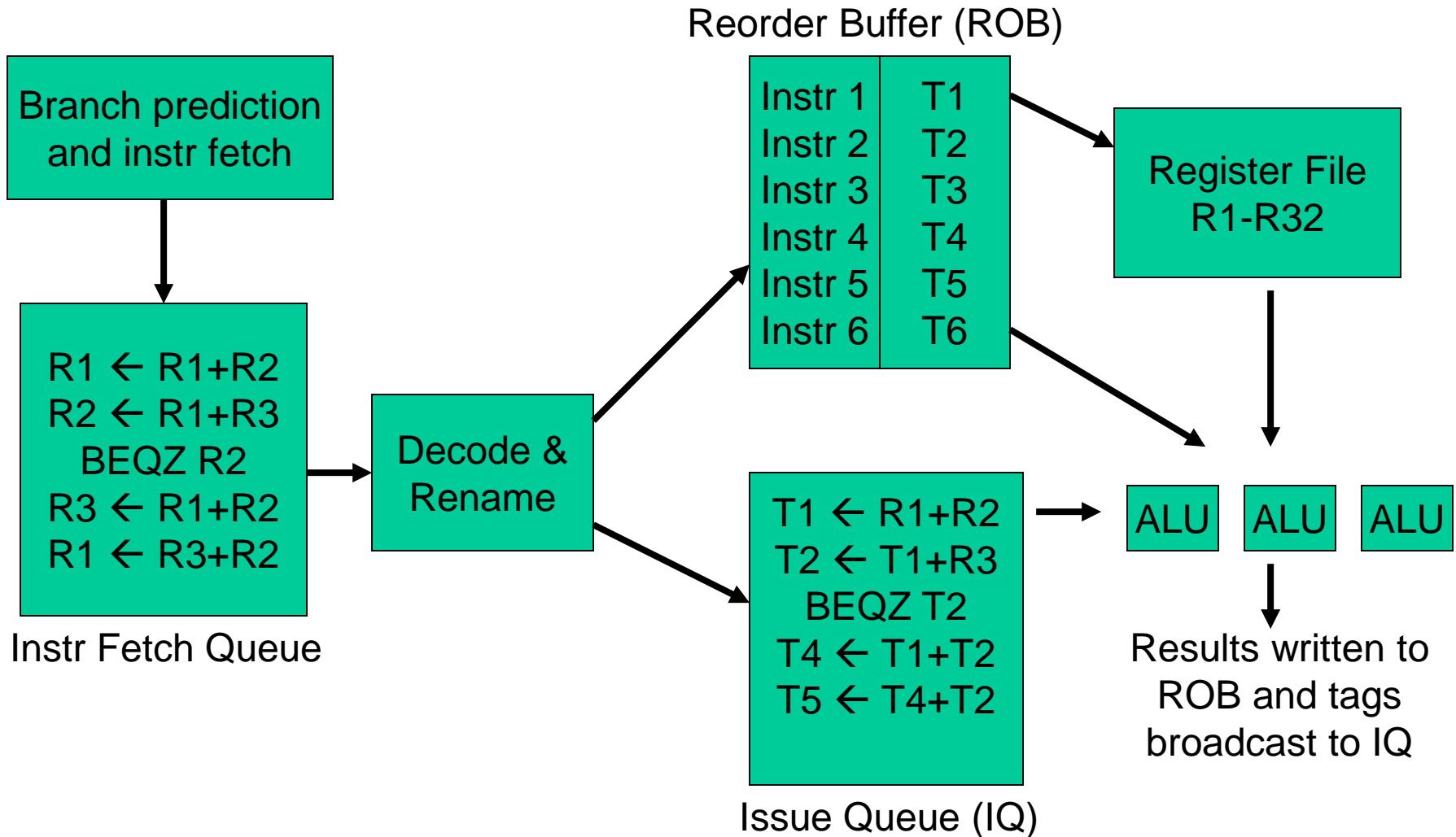


Lecture: Out-of-order Processors

- Topics: out-of-order implementations with issue queue, register renaming, and reorder buffer, timing, LSQ

An Out-of-Order Processor Implementation



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: ?

