

Computer Organization 2025
Assignment III

Due Date: 23:59, June 11, 2025

國立成功大學
National Cheng Kung University

Overview

- The assignment includes two parts:
  - 1. Programming Part
  - 2. Demo Part

- The goal is to enhance program execution performance through effective cache utilization.

- You will do the following steps to finish this assignment:
  - Implement a FIFO cache replacement policy on Spike
  - Reduce memory access overhead of the given two programs
  - Explain to TAs your works and design philosophy (including Assignment I & Assignment II)

Cache Optimization Fundamentals

- Common Replacement Policies (Hardware Perspective)
  - Replacement policies greatly affect cache hit/miss rate
  - The assignment requires implementing the **FIFO** replacement policy

- Algorithmic Level Method to Reduce Cache Miss (Software Perspective)
  - The main goal is to improve spatial or temporal locality
  - Common strategies:
    - **Loop Interchange:** Changes the order of accessing data, primarily to improve spatial locality on cache access
    - **Loop Fusion:** Combines operations on the same array across different loops to increase temporal locality
    - **Loop Tiling:** Breaks a large loop into smaller blocks (tiles) and organizes computations within each tile to enhance data reuse in the cache
  - This assignment requires reducing memory access overhead of the given two programs

Cache Simulation with Spike

- Cache simulators are represented by `cache_sim_t` class objects in spike
- The `cache_sim_t` class defines the behavior of the simulated caches. Its prototypes are in `riscv/cachesim.h`, and implementation is in `riscv/cachesim.cc`
- Important member functions include `access`, `check_tag`, and `victimize`

Performance Evaluation Methodology

- $Improvement\ Ratio = \frac{MemCycle_{original}}{MemCycle_{Improved}}$
- $MemCycle = cache_{hit} \times latency_{hit} + cache_{misses} \times penalty_{miss}$
- In this assignment, $latency_{hit}$ = 1 cycle; $penalty_{miss}$ = 100 cycles

```
Original version                          | Improved version
D$ Bytes Read: 24874040                   | D$ Bytes Read: 30530544
D$ Bytes Written: 4006529                 | D$ Bytes Written: 4504069
D$ Read Accesses: 5245118                 | D$ Read Accesses: 6595544
D$ Write Accesses: 871214                 | D$ Write Accesses: 995455
D$ Read Misses: 433720                    | D$ Read Misses: 48792
D$ Write Misses: 32478                    | D$ Write Misses: 27567
D$ Writebacks: 47520                      | D$ Writebacks: 40871
D$ Miss Rate: 7.622%                      | D$ Miss Rate: 1.006%
                                          |
Memory subsystem access overhead = 52269934 | Memory subsystem access overhead = 15150540

Improved ratio:  3.45003769
```

Example Performance Data and Calculation

Assignment

- Part 1: Programming Part (30%)
  - ➢Exercise 1 (10%): Implement **FIFO** data cache replacement policy
  - ➢Exercise 2 (20%): Enhancement of software programs to reduce memory access overhead
    - ❖Exercise 2-1 (10%): Matrix Transpose
    - ❖Exercise 2-2 (10%): 2D Matrix Multiplication
- Part 2: Demo Part (80%)
  - ➢Explain how your FIFO cache replacement policy works in Spike
  - ➢Present the design philosophy of your optimized programs, explaining the techniques and concepts applied **across all assignments**

Part 1: Exercise 1 (Implement FIFO Data Cache Replacement Policy, 10%)

- Implement FIFO data cache replacement policy to Spike
- Two files to modify in Spike source code
  - *{PATH_TO_SPIKE}*/riscv/cachesim.h
  - *{PATH_TO_SPIKE}*/riscv/cachesim.cc
- Recompilation is needed after implement FIFO replacement policy
  - Please refer to HW0 on NCKU Moodle
- You can add new data structures in your implementation
- After your implementation, copy both files to the `exercise1/` directory
- If you **cannot** provide a valid implementation for the FIFO data cache replacement policy, you can upload the random policy offered by the Spike simulator.

