In this assignment, you will leak information about a program by leveraging the instruction timing side channel of a gem5 processor. After demonstrating that you can leak this information, you will modify the processor to hide any potential leakage. You will also extend the RISC-V ISA to optimize the processor for common cases for which you determine the application doesn't leak any meaningful information. Your submission should include the **code/programs** associated with each part of the assignment, as well as a **written component** that you will answer as you go.

RISC-V developers have proposed an ISA extension called ZKT (Zero Knowledge Timing) to reduce side channels at the instruction granularity. At a high level, the ISA extension provides a software API to developers that ensures that the processor produces the correct output from some inputs to the instruction, but that the instruction executes the longest possible path to getting the output. That is, all instructions take the same amount of time to execute.

You will explore the motivation for the ZKT library by performing analyses in `gem5`. As you will have experienced in Homework 2, gem5 is a complicated tool. Using it or extending it straight out of the box can be tricky, so it will be useful to examine the gem5 cheat sheet and/or to attend/revist Lab 4 in order to familiarize yourself with the tool. Feel free to come by office hours for further clarifications, but note we may refer you to the available resources if they believe the answer can be found there!

Getting Started

Pull the assignment stencil from the course repository. This assignment is based on the RISC-V ISA, so you can start by compiling the gem5 project using `scons build/RISCV/gem5.debug`.

Submission

You are asked to submit the programming component and the write up to Gradescope. The programming component has an autograder with test cases to help facilitate your development process, but not all graded test cases are given. You are also expected to test your code independently!

To submit your code, create a subdirectory called "`submission`" in the base stencil directory. For each of the parts with programming components, you will create subdirectories for the files to submit with that part (e.g., `submission/part2/...`). Be sure to zip the *contents* of the `submission` directory to submit your code.

Each section of the assignment has a written component, which you should include in a single "Written Response" document. The purpose of these exercises is to encourage you to engage with the nuance of the programming task, so respond in as much detail as you feel is necessary. As a guiding principle, a strong response can be made in about one paragraph.

1 Leakage Mechanisms

Side channels are one of the fundamental security challenges to designing modern architectures. A side channel is information that can be inferred from the execution of a program without having access to application data. Recall from class that we described two side channel vulnerabilities that came from shared caches (Flush+Reload, Prime+Probe), but that these aren't the only means of information leakage in the architecture. For example, if we analyze a program, we can make observations of the timing of certain hardware events, processor and memory parallelism, etc. Then, we can use these observations to make inferences about precise behaviors of the program without needing privileges!

From Section 2.8 of the RISC-V spec, notice that we have access to three instructions in the **Timers and Counters** subsection that can be useful for learning program behavior. These instructions are `RDCYCLE`, `RDTIME`, and `RDINSTRET`, which returns the current cycle number in the processor as a 64-bit value, the current real-world time as a 64-bit value, and the number of currently retired instructions as a 64-bit value. In typical usage, these instructions are useful to do application testing, perform debugging, have event-driven behaviors, etc. For example, to estimate how much time has passed, a developer might implement something like:

```
unsigned long long start = asm(RDTIME);
// do work
unsigned long long elapsed = asm(RDTIME) - start;
return elapsed;
```

Note, usually there are headers/wrappers for the direct assembly call (i.e., time.h, etc). We won't use these directly for this assignment, but they serve as an "existence proof" that we can actually measure some of the stats that gem5 provides.

Your Task

Written Response. In a section clearly labeled "Leakage Mechanisms", please respond to the following.

1. Think back to the emulated processor that you implemented in HW1. In this model of computing, accessing memory is constant time, so attacks like Flush+Reload and Prime+Probe can't be performed. Do other side channels exist in this system? If so, what information can be leaked from the program's execution (i.e., what can you infer without knowing)? If not, what limits this processor's information leakage?

2. Recall that the side channels we examined all took advantage of a multi-processor system, so the spy program could run in parallel with the execution of the victim program. Suppose the device were a uniprocessor (single CPU) with a cache-based memory hierarchy, where all programs share time on the CPU and have their execution scheduled by the OS. Do you think a spy could learn any information from the victim program in this setting? If so, explain how the differences in environment impacts how the spy program would need to change. If not, explain how the environment hides information.

2 Analyzing Leaked Information

Suppose you have found an open source project with some behavior that is dependent on secret user data (e.g., `hw3-tests/leaky/leaky-prog.c`). With access to this source, you can read it, run it locally, try using a bunch of different inputs, etc. to study its behavior. But, if this program is run elsewhere, you will not be able to explicitly see the inputs to that particular source. Your job is to build an automated analysis tool that learns what the input to the program was based on the information leaked by the program's behavior.

