Inf2C Computer Systems

Coursework 2

Caches and Virtual Memory

Deadline: Thu, 30 Nov (Week 11), 16:00

Instructor: Boris Grot

TA: Priyank Faldu

The aim of this assignment is to write a parameterized simulator that models a cache and a Translation Look-aside Buffer (TLB). The simulator, written in C, will read a memory address trace and provide various statistics for the cache and TLB. An outline of the simulator is provided in `mem_sim.c` file to get you started. You are strongly advised to read up on TLB and caches in the lecture notes and course textbook, and to commence work as soon as possible.

This is the second of the two assignments for the Inf2C-CS course. It is worth 50% of the coursework marks and 20% of the overall course marks.

Please bear in mind the guidelines on academic misconduct, which are linked to from the online Undergraduate Year 2 Handbook http://web.inf.ed.ac.uk/infweb/student-services/ito/students/year2.

1 Simulator Design

The simulator consists of two components: cache and TLB.

Cache is a fast memory used for storing instructions and/or data that were recently accessed and, based on the principle of locality, are likely to be used in the near future. When the instruction or data is found in the cache, it saves the processor from a long-latency access to main memory.

Translation Look-aside Buffer (TLB) is a hardware structure which caches mappings from virtual to physical memory pages. If the mapping is found in the TLB, it

avoids a costly access to the page table to perform the translation.

The TLB sits between the processor and the cache(s). When the processor needs to access memory, it first queries the TLB with the virtual page number corresponding to the access address. If the virtual page number is found in the TLB, a TLB hit is registered and the associated physical page number is retrieved from the matching TLB entry. If the TLB does not contain the mapping for the requested virtual page number, then a TLB miss has occurred. In this case, the physical page number is obtained by accessing the page table maintained by the operating system. Once retrieved from the page table, the mapping is used for the access that triggered a TLB miss and is also inserted into the TLB. If TLB is full, then the *Least Recently Used (LRU)* entry is replaced.

Once the physical page number is retrieved either from the TLB or by accessing the page table, a physical address is generated by adding the page offset to the page number. This physical address is then forwarded to the cache and memory hierarchy to complete the load, store, or instruction fetch.

In practice, accessing a page table (often called a *page table walk*) is a complex operation. To keep the coursework tractable, the functionality is abstracted out in a simple function `dummy_translate_virtual_page_num` which takes a virtual page number (and *not* the virtual address) as the argument and returns the physical page number.

To encourage modular program development and testing, your simulator should support three execution modes: 1) `tlb-only`, 2) `cache-only` and 3) `tlb+cache`. **Note that all three are required and will be assessed individually.**

`tlb-only`: In this mode, you will only simulate the TLB. For every virtual address in the trace file, you will figure out the appropriate virtual page number from the address and query the TLB to see if the TLB contains the mapping. If it is a hit, you must update the LRU state. If it is a miss, you will call the provided function `dummy_translate_virtual_page_num` using the virtual page number to find the corresponding physical page number and update the TLB entry with this new mapping, while also updating LRU state appropriately. In the process, you will record the relevant hit/miss statistics for TLB.

`cache-only`: In this mode, you will only simulate the cache. For every virtual address in the trace file, you will convert the virtual address to the physical address using the provided function `dummy_translate_virtual_page_num`. Note that you should not simulate a TLB in this configuration. Once you get the physical page number and, in turn, the physical address, you should query the cache and update the relevant hit/miss statistics.

`cache+tlb`: In this mode, you will combine the cache and TLB. For every address, first you will consult the TLB using the virtual address, and then the cache using physical address as you would do in `tlb-only` and `cache-only` modes. Naturally, a TLB miss will require accessing the page table for the translation.

TLB and cache will use the following parameters:

Fixed parameters: The following parameters always stay the same.

1. Both virtual and physical addresses are 32-bits
2. Memory is byte addressable
3. TLB Organization: Unified (virtual mapping for pages containing data and instructions are kept in the same TLB)
4. TLB Mapping: Fully Associative with LRU replacement
5. Cache Organization: Unified (cache blocks holding data and instructions are kept in the same cache)
6. Cache Mapping: Direct Mapped

Variable parameters: The following parameters are passed as command-line arguments to the program and populated for you by the provided skeleton code.

