Out-of-order execution

Out-of-order execution (OoOE) is a computer processor optimization technique where instructions are executed based on the availability of their required data and execution units, rather than strictly by their original program order, to prevent performance-robbing "stalls" and increase instruction throughput. This dynamic instruction scheduling allows independent instructions to run in parallel while waiting for dependent instructions to complete, with hardware mechanisms like reservation stations and reorder buffers managing the process. The results are then committed in the original program order to maintain correct program behavior.

How it Works

1. **Instruction Fetch & Decode:** Instructions are fetched and decoded from the program as usual.

2. **Instruction Buffering:** Instructions are placed into an instruction buffer or reservation station, which holds instructions that are ready to execute.

3. **Dependency Checking:** Hardware continuously monitors dependencies between instructions. If an instruction needs data that hasn't been produced by a previous instruction yet, the processor can't execute it in program order.

4. **Dynamic Dispatch:** When an instruction's required operands are available, the processor dispatches it to an available execution unit, even if it's out of the original program sequence.

5. **Execution:** Instructions are executed as soon as their data and the necessary functional units are ready, not necessarily in their original order.

6. **Result Commit:** Results are buffered and then written back to the register file or memory in the correct program order, ensuring that the final program output is the same as if it had been executed sequentially.

Why it's Used

- **Reduces Stalls:**It minimizes processor idle time (stalls) caused by waiting for data or the results of a long-running instruction.
- **Exploits Parallelism:**It enables more instructions to execute simultaneously, increasing overall CPU performance.
- **Increases Throughput:**By keeping the processor busy, it improves the number of instructions completed per unit of time.

Key Hardware Components

- **Reservation Stations:**Hold instructions and their operands, waiting for dependencies to be met.
- **Reorder Buffers:**Store completed results, buffering them until they can be committed in the original program order.
- **Register Renaming:**A technique used to resolve data hazards by providing new temporary registers for instructions, allowing them to execute independently.

Speculative Execution

Speculative execution is a performance-boosting computer technique where a processor guesses the outcome of a branch instruction and executes subsequent instructions before the true path is known. If the prediction is correct, the work is already done, leading to faster execution. If the prediction is wrong, the speculatively executed instructions and their effects are discarded, and the processor restarts from the correct branch. This process keeps the processor pipeline full, avoids stalls, and significantly improves CPU performance by anticipating future needs.

How Speculative Execution Works

1. **Branch Prediction:**

   The processor's branch predictor makes an educated guess about which direction a conditional branch instruction will take.

2. **Speculative Execution:**

Based on the prediction, the CPU begins executing instructions along the predicted path.

3. **Verification:**

   The processor waits for the actual branch condition to be resolved.

4. **Commit or Discard:**

- **Correct Prediction:** If the prediction was accurate, the results of the speculatively executed instructions are committed, and they become part of the program's execution.
- **Incorrect Prediction:** If the prediction was wrong, the speculative results are discarded, and the processor rolls back to the correct execution path, re-executing instructions along the actual branch.

Why Speculative Execution is Used

- **Reduces Stalls:**

  It minimizes processor delays (stalls) caused by waiting for the outcome of branch instructions.

- **Hides Latencies:**

  It allows the processor to hide the time it takes to resolve long-latency operations and branch conditions.

- **Maximizes Throughput:**

  By keeping the instruction pipeline full of work, it enables more instructions to be executed concurrently.

- **Improves Performance:**

  It significantly boosts overall CPU performance by doing useful work ahead of time.

Key Components

- **Branch Predictor:** A hardware component that predicts the outcome of branches.
- **Reorder Buffer (ROB):** A buffer that temporarily stores the results of speculatively executed instructions before they are committed to the architectural state.