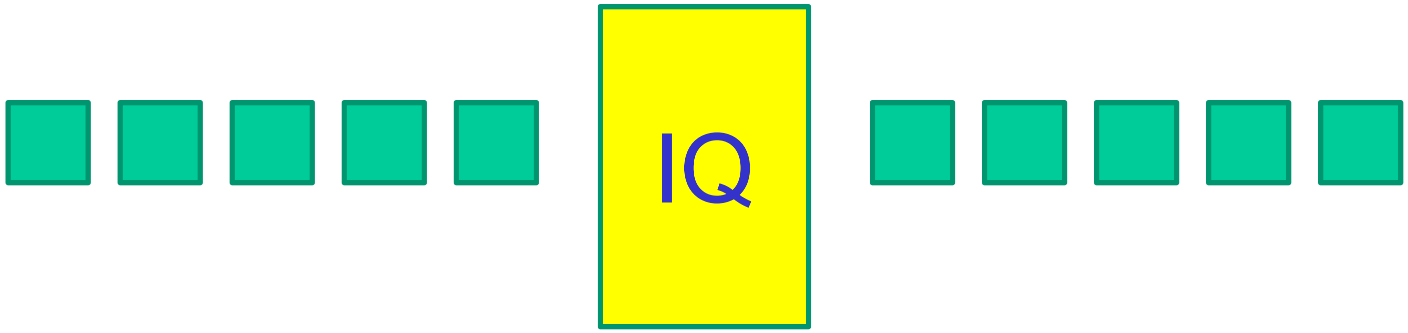
| Original code | | Renamed code |
|---|---|---|
| ADD R1, R2, R3 | | ADD P33, P2, P3 |
| LD R2, 8(R1) | | LD P34, 8(P33) |
| ADD R2, R2, 8 | | ADD P35, P34, 8 |
| ST R1, (R3) | | ST P33, (P3) |
| SUB R1, R1, R5 | | SUB P36, P33, P5 |
| LD R1, 8(R2) | Must wait | |
| ADD R1, R1, R2 | | |

## OOO Example



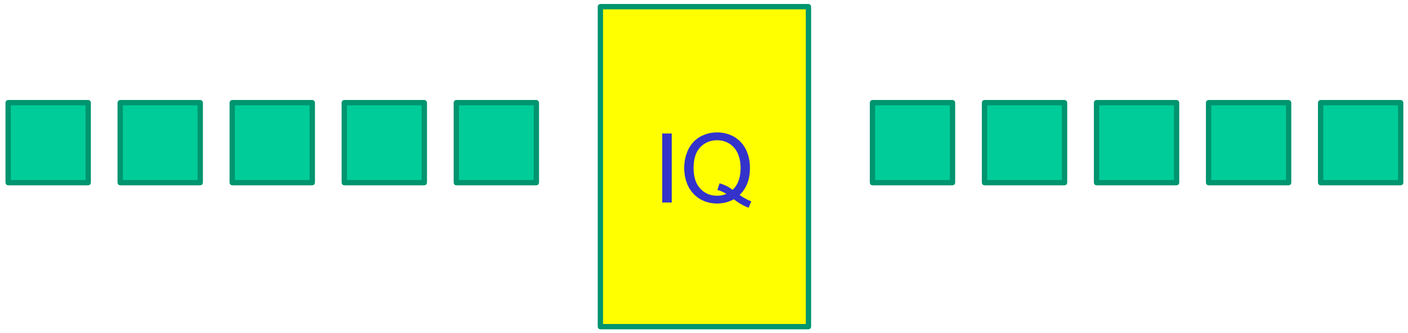
| Original code | Renamed code | InQ | Iss | Comp | Comm |
|---|---|---|---|---|---|
| ADD R1, R2, R3 | ADD P33, P2, P3 | | | | |
| LD R2, 8(R1) | LD P34, 8(P33) | | | | |
| ADD R2, R2, 8 | ADD P35, P34, 8 | | | | |
| ST R1, (R3) | ST P33, (P3) | | | | |
| SUB R1, R1, R5 | SUB P36, P33, P5 | | | | |
| LD R1, 8(R2) | | | | | |
| ADD R1, R1, R2 | | | | | |

