assign3.md

Assignment #3: Cache Emulation and Performance Evaluation

- SNU ECE-430.322
- Computer Organization, 2020 Spring
- **Due date: To be announced later**

In this assignment, you will be implementing cache in the RISC-V emulator, which will be used to evaluate the cache performance. Before you start this assignment, we strongly recommend you read the textbook Section 5.3 (The Basics of Caches) and Section 5.4 (Measuring and Improving Cache Performance). The lecture slide #12 would help your understanding as well.

Outline of Assignment #3: You will be extending `rv32emu` to implement two different cache structures: 1) a direct-mapped cache (Task1) and 2) a 4-way set-associative cache (Task2). Then based on your cache implementation done in Task1 and Task2, you will be evaluating an application computing the matrix multiplication (Task3). More specifically, for a given configuration (i.e., the row/column size of a matrix, or whether the application accesses the matrix in column-major orders or row-major orders), you will be analyzing which cache structure is better than the other.

Notes on Running Environment for Assignment #3

For Assignment #3, the compiler setup is the same as Assignment #1 but the emulator is not. You should download `rv32emu-assign3.tar.gz` and unzip it to the directory `BASE/rv32emu-assign3`.

Install: RISC-V Compiler

Follow the instruction that we provided in Assignment #1. Don't forget to setup the `PATH` environment variable.

```
BASE $ export PATH=`pwd`/xPacks/riscv-none-embed-gcc/8.2.0-3.1/bin:$PATH
```

As you have done in Assignment #1, if you can see the following message, your RISC-V compiler installation should be correct.

```
$ riscv-none-embed-gcc -v 2>&1 | tail -n1
gcc version 8.2.0 (xPack GNU RISC-V Embedded GCC, 64-bit)
```

Install: RISC-V Emulator

You will need to do the following to prepare the emulator. Different from Assignment #1, notice that we are going to use `rv32emu-assign3.tar.gz` for Assignment #3.

```
$ cd BASE
BASE $ wget http://compsec.snu.ac.kr:40404/downloads/rv32emu-assign3.tar.gz
BASE $ tar zxvf ./rv32emu-assign3.tar.gz
BASE/rv32emu-assign3 $ cd rv32emu-assign3
BASE/rv32emu-assign3 $ make
```

If you see the message like below, your `rv32emu` should be ready.

```
BASE/rv32emu-assign3 $ file emu-rv32i
emu-rv32i: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/Linux
3.2.0, BuildID[sha1]=d0ac207601f702e9ad4f556f73d1818221190c75, not stripped
```

Evaluation Target: Matrix Multiplication Application

Our target application does a fairly simple thing, matrix multiplication, which is implemented in `mat.c` that we provided. To clearly compare the cache performance, this `mat.c` can be configured with various settings:

- The size of matrix (i.e., the number of rows/columns in the square matrix) can be configured using the macro, `MAT_ORDER`
- If the matrix should be accessed row-by-row (i.e, row major if `ROW_MAJOR` is defined) or column-by-column (i.e., column major if `ROW_MAJOR` is not defined).

The `Makefile` that we provided produces executables mixing up the combination of such configurations. If you command `make mat`, it will be producing following 8 executables: `mat-row2`, `mat-col2`, `mat-row4`, `mat-col4`, `mat-row8`, `mat-col8`, `mat-row16`, and `mat-col16`.

```
BASE/rv32emu-assign3 $ make mat
riscv-none-embed-gcc -DMAT_ORDER=2 -DROW_MAJOR -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-row2 mat.c
riscv-none-embed-gcc -DMAT_ORDER=2 -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-col2 mat.c
riscv-none-embed-gcc -DMAT_ORDER=4 -DROW_MAJOR -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-row4 mat.c
riscv-none-embed-gcc -DMAT_ORDER=4 -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-col4 mat.c
riscv-none-embed-gcc -DMAT_ORDER=8 -DROW_MAJOR -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-row8 mat.c
riscv-none-embed-gcc -DMAT_ORDER=8 -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-col8 mat.c
riscv-none-embed-gcc -DMAT_ORDER=16 -DROW_MAJOR -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-row16 mat.c
riscv-none-embed-gcc -DMAT_ORDER=16 -march=rv32im -mabi=ilp32 -O0 -nostdlib -o mat-col16 mat.c
```

From the `make` command above, notice the changes in specifying `-DMAT_ORDER` and `-DROW_MAJOR` for each executable.

