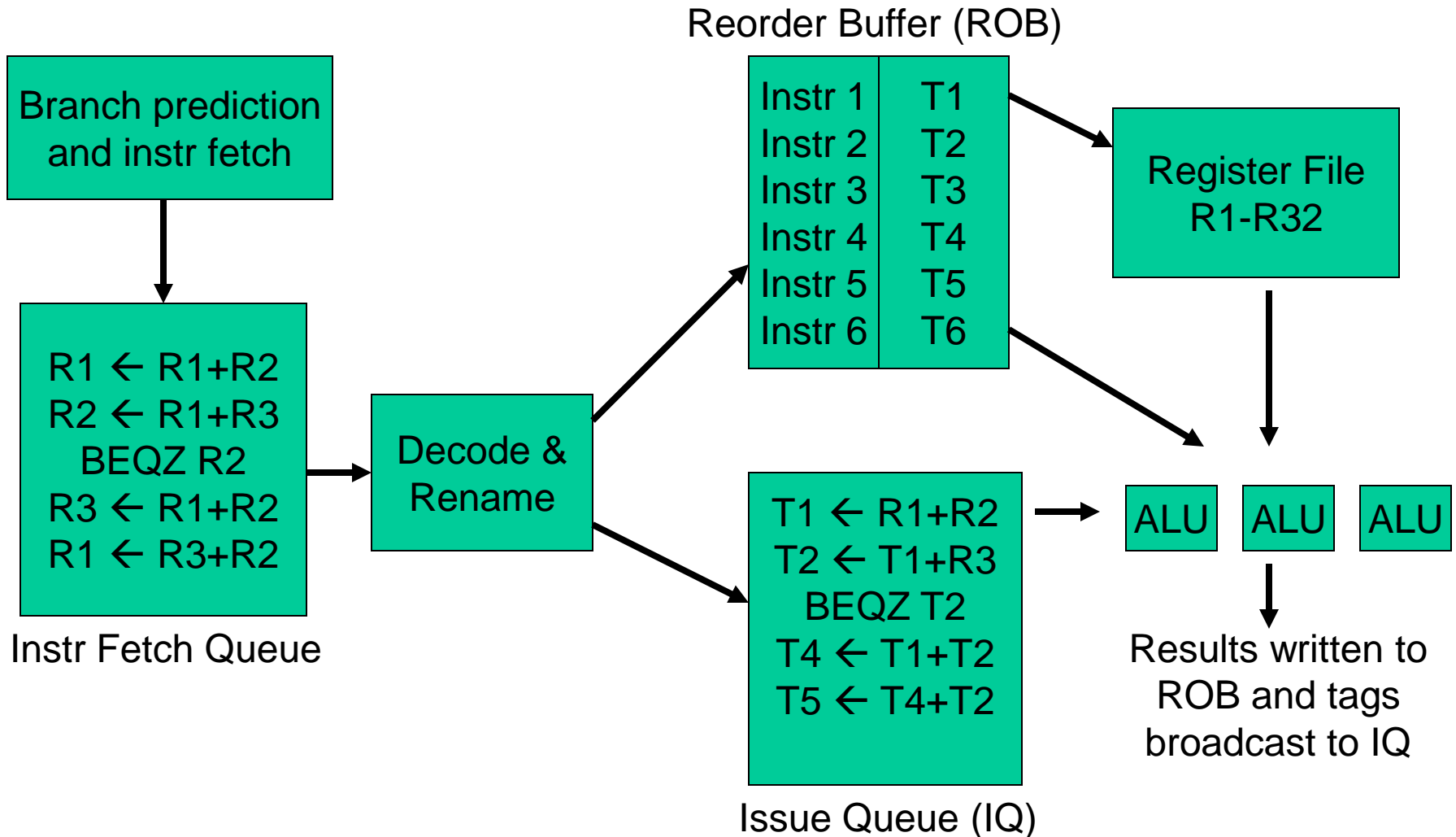# Lecture: Out-of-order Processors

- Topics: a basic out-of-order processor with issue queue,
  register renaming, and reorder buffer

An Out-of-Order Processor Implementation

Reorder Buffer (ROB)

