Thesis Research Proposal: Advancing CPU performance through Parallel Fetch and Decode Units for Enhanced Branch Prediction

I. Introduction:

A. Background

In the quest for ever-increasing processor performance, advancements in microarchitecture play a pivotal role. This thesis proposal focuses on exploring novel CPU architectures that integrate multiple fetch and decode units optimized for branch prediction to enhances overall system performance.

B. Research Objectives:

1.To design and analyze a CPU architecture featuring parallel fetch and decode units optimized for branch prediction.

2. To investigate advanced branch prediction algorithms tailored to the proposed CPU architecture.

3. To evaluate the real-world performance gains achieved by the designed CPU architecture through comprehensive benchmarking.

II. Literature Review

1. Review of current CPU architectures.

Examining the CPU architectures, particularly focusing on strategies for branch prediction and implementations with multiple fetch and decode units. Specifically addressing challenges in branch prediction accuracy and overall system performance. Below are some of the CPU architectures that use a similar architecture design.

1.Intel Pentium Pro (P6 Micro Architecture):

The Intel Pentium Pro, introduced in 1995, featured a complex pipeline structure with multiple fetch and decode units optimized for handling complex x86 instructions efficiently. The pipeline includes separate stages for instruction fetching, decoding, and execution. The Pentium PR’s micro architecture laid the foundation for subsequent inter processor and influenced the design of modern CPUs.

2. Intel Pentium 4 (NetBurst Microarchitecture):

The Intel Pentium 4, released in 2000, adopted the NetBurst microarchitecture, characterized by a long pipeline with multiple fetch and decode units. It featured a 20-stage pipeline, one of the longest in x86 CPU history. To facilitate high clock speeds and instruction throughput. The Pentium 4’s architecture aimed to exploit instruction-level parallelism and improve overall system performance.

3.IBM POWER7:

The IBM POWER7 processor, launched in 2010, showcased a high-performance CPU design with a superscalar out-of-order execution pipeline. It incorporated multiple fetch and decode units to enable parallel instruction processing and maximize throughput. The POWER7’ microarchitecture emphasized scalability and efficiently for server and enterprise computing workloads.

4.AMD Ryzen Processors (Zen Microarchitecture):

AMD Ryzen processors, based on the Zen microarchitecture, introduced a wide front-end capable of fetching and decoding multiple instructions per cycle. The Zen microarchitecture emphasized instruction throughput and efficiency, leveraging parallel fetch and decode units to enhance CPU performance. Ryzen processors targeted a wide range of applications, from gaming to content creation, with their versatile microarchitecture design.

1. Identifying Research Gaps:

III. Proposed Methodology:

1. CPU Architecture Design
2. Detailed Description of the Proposed CPU Architecture:

The proposed CPU architecture integrates multiple fetch and decode units to enhance instruction throughput and branch prediction accuracy. Parallel fetch units concurrently retrieve instruction blocks from memory, while corresponding decode units decode fetched instructions simultaneously. This widened front-end allows for efficient handling of instruction streams and improves the overall performance of the CPU.

In scenarios where branches are taken, speculatively fetched and decoded instructions are forwarded to the execution unit. These instructions are executed assuming the predicted branch target, enabling continued pipeline operation. Notably, the architecture's speculative fetch and decode stage benefits from the knowledge of the branch outcome from the preceding instruction that was executed by the execution unit. As a result, branch prediction cannot fail in this stage since the branch outcome is already known.

This speculative fetch and decode stage contribute to the formation of the data flow execution graph, representing the dependencies between instructions and their execution order. Instructions fetched and decoded in parallel form nodes in the data flow graph, with edges representing the flow of data and control dependencies between instructions. By accurately predicting branch outcomes and efficiently processing instructions in parallel, the architecture ensures that the data flow execution graph reflects the optimized execution path of the program.

Moreover, the fetch and decode units are equipped to recursively generate branch targets with interpreter loops. Interpreter loops are specialized sequences of instructions designed to efficiently handle frequent branches and loops encountered in program execution. The fetch and decode units identify these recurring patterns and proactively generate branch targets within the pipeline to facilitate rapid execution without the need for explicit branch prediction.

While this architecture may not be feasible for everyday computers due to its specialized design and potential complexity, it holds significant promise for architectures like RISC-V and microcontroller-based systems. These architectures often prioritize efficiency, simplicity, and predictability, making them ideal candidates for the proposed CPU architecture. By leveraging parallel fetch and decode units, advanced branch prediction mechanisms, and interpreter loops, these architectures can achieve improved performance and energy efficiency across a wide range of embedded and IoT applications.

