Project 1 Report

EEL4268 – 00379

Yousef Awad

November 2025

Contents

Testing Part A

We were told to test with the following specifications:

- Cache Associativity: 4
- Write-Back
- Replacement Policy: LRU

There the command is the following:

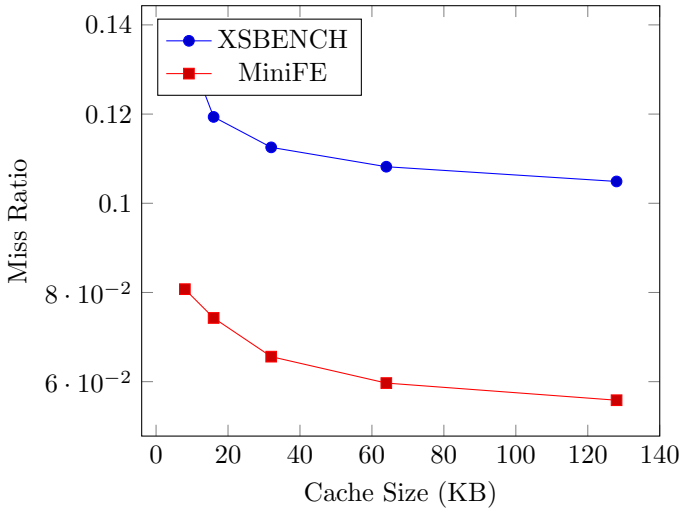
./SIM CACHE_SIZE 4 0 1 TRACE_FILE

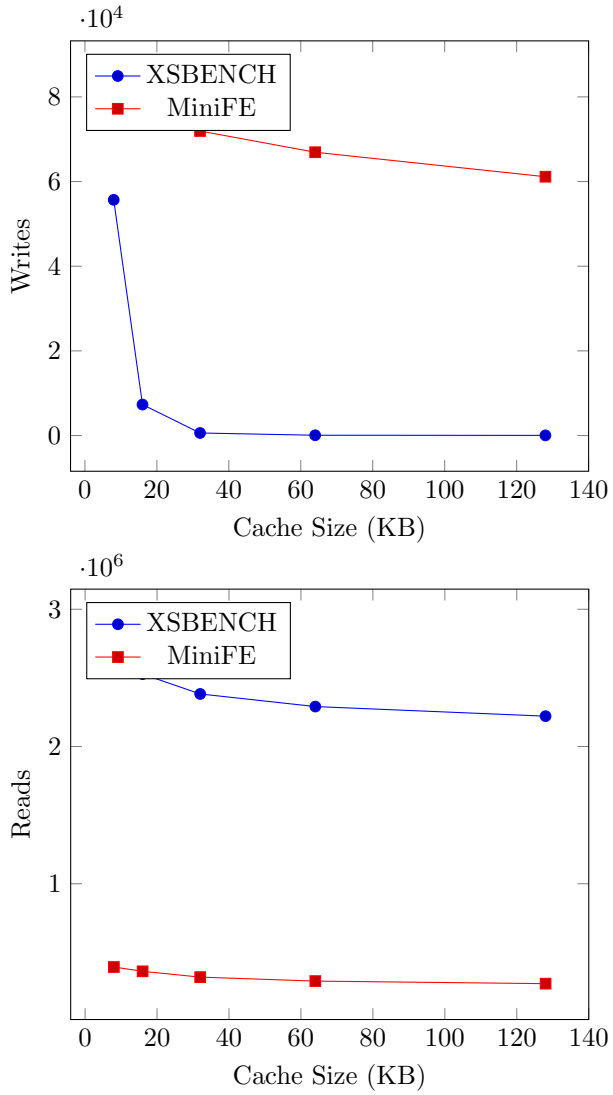
Here are the following plots

Cache-Size	Miss-Ratio	Writes	Reads
8	0.14	55,680	$2.89 \cdot 10^6$
16	0.12	7,318	$2.53 \cdot 10^6$
32	0.11	600	$2.38 \cdot 10^6$
64	0.11	69	$2.29 \cdot 10^6$
128	0.1	36	$2.22 \cdot 10^6$