## OOO Example



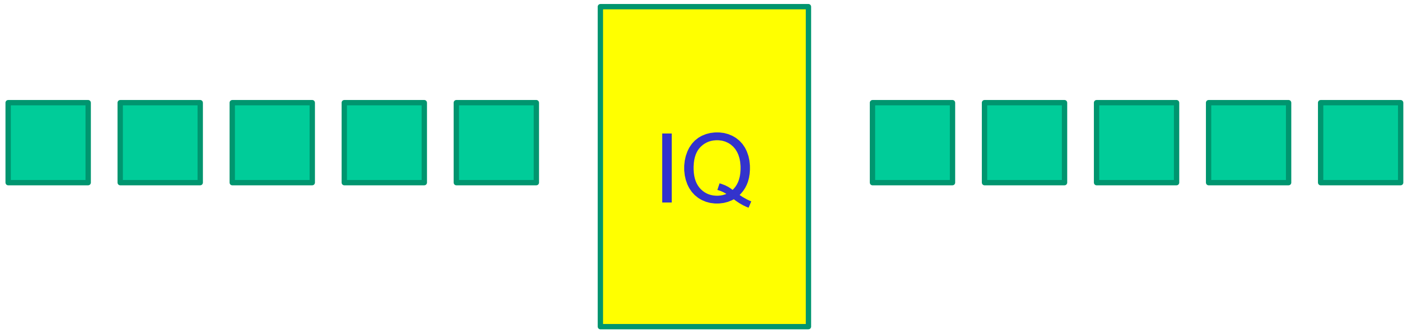
| Original code | | Renamed code | | InQ | Iss | Comp | Comm |
|---|---|---|---|---|---|---|---|
| ADD | R1, R2, R3 | ADD | P33, P2, P3 | i | i+1 | i+6 | i+6 |
| LD | R2, 8(R1) | LD | P34, 8(P33) | i | i+2 | i+8 | i+8 |
| ADD | R2, R2, 8 | ADD | P35, P34, 8 | i | i+4 | i+9 | i+9 |
| ST | R1, (R3) | ST | P33, (P3) | i | i+2 | i+8 | i+9 |
| SUB | R1, R1, R5 | SUB | P36, P33, P5 | i+1 | i+2 | i+7 | i+9 |
| LD | R1, 8(R2) | | | | | | |
| ADD | R1, R1, R2 | | | | | | |

## OOO Example



| Original code | | Renamed code | | InQ | Iss | Comp | Comm |
|---|---|---|---|---|---|---|---|
| ADD | R1, R2, R3 | ADD | P33, P2, P3 | i | i+1 | i+6 | i+6 |
| LD | R2, 8(R1) | LD | P34, 8(P33) | i | i+2 | i+8 | i+8 |
| ADD | R2, R2, 8 | ADD | P35, P34, 8 | i | i+4 | i+9 | i+9 |
| ST | R1, (R3) | ST | P33, (P3) | i | i+2 | i+8 | i+9 |
| SUB | R1, R1, R5 | SUB | P36, P33, P5 | i+1 | i+2 | i+7 | i+9 |
| LD | R1, 8(R2) | LD | P1, 8(P35) | i+7 | i+8 | i+14 | i+14 |
| ADD | R1, R1, R2 | ADD | P2, P1, P35 | i+9 | i+10 | i+15 | i+15 |

## OOO Example



| Original code | Renamed code | InQ | Iss | Comp | Comm | Prev Map |
|---|---|---|---|---|---|---|
| ADD R1, R2, R3 | ADD P33, P2, P3 | i | i+1 | i+6 | i+6 | P1 |
| LD R2, 8(R1) | LD P34, 8(P33) | i | i+2 | i+8 | i+8 | P2 |
| ADD R2, R2, 8 | ADD P35, P34, 8 | i | i+4 | i+9 | i+9 | P34 |
| ST R1, (R3) | ST P33, (P3) | i | i+2 | i+8 | i+9 | |
| SUB R1, R1, R5 | SUB P36, P33, P5 | i+1 | i+2 | i+7 | i+9 | P33 |
| LD R1, 8(R2) | LD P1, 8(P35) | i+7 | i+8 | i+14 | i+14 | P36 |
| ADD R1, R1, R2 | ADD P2, P1, P35 | i+9 | i+10 | i+15 | i+15 | P1 |

## Constraints Worth Remembering

- Don’t exceed rename width, issue width, commit width
- Make notes about a register’s previous mapping (so you can release it upon that instruction’s commit)
- Stall when out of registers
- Delay instructions with data dependences
- Factor in 5/6 stages for completion, depending on instr type
- InQ and Commit columns must monotonically increase; Issue and Complete times can be ooo

## Additional Details

- When does the decode stage stall? When we either run out of registers, or ROB entries, or issue queue entries
- Issue width: the number of instructions handled by each stage in a cycle. High issue width → high peak ILP
- Window size: the number of in-flight instructions in the pipeline. Large window size → high ILP
- No more WAR and WAW hazards because of rename registers – must only worry about RAW hazards

## Branch Mispredict Recovery

- On a branch mispredict, must roll back the processor state: throw away IFQ contents, ROB/IQ contents after branch
- Committed map table is correct and need not be fixed
- The speculative map table needs to go back to an earlier state
- To facilitate this spec-map-table rollback, it is checkpointed at every branch

## Waking Up a Dependent

- In an in-order pipeline, an instruction leaves the decode stage when it is known that the inputs can be correctly received, not when the inputs are computed

- Similarly, an instruction leaves the issue queue before its inputs are known, i.e., wakeup is speculative based on the expected latency of the producer instruction