1. `hierarchy_type`: tlb-only, cache-only or tlb+cache
2. `number_of_tlb_entries`: 8 or 16
3. `page_size`: 256 or 4096
4. `number_of_cache_blocks`: 256 or 2048
5. `cache_block_size`: 32 or 64

1.1 Trace and Code Files

You are provided with two files: 1) `mem_trace.txt` and 2) `mem_sim.c`. The memory trace file, `mem_trace.txt`, consists of a sequence of memory addresses. The code to read this trace file is provided for you. Your simulator should consume the addresses in the trace file one after another, in order. Each line in the trace file consist of two fields: 1) I/D, which specifies whether the address is for an instruction or data access and 2) 32-bit byte addressable virtual memory address. An example of the trace file is as follows:

I 8cda3fa8
I 8158bf94
D 8cd94c50
I 8cd94d64
D 8cd94c54

where the letter `I` indicates that it is an instruction address, `D` denotes a data address, and is followed by a 32-bit address in hex format.

The C file, `mem_sim.c`, contains the code for:

1. reading the command-line parameters and initializing the corresponding variables
2. reading the trace file
3. printing the output of simulation statistics

1.2 Output Format

For your convenience, the output of the program is already provided in `print_statistics(...)` function. You need to populate the parameters of the function with appropriate values. Following is the full list of all variables you need to populate.

1. `g_total_num_virtual_pages`: total number of virtual pages in the system
2. `g_num_tlb_tag_bits`: number of bits required to represent virtual page number in TLB
3. `g_tlb_offset_bits`: number of bits required to access offset within a page
4. `g_num_cache_tag_bits`: number of bits required for tag in cache
5. `g_cache_offset_bits`: number of bits required to access offset within a cache block
6. `g_result.tlb_data_hits`: total number of TLB hits while accessing data
7. `g_result.tlb_data_misses`: total number of TLB misses while accessing data
8. `g_result.tlb_instruction_hits`: total number of TLB hits while accessing instructions
9. `g_result.tlb_instruction_misses`: total number of TLB misses while accessing instructions
10. `g_result.cache_data_hits`: total number of cache hits while accessing data
11. `g_result.cache_data_misses`: total number of cache misses while accessing data
12. `g_result.cache_instruction_hits`: total number of cache hits while accessing instructions
13. `g_result.cache_instruction_misses`: total number of cache misses while accessing instructions

Summary: In this assignment, you are required to write a TLB and cache simulator which is fed by a trace of memory addresses. You will need to appropriately configure the TLB and cache data structures based on parameters that are passed as command-line arguments. You will collect the TLB and cache statistics, which will be output at the end of the simulation by the provided function.

1.3 Compiling and Running the simulator

Compile the simulator on the DICE machines with the command:

```
gcc -o mem_sim mem_sim.c -std=gnu99 -lm
```

This creates an executable `mem_sim`. Following are some valid command-line parameters to run the simulator.

```
./mem_sim tlb-only 16 4096 mem_trace.txt
```

where `mem_sim` is the executable name. `tlb-only` indicates you should simulate only TLB and not the cache. Total number of entries in TLB is 16 and each page is 4096 bytes. `mem_trace.txt` is the name of the tracefile which is located in the same directory as the `mem_sim`.

```
./mem_sim cache-only 4096 256 32 mem_trace.txt
```

`cache-only` indicates you should simulate only cache and not the TLB. Each page is 4096 bytes. Total number of cache blocks are 256 and each block is 32 bytes.

```
./mem_sim tlb+cache 16 4096 256 32 mem_trace.txt
```

`tlb+cache` indicates you should simulate both TLB and the cache. Total number of entries in TLB is 16 and each page is of 4096 bytes. Total number of cache blocks are 256 and each block is 32 bytes.

Make sure that your simulator compiles and runs on a DICE machine without errors and warnings.