	Cache-Size	Miss-Ratio	Writes	Reads
MiniFE	8	$8.07 \cdot 10^{-2}$	84,762	$3.94 \cdot 10^5$
	16	$7.43 \cdot 10^{-2}$	77,553	$3.62 \cdot 10^5$
	32	$6.56 \cdot 10^{-2}$	71,944	$3.2 \cdot 10^5$
	64	$5.97 \cdot 10^{-2}$	66,926	$2.91 \cdot 10^5$
	128	$5.58 \cdot 10^{-2}$	61,142	$2.72 \cdot 10^5$

And here it is graphed (with respect to miss ratio and cache size):




·10^4
XSBENCH
MiniFE
Writes
8
6
4
2
0
0
20
40
60
80
100
120
140
Cache Size (KB)
·10^6
XSBENCH
MiniFE
Reads
3
2
1
0
20
40
60
80
100
120
140
Cache Size (KB)

Testing Part B

We were told to test with the following specifications:

- Cache Associativity: 4
- Write-Through
- Replacement Policy: LRU

There the command is the following:

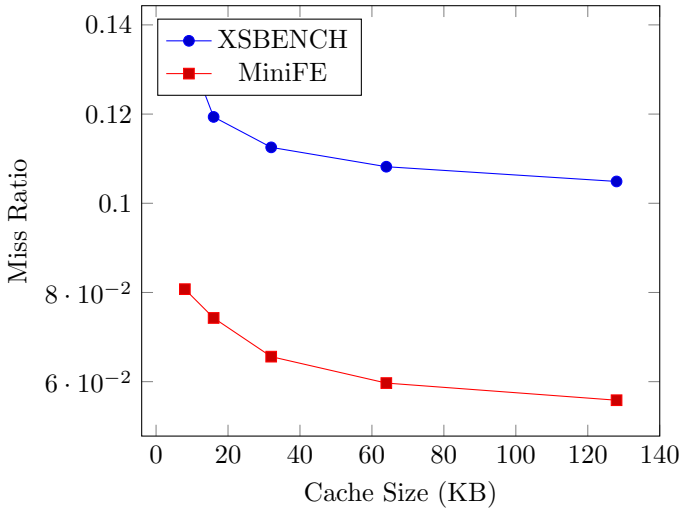
./SIM CACHE_SIZE 4 0 0 TRACE_FILE

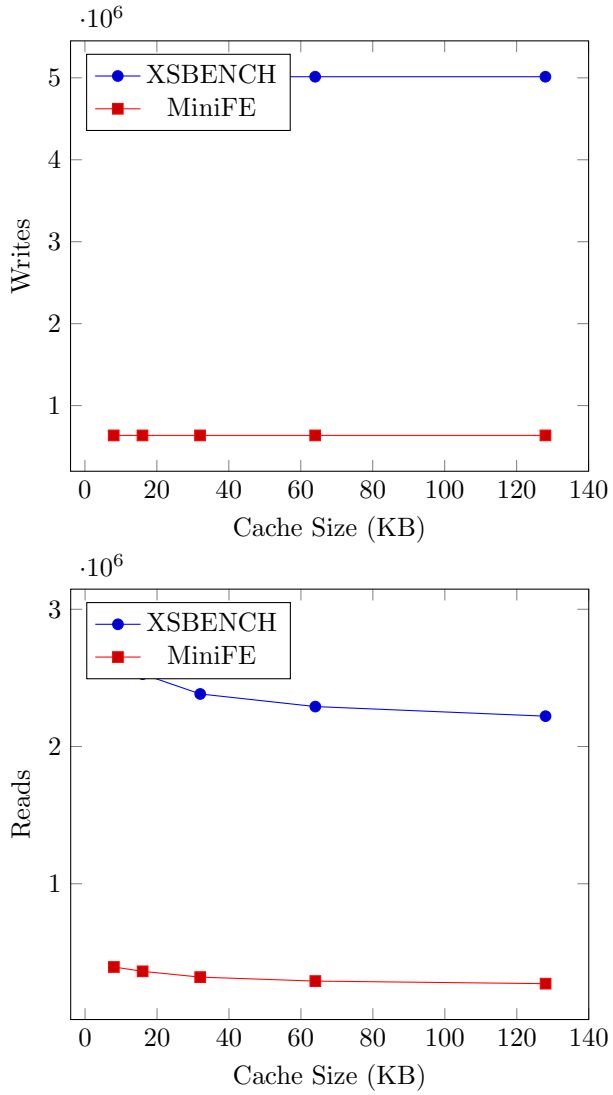
Here are the following plots

Cache-Size	Miss-Ratio	Writes	Reads
8	0.14	$5.01 \cdot 10^6$	$2.89 \cdot 10^6$
16	0.12	$5.01 \cdot 10^6$	$2.53 \cdot 10^6$
32	0.11	$5.01 \cdot 10^6$	$2.38 \cdot 10^6$
64	0.11	$5.01 \cdot 10^6$	$2.29 \cdot 10^6$
128	0.1	$5.01 \cdot 10^6$	$2.22 \cdot 10^6$

	Cache-Size	Miss-Ratio	Writes	Reads
MiniFE	8	$8.07 \cdot 10^{-2}$	$6.36 \cdot 10^5$	$3.94 \cdot 10^5$
	16	$7.43 \cdot 10^{-2}$	$6.36 \cdot 10^5$	$3.62 \cdot 10^5$
	32	$6.56 \cdot 10^{-2}$	$6.36 \cdot 10^5$	$3.2 \cdot 10^5$
	64	$5.97 \cdot 10^{-2}$	$6.36 \cdot 10^5$	$2.91 \cdot 10^5$
	128	$5.58 \cdot 10^{-2}$	$6.36 \cdot 10^5$	$2.72 \cdot 10^5$

And here it is graphed (with respect to miss ratio and cache size):





Note how the hit rate remains steady, with all being written to memory when compared to the write-back mode of cache, as well as how everything else remanis the same as in Part A.

Testing Part C

The command is the following:

