645 Final Project

Shogun Thomas and Zach McGrath

Approach

We chose to use python3 as our language in which to implement this project. For our design we chose to use three different files: main.py, trace_parser.py and processor.py.

The file main.py is the entry point to this project. As stated in the README, the way to run this project is

```
$> python3 main.py <trace file 0> <trace file 1> <trace file 2> <trace file 3>
```

Main will then get the parsed traces, sort them by timestamp first then processor ID, and loop through each call. This for loop in main is where we implemented our bus. Instead of making a file or class for it, we decided to just do this in main and keep things as abstract as possible. Most of the heavy lifting is done by each processor itself. Main takes care of all of the bus signals and decides which processor should be flushing.

The file trace_parser.py is responsible for parsing the trace files provided at runtime. It splits every trace into processor ID, timestamp, and hex address. It splits up the address into the index, tag and offset components, but converts them to decimal numbers for ease of debugging and indexing. The result of this function call is a list of tuples with 5 elements: processor ID, timestamp, tag, index and offset.

All the processor logic is contained in processor.py. This class holds the cache, a list of tuples. The index into the list is the index from the hex address in the trace files. Each index in this list is a tuple with the information (state, tag). This allows us to use only one list for all of the necessary information. The processor has three primary functions that are used by main which allows main to be a fairly simple file. These functions are execute, change_state_rw, change_state_bus. Execute is the function initially called by main on the processor ID of that cycle. It takes in read/write information and cache line information to determine what bus signal will be generated for a certain data line. Main can then call change_state_rw which then changes the state of that processor, the processor we perform the read/write on, given the same information as execute. We separated these two functions for readability and in order to separate main into two sections: determining what to do, then performing state changes. The final function, change_state_bus, is responsible for taking in a bus signal and changing the processor, the processor we take the data from, flush, etc., state for this signal. This is for all of the other processors that aren't executing.

Results

Our results are close to the desired results. The desired state results for P0 state is as following:

```
m: 9    o: 90    e: 12    s: 26    i: 375
```

Our results were

```
                    m: 9     o: 92    e: 10    s: 26    i: 375
```

The only divergence between our results and the desired results are the counts of states ‘Owner’ (o) and ‘Exclusive’ (e). We have 2 too many for ‘o’ and 2 few for ‘e’. We believe that the cause of this is some edge case where ‘o’ should be transferring to the ‘i’ state in order to go to the ‘e’ state later on.