Preliminary

- Cache specifications
  - 4 ways
  - 8 sets
  - 32 bytes cacheline
  - Miss penalty = 100 cycles
  - Hit latency = 1 cycle
- To enable data cache in Spike simulator
  - Add `–dc=<set>:<ways>:<cacheline>`

```
$ spike --isa=RV64GC --dc=8:4:32 $RISCV/riscv64-unknown-elf/bin/pk <your_program>
```

Part 1: Exercise 2-1 (Matrix Transpose, 10%)

- Input: Matrix size: $m \times n$, where m, n set to 1000 (1000×1000)
  - ➢Data Type: integer data and the values of the matrix elements are ranging from 0 to 1023
- Output: The performance data for both of the original and improved programs.
- Optimize the matrix transpose implementation to reduce memory access cycles.

```
=============== Exercise 2-1 ===============
Original version
D$ Bytes Read:            102247719
D$ Bytes Written:         49349943
D$ Read Accesses:         21301733
D$ Write Accesses:        8675759
D$ Read Misses:           1137185
D$ Write Misses:          702760
D$ Writebacks:            756764
D$ Miss Rate:             6.138%

Memory subsystem access overhead =  212132047 (
-----------------------------------
Improved version
D$ Bytes Read:            121078975
D$ Bytes Written:         50458427
D$ Read Accesses:         26012881
D$ Write Accesses:        8953944
D$ Read Misses:           253802
D$ Write Misses:          684730
D$ Writebacks:            712532
D$ Miss Rate:             2.684%

Memory subsystem access overhead =  127881493 (
-----------------------------------
Improved ratio:  1.658817409959391
Output Correctness:  Pass
-----------------------------------
```

Part 1: Exercise 2-2 (Matrix Multiply, 10%)

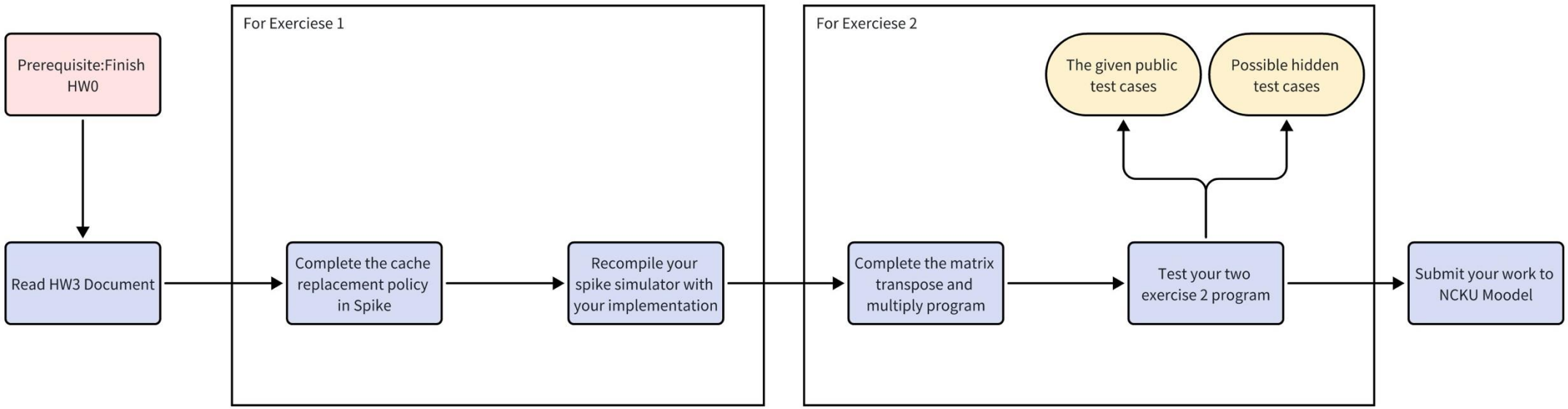
- Input: Two matrix sizes: $m \times n$, where m , n set to 100 (100×100)
  - ➢Data Type: integer data and the values of the matrix elements are ranging from 0 to 1023
- Output: The performance data for both of the original and improved programs.
- Optimize the matrix multiplication implementation to reduce memory access cycles

```
=============== Exercise 2-2 ===============
Original version
D$ Bytes Read:            66304079
D$ Bytes Written:         9311467
D$ Read Accesses:         14416460
D$ Write Accesses:        2216132
D$ Read Misses:           1220580
D$ Write Misses:          28107
D$ Writebacks:            66502
D$ Miss Rate:             7.507%

Memory subsystem access overhead =  140252605
-----------------------------------
Improved version
D$ Bytes Read:            104168227
D$ Bytes Written:         12577667
D$ Read Accesses:         23369163
D$ Write Accesses:        3031046
D$ Read Misses:           111400
D$ Write Misses:          26246
D$ Writebacks:            67375
D$ Miss Rate:             0.521%

Memory subsystem access overhead =  40027163 (
-----------------------------------
Improved ratio:  3.5039356898714007
Output Correctness:  Pass
-----------------------------------
```