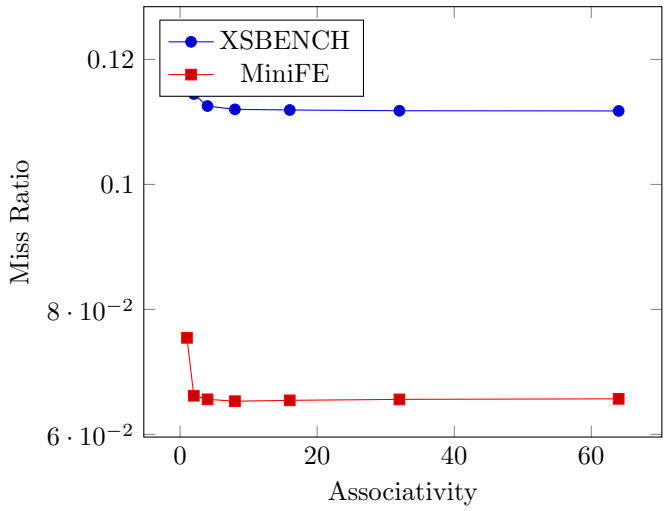
./SIM 32768 ASSOC 0 1 TRACE_FILE

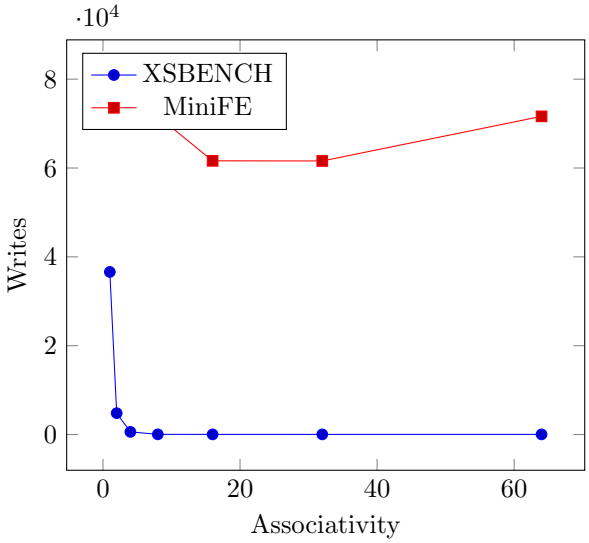
Here are the following plots

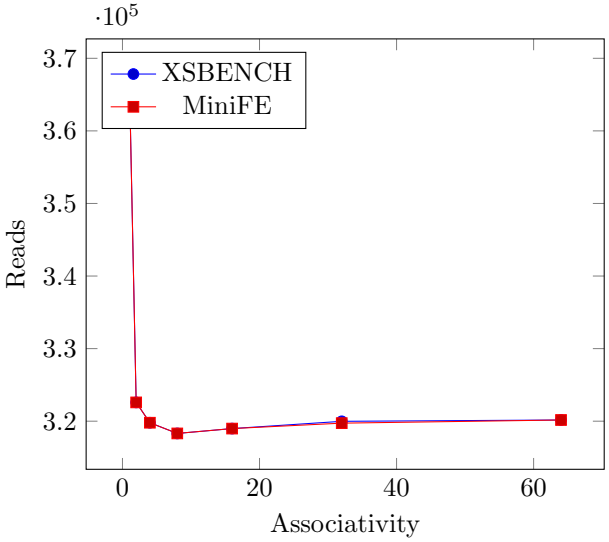
	ASSOC	Miss-Ratio	Writes	Reads
XSBENCH	1	0.12	36,601	$3.68 \cdot 10^5$
	2	0.11	4,814	$3.23 \cdot 10^5$
	4	0.11	600	$3.2 \cdot 10^5$
	8	0.11	44	$3.18 \cdot 10^5$
	16	0.11	35	$3.19 \cdot 10^5$
	32	0.11	35	$3.2 \cdot 10^5$
	64	0.11	35	$3.2 \cdot 10^5$

	ASSOC	Miss-Ratio	Writes	Reads
MiniFE	1	$7.54 \cdot 10^{-2}$	80,773	$3.68 \cdot 10^5$
	2	$6.62 \cdot 10^{-2}$	73,898	$3.23 \cdot 10^5$
	4	$6.56 \cdot 10^{-2}$	71,944	$3.2 \cdot 10^5$
	8	$6.53 \cdot 10^{-2}$	71,660	$3.18 \cdot 10^5$
	16	$6.54 \cdot 10^{-2}$	61,618	$3.19 \cdot 10^5$
	32	$6.56 \cdot 10^{-2}$	61,586	$3.2 \cdot 10^5$
	64	$6.57 \cdot 10^{-2}$	71,625	$3.2 \cdot 10^5$

And here it is graphed (with respect to miss ratio and cache size):




$\cdot 10^4$
XSBENCH
MiniFE
Writes
8
6
4
2
0
0
20
40
60
Associativity


$\cdot 10^5$
XSBENCH
MiniFE
Reads
3.7
3.6
3.5
3.4
3.3
3.2
0
20
40
60
Associativity

Testing Part D

The command is the following:

