HW07: Cache Performance
CSE 30321 Computer Architecture
Name(s): Aaron Wang and Ethan Little

Preamble:

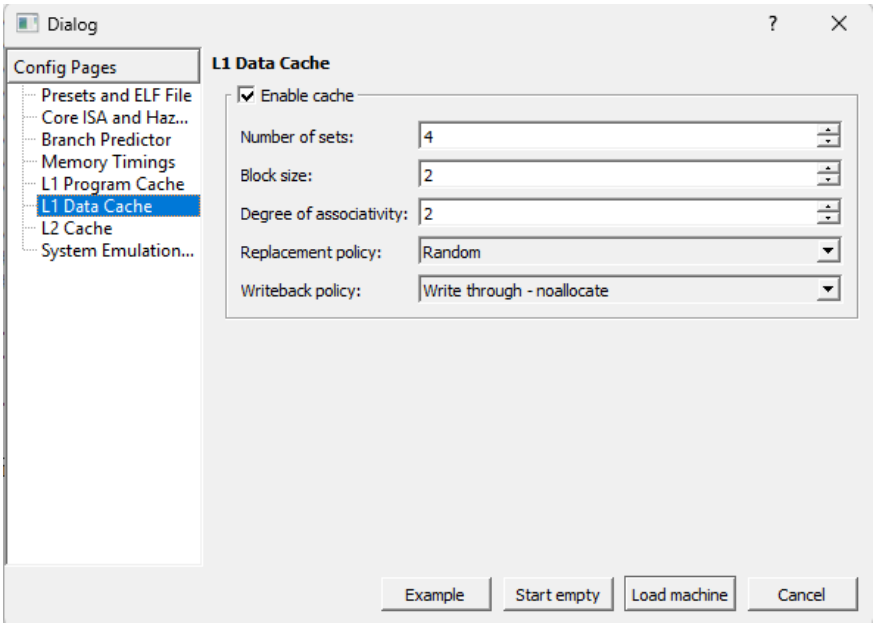
1. ***Copy this assignment from our Google Drive directory, not from Gradescope. Downloading the PDF from Gradescope may not preserve all answer boxes.***

2. ***Reminder:*** point totals differ across assignments mainly as a way to allow for partial credit, *but all assignments carry the same weight* (see the syllabus for details).

3. Enter your answers in the boxes provided. Each empty box should be filled in with an answer. You can either type your answers directly or insert images as long as they are legible. For solutions that require code, use a **fixed-width font (Consolas preferred) no smaller than 10 points with proper indentation**. Save your solutions as a single PDF file and upload them to Gradescope.

4. You are encouraged to work in groups of 2. Please type your names at the top of this document.

In this assignment, you will explore the performance of the **data** cache in QtRVSim for two short programs. For each question, you will start from the default configuration and attempt to change parameters to minimize the cache miss rate. To use the default configuration, start a *New simulation* and select "Pipelined with hazard unit and cache." Selecting this radio button will reset the L1 data cache to the default cache parameters in v0.9.8, as shown below:



You should also **disable the L1 Program Cache** (this should help speed up the simulation). The L2 cache will be disabled automatically with this configuration - **do not enable it (yet).**

Here are the changes you are allowed to make, and to what degree, for each problem:

Design Change	Problem 1 (hw06_arr.S)	Problem 2 (hw07_class.S)		
		Q1	Q2 (same as Q1)	Q3
Number of sets	Max 16	Default	Default	Any
Block size	Max 16	Default	Default	Any
Associativity	Max 16	Default	Default	Any
Replacement policy	Any	Default	Default	Any
Writeback policy	Any	Write through-write allocate	Write through-write allocate	Any
L2 Number of sets	No L2	No L2	No L2	Any
L2 Block size				Any
L2 Associativity				Any
L2 Replacement policy				Any
L2 Writeback policy				Any

Changes you may **not** make, for any problem, unless specified otherwise in a question:

- Any changes to the code,
- Memory access time (leave default),
- Program cache (**disable it**),
- Any changes to the core configuration.

