Lab introduction

In the fast-paced world of modern computer architecture, optimizing the performance of processors has become a paramount challenge. As computational tasks have grown in complexity and size, the need for ever-increasing processing speeds has driven the evolution of microprocessors. One critical aspect of this evolution is branch prediction—a technique that plays a pivotal role in enhancing the efficiency of modern CPUs.

Branch prediction is not merely an esoteric concept for computer engineers; it is a fundamental technology that underpins the functionality of nearly every computer system, from your personal laptop to the supercomputers powering scientific research. At its core, branch prediction is the art and science of guessing the outcome of conditional branches in a program before their actual results are known. This seemingly small leap of intuition carries immense importance in the world of computer architecture and performance optimization.

To comprehend the significance of branch prediction, it is essential to consider the nature of modern software. Software applications are inherently full of decision points, often in the form of conditional statements like if-else constructs and loops. These decisions, or branches, determine the flow of a program and can dramatically affect its execution time. When a processor encounters a branch instruction, it must choose a direction to follow—either taken or not taken. Making this choice correctly can lead to substantial performance improvements, while incorrect predictions can result in significant slowdowns.

Imagine a web server handling multiple incoming requests simultaneously. Each request may involve a complex series of decisions, such as database queries, conditionals, and loops. Accurate branch prediction can mean the difference between efficiently serving multiple clients and experiencing severe bottlenecks. Similarly, in gaming consoles or graphics processing units (GPUs), where real-time rendering and responsiveness are crucial, branch prediction can directly impact the gaming experience.

Furthermore, energy efficiency is a growing concern in modern computing. Processors that make accurate branch predictions can execute fewer unnecessary instructions, reducing power consumption and extending battery life in mobile devices. This is especially relevant in an era where mobile computing has become ubiquitous.

In this lab, you will delve into the fascinating world of branch prediction. You will explore the different strategies and algorithms used by processors to predict branches accurately.

You are given two design and implementation tasks, alongside an extra task that will provide 1/10 extra marks for anyone that achieves it, as well as 1/10 more extra marks for whomever is the best in the class:

1. Design and implement a simple TAKEN / NOT TAKEN branch predictor as shown in class **(2 marks)**.

2. Design and implement an improved version of the branch predictor that uses a 4-state state machine to predict branches **(4 marks)**.

3. Extra: Improve the accuracy of your branch predictor as much as you can. If you increase the accuracy versus the 4-state branch predictor and you achieve the best accuracy of the entire class, you will gain **an extra mark**. Good luck!

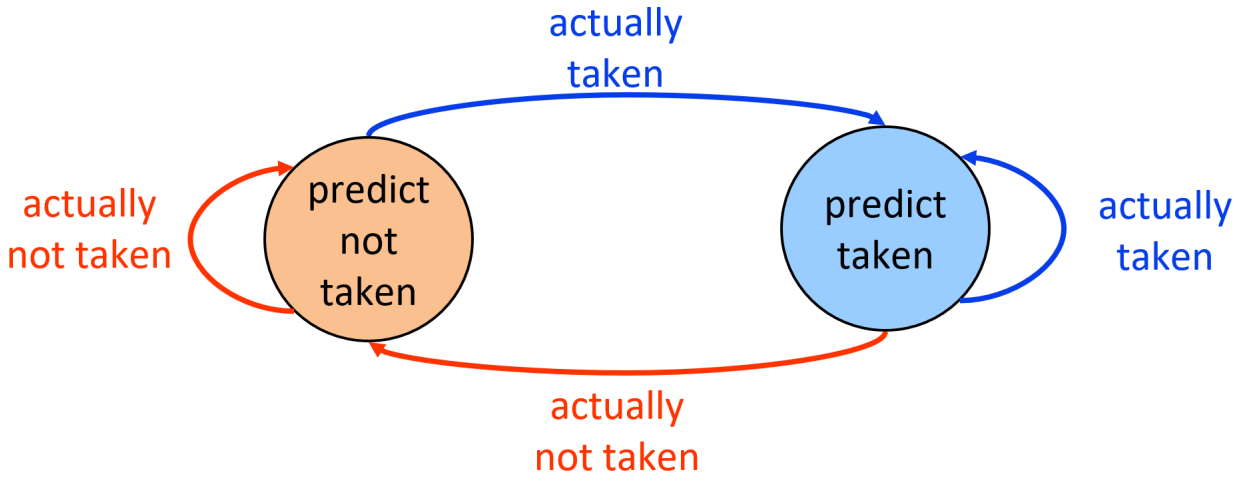
4. Write a technical report that contains all the results from the lab **(4 marks)**.