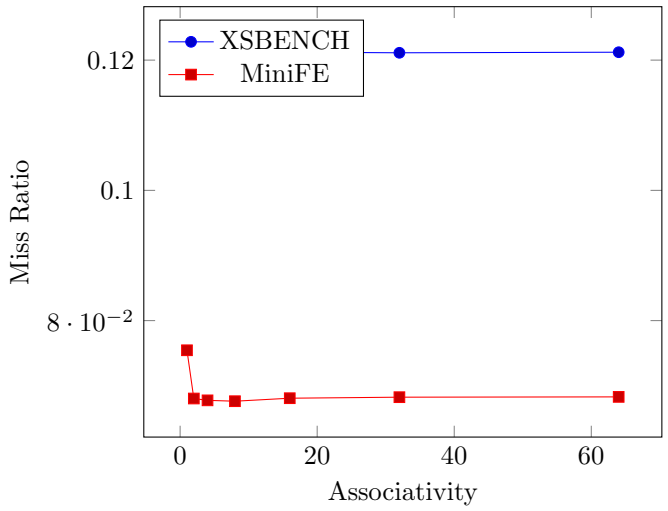
./SIM CACHE_SIZE 4 1 0 TRACE_FILE

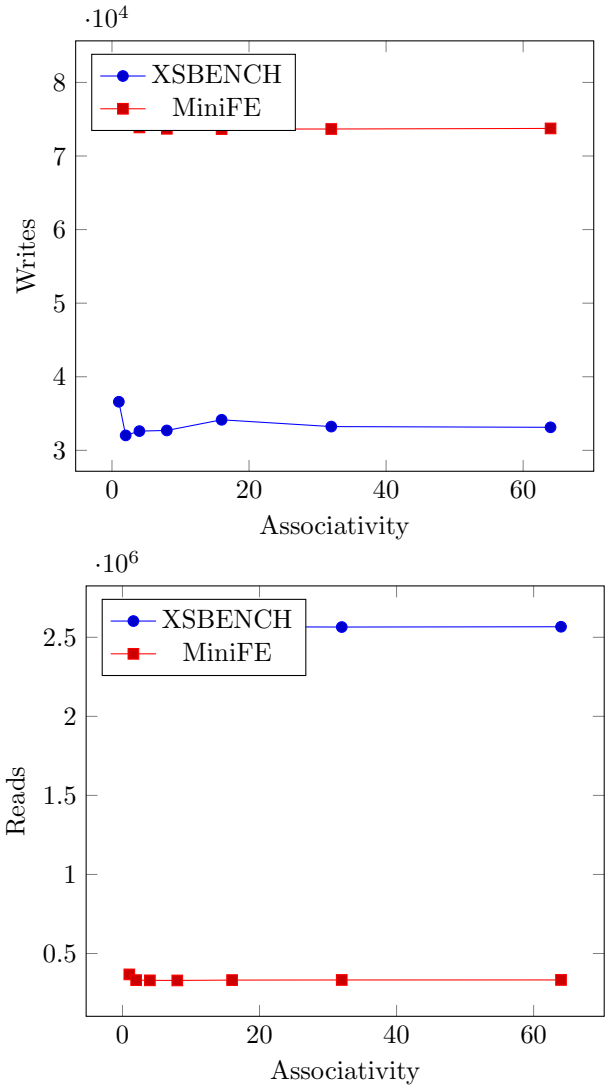
Here are the following plots

	ASSOC	Miss-Ratio	Writes	Reads
XSBENCH	1	0.12	36,601	$2.6 \cdot 10^6$
	2	0.12	32,029	$2.55 \cdot 10^6$
	4	0.12	32,613	$2.56 \cdot 10^6$
	8	0.12	32,701	$2.56 \cdot 10^6$
	16	0.12	34,150	$2.57 \cdot 10^6$
	32	0.12	33,228	$2.56 \cdot 10^6$
	64	0.12	33,128	$2.57 \cdot 10^6$

	ASSOC	Miss-Ratio	Writes	Reads
MiniFE	1	$7.54 \cdot 10^{-2}$	80,773	$3.68 \cdot 10^5$
	2	$6.8 \cdot 10^{-2}$	75,554	$3.32 \cdot 10^5$
	4	$6.78 \cdot 10^{-2}$	73,881	$3.3 \cdot 10^5$
	8	$6.76 \cdot 10^{-2}$	73,699	$3.3 \cdot 10^5$
	16	$6.81 \cdot 10^{-2}$	73,661	$3.32 \cdot 10^5$
	32	$6.82 \cdot 10^{-2}$	73,669	$3.33 \cdot 10^5$
	64	$6.83 \cdot 10^{-2}$	73,748	$3.33 \cdot 10^5$

And here it is graphed (with respect to miss ratio and cache size):




·10^4
8
7
6
5
4
3
Writes
XSBENCH
MiniFE
0
20
40
60
Associativity
·10^6
2.5
2
1.5
1
0.5
Reads
XSBENCH
MiniFE
0
20
40
60
Associativity