Your Task

Programming. To study the behavior, you should run `build/RISCV/gem5.debug -d <output directory> configs/example/hw3/isa-assignment.py --argv <program input>`. The parameter specified by argv is passed to the leaky-prog. As you have found during labs and Homework 2, the performance of the run is written to the `stats.txt` file, which you can use to analyze the behavior of the run. This file is very comprehensive, so feel free to parse through it to try to understand what's happening under the hood of the simulation. Then, modify `analyze-program.py`, which should take *the full path* of a `stats.txt` file as input and print one of **supersecretdata** or **data is not secret** as output based what sort of input would have generated that stats file. However, due to the APIs available to you as an end-user, **you may only use the `simTicks`, `simSeconds`, and `simInsts` fields** in your analysis!

Concretely, the autograder will evaluate your solution by calling `python3 analyze-program.py /path/to/m5out/stats.txt` for various hardware configurations and will expect `supersecretdata` to be printed if this was the input provided. It is possible to implement this in 15 lines of Python code. Be sure that any files written to this part of your submission are in the `submission/part1` directory in your submission.

Written Response. In a section clearly labeled "Analyzing Leaked Information", please respond to the following.

1. Describe the side channel that your `analyze-program.py` is leveraging. Which properties of the CPU impact this side channel? Where else might information be leaked?
2. Your `analyze-program.py` file gets to learn all sorts of values because gem5 reports them all. In practice, if you were leaking information about the program you would need to set up a spy to run alongside the victim leaky-prog. Examine how the simulator is configured in `configs/example/hw3/isa-assignment.py`. What about this configuration is conducive to leaking information, and what about this environment would make it challenging to set up the spy?
3. Examine the output in the `stats.txt` file. Beyond `simTicks`, `simSeconds`, and `simInsts`, find one field that you could conceivably deduce from a spy process. Describe how you might leak this value, and describe any information you may be able to infer from this value. You may assume whatever architecture you would like to leak this information, but make sure to explain your assumed environment.

3 Hiding Leaked Information

Given your work in Part 1, your task is to change the `O3CPU181CA` processor to support the ZKT principles such that an attacker cannot infer any information from the `IntMultDiv` functional unit. Your new processor should compile with the gem5 simulator, and you should develop a way of testing your processor to demonstrate that it now hides the timing of multiplication- or division- bound applications.

The `O3CPU181CA` is defined in the `src/cpu/o3/BaseCPU.py`. You will notice that this processor follows the same definition as the standard `BaseO3CPU`, but uses a custom `FUPool` (functional unit pool) for the purposes of this assignment. This functional unit is defined in the `src/cpu/o3/FUPool.py`.

Your Task

Programming. Find the definition of the problematic unit in the `src/cpu/o3` directory that leaks information about the program's execution. Once you have found the definition, you should modify the relevant components in the `FUPool181CA` to hide any potential leakage. Note, this change will be small (1 line-of-code). Your changes must recompile with gem5 and hide any potential leakage that was found in your analysis from "Analyzing Leaked Information".

Be sure to submit any files that have been modified in your solution are added to the `submission/part2` subdirectory.

Written Response. In a section clearly labeled "Hiding Leaked Information", please respond to the following.

1. Extrapolate on the role of the Functional Unit Pool. What do you think this refers to in the context of the processor? You may find it helpful to look at the source files of the other processor models (e.g., `src/cpu/simple`, `src/cpu/minor`, etc) to help with your answer.
2. Modify the `configs/example/hw3/isa-assignment.py` script so that you are using the `O3CPU` as opposed to the `O3CPU181CA`. What is different about how these two CPUs are implemented? Further, try making the processor speed much faster (10s of GHz) and much slower (10s of MHz) for both processors. What do you find? Provide a hypothesis about what is happening in the processor that would support this finding.
3. Suppose you are setting out to mitigate the side channels from "Analyzing Leaked Information" exhibited by the `leaky-prog`. We could either take a microarchitectural approach, or we could potentially re-write the application to avoid leaking information. Describe an architecture design that could be used to hide the information leaked by this program. What are some of the advantages/limitations of such an architecture? How would this compare to trying to write less leaky software? Answer in 3-5 sentences.

4 Processor Optimization

After making your modifications to the functional unit, you notice that you start to have quality of service (QoS) issues with your processor in that performance degrades beyond a tolerable amount. Your team does some exploration of the sensitive applications and data and determines that multiplying 64 bit values by zero is a very common occurrence. They consult a security expert who proves that no meaningful information can be gleaned from knowing that some sensitive data is multiplied by zero. Given this, your team tasks you with modifying the gem5 processor model to see the performance benefits of optimizing multiplication by 0. For the sake of simplicity in processor logic, your team determines that the easiest way forward is to modify the ISA to add a zero multiplication instruction.