Problem 1

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

$R1 \leftarrow R2 + R3$

$R3 \leftarrow R4 + R5$

BEQZ R1

$R1 \leftarrow R1 + R3$

$R1 \leftarrow R1 + R3$

$R3 \leftarrow R1 + R3$

Problem 1

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

R1 \leftarrow R2+R3

R3 \leftarrow R4+R5

BEQZ R1

R1 \leftarrow R1 + R3

R1 \leftarrow R1 + R3

R3 \leftarrow R1 + R3

T1 \leftarrow R2+R3

T2 \leftarrow R4+R5

BEQZ T1

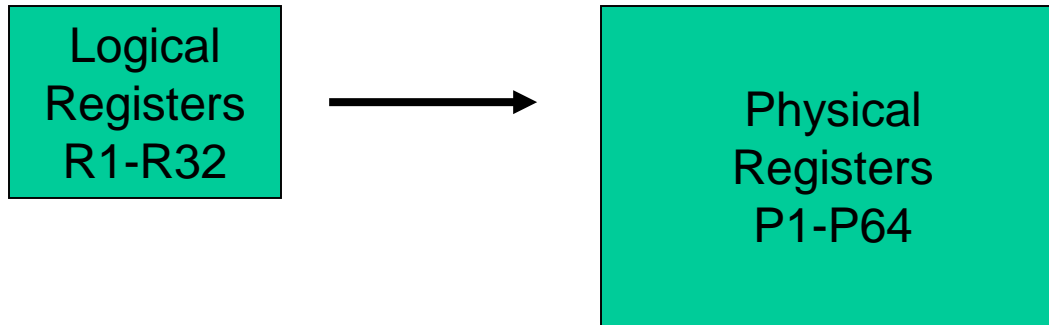
T4 \leftarrow T1+T2

T1 \leftarrow T4+T2

T2 \leftarrow T1 +R3

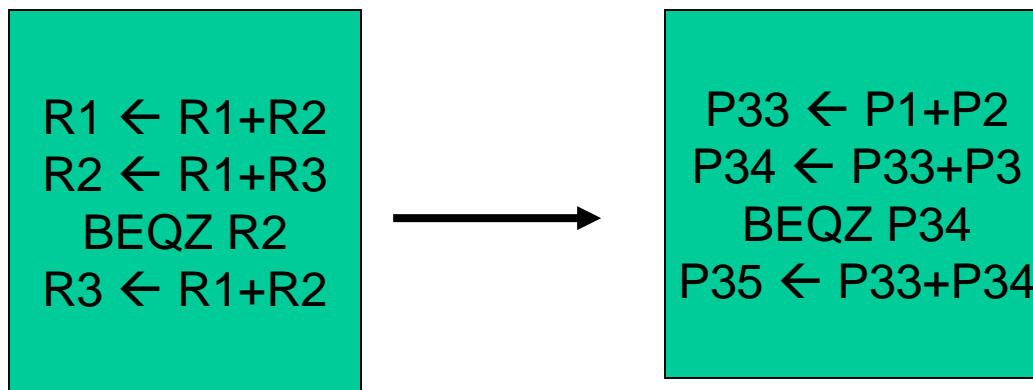
Managing Register Names

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



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

Instructions stop entering the pipeline when P64 is assigned

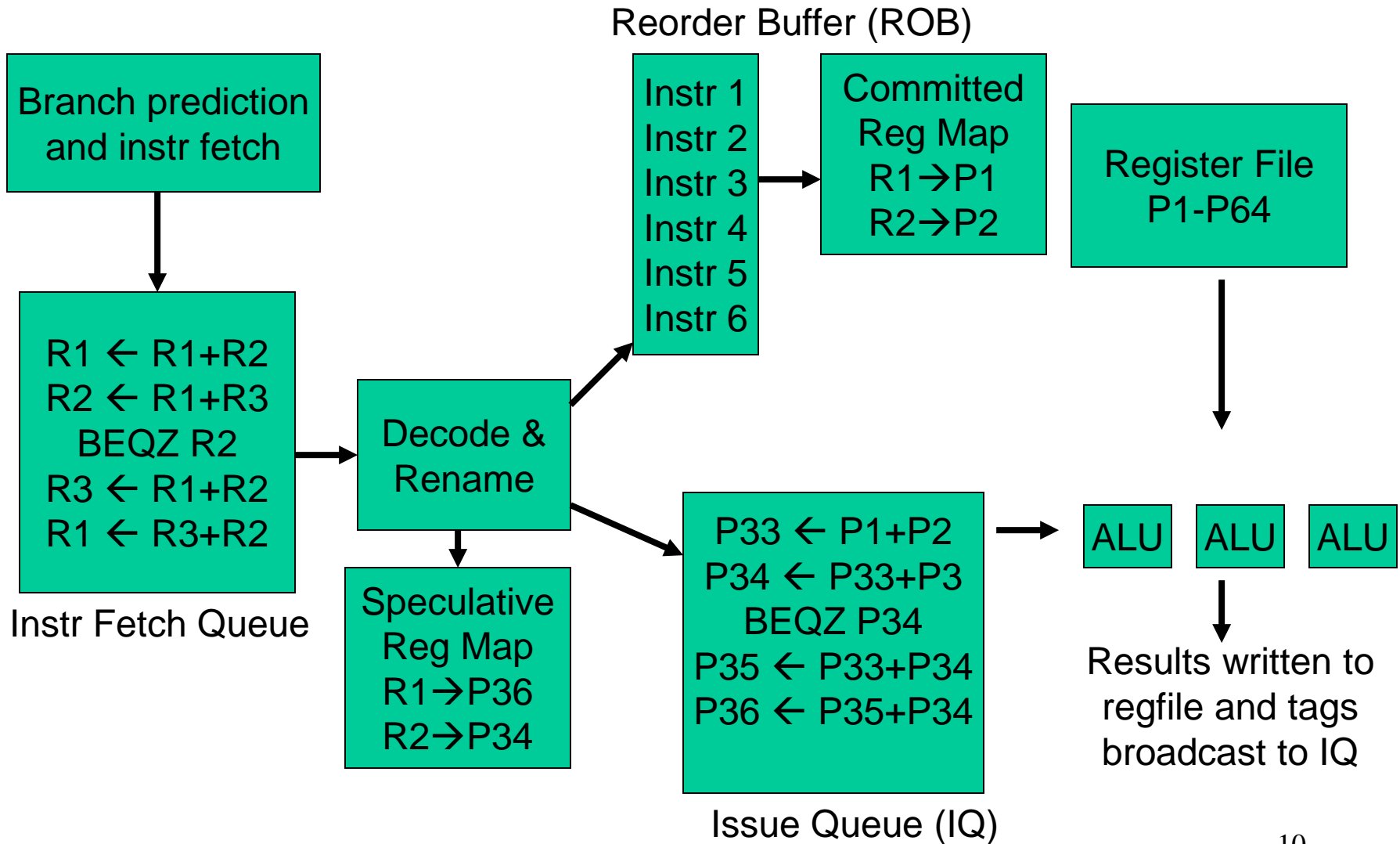


What happens on commit?

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 9

The Alpha 21264 Out-of-Order Implementation



Problem 2

- Show the renamed version of the following code:
Assume that you have 36 physical registers and 32 architected registers

$R1 \leftarrow R2 + R3$

$R3 \leftarrow R4 + R5$

BEQZ R1

$R1 \leftarrow R1 + R3$

$R1 \leftarrow R1 + R3$

$R3 \leftarrow R1 + R3$

$R4 \leftarrow R3 + R1$

Problem 2

- Show the renamed version of the following code:
Assume that you have 36 physical registers and 32 architected registers

R1 \leftarrow R2+R3

R3 \leftarrow R4+R5

BEQZ R1

R1 \leftarrow R1 + R3

R1 \leftarrow R1 + R3

R3 \leftarrow R1 + R3

R4 \leftarrow R3 + R1

P33 \leftarrow P2+P3

P34 \leftarrow P4+P5

BEQZ P33

P35 \leftarrow P33+P34

P36 \leftarrow P35+P34

P1 \leftarrow P36+P34

P3 \leftarrow P1+P36

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?