Main.cpp

```
/*
 * Made by Yousef Awad in C++ using CMake for the compilation.
 * Yay!
 */
/*
 * The following is all of the return types and what they signify:
 * 0: Program ran succesfully
 * 1: Error was found in the program
 *
 */
#include <fstream>
#include <iostream>
#include <cstring>

// The following are set by the command line
struct cli_defines {
  long long int CACHE_SIZE;
  long long int NUM_SETS;
  long long int ASSOC;
  long long int BLOCK_SIZE = 64;    // 64 byte block size is
    default
  long long int REPLACEMENT_POLICY; // 0 = LRU, 1 = FIFO
  long long int WRITE_POLICY;       // 0 = write-through, 1 = write
    -back
};

// dynamic cache structure variables below
struct dynamic_cache {
  long long int **tag;
  long long int **lru_pos;
  bool **dirty;
  bool **valid;
  long long int **fifo_counter;
  long long int global_counter = 0;
};


void init_cache(cli_defines *cli, dynamic_cache *cache)
{
  // ok so im allocating an array of pointers for each set
  cache->tag = new long long int *[cli->NUM_SETS];
  cache->lru_pos = new long long int *[cli->NUM_SETS];
  cache->dirty = new bool *[cli->NUM_SETS];
  cache->valid = new bool *[cli->NUM_SETS];
  cache->fifo_counter = new long long int *[cli->NUM_SETS];

  // and for each set we allocate arrays for all ways
  for (int i = 0; i < cli->NUM_SETS; i++)
  {
    cache->tag[i] = new long long int[cli->ASSOC];
    cache->lru_pos[i] = new long long int[cli->ASSOC];
    cache->dirty[i] = new bool[cli->ASSOC];
    cache->valid[i] = new bool[cli->ASSOC];
    cache->fifo_counter[i] = new long long int[cli->ASSOC];