Instr 1	T1
Instr 2	T2
Instr 3	T3
Instr 4	T4
Instr 5	T5
Instr 6	T6

Branch prediction and instr fetch

Register File
R1-R32

R1 ← R1+R2
R2 ← R1+R3
BEQZ R2
R3 ← R1+R2
R1 ← R3+R2

Decode & Rename

T1 ← R1+R2
T2 ← T1+R3
BEQZ T2
T4 ← T1+T2
T5 ← T4+T2

ALU ALU ALU

Instr Fetch Queue

Results written to ROB and tags broadcast to IQ

Issue Queue (IQ)

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

3

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: ?
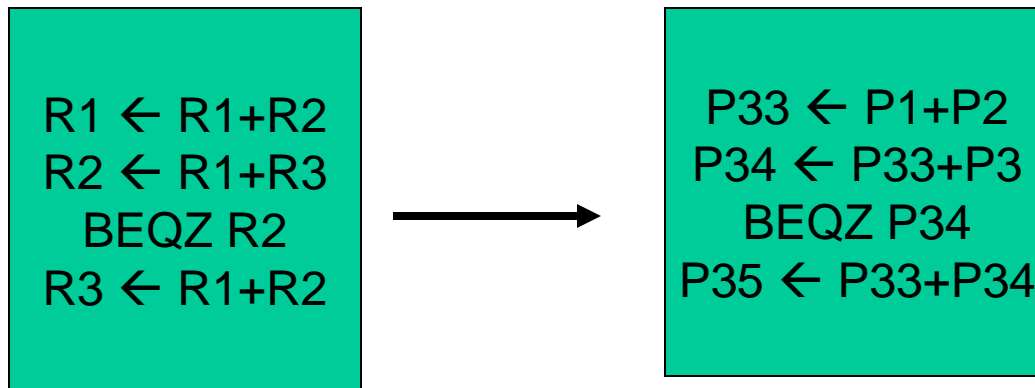
Managing Register Names

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

Logical Registers R1-R32	→	Physical Registers P1-P64
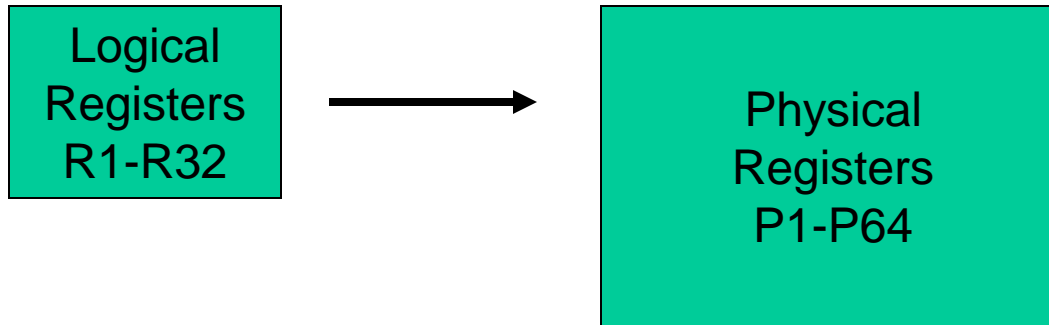
At the start, R1-R32 can be found in P1-P32
Instructions stop entering the pipeline when P64 is assigned