Suggested Workflow



Test Your Assignment

- Use `make check` under directory ` CO_StudentID_HW3/exercise2/` for checking the algorithm correctness and the memory access overhead
- If the result is incorrect, you will not get the scores.

```
============== Exercise 2-1 ==============
Original version
D$ Bytes Read:            102247719
D$ Bytes Written:         49349943
D$ Read Accesses:         21301733
D$ Write Accesses:        8675759
D$ Read Misses:           1137185
D$ Write Misses:          702760
D$ Writebacks:            756764
D$ Miss Rate:             6.138%

Memory subsystem access overhead =  212132047 (cpu cycle)
-----------------------------------
Improved version
D$ Bytes Read:            121078975
D$ Bytes Written:         50458427
D$ Read Accesses:         26012881
D$ Write Accesses:        8953944
D$ Read Misses:           253802
D$ Write Misses:          684730
D$ Writebacks:            712532
D$ Miss Rate:             2.684%

Memory subsystem access overhead =  127881493 (cpu cycle)
-----------------------------------
Improved ratio:  1.658817409959391
Output Correctness:  Pass
-----------------------------------
============== Exercise 2-2 ==============
Original version
D$ Bytes Read:            66304079
D$ Bytes Written:         9311467
D$ Read Accesses:         14416460
D$ Write Accesses:        2216132
D$ Read Misses:           1220580
D$ Write Misses:          28107
D$ Writebacks:            66502
D$ Miss Rate:             7.507%

Memory subsystem access overhead =  140252605 (cpu cycle)
-----------------------------------
Improved version
D$ Bytes Read:            104168227
D$ Bytes Written:         12577667
D$ Read Accesses:         23369163
D$ Write Accesses:        3031046
D$ Read Misses:           111400
D$ Write Misses:          26246
D$ Writebacks:            67375
D$ Miss Rate:             0.521%

Memory subsystem access overhead =  40027163 (cpu cycle)
-----------------------------------
Improved ratio:  3.5039356898714007
Output Correctness:  Pass
-----------------------------------
```