$R1 \leftarrow R2 + R3$

$R1 \leftarrow R1 + R5$

BEQZ R1

$R1 \leftarrow R4 + R5$

$R4 \leftarrow R1 + R7$

$R1 \leftarrow R6 + R8$

$R4 \leftarrow R3 + R1$

$R1 \leftarrow R5 + R9$

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 \leftarrow R2+R3

R1 \leftarrow R1+R5

BEQZ R1

R1 \leftarrow R4 + R5

R4 \leftarrow R1 + R7

R1 \leftarrow R6 + R8

R4 \leftarrow R3 + R1

R1 \leftarrow R5 + R9

P33 \leftarrow P2+P3

P34 \leftarrow P33+P5

BEQZ P34

P35 \leftarrow P4+P5

P36 \leftarrow P35+P7

P1 \leftarrow P6+P8

P33 \leftarrow P3+P1

P34 \leftarrow P5+P9

Problem 3

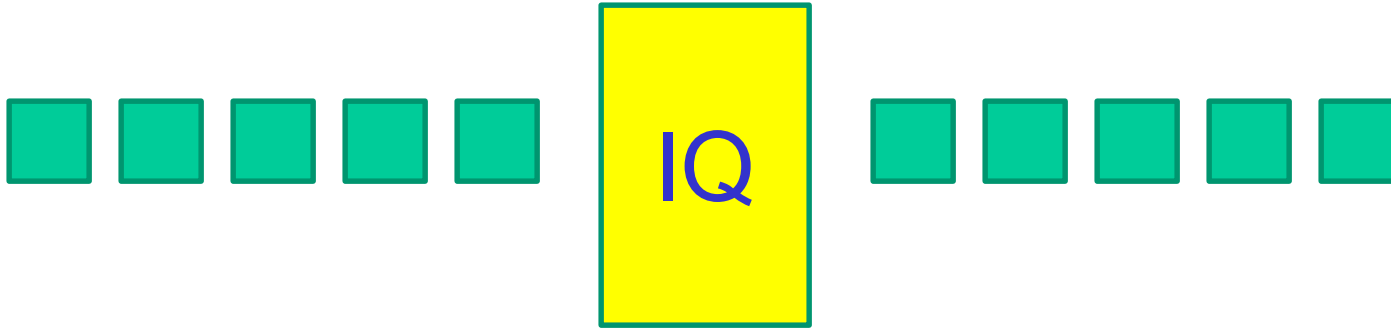
- Show the renamed version of the following code:
Assume that you have 36 physical registers and 32 architected registers. When does each instr leave the IQ?

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

Width is assumed to be 4.

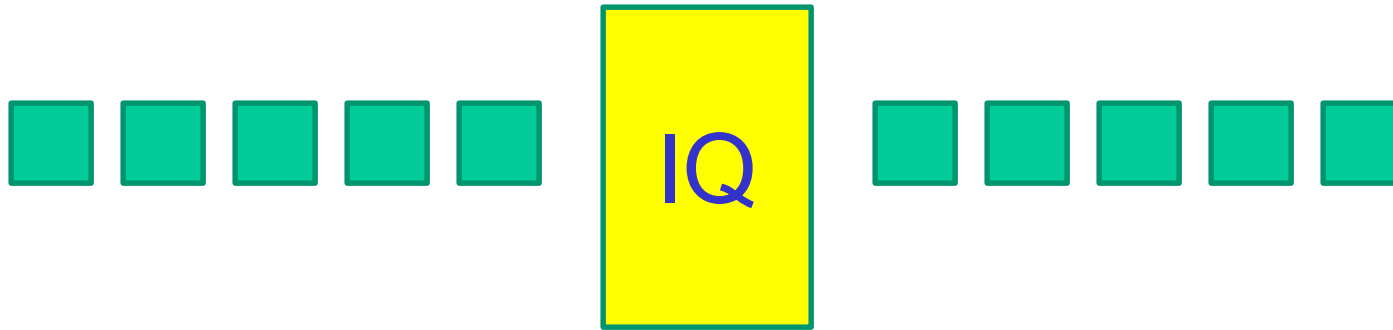
j depends on the #stages between issue and commit.

