Branch Prediction in ARM Processors

Contents

Introduction

Modern processors use **branch prediction** to minimize pipeline stalls caused by conditional instructions (e.g., loops, if-else). In ARM processors, which power mobile and embedded systems, branch prediction is a key feature for achieving **high performance**, **energy efficiency**, and **accuracy**.

The Need for Branch Prediction

- **Problem:** When a branch instruction is encountered, the processor must decide:
  - **Taken:** Execute the branch.
  - **Not-taken:** Continue executing sequential instructions.
- **Challenge:** Deciding which path to take delays execution.
- **Solution:** Use branch prediction to predict the outcome and fetch instructions speculatively.

1 Purpose of Branch Predictor

The purpose of a branch predictor is to:

1. Reduce pipeline stalls caused by branch mispredictions.
2. Maintain high throughput in pipelined architectures.
3. Optimize performance in **mobile**, **IoT**, and **high-performance** systems.

Pipeline Delay

Without branch prediction, the processor stalls to determine the outcome of a branch:

$$\text{Delay (cycles)} = \text{Pipeline Depth} \times \text{Branch Misprediction Rate}$$

2 Types of Branch Prediction in ARM Processors

ARM processors implement two main types of branch prediction: **static** and **dynamic**.

2.1 Static Branch Prediction

Static Prediction Rules:

- **Backward Branches:** Predicted as **taken** (e.g., loops).
- **Forward Branches:** Predicted as **not-taken**.

Advantages:

- Simple to implement.

- No hardware overhead.

Limitations:

- Cannot adapt to branch behavior.

2.2 Dynamic Branch Prediction

Dynamic prediction adapts to **runtime branch behavior** using:

- **History-based predictors.**
- **Hybrid models** (e.g., global + local history).

3 Branch Prediction Architecture in ARM Processors

ARM processors utilize **multi-level architectures** for branch prediction.

3.1 Branch Target Buffer (BTB)

The **BTB** is a cache storing:

- Branch instruction addresses.
- Predicted target addresses.

When a branch instruction is fetched:

$$\text{Predicted Target Address} = \text{BTB}[\mathit{Branch\,PC}]$$

Role of BTB:

- Reduces latency by prefetching target instructions.
- Updates dynamically with new branches.

3.2 Pattern History Table (PHT)

The **PHT** tracks branch outcomes using **2-bit saturating counters**:

1. **Strongly Taken (ST):** Always predict taken.
2. **Weakly Taken (WT):** Predict taken, but can switch.
3. **Weakly Not-Taken (WNT):** Predict not-taken, but can switch.
4. **Strongly Not-Taken (SNT):** Always predict not-taken.

$$\text{Next State} = f(\text{Current State}, \text{Branch Outcome})$$

Example Transition:

$$\text{WT} \xrightarrow{\text{Not-Taken}} \text{WNT}$$

3.3 Global and Local Predictors

- **Global History Register (GHR):**
  - Tracks outcomes of the last N branches.
  - Correlates branch predictions with global patterns.

  Example:

  $$\text{GHR} = [T, T, NT, T] \quad \text{(T: Taken, NT: Not-Taken)}$$

- **Local History Table (LHT):**
  - Tracks branch-specific history.
  - Useful for branches with unique behavior.

3.4 Return Address Stack (RAS)

The **RAS** predicts the return address for function calls:

1. Pushes return addresses during function calls.
2. Pops addresses during returns.

4 How Branch Prediction Works in ARM

4.1 Pipeline Stages with Branch Prediction

1. **Fetch Stage:**
   - The branch predictor identifies branch instructions.
   - Checks the **BTB** and **PHT** for prediction data.
2. **Prediction Stage:**
   - If a branch is found in the BTB:

     $$\text{Target Address} \rightarrow \text{Prefetch}$$

   - Otherwise:
     - Use static prediction.
3. **Execution Stage:**
   - Executes the branch.
   - If the prediction was wrong:
     - Flush the pipeline.
     - Fetch correct instructions.

5 ARMv8 and ARMv9 Enhancements

5.1 ARMv8-A (AArch64)

Key Features:

- **Improved BTB:** Larger size to store more branches.
- **Advanced Predictors:** Combines local and global history.