Output (if pass)

```
============== Exercise 2-1 ==============
Original version
D$ Bytes Read:            102247719
D$ Bytes Written:         49349943
D$ Read Accesses:         21301733
D$ Write Accesses:        8675759
D$ Read Misses:           1137185
D$ Write Misses:          702760
D$ Writebacks:            756764
D$ Miss Rate:             6.138%

Memory subsystem access overhead =  212132047 (cpu cycle)
-----------------------------------
Improved version
D$ Bytes Read:            46214339
D$ Bytes Written:         41337375
D$ Read Accesses:         9295055
D$ Write Accesses:        6673296
D$ Read Misses:           64515
D$ Write Misses:          546632
D$ Writebacks:            564360
D$ Miss Rate:             3.827%

Memory subsystem access overhead =  76471904 (cpu cycle)
-----------------------------------
Improved ratio:  2.773986731126768
Output Correctness:  Fail
-----------------------------------
============== Exercise 2-2 ==============
Original version
D$ Bytes Read:            66304079
D$ Bytes Written:         9311467
D$ Read Accesses:         14416460
D$ Write Accesses:        2216132
D$ Read Misses:           1220580
D$ Write Misses:          28107
D$ Writebacks:            66502
D$ Miss Rate:             7.507%

Memory subsystem access overhead =  140252605 (cpu cycle)
-----------------------------------
Improved version
D$ Bytes Read:            1928931
D$ Bytes Written:         1174355
D$ Read Accesses:         324338
D$ Write Accesses:        179113
D$ Read Misses:           5323
D$ Write Misses:          18915
D$ Writebacks:            20841
D$ Miss Rate:             4.814%

Memory subsystem access overhead =  2903013 (cpu cycle)
-----------------------------------
Improved ratio:  48.312771937294116
Output Correctness:  Fail
-----------------------------------
```

Output (if fail)

Test Your Assignment

If your program didn't pass, run the following command:

- Use `make diffex2-1` to check the ouput:
  - This will compare your matrix transpose output with correct answer

- Use `make diffex2-2` to check the output:
  - This will compare your matrix multiplication output with correct answer

```
$ make diffex2-1
diff --git a/exercise2_1/ans.output b/exercise2_1/stu.output
index 03a72d0..cd90b93 100644
--- a/exercise2_1/ans.output
+++ b/exercise2_1/stu.output
@@ -1 +1 @@
0 989 954 919 884 849 815 780 755745 710 675 641 606 571 536 501 467 432 397 362
327 293 258 223 188 153 119 84 49 14 1003 969 934 899 864 829 795 760 725 690
655 621 586 551 516 481 446 412 377 342 307 272 238 203 168 133 98 64 29 1018
983 948 914 879 844 809 774 740 705 670 635 600 566 531 496 461 426 392 357 322
287 252 218 183 148 113 78 44 9 998 963 928 893 859 824 789 754 719 685 650 615
580 545 511 476 441 406 371 337 302 267 232 197 163 128 93 58 23 1013 978 943
908 873 839 804 769 734 699 665 630 595 560 525 491 456 421 386 351 316 282 247
212 177 142 108 73 38 3 992 958 923 888 853 818 784 749 714 679 644 610 575 540
505 470 436 401 366 331 296 262 227 192 157 122 88 53 18 1007 972 938 903 868
833 798 763 729 694 659 624 589 555 520 485 450 415 381 346 311 276 241 207 172
137 102 67 33 1022 987 952 917 883 848 813 778 743 709 674 639 604 569 535 500
465 430 395 361 326 291 256 221 186 152 117 82 47 12 1002 967 932 897 862 828
793 758 723 688 654 619 584 549 514 480 445 410 375 340 306 271 236 201 166 132
97 62 27 1016 982 947 912 877 842 808 773 738 703 668 633 599 564 529 494 459
425 390 355 320 285 251 216 181 146 111 77 42 7 996 961 927 892 857 822 787 753
718 683 648 613 579 544 509 474 439 405 370 335 300 265 231 196 161 126 91 56 22
1011 976 941 906 872 837 802 7
```

Example output of `make diffex2-1`

Part 2: Demo

- Demo session on
  - June 19-20, 2025 (all days)
- Demo session at
  - Room 65704 (Advanced Systems Research Lab)
- We will download the code you submited. You don't need to prepare PC in demo session
- Please register your preferred time on the Google Sheet on NCKU Moodle