```

```
54      // FINALLY we initialize all the blocks in this set to default/
        empty state
55      for (int j = 0; j < cli->ASSOC; j++)
56      {
57        cache->tag[i][j] = -1;
58        cache->lru_pos[i][j] = j;
59        cache->dirty[i][j] = false;
60        cache->valid[i][j] = false;
61        cache->fifo_counter[i][j] = -1;
62      }
63    }
64  }
65
66  // Free's all used memory
67  void cleanup(cli_defines *cli, dynamic_cache *cache)
68  {
69    for (int i = 0; i < cli->NUM_SETS; i++)
70    {
71      delete[] cache->tag[i];
72      delete[] cache->lru_pos[i];
73      delete[] cache->dirty[i];
74      delete[] cache->valid[i];
75      delete[] cache->fifo_counter[i];
76    }
77    delete[] cache->tag;
78    delete[] cache->lru_pos;
79    delete[] cache->dirty;
80    delete[] cache->valid;
81    delete[] cache->fifo_counter;
82  }
83
84  void update_lru(long long int add, cli_defines *cli, dynamic_cache
        *cache)
85  {
86    // breaking down a memory address into cache components
87    long long int set = (add / cli->BLOCK_SIZE) % cli->NUM_SETS;
88    long long int tag_accessing = add / cli->BLOCK_SIZE;
89
90    // finding which way was accessed
91    long long int accessed_way = -1;
92    for (long long int i = 0; i < cli->ASSOC; i++)
93    {
94      if (cache->valid[set][i] && cache->tag[set][i] == tag_accessing
        )
95      {
96        accessed_way = i;
97        break;
98      }
99    }
100
101   if (accessed_way == -1)
102   {
103     return;
104   }
105
106   long long int old_position = cache->lru_pos[set][accessed_way];
107
```

```
  // movin down all da all blocks with position < old_position down by 1
  for (long long int i = 0; i < cli->ASSOC; i++)
  {
    if (cache->lru_pos[set][i] < old_position)
    {
      cache->lru_pos[set][i]++;
    }
  }

  // ok so im setting the accessed block to MRU (position 0)
  cache->lru_pos[set][accessed_way] = 0;
}

long long int find_lru_victim(long long int set, cli_defines *cli, dynamic_cache *cache)
{
  long long int victim = 0;
  long long int max_pos = cache->lru_pos[set][0];

  for (long long int i = 1; i < cli->ASSOC; i++)
  {
    if (cache->lru_pos[set][i] > max_pos)
    {
      max_pos = cache->lru_pos[set][i];
      victim = i;
    }
  }
  return victim;
}

long long int find_fifo_victim(long long int set, cli_defines *cli, dynamic_cache *cache)
{
  long long int victim = 0;
  long long int min_counter = cache->fifo_counter[set][0];

  for (long long int i = 1; i < cli->ASSOC; i++)
  {
    if (cache->fifo_counter[set][i] < min_counter)
    {
      min_counter = cache->fifo_counter[set][i];
      victim = i;
    }
  }
  return victim;
}

long long int find_invalid_block(long long int set, cli_defines * cli, dynamic_cache *cache)
{
  for (long long int i = 0; i < cli->ASSOC; i++)
  {
    if (!(cache->valid[set][i]))
    {
      return i;
    }
```

```
  }
  return -1;
}

void simulate_access(char opcode, long long int add, unsigned long long int* hit, unsigned long long int *miss, unsigned long long int *memory_writes, unsigned long long int *memory_reads, cli_defines *cli, dynamic_cache *cache)
{
