CMPEN 431: Computer Architecture, Spring 2019

Course Project #3
Out-of-Order Scheduling

Project Goal

You are required to implement a program in Python which performs dynamic (OoO) scheduling of instructions with conservative load-store ordering on a restricted set of simplified instruction set.

Details

You are provided with a framework which takes a sequence of instructions and generates a schedule describing how each instruction will propagate through an OoO processor's pipeline stages. Your task is to implement the internals of such an OoO scheduler. Specifically, the provided framework implements "Fetch" and "Decode" pipeline stages and you are required to implement the remaining stages (Rename, Dispatch, Issue, Writeback and Commit). File input/output is already taken care of by the framework so you can focus on the various pipeline stages.

Your program will take two command line arguments: input filename and output filename.

Input file format

The input file will consist of a first line with two comma-separated (no whitespace) positive integers followed by between 1-256 lines of a format described below.

The two integers on the first line will specify the number of physical registers in the system and the issue width of the machine. **All machine resources will match issue width**. The number of physical registers will always be greater than 32.

Each subsequent line will contain one of the following "instructions"

```
R,<REG>,<REG>,<REG>
I,<REG>,<REG>,<IMM>
L,<REG>,<IMM>,<REG>
S,<REG>,<IMM>,<REG>
```

where {R,I,L,S} are the capital letters R, I, L, and S, {,} is comma, <REG> is a positive integer value between 0 and 31, inclusive and <IMM> is a POSITIVE integer value between 0 and 65535, with both <REG> and <IMM> encoded as decimals. The first <REG> is the destination for R, I, and L. Memory (not modeled) is the destination for S.

Output file format

Assuming that the first instruction is always fetched in cycle 0, for each instruction in the input file, produce a corresponding line in output file of the form:

`<FE>,<DE>,<RE>,<DI>,<IS>,<WB>,<CO>`

where all comma-separated fields are non-negative integers encoded as decimals and represent the cycle in which the associated instruction completes the specified stage. **For simplicity, assume that S instructions occupy WB in the cycle after they issue.**

Miscellaneous

Note following restrictions with regards to register freeing and conservative load-store scheduling apply:

- Registers freed at commit in a cycle are not available on the free list until the following cycle
- Potentially dependent memory operations cannot issue in the same cycle.
- Assume that all memory operations hit.
- Assume that all instructions writeback in the cycle following their issue.
- Assume that the initial architectural to physical mappings are A0→P0, A1→P1, …, A31→P31 and all other physical registers are on the free list in increasing register order (i.e. the next register consumed in renaming will always be P32).

Framework

A barebones Python framework for the above task has been provided. A set of 5 different instruction sequences to be scheduled has been provided in `inputs` directory. The framework can be invoked use the following command:

```
# Extract project files archive and navigate to project directory.
make
```

Output files will be generated in `outputs` directory. Expected output for all given instruction sequences is available in `outputs-correct` directory.

Framework generates verbose output on console. You can optionally disable debug messages by commenting out the following line: `main.py:24`.

Submission Requirements

This is an individual project.

You will submit all source files (*.py files, Makefile*)* of your implementation as a single *tar* archive. Make sure your code works on CSE machines without errors.

You are free to make changes to the provided framework.

The following command will be used to execute your code (followed by analysis of generated output files):

```
# Extract project files archive and navigate to project directory.
rm outputs/*
make
```