For instance, `mat-row4` is an executable accessing the matrix row-by-row, where the matrix order is 4. If you run `mat-row4` with `emu-rv32i` (which does not implement cache), it will print the result of matrix multiplication.

```
BASE/rv32emu-assign3 $ ./emu-rv32i ./mat-row4
0 14 28 42
0 20 40 60
0 26 52 78
0 32 64 96

>>> Execution time: 361965 ns
>>> Instruction count: 4743 (IPS=13103476)
>>> Jumps: 345 (7.27%) - 112 forwards, 233 backwards
>>> Branching T=248 (78.48%) F=68 (21.52%)
```

Task1: Implementing Direct-Mapped Cache

In Task1, you will implement a direct-mapped cache by modifying `rv32emu`. In particular, you can modify two files, `emu-rv32i.c` and `emu-dir-cache.c`, and we inlined a comment, `// TODO: Assignment #3`, where you need to fill up the code for this task. Any other files are not allowed to modify (You may modify other files during the development, but your final implementation should be working without any modification on other files).

Your direct-mapped cache will be overlaying the ram (declared in line 129 of `emu-rv32i.c`), such that all the memory access should be first checked through your cache. To do so, you will need to modify all helper functions accessing the memory---i.e., `target_read_u8`, `target_read_u16`, `target_read_u32`, `target_write_w8`, `target_write_w16`, and `target_write_w32`.

Your cache will have following three interface functions:

- `cache_init()` initializes all necessary structures for cache.
- `uint32_t cache_read(uint32_t addr)` reads the cache block corresponding to the address `addr`, and returns the cached value. You will accordingly need to handle if the corresponding cache block is missing. You will also need to update the statistics counter, `num_cache_hit` and `num_cache_miss` (declared in `emu-cache.h`).
- `void cache_write(uint32_t addr, uint32_t value)` writes `value` to the cache block corresponding to the address `addr`. You will accordingly need to handle if the cache block has to be evicted to the ram.

Your cache size is 256 bytes in total (declared as `CACHE_SIZE` in `emu-cache.h`), and your cache is an array of cache blocks (declared as `struct cache_block cache[NUM_CACHE_BLOCKS]` in `emu-dir-cache.c`. We define that each cache block has 4 bytes of data, so the total number of cache blocks (declared as `NUM_CACHE_BLOCKS` in `emu-cache.h`) is 64.

Your cache will be interpreting the address as follows:

TAG	INDEX	OFFSET

TAG	INDEX	OFFSET
TAG	log(`NUM_CACHE_BLOCKS`)	G
24 bits	6 bits	2 bits

- `G` has 2 bits since the each cache block has 4 bytes.
- `INDEX` has 6 bits. This is because this cache is direct-mapped cache, so the index is the log of total number of cache blocks (i.e., log(64)).
- `TAG` has 24 bits: 32 (the bit width of address) - 6 (the bit width of INDEX) - 2 (the bit width of G).
- Worth noting that our cache does not have `block offsets`.

Each cache block's representation is (declared in `emu-dir-cache.c`):

```
// emu-dir-cache.c
struct cache_block
{
  uint32_t tag:TAG_BIT_WIDTH;
  uint32_t valid:1;
  uint32_t data;
};
```

- Each cache block has 24 bits of `tag`, 1 bit of `valid`, 32 bits of `data`.
- Since each cache block has 4 bytes data (i.e., `uint32_t data`), `G` has 2-bits.

To summarize, your goal of Task1 is to modify two files, `emu-rv32i.c` and `emu-dir-cache.c`, so as to implement a direct-mapped cache following the instructions above. If not specifically mentioned in Task1's instruction, you can decide your own designs of direct-mapped cache.