Important: when making changes, they may not "take" unless you first make the change, and then click outside the text field you just changed. (This *seems* to be fixed in v0.9.8, but FYI just in case.)

Problem 1

Note: there is a minimal number of misses that can be achieved given the parameters you are allowed to change. Think carefully about the access pattern and what accesses *must* be misses. A well-reasoned, non-minimal answer can still receive full credit.

i. Come up with a plan to minimize the data cache miss rate for the hw06_arr.S program. You may change any of the design parameters listed in the table, within the ranges listed. Note you may **not** enable the L2 cache for this problem. Explain the change(s)

you propose to make, and why you believe that will minimize the miss rate. Be specific in your explanation, describing the expected effects the changes will have on the loads and stores *in this program*.

We propose that we up the block size as much as possible, so that spatial locality gives hits more often. Since we are accessing adjacent elements in an array, increasing the block size from 2 to 16 allows hits to occur 15 out of 16 times instead of 1 out of 2.

Since our program does not use that much data, we can make the cache big enough to hold all the data after filling the array, meaning capacity misses aren't an issue. Then we should maximize the number of sets and associativity to ensure all reads are hits after the initial filling of the array.

Also changing the writeback policy to writeback ensures that all of the misses when filling are written to the cache, eliminating any misses during the array summing.

Finally using LRU as the replacement policy, we are able to achieve the minimum data cache miss rate as it keeps the blocks we need in data.

ii. Run the hw06_arr.S program from start to finish, recording the same statistics as from HW06, and calculate AMAT and AMAT speedup.

Hits: 1984
Misses: 65
Hit rate: 96.828%
AMAT: (1 clock cycle * 1 ns per cycle) + (1-96.828%) *(10 clock cycles * 1 ns per cycle) = 1.317 ns
AMAT speedup:
Original AMAT: (1 clock cycle * 1 ns per cycle) + (1-25.037%) *(10 clock cycles * 1 ns per cycle) = 8.496 ns
8.49/1.317 = 6.45x speedup

iii. Explain why *each* load and store in the program was a hit or a miss. (See the solution to HW06 for an example write-up that addresses each load and store.)

The loading of the array size on line 22 is a miss because it is the first access to that address. Then the first store on line 32 for each block when filling the array is a miss since it is the first access, while the other 15 out of 16 stores in each block are hits due to spatial locality from the first store. Since our cache is large enough to hold all the data from the fill loop, every load in the sum loop on line 45 is a hit due to temporal locality (if it was initially the first access out of the 16 writes) or spatial locality (if it was initially one of the other 15 writes).

This gives us 1 + 64 = 65 misses and 15 * 64 + 1024 = 1984 hits.

iv. Do you believe your plan from ***1.i*** resulted in the minimal number of misses for this cache? Justify your answer. If you believe you could have done better, explain how and why, and re-run the program once with the updated design, recording just the hit rate.

Yes I believe we have achieved the best design given the limitations. The only misses are the first of every block, so the only way to better this is to increase the block size.

Problem 2

In this problem, the program has a less "ideal" access pattern. **Read through the code and ensure you understand what it does**, including the comments at the top of the program – this is important so you don't change the meaning of the program (or data) for P2Q2. Note how the array size is no longer adjacent to the array in memory.

Question 1

Run the hw07_class.S program from start to finish with the cache setup described in the table, recording the same statistics as from HW06, and calculate AMAT. Calculate AMAT using the same memory assumptions as from HW06. Explain why *each* load and store in the program was a hit or a miss. (See the solution to HW06 for an example write-up that addresses each load and store.)

Hits: 0 Misses: 2049 Hit rate: 0% AMAT: (1 clock cycle * 1 ns per cycle) +(1-0%)*(10 clock cycles * 1 ns per cycle) = 11 ns Explanation for all hits/misses: The first miss occurs when we load in the word from arr_max. The next 1024 misses occur from: sw x8, 0(x9) Since each word is separated by 32 words in between, they will never be in the same block, so it won't be cached. It then returns and does the same thing for every adjacent word. The previously used blocks will be taken out of the cache by the time the program tries to access it and will have to go back to memory to find it. This thus results in 2049 misses and 0 hits.