  long long int set = (add / cli->BLOCK_SIZE) % cli->NUM_SETS;
  long long int tag_accessing = add / cli->BLOCK_SIZE;

  bool is_hit = false;
  long long int hit_way = -1;

  // checkin for cache hit
  for (long long int i = 0; i < cli->ASSOC; i++)
  {
    if (cache->valid[set][i] && tag_accessing == cache->tag[set][i])
    {
      // Cache hit!!
      (*hit)++;
      is_hit = true;
      hit_way = i;

      if (cli->REPLACEMENT_POLICY == 0) // LRU
      {
        update_lru(add, cli, cache);
      }
      else // FIFO
      {
        // FIFO does nothing...
      }

      // handling the write
      if (opcode == 'W')
      {
        if (cli->WRITE_POLICY == 1)
        {
          // Write-back: mark dirty
          cache->dirty[set][i] = true;
        }
        else
        {
          // write-through!!! write to memory
          (*memory_writes)++;
        }
      }

      break;
    }
  }

  // handle cache miss
  if (!is_hit)
  {
```

```
214     (*miss)++;
215     (*memory_reads)++; // ALWAYS read from memory on miss
216 
217     long long int victim_way;
218 
219     // trying to find an invalid block first
220     victim_way = find_invalid_block(set, cli, cache);
221 
222     // if theyre all valid, need to evict
223     if (victim_way == -1)
224     {
225       if (cli->REPLACEMENT_POLICY == 0)
226       {
227         victim_way = find_lru_victim(set, cli, cache);
228       }
229       else
230       {
231         victim_way = find_fifo_victim(set, cli, cache);
232       }
233 
234       // if im evicting a dirty block in write-back, write to
    memory
235       if (cache->valid[set][victim_way] && cache->dirty[set][
    victim_way] && cli->WRITE_POLICY == 1)
236       {
237         (*memory_writes)++;
238       }
239     }
240 
241     // install that new block
242     cache->tag[set][victim_way] = tag_accessing;
243     cache->valid[set][victim_way] = true;
244 
245     // set dirty bit
246     if (opcode == 'W' && cli->WRITE_POLICY == 1)
247     {
248       cache->dirty[set][victim_way] = true;
249     }
250     else
251     {
252       cache->dirty[set][victim_way] = false;
253     }
254 
255     // write-through!!!: write to memory on write
256     if (opcode == 'W' && cli->WRITE_POLICY == 0)
257     {
258       (*memory_writes)++;
259     }
260 
261     // update replacement policy
262     if (cli->REPLACEMENT_POLICY == 0)
263     {
264       update_lru(add, cli, cache);
265     }
266     else
267     {
268       cache->fifo_counter[set][victim_way] = (cache->global_counter
```

```
    )++;
269     }
270   }
271 }
272
273 int main(int argc, char *argv[])
274 {
275
276   if (argc > 1 && !std::strcmp(argv[1], "--help"))
277   {
278     std::cout << "General Usage: ./SIM <CACHE_SIZE> <ASSOC> <
      REPLACEMENT> <WB> <TRACE_FILE>" << std::endl;
279     std::cout << "Meaning of each term:" << std::endl;
280     std::cout << " <CACHE_SIZE>: Size of the total cache in the '
      CPU', in Bytes (32KB is 32768)" << std::endl;
281     std::cout << "      <ASSOC>: The associativity of the cache
      setup (8-way is 8)" << std::endl;
282     std::cout << "<REPLACEMENT>: The replacement policy of the