OOO Example



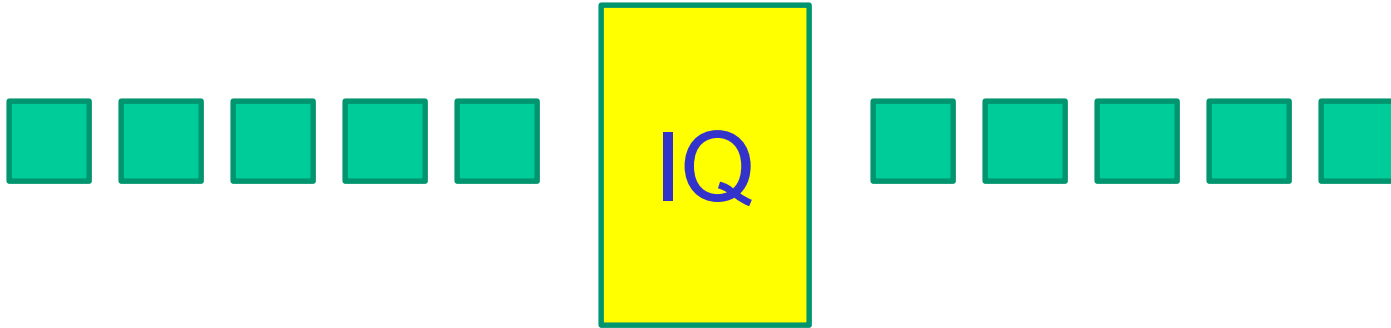
- Assume there are 36 physical registers and 32 logical registers, and width is 4
- Estimate the issue time, completion time, and commit time for the sample code

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

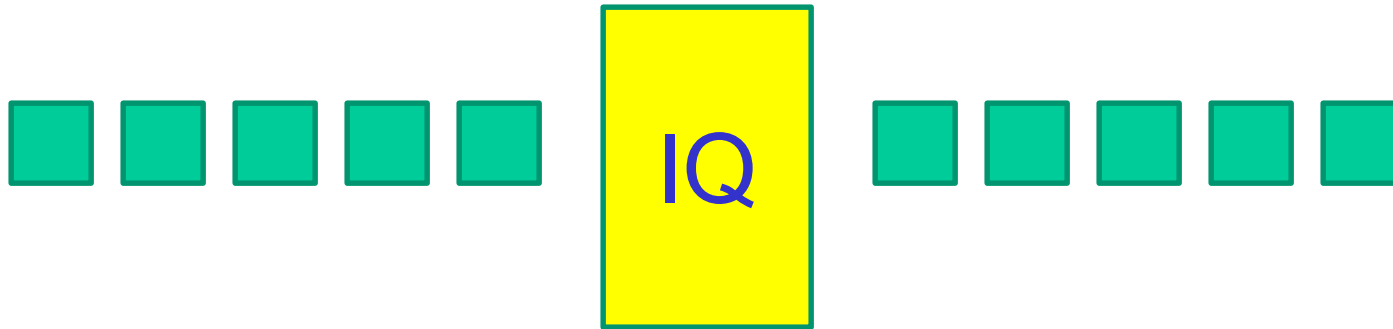


Original code

```
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

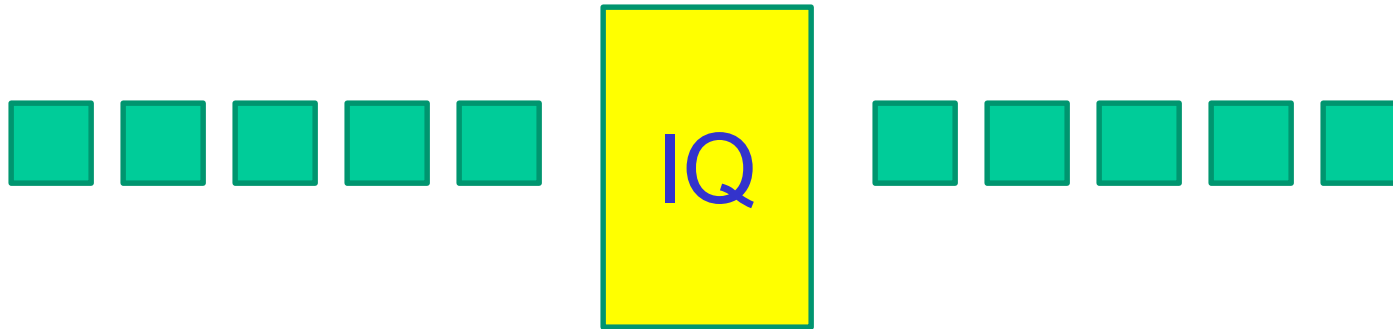
```
ADD  R1, R2, R3
LD   R2, 8(R1)
ADD  R2, R2, 8
ST   R1, (R3)
SUB  R1, R1, R5
LD   R1, 8(R2)
ADD  R1, R1, R2
```

Must wait

Renamed code

