ACA PROJECT *Part_2*

Name:T.Pranav **Roll No:22cs02006**
M V V V S Raju **22cs02003**

Evaluation of Replacement Policies under Varying BTB Sizes

Objective
This report investigates how Branch Target Buffer (BTB) size influences the performance of different replacement algorithms in a processor's branch prediction mechanism.

Role of BTB
The BTB stores the target addresses of recently executed branch instructions, enabling faster branch prediction and reducing control hazards within the instruction pipeline.

Importance of BTB Size and Policy
Since the BTB has limited storage capacity, both its size and the replacement policy used to manage entries are critical in maintaining high prediction accuracy.

Scope of the Study
This study examines how varying BTB capacities — 128, 256, and 512 entries — affect prediction performance under different replacement strategies.

Replacement Algorithms Analyzed
The report evaluates three widely used replacement policies:

- **Least Recently Used (LRU):** Retains recently accessed entries by evicting the least recently used ones.
- **Random:** Selects a BTB entry for replacement randomly, offering simplicity and low hardware overhead.
- **First-In-First-Out (FIFO):** Replaces entries in the order they were inserted, ignoring usage patterns.

Overall Aim:
The goal is to identify which replacement algorithm performs most effectively under different BTB sizes and to understand the relationship between buffer capacity and replacement policy efficiency.

Replacement Policies Evaluated

1. **LRU (Least Recently Used)**: Replaces the entry that hasn't been accessed for the longest time, exploiting temporal locality
2. **Random**: Selects a victim entry randomly, providing a simple baseline with no bookkeeping overhead
3. **FIFO (First-In-First-Out)**: Replaces entries in the order they were inserted, ignoring access patterns

Experimental Configuration

- **BTB Sizes**: 128, 256, and 512 entries
- **Benchmarks**: anagram.alpha, gcc, Compress, and Go
- **Predictor**: bimod (bimodal branch predictor)
- **Metric**: Branch prediction accuracy (higher is better)

bimod:
Anagram.alpha

BTB Replacement Policy	LRU	Random	FIFO
128 entries	0.6489	0.6444	0.6342
256 entries	0.6501	0.6501	0.6421
512 entries	0.6501	0.6501	0.6467

Gcc:

BTB Replacement Policy	LRU	Random	FIFO
128 entries	0.8492	0.8599	0.7525
256 entries	0.8687	0.8655	0.7884
512 entries	0.8769	0.8749	0.8208

Compress:

BTB Replacement Policy	LRU	Random	FIFO
128 entries	0.9680	0.9676	0.9674
256 entries	0.9680	0.9678	0.9674
512 entries	0.9680	0.9678	0.9678

GO:

BTB Replacement Policy	LRU	Random	FIFO
128 entries	0.7709	0.7660	0.6875
256 entries	0.7832	0.7813	0.7212
512 entries	0.7889	0.7886	0.7476

Reason for Choosing Bimod:

The bimodal predictor was chosen because it represents a simple yet widely used baseline in branch prediction studies. It uses a single level of 2-bit saturating counters, making it easier to isolate and evaluate the impact of BTB replacement policies without interference from complex predictor dynamics such as local or global history.
By keeping the branch predictor constant and uncomplicated, any variations in performance can be directly attributed to the BTB replacement strategy and buffer size, ensuring a fair and clear comparison of LRU, Random, and FIFO policies.

Experimental Results Overview

The concept of branch footprint refers to the number of branch target addresses that remain active within a short period of time.

When the BTB size becomes larger than the branch footprint, the hit rate saturates and becomes independent of the replacement algorithm used.

The following trends were observed across benchmarks:

- In the Anagram.alpha benchmark, this behavior is observed at a BTB size of 256 entries.
- Beyond 256 entries, the hit rate remains constant across all replacement policies, indicating that the branch footprint fits entirely within the BTB.
- Therefore, increasing the BTB size further does not improve prediction accuracy, as all necessary branch targets are already captured.
- GCC and Go benchmarks show strong improvement in prediction accuracy with increasing BTB size and exhibit clear sensitivity to the replacement policy, especially at smaller capacities.
- In the case of GCC, the Random replacement policy achieved a higher hit rate than LRU at 128 entries.
- This occurs because GCC contains complex and irregular branching behavior, with a less predictable control flow and frequent changes in branch targets.
- Under such conditions, the LRU policy tends to evict entries more aggressively when almost every branch address is new, leading to frequent replacements in asmall BTB.

- The Random policy, on the other hand, avoids this deterministic eviction pattern, occasionally resulting in a slightly higher hit rate at smaller BTB sizes.

- *Compress*: Exhibits consistently high accuracy regardless of BTB size or replacement strategy, indicating a small, predictable branch working set.

Anagram.alpha: Accuracy saturates at 256 entries (0.6501) across all policies, suggesting the working set fits comfortably in medium-sized BTBs. At 128 entries, LRU (0.6489) marginally outperforms Random (0.6444) and FIFO (0.6342).

GCC: Shows strong sensitivity to both BTB size and replacement policy. Notably, Random (0.8599) outperforms LRU (0.8492) at 128 entries, though LRU dominates at larger sizes (0.8769 at 512 entries vs 0.8749 for Random). FIFO consistently underperforms across all configurations.

Compress: Exhibits uniformly high accuracy (0.9680 for LRU across all sizes), indicating a small, predictable branch working set. All policies perform similarly, with minimal variation.

Go: Demonstrates clear improvement with increasing BTB size. LRU consistently leads (0.7889 at 512 entries), while FIFO shows the weakest performance (0.7476 at 512 entries).

Analysis:

->LRU generally provides the best performance by exploiting temporal locality and retaining recently accessed branch targets. However, the GCC benchmark at 128 entries reveals that Random replacement can occasionally outperform LRU, likely due to its non-deterministic eviction preventing pathological access pattern conflicts when capacity is constrained.

->FIFO consistently shows inferior performance due to its lack of awareness regarding access frequency or recency, causing premature eviction of active entries.

->As BTB size increases, prediction accuracy generally improves across all policies, though the magnitude of improvement varies by workload. Compress shows minimal sensitivity to BTB size, while gcc and go demonstrate substantial gains with larger buffers.

Conclusion:

LRU remains the most robust choice for BTB replacement due to consistent performance advantages across diverse workloads. However, workload characteristics and BTB capacity can influence optimal policy selection, as demonstrated by Random replacement policy success in gcc at smaller capacities. These findings emphasize that BTB management strategies should consider both architectural constraints and program behavior rather than applying a universal approach.