Task2 can be built with the command `make emu-rv32i-dir-cache`, which produces an executable, `emu-rv32i-dir-cache`.

```
BASE/rv32emu-assign3 $ make emu-rv32i-dir-cache
gcc-7 -O3 -Wall -DUSE_CACHE -o emu-rv32i-dir-cache emu-rv32i.c emu-dir-cache.c -lelf
```

To run the application (e.g., `mat-row4`) with the emulator supporting this direct-mapped cache, you can execute `emu-rv32i-dir-cache` as follows.

```
BASE/rv32emu-assign3 $ ./emu-rv32i-dir-cache ./mat-row4
0 14 28 42
0 20 40 60
0 26 52 78
0 32 64 96

>>> Execution time: 350897 ns
>>> Instruction count: 4743 (IPS=13516786)
>>> Jumps: 345 (7.27%) - 112 forwards, 233 backwards
>>> Branching T=248 (78.48%) F=68 (21.52%)
>>> cache hit ratio: 0.xxxx (hit:yyyy, miss:zzzz)
```

- The output of `emu-rv32i-dir-cache` should be the same as the output of `emu-rv32i` (except the lines starting with `>>>`), because that shows your cache overlay (with the direct-mapped cache) preserves the execution correctness.
- The last line of output should show the cache hit ratio (we intentionally blinded numbers). Check how this is printed in `main()` of `emu-rv32i.c`. This also implies that you need to accordingly update `num_cache_hit` and `num_cache_miss` depending on the cache hit/miss events.

Task2: Implementing 4-Way Set Associative Cache

Task2's requirements are mostly the same as Task1, but with following key differences:

- You can only modify `emu-rv32i.c` and `emu-set-cache.c` (not `emu-dir-cache.c`)
- You will implement 4-way set associative cache (not direct-mapped cache).

Since Task2 is 4-way set associative cache, its address interpretation is also different: 2-bits more TAG bits and 2-bits less INDEX bits than that of direct-mapped cache.

TAG	INDEX	OFFSET
26 bits	4 bits	2 bits

- `G` has 2 bits since the each cache block has 4 bytes
- `INDEX` has 4 bits. Because this cache is 4-way set associative cache, each index can be mapped to four different cache blocks. Thus, the bit width of `INDEX` is 4 (i.e., log(`NUM_CACHE_BLOCKS`)) - 2 (i.e., log(`NUM_WAY`)).
- `TAG` has 26 bits: 32 (the bit width of address) - 4 (the bit width of `INDEX`) - 2 (the bit width of `G`).

Since Task2 is a set associative cache, we need to design a cache replacement policy. To this end, we implement an **pseudo** version of LRU-based cache replacement policy (i.e., not precisely following the LRU policy, but trying to be close to the LRU policy). To do so, we maintain 2-bits of `counter` in each `cache_block` , which keeps track of the **approximated** time of the last access. So this `counter` can be used to **approximate* which cache block is the least recently used one . We do not restrict any rules on how this counter should be designed, so feel free to come up with your own design.

```
// emu-set-cache.c
struct cache_block
{
  uint32_t tag:TAG_BIT_WIDTH;
  uint32_t valid:1;
  uint32_t data;
  uint32_t counter:2;
};
```

Build and running processes are similar to Task1, but the executable name is `emu-rv32i-set-cache` for Task2.

```
BASE/rv32emu-assign3 $ make emu-rv32i-set-cache
gcc-7 -O3 -Wall -DUSE_CACHE -o emu-rv32i-set-cache emu-rv32i.c emu-set-cache.c -lelf
```

```
BASE/rv32emu-assign3 $ ./emu-rv32i-set-cache ./mat-row4
0 14 28 42
0 20 40 60
0 26 52 78
0 32 64 96

>>> Execution time: 372023 ns
>>> Instruction count: 4743 (IPS=12749211)
>>> Jumps: 345 (7.27%) - 112 forwards, 233 backwards
>>> Branching T=248 (78.48%) F=68 (21.52%)
>>> cache hit ratio: 0.xxxx (hit:yyyy, miss:zzzz)
```

- The output of `emu-rv32i-set-cache` should be the same as the output of `emu-rv32i` (except the lines starting with `>>>`).
- The last line of output should show the cache hit ratio (we intentionally blinded the numbers).

