Project 1

- Implement a trace cache in 16-wide superscalar using sim-outorder
- Evaluate the performance in terms of IPC
- Evaluate hit/miss rates and utilization of the trace cache
- Due: by the end of the term,
- But work as quickly as you can since there will be more projects coming 😊
Hints

- Copy the simplescalar to your work area and compile by typing the following commands:
  - `cp -f -r /stash/akkary/ss3 .`
  - `cd ss3`
  - `make clean`
  - `make`

- Test by running a small binary, e.g.
  - `sim-outorder /stash/akkary/tests/bubblesort`
Hints

- Study until you completely understand the functions `sim_main()` and `ruu_fetch()` in `sim-outorder.c`
- Start coding the trace cache fill logic in `ruu_fetch()`. For that you need:
  - The beginning address of the first instruction of next 16 instruction trace
  - The direction of the branches in the trace
  - Do not place any limit on the number of branches in the trace
  - You need to account for branch mispredictions
Coding the fill buffer

- Use a large fill buffer, with a tail pointer
- Advance tail when an instruction is fetched, storing when you may need in the fill buffer, e.g. instruction address, branch information, etc…
- Mark instructions when they are retired and clear instructions after branch mispredictions
- When a trace cache block commits, e.g. 16 instructions, the block is ready to write into the trace cache
- The buffer has to be large enough to store all instructions in the pipeline until at least one trace block is retired, e.g. 16 instructions retired for a 16-wide trace block
- Each instruction is identified in the simulator with a unique ID stored in ptrace_seq variable in the pipeline and the pipeline buffers, e.g. fetch queue, RUU, etc…
  - So ptrace_seq is very handy for identifying which instructions in the fill buffer are committed and which instructions are flushed due to a branch misprediction
Handling branch recovery

- See ruu_recover()
- Fill buffer can be recovered by calling a fill_recover() function from ruu_recover(), passing the mispredicted branch ptrace_seq
Trace cache

- 2-d array: tc[ways][sets]
  - Blocks = ways*sets
  - Instructions = blocks x 16

- LRU implementation:
  - Save in each block time of last read/fill cycle from sim_cycle global variable
Implementing timing

- On every instruction fetch, check the trace cache for a hit
- If hit, update sim_cycle for LRU
- If miss, delay writing the instruction in fetch queue by a fixed delay
  - Implement a knob to set this delay from the command line
  - Delay models the additional time needed to decode instructions fetched from a conventional cache