Data structures for Tomasulo’s algorithm

• **Instruction status:** Which stage the instruction is in.

• **Functional unit status:** “Busy” \( \Rightarrow \) FU is busy executing an instruction

• **Reservation-station status**
  - Busy – reservation station is in use
  - Op – what instruction is the FU busy with
  - \( F_i \) – destination register
  - \( V_j, V_k \) – source values
  - \( Q_j, Q_k \) – functional units producing src values

• **Register result status:** Which FU is going to write each register.

The memory unit has three sets of reservation stations, not one:

• The *load* reservation stations (for pending loads).
• The *store* reservation stations (for pending stores).
• The *conflict queue*, for later instructions with the same addresses as pending loads and stores.

Here is how the conflict queue is used.

• When a new memory-reference instruction \( i \) is issued, its effective address (if available) is checked against addresses in the *load* and *store* reservation stations.

• If there is a conflict with some instruction \( j \), then \( i \) is issued and is queued in the conflict queue.

• Also, if \( i \) uses an address based on an index register that is not ready, \( i \) is queued in the conflict queue.

• When \( j \) eventually executes (and any index registers involved become ready), the conflict disappears. Then \( i \) is transferred to a *load or store* reservation station.
Thus, instructions from the load and store reservation stations can be processed independently and asynchronously. (No two instructions simultaneously in these units ever have the same address.)

However, it is simplest to process instructions in the conflict queue in order of arrival.

The 360/91, though, had complex hardware to allow instructions from the conflict queue to be processed out of order. Only two or more requests for the same address were kept in sequence.

*Examples of Tomasulo’s algorithm*

I want to go through a couple of examples from http://www.cs.arizona.edu/classes/cs576/fall02/Lecture11.ppt.

Although these are very good examples, they use three rules that are different from the ones we have been using:

- IF+ID+IS takes only one cycle (!).
- An instruction that is waiting on an operand does not begin execution until the cycle after the operand is written back (WB stage).
- The time to complete a load may vary, depending on whether there is an L1 or L2 cache miss.

These three rules apply *only* to these examples. They do *not* apply to any homework or exam questions!

The figure at the top of the next page shows the schedule that would be achieved using our rules.
Cost of register renaming

The major drawback of Tomasulo’s algorithm is that it requires a lot of hardware.

When a tag comes across the CDB, each reservation station and each register must check to see if the tag belongs to it.

This is basically an associative memory—the value coming across the bus will be stored anywhere there is a matching tag.

How could we reduce the cost of this hardware?

This would essentially give us Thornton’s algorithm with reservation stations:

- Instructions could be dispatched if reservation stations were available, even if the FU was not free.
- This allows several instructions to await operands at the functional unit.

Example:  

MUL.D  F1, F2, F3
MUL.D  F4, F5, F6
How do reservation stations help here?

Values can be copied into reservation stations. This helps avoid WAR hazards.

Example:  
```
DADD     R1, R2, R3
DADDI    R2, R4, #1
```

Assume R2 and R4 are ready, but not R3.

How do reservation stations help here?

Simulations by Weiss and Smith showed that this scheme produced slightly more than half as much speedup as Tomasulo’s algorithm (vs. the CDC 6600).

**Precise interrupts**

Out-of-order execution and interrupts don’t mix! When an interrupt occurs, O/S wants *precise state*.

*Precise state* means—

- All instructions *before* the instruction causing the interrupt have completed
- All instructions *after* have not completed

Why are precise interrupts important?

- The OS needs to save the state of the process and restart the process after servicing the interrupt.  
  Restart occurs from the PC of interrupted instruction  
  The restart state must reflect only changes up to that PC!
- Programmer debugging – user presumes sequential program
Why is OOO execution bad for precise interrupts?

Three categories of interrupts

“External interrupts.”

- I/O-device request
- Timer interrupt
- Power failing

“Exceptions”

- Arithmetic overflow, divide-by-0, etc.
- Page fault

“O/S calls”

- Also called system call or trap
- Initiated via an explicit instruction in ISA

Synchronous vs. asynchronous interrupts\(^1\)

- A *synchronous* interrupt is a function of program and memory state.
- An *asynchronous* interrupt happens independent of the program/memory state.

External interrupts are ______________, exceptions are __________, O/S calls are ______________ and user-initiated

User request vs. coerced interrupts

- From user program, e.g.,
- From OS or hardware, e.g.,

User-maskable vs. nonmaskable interrupts

- User maskable—can be (temporarily) ignored, e.g.,

---

\(^1\) This classification scheme is due to Hill, Smith, Sohi, and Wood of the University of Wisconsin (ECE/CS 752 lecture notes).
- Nonmaskable—must be handled, e.g.,

*Within an instruction vs. between instructions*
- Within—must be dealt with to complete an instruction, e.g.,
- Between—not part of an instruction, e.g.,

*Resume vs. Terminate*
- Resume—must transparently return to user process, e.g.,
- Terminate—give up and die, e.g.,