**Install SimpleScalar**

- Click “Tools”
- Download simplesim-3v0e.tgz
- Click “Benchmarks”
- Download “Instructor benchmarks”
- In Linux command window:

  ```
  zcat simplesim-3v0e.tgz | tar -xvf -
cd simplesim-3.0
make config-alpha
make
  
  #Set up
  cd ..
zcat instruct-progs.tar.gz | tar -xvf -
mv benchmarks simplesim-3.0/
  ```

**Cache Experiment**

The following command runs the simulator on the cc1 benchmark, with a direct-mapped L1 cache specified.

```
```

Each cache is specified as `<name>:<nsets>:<bsize>:<assoc>:<repl>`. The direct mapped cache has 128 sets, with 1 block of 64 bytes per set, making it an 8 kB cache. A 2-way set associative cache of the same total size would be simulated using the following command.

```
./sim-outorder -cache:il1 il1:64:64:2:1 -cache:dl1 dl1:64:64:2:1 benchmarks/cc1.alpha -O benchmarks/1stmt.i
```

Here are commands to run the “go” and “anagram” benchmarks.

```
```
Fill out the following table by simulating benchmarks with different cache configurations. The total size of the cache should be 8 kB in all cases, and the block size should be 64 bytes.

<table>
<thead>
<tr>
<th></th>
<th>I-Cache Miss Rate</th>
<th>D-Cache Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direct 2-Way 4-Way</td>
<td>Direct 2-Way 4-Way</td>
</tr>
<tr>
<td>cc1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td></td>
<td></td>
</tr>
<tr>
<td>anagram</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now, assume the following cycle times for processors with each cache:
- Direct Mapped: 0.29 ns
- 2-Way: 0.31 ns
- 4-Way: 0.51 ns
- 4-Way (2-cycle): 0.29 ns

For the 2 cycle cache, add the following options to the simulator command (before the benchmark is specified):
```
-cache:dl1lat 2 -cache:ll1lat 2
```

Fill out the following table with the total time. This will be the number of cycles times the cycle time.

<table>
<thead>
<tr>
<th></th>
<th>Direct Mapped</th>
<th>2-Way</th>
<th>4-Way</th>
<th>4-Way (2 cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anagram</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions to think about:
- For each benchmark what configuration worked best for performance?
- Did the lowest miss rate always result in the best performance?
- In what situations would a 2-Cycle 4-Way cache work better than a 1-cycle direct mapped cache?
- In what situations would a 1-cycle direct mapped cache work better than a 2-Cycle 4-Way cache?

Branch Predictor Experiment

The following command explicitly sets a GAg branch predictor.

```
./sim-outorder -bpred:2lev 1 65536 16 0 benchmarks/cc1.alpha -O benchmarks/1stmt.i
```
The branch predictor is specified as `<# entries in first level> <# entries in 2nd level> <width of shift register(s)> <xor history>` (compactly, `<N> <M> <W> <X>`). Please experiment with the following types of 2-level branch predictor:

- `GAg : 1, M, W, 0` where `M = 2^W`
- `GAp : 1, M, W, 0` where `M = C * 2^W`, `C` is the number of per-address prediction tables
- `PAg : N, M, W, 0` where `M = 2^W`
- `PAp : N, M, W, 0`

Each branch predictor should have the same size in the second level, where N=4 and C=2.

Fill out the following table with the prediction rate (direction):

<table>
<thead>
<tr>
<th></th>
<th>GAg</th>
<th>GAp</th>
<th>PAg</th>
<th>PAp</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anagram</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fill out the following table with the performance in cycles:

<table>
<thead>
<tr>
<th></th>
<th>GAg</th>
<th>GAp</th>
<th>PAg</th>
<th>PAp</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anagram</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>