```
ADD  P33, P2, P3
LD   P34, 8(P33)
ADD  P35, P34, 8
ST   P33, (P3)
SUB  P36, P33, P5
```

OOO Example



Original code

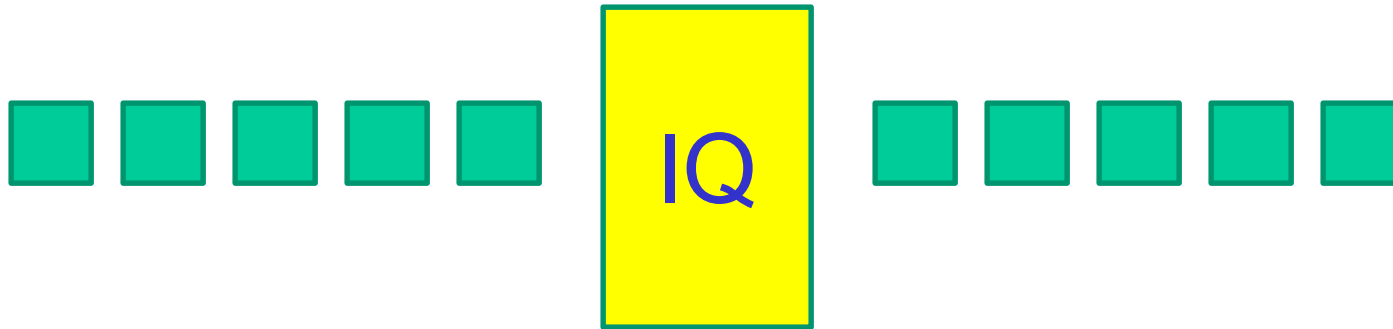
```
ADD  R1, R2, R3
LD   R2, 8(R1)
ADD  R2, R2, 8
ST   R1, (R3)
SUB  R1, R1, R5
LD   R1, 8(R2)
ADD  R1, R1, R2
```

Renamed code

```
ADD  P33, P2, P3
LD   P34, 8(P33)
ADD  P35, P34, 8
ST   P33, (P3)
SUB  P36, P33, P5
```

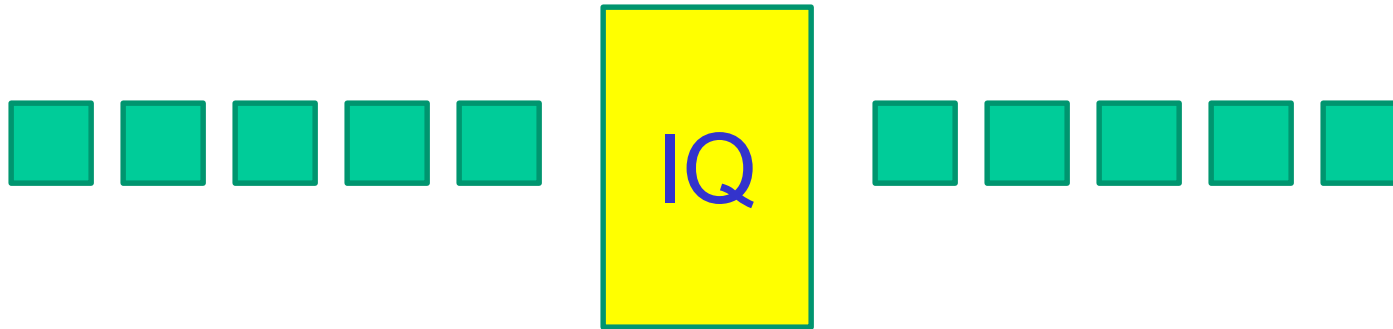
InQ Iss Comp Comm

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

Title

- Bullet