


# **Speculative execution; Intro to multiple-issue**



# Review: Tomasulo's

## Issue

If reservation station available, issue to station; tracking source/dest data

Source can be register (in which case, value is available) or reservation station (in which case, update res. station tags to wait for that result)

If dest is register, update register file tags to say data will come from this res. station

Reservation stations provide HW registers  
Q fields in reservation stations/register file tags play role  
of reg. renaming

## Execute

If operands available, execute instr; otherwise wait

## Write

Once result is ready, send on CDB (update registers, reservation stations)

# Example 3: two stores





What hazards arise from reordering stores?  
How can we modify Tomasulo's to resolve the hazards?



# Speculation and the ROB

Tomasulo's allows instructions to execute and be committed out of order

Handles WAW, WAR hazards for registers... more complicated for stores

Cannot handle branches very well

What if instructions could execute out of order but had to commit in-order?

Re-order buffer (ROB) helps us do this

Stores instructions in the order they are issued

Only commits (writes result of) instruction when it's next in the ROB

Allows us to execute before we know the result of branch (hence speculative)

# The speculative CPU with ROB



# Four stages of execution

## Issue

If reservation station and ROB available, issue to both; update control entries

## Execute

If operands available, execute instr; otherwise wait (for stores: this stage only computes effective address)

## Write

When result is available, send on CDB (update ROB, reservation stations)

## Commit

If normal commit or store: update register/memory and remove instr from ROB

If incorrectly predicted branch: flush ROB and reservation stations, fetch correct instr



# Example 4: speculative branch





What is challenging about OOO/speculative execution for CISC architectures?

What can be done at the uarch level to account for this?

# From ISA lecture: micro-ops

Translation of complex machine instruction (macro-op) to multiple steps

Microarchitecture dependent (Intel doesn't provide documentation on this)

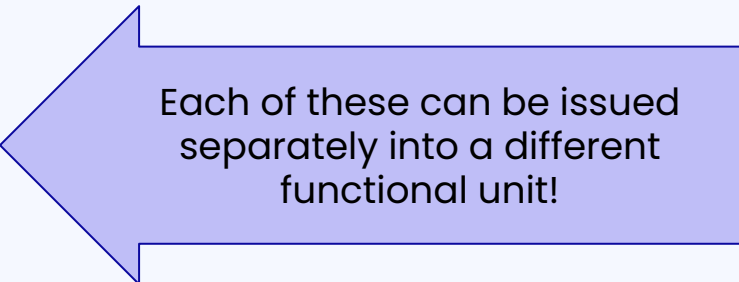
For example, `addq 8(%rdi) %rax` might be translated into:

add 8 to rdi

load that address from memory

add that value to rax

store the result in rax



Each of these can be issued separately into a different functional unit!

x86 processors have had a uop decoder since the Pentium Pro (1997)

Even RISC processors decode into uops – driven by design of FUs



What is the theoretical best CPI for our OOO  
CPU (with or without speculation)?

# Multiple-issue

Allows for multiple instructions to be issued at the same time

Dynamically (by processor): superscalar

Multiple variations: in-order, OOO, OOO + speculative

Statically (by compiler): Very Long Instruction Word (VLIW), EPIC

We'll come back to this very soon



Potentially how many instructions can we issue at once if we have the following FUs:

- 1 load
- 1 store
- 1 integer ALU
- 1 FP add/sub
- 1 FP mul/div

# Issuing two instrs at once

lw t0 8(s0)

add t2, t0, t1

- What do the reservation stations/ROB look like?
- What does the hardware need to check in a *single cycle*?

# Superscalar steps

1. Make sure there is room in the ROB (if speculative) and a reservation station for every instruction that *might* be in the next issue bundle (if necessary, bundles can be broken)
2. Analyze all dependences between instructions in bundle (**done in hardware in one cycle – HW grows quadratically in complexity w/ bundle size!**)
3. Update reservation stations info and ROB entries for all instructions in bundle and send them off to the FUs



# Example

loop:

```
lw t0, 0(s0)
```

```
addi t0, t0, 1
```

```
sw t0, 0(s0)
```

```
addi s0, s0, 8
```

```
bne t0, t1, loop
```