      cache (0 for LRU, 1 for FIFO)" << std::endl;
283     std::cout << "         <WB>: Write-Back Policy for cache setup
      (0 for write-through, 1 for write-back)" << std::endl;
284     std::cout << " <TRACE_FILE>: Location of the *.t trace file,
      relative to the binary (or with absolute path)" << std::endl;
285     std::cout << std::endl;
286     std::cout << "Example of a 32KB cache size, with 8-way
      associativity, with LRU replacement, no write-back, and a trace
       file at the same location of the binary called foo.t:" << std
      ::endl;
287     std::cout << std::endl;
288     std::cout << "./SIM 32768 8 0 0 foo.t" << std::endl;
289     return 0;
290   }
291
292   if (argc != 6)
293   {
294     std::cerr << "Usage: ./SIM <CACHE_SIZE> <ASSOC> <REPLACEMENT> <
      WB> <TRACE_FILE>" << std::endl;
295     std::cerr << "To find meaning of all arguments, please do './
      SIM --help'" << std::endl;
296     return 1;
297   }
298
299   cli_defines cli;
300   dynamic_cache cache;
301
302   // parse arguments
303   cli.CACHE_SIZE = std::atoi(argv[1]);
304   cli.ASSOC = std::atoi(argv[2]);
305   cli.REPLACEMENT_POLICY = std::atoi(argv[3]);
306   cli.WRITE_POLICY = std::atoi(argv[4]);
307   const char *trace_file = argv[5];
308
309   // statistic variables
310   unsigned long long int hit = 0;
311   unsigned long long int miss = 0;
312   unsigned long long int memory_reads = 0;
313   unsigned long long int memory_writes = 0;
```

```
314
315   // calculate number of sets
316   cli.NUM_SETS = cli.CACHE_SIZE / (cli.BLOCK_SIZE * cli.ASSOC);
317
318   // self explanatory lulz
319   init_cache(&cli, &cache);
320
321   char opcode;
322   long long int add;
323
324   // Opening file for reading
325   std::ifstream file;
326   file.open(trace_file);
327
328   if (!file.is_open())
329   {
330     std::cerr << "Error: Could not open the trace file." << std::
      endl;
331     cleanup(&cli, &cache);
332     return 1;
333   }
334
335   // Reading the file inputs
336   while (file >> opcode >> std::hex >> add)
337   {
338     simulate_access(opcode, add, &hit, &miss, &memory_writes, &
      memory_reads, &cli, &cache);
339   }
340
341   // Closing the file
342   file.close();
343
344   if (cli.WRITE_POLICY == 1)
345   {
346     for (long long int i = 0; i < cli.NUM_SETS; i++)
347     {
348       for (long long int j = 0; j < cli.ASSOC; j++)
349       {
350         if (cache.valid[i][j] && cache.dirty[i][j])
351         {
352           memory_writes++;
353         }
354       }
355     }
356   }
357
358   // Calculate miss ratio
359   long long int total = hit + miss;
360   double miss_ratio = (total > 0) ? (double)miss / total : 0.0;
361
362   // Print statistics
363   std::cout << "Miss ratio: "  << miss_ratio << std::endl;
364   std::cout << "write " << memory_writes << std::endl;
365   std::cout << "read " << memory_reads << std::endl;
366
367   cleanup(&cli, &cache);
368
```

```
  return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16.0)
project(SIM LANGUAGES CXX)

set(TARGET_NAME SIM)

add_executable(${TARGET_NAME}
  src/main.cpp
)

set(CMAKE_CXX_STANDARD 25)
set(CMAKE_CXX_STANDARD_REQUIRED True)
```