Question 2

Explain how you can improve the cache performance by rewriting (reorganizing) the assembly while using the cache setup from P2Q1. Your explanation should include why the *code organization* from P2Q1 led to the (terrible) hit rate in P2Q1 and address how reorganized code could alleviate this, and why. Rewrite (reorganize) the code and again use simulation data to justify your answer, recording hits, misses, and hit rate (no need

to calculate AMAT). Note: you may not change the array organization (array of student records), only how the *code* is organized. You are allowed to change register usage, as long as the code behaves the same way and the memory/data organization is unchanged.

Explanation: To reorganize the assembly code, we propose writing to adjacent blocks at the same time, rather than in two separate loops. AKA rather than writing a to 400 at the beginning and writing 0 to 404 halfway through, we write to both of them in sequence, so the cache can determine the spatial locality of that block.

Rewritten code that will improve cache performance:

```
.globl _start
.option norelax
.text
_start:
        la x5, arr              //starting address of array
        la x6, arr_max
        lw x6, 0(x6)
        li x7, 0        //loop index, i = 0
        li x8, 10               //we will first fill a large array with student ID numbers,
starting at 10
        li x11, 16              // to find remainder
        li x3, 0        // counter to remainder from
fill_loop:
        bge x7, x6, done_fill
        slli x9, x7, 2    //i * 4
        add x9, x9, x5 //&(arr + i * 4)
        sw x8, 0(x9)    //arr[i] = student id number
        addi x9, x9, 4  //next record is section number
        rem x10, x3, x11 // calculate section number
        sw x10, 0(x9)  //arr[i + 1] = section number
        addi x7, x7, 32 // increment
        addi x3, x3, 1  // increment section counter
        addi x8, x8, 1  // increment student id
        j fill_loop
done_fill:
        ebreak
.data
.org 0x300
arr_max:
        .word 32768

.org 0x400
arr:
        .word  1
```

Results from simulation:
Hits: 1024
Misses: 1025
Hit rate: 49.976%

Question 3

Explain how to improve the cache performance by using the original assembly, within the unbounded cache design space. Your explanation should include why the cache design from P2Q1 led to the (terrible) hit rate in P2Q1 and address what design parameter(s) could alleviate this, and why. Then, change the cache design to demonstrate an improved miss rate using a combination of at least 2 cache design parameters. You do not need to attempt to minimize miss rates for this problem, but your changes should demonstrate what will lead *toward* optimal miss rates. Again, use simulation data to justify your answer, recording hits, misses, and hit rate (no need to calculate AMAT).

Note: QtRVSim gets quite slow for very large cache sizes.

Explanation:

The cache needs to be able to hold all the data at once for minimized miss rate.
To this end, we need to store 1024x32 words in the cache. There are 1024 chunks of information that are offset by 32 words each.

Maximize the block amount while keeping it a power of 2: 64, so that every set holds a whole number of array data indexes, for efficient spatial locality caching.

Set and associativity size can be chosen as any power of 2, such that when Blocks x Sets x Associativity = 1024x32 words or the cache can hold that many words.

Thus we chose:
Block Size: 64
Sets: 32
Associativity: 32

We also used write back, so that it would all be stored in the cache and LRU to keep the most recently used (just to make sure that the first lw for the address did not have priority).

Simulation parameters that will justify your explanation:
Num sets: 32
Block Size: 64
Associativity: 32
replace policy: LRU
write policy: write back
L2: Did not usResults from simulation:

Hits: 1536
Misses: 513
Hit rate: 74.963%

What to Turn In

Fill in the boxes on the previous pages with answers to all the questions. Save your Google Doc as a PDF and upload it to **Gradescope** for grading. Note Gradescope can be accessed via our Canvas site, or by visiting gradescope.com. Below is a checklist for this assignment:

Problem 1 (30 points)

	Deliverable	Points
1.	Complete i-iv	30

Problem 2 (40 points)

	Deliverable	Points
1.	Question 1	10
2.	Question 2	15
3.	Question 3	15