Submission of Your Assignment

- Compress your source code within the folder into a zip file.
- Submit your assignment to NCKU Moodle.
- NOTE: **Change the zip and directory name to your student ID number, e.g., F12345678_HW3.zip**.
- The zip file should contain these files shown in the figure

```
F12345678_HW3.zip
└── F12345678_HW3
    ├── README.md
    ├── exercise1
    │   ├── cachesim.cc
    │   └── cachesim.h
    └── exercise2
        ├── exercise2_1
        │   ├── matrix_transpose.c
        │   ├── matrix_transpose_improved.c
        │   └── testbench_driver.c
        ├── exercise2_2
        │   ├── matrix_multiply.c
        │   ├── matrix_multiply_improved.c
        │   └── testbench_driver.c
        ├── makefile
        ├── test_exercise2_1.py
        └── test_exercise2_2.py
```

Scoring Criteria

Part 1: (Programming) 30%

1. Implement FIFO data cache replacement policy on Spike (10%)
2. Reduce the memory access overhead of matrix transpose (10%)
   - Improved > 1.6 (10/10)
   - Improved > 1 (5/10)
3. Reduce the memory access overhead of matrix multiplication (10%)
   - Improved > 3 (10/10)
   - Improved > 1.5 (5/10)

- If you cannot implement FIFO replacement policy, you can still work on other questions
- The public test cases account for 60% and the hidden test cases account for 40%.

Scoring Criteria

Part 2: (Demo) 80%

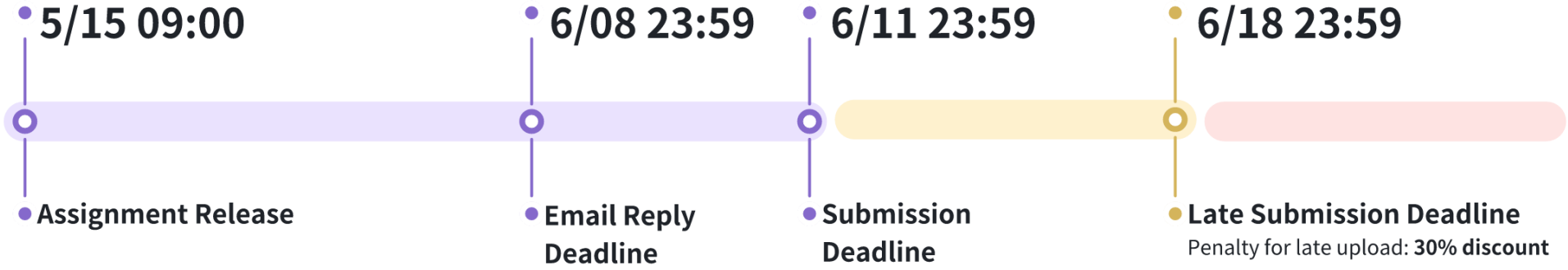
1. Describe the workflow and mechanism in Spike, related to cache simulation. (20%)
2. Describe the concept behind your modified matrix transpose algorithm. (20%)
3. Describe the concept behind your modified matrix multiplication algorithm. (20%)
4. Describe the concept behind your design philosophy of previous **Assignment I** and **Assignment II**. (20%)

Scoring Criteria – About README.md

- README.md is **mandatory**.
- You can include draft answers to the four questions of the demo part in your README.md
- During the demo session, you are allowed to reference your README file to help you answer TA's questions.
- Please maintain the filename as 'README.md' only. Do not modify it by adding any additional information such as your student ID.
- Although it does not contribute to your score, failing to submit **README.md** including wrong filename will result in a score of **zero** for this assignment.

Important Note

- Penalty for late upload
  - ➢30% discount
  - ➢Within seven days of the given deadline.
- **Incorrect format** and **Inconsistent of the folder structure** will lose **10** points.
- A high similarity score between your code and someone else's will result in a score of **zero** for this assignment.





Thank you for listening.
The quiz will start at 9:40 AM.