Your Task

Programming. You will extend your processor to introduce a new class of instruction. Note, **there are several moving pieces to complete this step**. While no individual modification will require more than one-to-two lines of code to be added, you will want to ensure that all of your intermediate changes compile in the context of the larger simulator.

In `src/cpu/op_class.hh`, gem5 defines a set of instruction types that links a particular instruction to a particular hardware device. These are tied to an enumerated device, which is declared in `src/cpu/FuncUnit.py`. Add a zero multiplication `OpClass` and `FuncUnit` to the gem5 base processor. Doing so requires updating the backend source both in the C++ backend and in the Python backend, but should only require changing two lines of code. *Be sure that the simulator compiles before moving on.*

Once you've added this new `OpClass` to the processor, be sure to extend one of the components in the pool of functional units to be able to interpret your new zero multiplication `OpClass`. Again, ensure that the simulator compiles before moving on.

Then, come up with your new instruction. Start with the RISC-V Card. First, notice that multiplication is expressed as a Standard Extension to the RISC-V ISA. In particular, look at the RV64M standard extension. These instructions all have an opcode of 0x3B, and use the funct3 field to switch on which decoded instruction to execute. Given this, come up with a format for your new `zmul` instruction within this instruction layout.

Then, look at the RISC-V decoder in gem5 (`src/arch/riscv/isa/decoder.isa`). This file can be intimidating, but it is essentially a large JSON file that the gem5 processor uses to determine which format a particular instruction is conforming to. Find where addition, subtraction, and multiplication is defined in the decoder file. Given your new instruction format, implement the `zmul` instruction in the decoder.isa file. For the most part, feel free to copy the syntax for similar instructions, but make sure that Rd is set to what you expect at the end of a zero multiplication instruction. Like the multiplication instruction, your instruction should specify the newly created zero multiplication `OpClass`. Ensure that gem5 compiles with this new instruction in the decoder logic.

Be sure to include your updated `op_class.hh`, `decoder.isa`, and any other files that needed to be modifed to your `submission/part3` directory.

5 Exposing `zmul` to Software

Now your processor understands zero multiplication to be a much faster operation. Unfortunately, the compiler still doesn't know that this instruction exists. Redesigning the compiler is an arduous task (and is out of the scope of this class!), so instead you hope to do a post-compilation scan of the binary, and will modify the relevant instruction bytes to now be the zero multiplication bytes.

Your Task

Programming. Examine the files in the `hw3-tests/zero-mult/` directory. Notice that there is a RISC-V binary file that directory corresponds to the `zero-mult.c` source file. Write a program called `convert2zmul.py` that converts all of the multiplication instructions in this file to your new zero multiplication instruction. If done correctly, when running gem5 with this file, your program will print out that the output of each multiplication operation is zero. (Hint: Remember endian-ness!)

Include your `convert2zmul.py` file in `submission/part4`.

Written Response. Use what you learned in this assignment to draw some conclusions about RISC versus CISC ISAs, gem5 as a simulation tool, etc. from the following prompts. In a a section clearly labeled "Exposing `zmul` to Software", please respond to the following.

1. By adding the zmul instruction to the RISC-V ISA, you've essentially implemented a lightweight RISC-V extension. Consider what changes were required to the gem5 processor. Do you think that implementing a comprehensive set of special instructions for optimization special case arithmetic warrants an ISA extension? Why or why not? More generally, what do you think is the standard of what an ISA extension should contribute in order to be incorporated into a RISC-V processor? You may want to consider Chapter 21 of the RISC-V specification in your answer.

2. Think about gem5 as a research tool as compared to the processor emulator that you developed in Homework 1. In particular, consider the level of detail required to simulate behaviors in hardware for each of these tools. When might a researcher opt to use one tool versus the other? In particular, suppose you were trying to study whether or not a new hardware component suffers from a side channel. Describe the thought process for choosing a simulation tool in 3-5 sentences.

3. As we saw in class, CISC ISAs like x86 have a more robust set of instructions that allow the programmer to use various parts of the underlying hardware to optimize the execution of a program. However, some of these instructions (like `clflush`) enable easier access to implement side channel attacks like Flush+Reload. Describe what incentives Intel has to make this design decision in the x86 ISA and the implications of these incentives on the security/privacy of the end-user.

6 Reflection

In a a section clearly labeled “Reflection”, please respond to the following.

1. What were your main takeaways from this assignment?
2. What suggestions do you have for improving the assignment in the future?
3. What questions do you still have about ISA design and/or side channels?