CS 2410 – Project 2 Frequently Asked Questions

Q: How to interoperate with cache and memory?

A: In project 2, you will use SST's existing cache and memory models. You do not need to implement these models. The description on the last page of the project write-up shows how to connect the cache and the memory together through SST links. The focus of this project is the Tomasulo scheduler that is used by a dynamically scheduled out-of-order execution CPU. The project also shows the power of SST through reuse of existing and tested simulation models (the cache and memory!).

For this project, you will need memory read/write functions that access the memory hierarchy. The memory hierarchy has variable latency since an access might hit or miss in the L1 cache. These functions are the same as you used in project 1. In fact, you can try this in your existing project 1. Simply modify project 1 to include cache (as shown below), and then try rerunning it. It should work with the cache!

Q: How to set cache parameters?

A: The project 2 write-up specifies that two cache parameters can be set from the simulator's JSON configuration file: cache associativity (`associativity`) and cache capacity (`cache_size`). These values can be read from the JSON configuration, and then used to set the SST cache model's corresponding parameters in the SST Python script. Here is an example of how to set the SST cache model parameters in the SST Python script:

```
import json
with open(<configuration file>, 'r') as inp_file:
  sim_config = json.load(inp_file)

cache_config = sim_config.get("cache")
l1_cache = sst.Component("l1cache", "memHierarchy.Cache")
# Setting cache parameters
l1_cache.addParams({
  "cache_line_size":16, # Same as block size
  "associativity":cache_config.get("associativity"),
  "cache_size":cache_config.get("size"),
  "cache_frequency": "1GHz", # Same as cpu
  "access_latency_cycles": 1,
  "L1": True
})
```

Q: How to handle loads and stores? Do I need to model the data and the addresses?

A: You can modify project 1 to generate instruction traces. To do so, simply change your project 1 to output each instruction executed that is not a control transfer. The output is just the hexadecimal encoding of the instruction (in ASCII). For loads and stores, however, the trace also needs to include the addresses accessed by memory operations. The cache needs to the addresses to correctly access cache/memory. Data values are not needed. For load/store instructions, output both the encoded instruction and the address loaded/stored. E.g., suppose your trace generator executes the instruction `lw $r1,$r0` and the value held in `$r0` is `0x000C`. The trace generator would output this line:

```
4100 000C
```