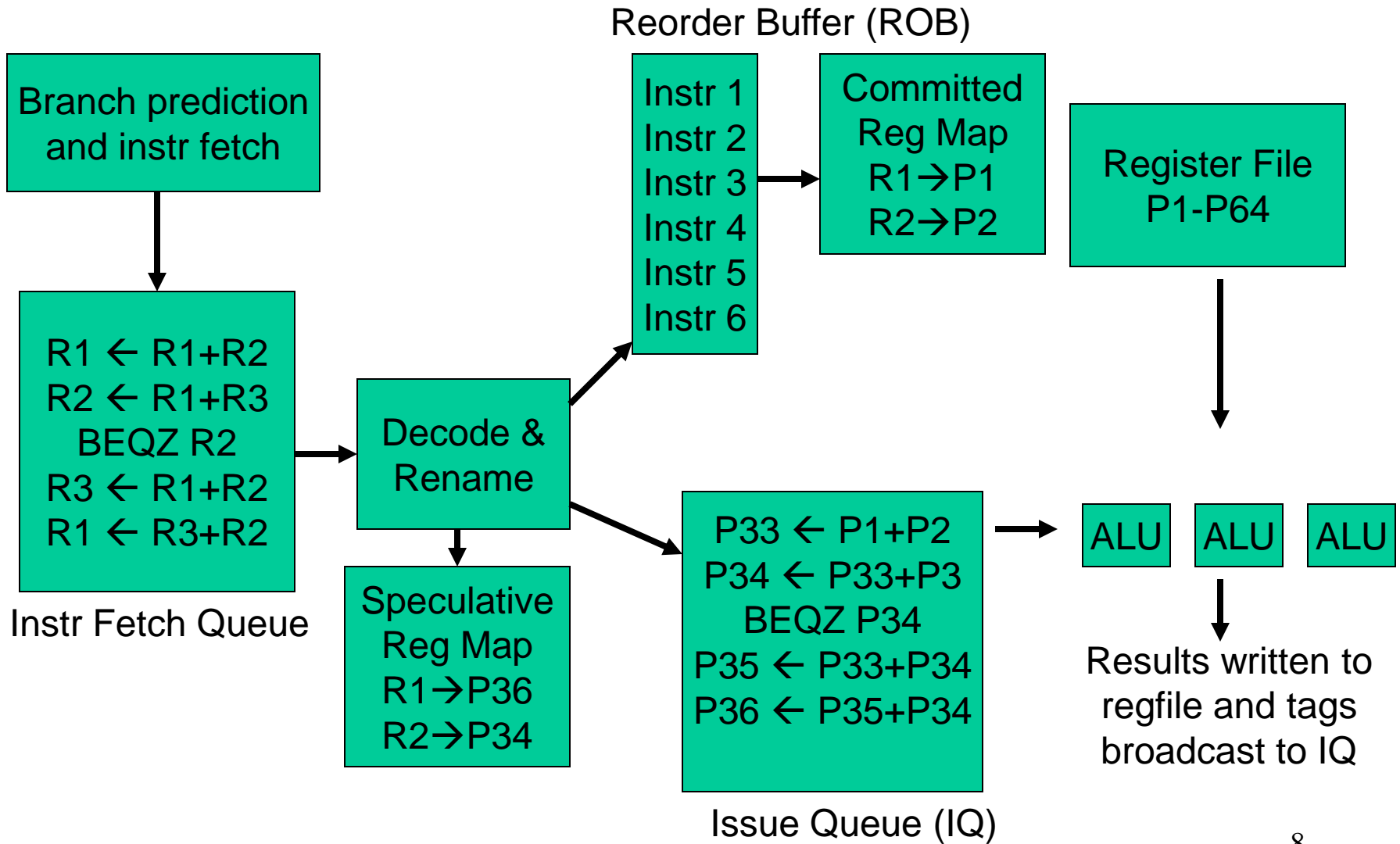| R1 ← R1+R2 R2 ← R1+R3 BEQZ R2 R3 ← R1+R2	→	P33 ← P1+P2 P34 ← P33+P3 BEQZ P34 P35 ← P33+P34

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

Reorder Buffer (ROB)

Branch prediction and instr fetch

| Instr 1 |
| Instr 2 |
| Instr 3 |
| Instr 4 |
| Instr 5 |
| Instr 6 |

Committed Reg Map
R1→P1
R2→P2

Register File P1-P64

R1 ← R1+R2
R2 ← R1+R3
BEQZ R2
R3 ← R1+R2
R1 ← R3+R2

Instr Fetch Queue

Decode & Rename

Speculative Reg Map
R1→P36
R2→P34

P33 ← P1+P2
P34 ← P33+P3
BEQZ P34
P35 ← P33+P34
P36 ← P35+P34

ALU   ALU   ALU

Results written to regfile and tags broadcast to IQ

Issue Queue (IQ)

8

Title

- Bullet