All of this work will have to be written into a technical report that documents everything you have done to design and implement the branch predictors. You will also have to answer the questions that are given to you in the following sections. The implementation of the branch predictors can be done in any programming language (although Python is preferable), but it must be in English. The report must also be in English. You will have to submit the PDF of the report alongside a .zip file with the source code on the PDU.

The deadline of this Lab will be: October 9th, 2023

Task 1: Simple 1-bit TAKEN/NOT TAKEN Branch Predictor

In this first task, you will embark on the journey of creating a fundamental branch predictor: the 1-bit taken/not taken predictor. This type of predictor makes predictions based solely on the history of whether a particular branch instruction was taken or not taken in the past. It's a simple yet crucial concept in the world of branch prediction.



Task 1.1: Trace file format

You will be provided trace files that contain the address of each branch instruction that was executed, as well as whether it was taken (T) or not taken (N). As such, each line of the

trace file will follow this format: [Branch address][$Taken|Not\ taken(T|N)$] (e.g. 12345678 T). Your objective is to implement a 1-bit branch predictor that uses this information to predict the outcome of each branch.

Task 1.2: Implementing the 1-bit branch predictor

Your task is to create a simple program or script that reads the trace file, maintains a history of branch outcomes (with a limited size), and makes predictions based on that history. You will need to decide on an initial prediction for each branch (taken or not taken) and then update this prediction as new information from the trace file becomes available.

