**CS 4290: Advanced Computer Organization**

**Project 2**

**Due: 07/07/2020, 11:55 pm, EST**

**Important policies:**

1. Sharing of code between students is viewed as cheating and will receive appropriate action in accordance with University policy.
2. It is acceptable for you to compare your results, and only your results, with other students to help debug your program. It is not acceptable to collaborate either on the code development or on the final experiments.
3. You should do all your work in the C or C++ programming language and should be written according to the C99 or C++11 standards, using only the standard libraries.
4. Unfortunately, experience has shown that there is a very high chance that there are errors in this project description. The online version will be updated as errors are discovered. It is your responsibility to check the website often and download new versions of this project description as they become available.

**Project Description**

In this project, we will be designing a simulator that implements a simplified version of Tomasulo’s algorithm. You are tasked to design the register alias table and reservation stations.

Each line of the trace file is in the following format

OP RD RS RT

* OP indicates the operation type. There will be multiple different types of instructions, each with a different latency (i.e. ADD takes 2 cycles, DIV takes 15 cycles, MEM takes 20).
* RD is the destination register number (i.e. 1 for R1)
* RS in the first source register number. If there is none (for example, an immediate value), the “number” is -1.
* RT is the second source register number.

You will be provided driver code that will convert the traces to instructions. You are only responsible for filling in the following functions (along with any other classes and variables as needed).

FUs are pipelined. You start from cycle number 1.

**Specification of Simulator**

Explanation of functions you need to fill in:

void scheduler\_unified\_init(int num\_registers, int rs\_size)

* This function is called if the type of scheduler uses a unified reservation station.

void scheduler\_per\_fu\_init(int num\_registers, int rs1, int rs2, int rs3)

* This function is called if the type of scheduler uses per-FU reservation stations.

bool scheduler\_try\_issue(op\_type op, int dest, int src1, int src2)

* This function tries to issue a new instruction with the given arguments. If successful, return true, if not, return false (i.e the RS is full).

void scheduler\_step()

* Increment the clock by 1 step

void scheduler\_start\_ready()

* Start any ready instructions

void scheduler\_clear\_completed()

* Clear completed instructions

bool scheduler\_completed()

* Return true if all instructions are completed and cleared.

void scheduler\_complete()

* Complete the scheduler and compute any final stats

**Statistics (output)**

The simulator outputs the following statistics after completion of the run:

1. Number of instructions
2. Number of cycles
3. Instructions Per Cycle (IPC)
4. Number of times the issue was stalled (issue failed)
5. Maximum number of instructions started at once
6. Maximum number of instructions completed (wrote back) at once
7. Maximum number of instructions active per FU

**Validation**

Several test traces will be provided along with the correct output. You must run your simulator and debug it until it matches 100% all the statistics in the validation outputs posted on the website. It is highly recommended that you first try to work through each trace for each scheduler type by hand.

**Experiments**

For each benchmark in the traces directory, design a instruction scheduler subject to the following goals:

1. You have a total budget of 10 entries for the reservation stations (10 all in the unified RS, or X for each RS that sums up to 10 entries)
2. The scheduler should have the highest possible IPC at the end. You may vary any parameter (RS type, entries per RS).

**Deliverables**

What to hand in via T-Square:

* **schedulersim.hpp, schedulersim.cpp**
  + The commented source code for the simulator program itself (you only need to provide schedulersim.hpp, schedulersim.cpp).
* **<gt\_username>\_prj2.pdf**
  + A document with the design results of the experiments for each trace file, with a persuasive argument of the choices that were made. (An argument may be as simple as an explanation of the search procedure used to find the designs and a statement about why the procedure is complete.) This argument should include output from runs of your program. (There are multiple answers for each trace file, so I will know which students have "collaborated" inappropriately!)

Remember that your code must compile and run on a current variant of Linux (i.e., Debian, Red Hat, Ubuntu) running on an x86 architecture (i.e., Intel or AMD).

Late submissions will be deducted 25% per day

**Grading Rubric**

0% You do not hand in anything

+50% Your simulator doesn't run, does not work, but you hand in significant commented code

+35% Your simulator matches the validation outputs

+15% You ran all experiments and found the best-performing schedulers