Potential output looks as below for the following input

```
./mem_sim tlb+cache 16 4096 256 32 mem_trace.txt
```

```
input:trace_file: mem_trace.txt
input:hierarchy_type: tlb+cache
input:number_of_tlb_entries: 16
input:page_size: 4096
input:number_of_cache_blocks: 256
input:cache_block_size: 32

NumPageTableAccesses:40
TotalVirtualPages:1048576
TLBTagBits:20
TLBOffsetBits:12
TLB:Accesses:80
TLB:data-hits:20, data-misses:20, inst-hits:20, inst-misses:20
```

```
TLB:total-hit-rate:50.00%
CacheTagBits:19
CacheOffsetBits:5
Cache:data-hits:20, data-misses:20, inst-hits:20, inst-misses:20
Cache:total-hit-rate:50.00%
```

Note that the values in the sample output are just random numbers and are not intended to reflect anything about the correct output. NumPageTableAccesses statistic is automatically generated based on how many times you call `dummy_translate_virtual_page_num` function. Ideally, it should be equal to the total number of TLB misses if TLB is simulated or total number of memory accesses. Also, if TLB or Cache is not simulated, the corresponding statistics are not printed by the output function.

1.4 Submission

You should submit a copy of your simulator by 4pm on November 30, 2017 using the below command at a command-line prompt on a DICE machine.

```
submit inf2c-cs 2 mem_sim.c
```

Unless there are special circumstances, **late submissions are not allowed**. Please consult the online undergraduate year 2 student handbook for further information on this.

You can submit more than once up until the submission deadline. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.

Warning: Unfortunately the `submit` command will technically allow you to submit late even if you submitted before the deadline. Don't do this! We can only retrieve the latest version, which means you will be penalized for submitting late.

For additional information about late penalties and extension requests, see the School web page below. Do **not** email any course staff directly about extension requests; you must follow the instructions on the web page.
http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests.

1.5 FAQ

1. You are expected to dynamically allocate memory for the TLB and cache data structures. Submissions that use statically allocated memory (e.g., statically declared arrays of fixed size) will be penalized.

2. You may find it easier to start the coursework by first implementing the cache-only mode of the simulator. For development and testing purpose only, you may feed the addresses from the trace file directly to the cache without translating

them. As virtual addresses and physical addresses are both of 32-bits, the cache interface should not require any modifications. Once you are happy with the cache implementation, you can add address translation. We recommend first doing translation without the TLB, and then with the TLB. Note that your final cache-only simulator must work with translated addresses.

3. To verify correctness of your implementation, you can write your own trace files with easily predictable hit rates. For instance, think of a trace with four accesses that uses four different addresses and has a 50% cache hit rate.

4. Your simulator will be tested with trace files different from the one provided.

5. Your implementation will be assessed on correctness and the code will be assessed on quality and readability. The latter includes using modular programming practices, quality comments, meaningful variable names and a clean presentation.

6. You can use the C library functions available from the header files that are already included in `mem_sim.c`. You may not use any library functions beyond these (i.e., do not include other C header files).

7. Your simulator must compile and run on DICE without warnings or errors when compiled with `gcc -o mem_sim mem_sim.c -std=gnu99 -lm`

8. You can submit more than once up until the submission deadline. Late submissions are not allowed and will receive a mark of 0.

2 Similarity Checking and Academic Misconduct

You must submit your own work. Any code that is not written by you must be clearly identified and explained through comments at the top of your files. Failure to do so is plagiarism. Detailed guidelines on what constitutes plagiarism can be found at: http://web.inf.ed.ac.uk/infweb/admin/policies/guidelines-plagiarism. All submitted code is checked for similarity with other submissions using the MOSS [1] system. MOSS has been effective in the past at finding similarities. It is not fooled by name changes and reordering of code blocks.

3 Questions

If you have any questions about this assignment, you may talk to Inf2C-CS lab demonstrators, tutor or the course instructor. Alternatively, you may also post questions on the online course discussion forum at https://piazza.com/ed.ac.uk/fall2017/inf2ccs/home.

November 14, 2017

[1] http://theory.stanford.edu/~aiken/moss/