Unlike approaches like Tomasulo's approach, which focuses on dynamic scheduling and out-of-order execution to handle data dependencies and maximize instruction-level parallelism, the proposed architecture emphasizes parallelism in the fetch and decode stages to improve instruction throughput and branch prediction accuracy. While Tomasulo's approach dynamically schedules instructions based on data availability, the speculative fetch and decode stage of the proposed architecture operates at an earlier stage of the pipeline, leveraging parallelism to accelerate instruction processing and prediction accuracy.

To address potential mispredictions, the architecture incorporates speculative execution rollback and recovery mechanisms. If a branch prediction is incorrect, speculatively fetched and decoded instructions are squashed, and the correct target address is fetched and decoded to resume execution from the correct program path. These mechanisms ensure that mispredicted branches do not disrupt pipeline operation unnecessarily, maintaining efficient execution of instructions and preserving the integrity of the data flow execution graph.

Overall, the proposed CPU architecture optimizes parallelism in the fetch and decode stages to improve instruction throughput and branch prediction accuracy while facilitating the formation of an accurate data flow execution graph. By integrating advanced prediction mechanisms, speculative execution handling, and interpreter loops for branch target generation, the architecture aims to deliver superior performance across diverse workloads and usage scenarios, particularly in RISC-V and microcontroller-based systems where efficiency and predictability are paramount.

1. Implementation Plan:
2. Custom CPU Header File Creation Development of Custom CPU Header File:

The implementation will commence with the creation of a custom CPU header file within the gem5 source code. This header file will define the specifications and characteristics of the proposed CPU architecture, including pipeline structure, fetch and decode units, branch prediction mechanisms, and other relevant parameters. The custom CPU header file will serve as the foundation for modeling and simulating the proposed CPU architecture within the gem5 simulation framework.

1. Integration with Different ISAs Testing with Different Instruction Set Architectures (ISAs):

Once the custom CPU header file is created, it will be integrated into the gem5 simulation environment to facilitate testing with various ISAs. gem5 supports multiple ISAs, including ARM, x86, and RISC-V, among others. The custom CPU model will be instantiated and simulated using different ISAs to evaluate its compatibility, performance, and behavior across diverse instruction set architectures. This testing phase will provide valuable insights into the versatility and adaptability of the proposed CPU architecture across different ISA specifications.

1. Further Microcontroller Testing with QEMU Utilizing QEMU for Microcontroller Testing:

For further validation and testing, the custom CPU architecture can be evaluated using QEMU, a versatile CPU emulator supporting a wide range of ISAs, including those commonly used in microcontrollers. QEMU enables software developers to test their code against emulated microcontroller environments, facilitating rapid prototyping, debugging, and performance analysis. By leveraging QEMU's capabilities, additional validation of the custom CPU architecture's behavior in microcontroller-like scenarios can be conducted, ensuring compatibility and functionality across diverse embedded systems.

1. Evaluation Using gem5 Metrics Utilization of gem5 Metrics for Evaluation:

gem5 offers a rich set of performance metrics and statistics that can be leveraged to evaluate the effectiveness and efficiency of the custom CPU architecture's pipeline structure. Key gem5 metrics to be considered include:

IPC (Instructions Per Cycle): This metric measures the average number of instructions executed per cycle, providing insights into the overall throughput of the CPU pipeline. Higher IPC values indicate improved pipeline efficiency and performance.

Branch Prediction Accuracy:

gem5 provides metrics for assessing the accuracy of branch prediction mechanisms, including branch prediction hit rates, miss rates, and prediction accuracy percentages. These metrics quantify the effectiveness of the pipeline's branch prediction strategies in reducing branch mispredictions and improving instruction flow.

Pipeline Stall Analysis:

gem5 enables detailed analysis of pipeline stalls, including the reasons for stalls such as data hazards, control hazards, and structural hazards. Understanding the sources and frequency of pipeline stalls helps identify potential bottlenecks and areas for optimization within the pipeline structure.

Cycle Time and Latency:

gem5 provides metrics for evaluating cycle time and latency within the CPU pipeline. Analyzing cycle time and latency metrics helps assess the overall efficiency and responsiveness of the pipeline architecture in executing instructions and handling workload variations.

1. Custom Metric Development:

In addition to standard gem5 metrics, custom performance metrics can be developed to provide further insights into specific aspects of the custom CPU architecture's pipeline structure. These custom metrics may include measures of resource utilization, instruction scheduling efficiency, and power consumption, tailored to the unique characteristics and objectives of the proposed CPU architecture.