5.2 ARMv9

New Enhancements:

- **Neural Predictors:** Leverages machine learning for irregular patterns.
- **Thread-Specific Predictors:** Separate tables for multi-threaded execution.

6 Key Features in ARM Branch Predictors

6.1 Low Latency Prediction

Predictions occur at the fetch stage to minimize stalls.

6.2 Energy Efficiency

Optimized for mobile devices where power consumption is critical.

6.3 High Accuracy

Combines global and local predictors to improve prediction rates.

7 Common Metrics for Evaluating Branch Predictors

1. **Branch Prediction Accuracy (BPA):**

$$\text{BPA} = \frac{\text{Correct Predictions}}{\text{Total Branches}} \times 100$$

2. **Misprediction Penalty:** Cycles lost due to incorrect predictions.
3. **BTB Hit Rate:**

$$\text{BTB Hit Rate} = \frac{\text{Branches Found in BTB}}{\text{Total Branches}}$$

8 Example of Branch Prediction in ARM

8.1 Backward Branch in a Loop

```
MOV R0, #0              ; Initialize counter
LOOP:
ADD R0, R0, #1          ; Increment counter
CMP R0, #10             ; Compare counter with 10
BNE LOOP                ; Branch if not equal (backward branch)
```

- **Static Prediction:** Predicts as **taken**.
- **Dynamic Prediction:** Adapts to runtime behavior for higher accuracy.

9 Conclusion

ARM processors use advanced branch prediction mechanisms to achieve high performance, low latency, and energy efficiency. Techniques like **dynamic prediction**, **neural predictors**, and **hybrid models** ensure ARM architectures excel in modern workloads.

Speculative Execution in ARM Processors

Contents

Introduction

Speculative Execution is a performance optimization technique used in modern processors, including ARM processors, to execute instructions before the final outcome of a decision (e.g., a branch) is known. By predicting the path of execution, the processor can execute instructions speculatively, reducing delays caused by pipeline stalls.

1 Purpose of Speculative Execution

1. **Increase Instruction Throughput:**
    - Minimizes idle time in the pipeline.
    - Keeps the pipeline filled with instructions.
2. **Reduce Branch Penalty:**
    - Branch instructions introduce delays due to uncertainty about which path to execute.
    - Speculative execution helps mitigate this delay.
3. **Optimize Multi-Stage Pipelines:**
    - ARM processors with deeper pipelines benefit significantly from speculation as it minimizes stalls.

2 How Speculative Execution Works

1. **Instruction Fetch Stage:**
    - The processor fetches instructions for both possible outcomes of a branch.
2. **Prediction Stage:**
    - The **branch predictor** predicts the most likely branch to be taken.
    - Example: Global History Register (GHR) or Branch Target Buffer (BTB).
3. **Execution Stage:**
    - Executes instructions along the predicted path speculatively.
    - If the prediction is correct, the results are committed.
    - If incorrect, the pipeline is flushed, and the correct path is executed.

3 Mathematical Representation of Speculative Execution

3.1 Pipeline Performance Without Speculation

Let:

- I: Total number of instructions.
- C: Cycles per instruction (CPI).
- P_m: Probability of branch misprediction.
- T_m: Misprediction penalty in cycles.

The total execution time $T_{\text{no-speculation}}$ is given by:

$$T_{\text{no-speculation}} = I \times C$$

3.2 Pipeline Performance With Speculation

Speculative execution introduces potential branch misprediction penalties. The execution time becomes:

$$T_{\text{speculation}} = I \times C + P_m \times T_m \times \text{Branch Instructions}$$

3.3 Performance Gain

$$\text{Performance Gain} = \frac{T_{\text{no-speculation}} - T_{\text{speculation}}}{T_{\text{no-speculation}}} \times 100$$

4 Components Supporting Speculative Execution in ARM Processors

4.1 Branch Predictors

- Predicts the direction (taken/not-taken) and target of branch instructions.
- Reduces uncertainty in pipeline execution.

4.2 Branch Target Buffer (BTB)

- Caches target addresses of recently executed branches.
- Provides quick access to likely branch targets.

4.3 Reorder Buffer (ROB)

- Holds speculative results until they are verified as correct.
- Ensures precise exception handling.

