Hardware-Based Speculation

As we exploit more instruction-level parallelism, control dependencies become increasingly limiting. Branch prediction alone may not generate sufficient ILP for wide-issue processors that need to execute a branch every cycle.

Hardware speculation overcomes these limitations by executing instructions as if branch predictions were always correct, with mechanisms to handle incorrect speculation.

1. **Dynamic Branch Prediction**

   Chooses which instructions to execute

2. **Speculative Execution**

   Allows execution before control dependencies are resolved, with ability to undo incorrect speculation

3. **Dynamic Scheduling**

   Handles scheduling of different basic block combinations

This approach essentially implements data flow execution: operations execute as soon as their operands are available.

Extending Tomasulo's Algorithm for Speculation

- To support speculation, we must separate result bypassing (needed for speculative execution) from instruction completion. This separation allows an instruction to execute and bypass results without performing irreversible updates until we know the instruction is no longer speculative.
- The key insight: allow instructions to execute out of order but force them to commit in order, preventing any irrevocable action (state updates or exceptions) until an instruction commits.
- This requires an additional hardware buffer—the reorder buffer (ROB)—to hold results between execution completion and instruction commit.

Reorder Buffer (ROB)

1

Purpose

Holds instruction results between execution completion and commit

Serves as a source of operands for instructions, similar to reservation stations

Integrates the function of the store buffer from Tomasulo's algorithm

2

Structure

Each ROB entry contains four fields:

- Instruction type (branch, store, or register operation)
- Destination field (register number or memory address)
- Value field (holds result until commit)
- Ready field (indicates completion status)

Unlike Tomasulo's algorithm where register files are updated immediately, with speculation, registers are only updated when instructions commit (when we know definitively they should execute).

Instruction Execution Steps with Speculation

Issue

Get instruction from queue if there's an empty reservation station and ROB slot

Send available operands from registers or ROB to reservation station

Update control entries and allocate ROB entry for the result

Execute

Monitor CDB for unavailable operands (checking for RAW hazards)

When both operands are available, execute the operation

For stores, calculate effective address at this stage

Write Result

Write result to CDB (with ROB tag) and from CDB into ROB

Update any waiting reservation stations

For stores, write value to be stored into ROB entry if available

Commit

Update register/memory with result when instruction reaches ROB head

For incorrect branch predictions, flush ROB and restart at correct path

Remove instruction from ROB after commit

Precise Exceptions with Speculation

Key Difference from Tomasulo's Algorithm

With speculation, no instruction after the earliest uncompleted instruction is allowed to complete. This enables precise exceptions.

If an instruction causes an exception, we wait until it reaches the ROB head and then take the interrupt, flushing any pending instructions.

Handling Exceptions

- Exceptions are recorded in the ROB but not recognized until commit
- If a speculated instruction raises an exception but shouldn't have executed (due to branch misprediction), the exception is flushed
- Only when an instruction reaches the ROB head do we know it's no longer speculative and the exception should be taken

This approach maintains a precise interrupt model while allowing dynamic execution—a significant advantage over basic Tomasulo's algorithm where instructions might complete out of order.

Memory Hazard Handling with Speculation

Store Instruction Handling

Unlike in Tomasulo's algorithm, stores update memory only when they reach the ROB head, ensuring memory isn't updated until instructions are no longer speculative.

WAW and WAR Hazards

Eliminated through memory because actual memory updates occur in order when a store reaches the ROB head.

RAW Hazards

Maintained by two restrictions:

1. Not allowing loads to execute if any active store has a matching destination
2. Maintaining program order for effective address computation

Some processors bypass values directly from store to load when RAW hazards occur.

Example

Reorder Buffer

#	Busy	Instruction		State	Dest.	Value
1		L.D	F6,32(R2)	Commit	F6	M[32+R[2]]
2		L.D	F2,44(R3)	Commit	F2	M[44+R[3]]
3	Y	MUL.D	F0,F2,F4	Execute	F0	
4	Y	SUB.D	F8,F2,F6	Write result	F8	#2 - #1
5	Y	DIV.D	F10,F0,F6	Execute	F10	
6	Y	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Reservation Station

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest.	A
Load1								
Load2								
Add1								
Add2								
Add3								
Mult1	Y	MUL	M[44+R[3]]	F[4]			#3	
Mult2	Y	DIV		M[32+R[2]]	#3		#5	