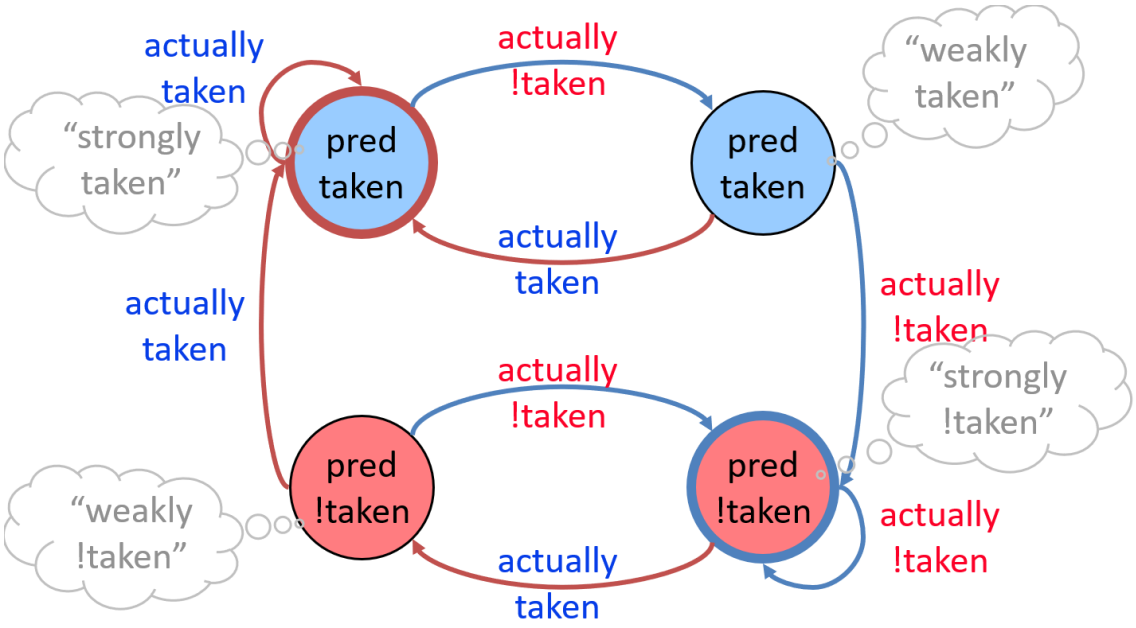
Task 1.3: Evaluation

After implementing your 1-bit branch predictor, you will need to evaluate its performance. Use the trace file to check how accurate your predictor's predictions are compared to the actual outcomes in the trace. Calculate metrics such as prediction accuracy, misprediction rate, and any other relevant statistics. During your evaluation you must answer the following questions (they must be included in the PDF report you must submit):

1. How did you initialize your 1-bit predictor? Did you use a default prediction for all branches, or did you employ a specific strategy?
2. What data structures did you use to maintain the history of branch outcomes? How did you update these data structures as you processed the trace file? Try using different limited sizes for these structures (like 1 entry, 2 entries, 4 entries, 8 entries and 16 entries). Does the accuracy of the predictor change?
3. How accurate was your 1-bit branch predictor? What metrics did you use to measure its performance?
4. Can you identify any patterns or trends in the trace file that might have influenced the accuracy of your predictor?
5. What are the limitations of a 1-bit taken/not taken predictor, and how might it be improved?

Task 2: 2-bit/State Machine Branch Predictor

Building upon your understanding of branch prediction gained in Task 1, you will now delve into a more sophisticated branch prediction technique: the 2-bit branch predictor. This task introduces you to the concept of using a state machine to introduce some hysteresis to the branch predictor. In this task, you will design, implement, and evaluate a 2-bit branch predictor. Unlike the 1-bit predictor, which simply switches between taken and not taken predictions, the 2-bit predictor keeps track of the history of outcomes for each branch and makes predictions based on this history. This can lead to more accurate predictions and better overall performance.



Task 2.1: Understanding the 2-bit predictor

Before you start implementing, make sure you understand how a 2-bit branch predictor works. It uses a two-bit counter to maintain the prediction history for each branch. The counter can take on four states: strongly taken, weakly taken, weakly not taken, and strongly not taken. These states help the predictor adapt to changing patterns of branch behavior.

Task 2.2: Implementation

Design and implement a 2-bit branch predictor using your preferred programming language. You will need to read the same trace files from Task 1, but this time, your predictor should use the history of each branch's outcomes to make predictions.

Task 2.3: Evaluation

After implementing your 2-bit branch predictor, it's time to evaluate its performance. Use the trace file once again to assess how well your predictor's predictions align with the actual branch outcomes. Calculate prediction accuracy, misprediction rates, and any other relevant metrics to gauge the effectiveness of your predictor. During your evaluation you must answer the following questions (they must be included in the PDF report you must submit):

1. How did you design your 2-bit branch predictor? Describe the data structures and algorithms you used to maintain the prediction history for each branch. Try using different limited sizes for these structures (like 1 entry, 2 entries, 4 entries, 8 entries and 16 entries). Does the accuracy of the predictor change?
2. What is the significance of the four states in the 2-bit counter (strongly taken, weakly taken, weakly not taken, and strongly not taken)? How do they contribute to prediction accuracy?
3. How does the accuracy of your 2-bit predictor compare to the 1-bit predictor from Task 1? What factors might have contributed to the differences in accuracy?
4. Can you identify any specific patterns or behaviors in the trace file that your 2-bit predictor handled particularly well or poorly?
5. Reflect on the advantages of using a 2-bit branch predictor over a 1-bit predictor. In what scenarios or applications might a 2-bit predictor be especially beneficial?