4.4 Return Address Stack (RAS)

- Tracks return addresses for function calls, ensuring accurate predictions for nested calls.

5 ARM Speculative Execution Mechanism

5.1 Step-by-Step Process

1. **Fetch Instructions Speculatively:**
   - Fetch instructions along the predicted path.
2. **Execute Speculatively:**
   - Execute the instructions without committing results.
   - Maintain speculative results in temporary buffers (e.g., ROB).
3. **Verify Prediction:**
   - Check the branch prediction outcome.
   - If correct, commit the results.
   - If incorrect, flush the pipeline and re-execute.

6 Equations for Speculative Execution in ARM

6.1 Correct Prediction

If P_c is the probability of a correct branch prediction:

$$\text{Execution Time (Correct Prediction)} = I \times C$$

6.2 Incorrect Prediction

If P_m is the probability of a misprediction:

$$\text{Misprediction Time} = P_m \times T_m \times \text{Branch Instructions}$$

6.3 Overall Execution Time

Combining both cases:

$$T_{\text{speculative}} = I \times C + P_m \times T_m \times \text{Branch Instructions}$$

7 Security Implications of Speculative Execution

7.1 Side-Channel Attacks

- Exploit speculative instructions that access restricted data.
- Example: **Spectre** and **Meltdown** attacks.

7.2 Mitigations in ARM

- **Speculation Barriers:** ARM introduced 'CSDB' (Context Synchronization Barriers) to limit speculative execution.
- **Branch Target Isolation (BTI):** Prevents exploitation of mispredicted branch targets.

8 ARM Architectures with Speculative Execution

8.1 ARMv7

- Uses basic speculative execution.
- Optimized for mobile devices with short pipelines.

8.2 ARMv8

- Advanced dynamic branch predictors.
- Speculative execution supports **AArch64** (64-bit).

8.3 ARMv9

- Introduces **neural-based predictors** for better accuracy.
- Mitigates speculative vulnerabilities using **branch protection mechanisms**.

9 Advantages of Speculative Execution

1. **Improved Performance:**
   - Keeps the pipeline busy, maximizing throughput.
2. **Reduced Branch Penalty:**
   - Mitigates the performance loss caused by branch instructions.
3. **Efficient Resource Utilization:**
   - Ensures all pipeline stages remain active.

10 Challenges of Speculative Execution

1. **Branch Misprediction Penalty:**
   - Requires flushing the pipeline, which wastes cycles.
2. **Hardware Complexity:**

- Speculative execution introduces additional hardware like BTBs, ROBs, and RAS.

3. **Security Concerns:**
   - Speculative instructions may leak sensitive data through side channels.

11 Example: Speculative Execution in ARM Assembly

```
CMP R1, R2            ; Compare two registers
BEQ TARGET            ; Branch to TARGET if equal
MOV R3, #1            ; Speculatively execute this instruction
TARGET:
ADD R4, R3, #10       ; Add 10 to R3
```

Explanation:

- The processor speculates whether the branch (`BEQ`) will be taken.
- Executes `MOV R3, #1` speculatively.
- If the prediction is correct, the results are committed. Otherwise, the pipeline is flushed.

12 Metrics for Evaluating Speculative Execution

12.1 Speculation Efficiency (SE)

$$\text{SE} = \frac{\text{Correct Speculative Instructions}}{\text{Total Speculative Instructions}} \times 100$$

12.2 Branch Prediction Accuracy (BPA)

$$\text{BPA} = \frac{\text{Correct Branch Predictions}}{\text{Total Branches}} \times 100$$

12.3 Misprediction Penalty (MP)

$$\text{MP} = T_m \times P_m$$

13 Future of Speculative Execution in ARM

1. **Neural Predictors:**
   - Use machine learning models to improve branch prediction accuracy.
2. **Context-Aware Execution:**
   - Adapt speculative execution based on workload characteristics.

3. **Secure Speculation:**

    - Develop hardware and software mechanisms to prevent speculative vulnerabilities.

14 Conclusion

Speculative execution is a cornerstone of modern ARM processors, enabling high-performance computing by reducing pipeline stalls. Despite its challenges, such as branch misprediction penalties and security risks, ARM's advanced predictors and mitigation techniques ensure efficient and secure execution for a wide range of applications.