Task3: Comparison Study on Cache Performance

Task3 asks you to write `report.pdf` after evaluating your cache implementations with applications computing the matrix multiplication.

Specifically, your `report.pdf` should answer following questions (No need to be a long answer. Please try to be as short as possible).

- Q1. What's the impact of row-by-row or column-by-column accesses? Given the cache hit ratio you observed, can you interpret those numbers? You may mention 1) the characteristics of matrix multiplication algorithms, 2) how each cache policy in Task1 and Task2 impacts the cache hit ratio, etc.
- Q2. What's the impact of matrix order (i.e., `MAT_ORDER`) to the cache hit ratio of direct-mapped cache and 4-way set-associative cache, respectively? Do you have any interesting findings that you want to share? You may try to relate your answer with the concept of `working set` .
- Q3. Any thoughts on how to further improve the cache performance? How should you change the cache design?

More Notes on Task1 and Task2

- If not specifically mentioned in this instruction, you may make your own design decisions. For instance, you may go for either inclusive cache or non-inclusive cache, or go for either write-back or write-through cache. This decision is completely up to you.
- Our cache designs in Task1 and Task2 **do not have block offset**.
- Do not print any from your code. All the printing should only be done by the provided code. If you print something from your code, it would interfere the automated grading process.
- You only implement a single version of `emu-rv32i.c`, which works for both Task1 and Task2.
- Double-check if your cache implementation preserves the execution correctness. In other words, the result of matrix multiplication should be correct! To ease your checking process, we provided the script that can be invoked using `make check` command. It runs all the combination of execution to be checked, and compare if the execution results are correct. If everything is done right, then you should see the "PASS" message. You will get the "FAIL" message otherwise.

```
BASE/rv32emu-assign3 $ make check
[+] Compare: [./native-mat-col2] vs. [./emu-rv32i-dir-cache ./mat-col2]
[+] Compare: [./native-mat-col2] vs. [./emu-rv32i-set-cache ./mat-col2]
[+] Compare: [./native-mat-row2] vs. [./emu-rv32i-dir-cache ./mat-row2]
[+] Compare: [./native-mat-row2] vs. [./emu-rv32i-set-cache ./mat-row2]
[+] Compare: [./native-mat-col4] vs. [./emu-rv32i-dir-cache ./mat-col4]
[+] Compare: [./native-mat-col4] vs. [./emu-rv32i-set-cache ./mat-col4]
[+] Compare: [./native-mat-row4] vs. [./emu-rv32i-dir-cache ./mat-row4]
[+] Compare: [./native-mat-row4] vs. [./emu-rv32i-set-cache ./mat-row4]
[+] Compare: [./native-mat-col8] vs. [./emu-rv32i-dir-cache ./mat-col8]
[+] Compare: [./native-mat-col8] vs. [./emu-rv32i-set-cache ./mat-col8]
[+] Compare: [./native-mat-row8] vs. [./emu-rv32i-dir-cache ./mat-row8]
[+] Compare: [./native-mat-row8] vs. [./emu-rv32i-set-cache ./mat-row8]
[+] Compare: [./native-mat-col16] vs. [./emu-rv32i-dir-cache ./mat-col16]
[+] Compare: [./native-mat-col16] vs. [./emu-rv32i-set-cache ./mat-col16]
[+] Compare: [./native-mat-row16] vs. [./emu-rv32i-dir-cache ./mat-row16]
[+] Compare: [./native-mat-row16] vs. [./emu-rv32i-set-cache ./mat-row16]
[+] PASS: All execution results seem to be correct!
```

Submission

Prepare your submission file, `assign3.zip`, which **only** zips following four files:

1. `emu-rv32i.c` (for Task1 and Task2)
2. `emu-dir-cache.c` (for Task1)
3. `emu-set-cache.c` (for Task2)
4. `report.pdf` (for Task3)

Please note that you submit a single version of `emu-rv32i.c`, which works for both Task1 and Task2. Once your `assign3.zip` is ready, submit it through our submission website.