Extra Task 3: Improving Your Branch Predictor

In addition to the primary tasks involving the implementation of 1-bit and 2-bit branch predictors, an extra credit opportunity is presented, wherein students have the prospect of acquiring an additional mark. This additional mark is attainable not merely through participation but through the development of an exceptional branch predictor design that outperforms those of peers in the class. This task encourages students to channel their creativity and ingenuity into the realm of branch prediction. For this extra credit task, you are tasked with designing an innovative branch predictor that goes beyond the basic 1-bit and 2-bit predictors discussed in the previous tasks. Your predictor should aim for higher accuracy and better adaptability to various branch behavior patterns. Think of this as a chance to experiment with novel ideas in branch prediction. The rules for the competition are the following:

1. To be eligible for the extra credit mark, you must complete and submit your innovative branch predictor, as well as write its own section in your technical report.

2. The student with the most accurate and effective predictor in the class will be awarded an additional mark, making a total of two extra credit marks.

3. In case of a tie, where multiple students achieve the highest accuracy, no one will receive the additional extra mark (this is to avoid you copying from each other, so that you actually compete for it).

4. The improvements you achieve must not come from using a larger branch table. At normal table sizes (such as 32 or 64) your branch predictor should be better than the 2-bit predictor. All the predictors in the class will be evaluated with a 128 entries table.

Task 3.1: Innovative design

Begin by brainstorming and designing your innovative branch predictor. Consider factors such as additional state information, new algorithms, or different ways of capturing and utilizing branch history. Your goal is to create a predictor that outperforms both the 1-bit and 2-bit predictors in terms of prediction accuracy.

Task 3.2: Implementation

Once you've designed your predictor, implement it using your preferred programming language. Make sure it can read the same trace files used in the previous tasks and make predictions based on its unique design.

Task 3.3: Evaluation and competition

Evaluate the performance of your innovative branch predictor by using the trace file provided. Calculate prediction accuracy, misprediction rates, and any other relevant metrics. The real competition begins when you compare your predictor's performance with your fellow students' predictors. While you are evaluating your newly designed predictor, you will have to answer the following questions in your report:

1. What innovative ideas and techniques did you incorporate into your branch predictor's design? How do these innovations contribute to its performance?

2. How does the accuracy of your innovative predictor compare to the 1-bit and 2-bit predictors implemented in the core tasks?

3. Can you identify any specific scenarios or types of branch behavior where your predictor excels?

4. What challenges did you encounter while implementing and testing your innovative predictor, and how did you address them?

5. Reflect on the potential real-world applications or scenarios where your predictor's design might prove particularly valuable.

Task 4: Writing Your Report

In Task 4, you will synthesize the knowledge and experience gained throughout this lab into a comprehensive technical report. This report will serve as a structured document that presents your findings, analyses, and insights related to branch prediction. It will encompass all aspects of the lab, from the foundational concepts to the innovative predictors you've designed and evaluated. The report must contain the following sections:

1. **Abstract (150-200 words):** Provide a concise summary of the entire report. Highlight the key objectives, methodologies, and significant findings of your branch prediction experiments.
2. **Introduction:** Introduce the importance of branch prediction in modern computer architecture, mention the objectives and scope of the lab, and briefly describe the tasks undertaken in the lab, including the design and evaluation of branch predictors.
3. **Results:** One subsection for each task where you provide all of the information that you gathered throughout the lab, and where you answer all of the posed questions.
4. **Conclusion:** Summarize the main findings and insights from each task, reflect on the significance of branch prediction in computer architecture, discuss the limitations and potential improvements of the predictors you implemented, and highlight any practical applications or scenarios where branch prediction plays a critical role.

This technical report will serve as a comprehensive record of your work in this lab, showcasing your understanding of branch prediction concepts, your programming skills, and your ability to analyze and communicate technical information effectively. It's an opportunity to demonstrate your mastery of the subject and present your findings in a professional manner. As such, here are some more guidelines that should be followed:

1. Your report should be well-structured, organized, and clearly written.
2. Use appropriate headings and subheadings to delineate sections and subsections.
3. Support your findings with data, charts, graphs, or any other relevant visual aids.
4. Provide explanations for the choices made during the design of predictors.
5. Discuss any challenges encountered during the lab and how you addressed them.
6. Be concise and precise in your writing, focusing on the key aspects of each task.
7. Cite sources and references as needed, particularly if you consulted external materials.