**LAB REPORT**

**IT3280E– 152049– Assembly Language and Computer Architecture Lab**

**Lab 12: Cache memory**

**Assignment 1:**

1. Open the program row-major.asm. This program will traverse a 16 by 16 element integer matrix in row-major order, assigning elements the values 0 through 255 in order. It performs the following algorithm:

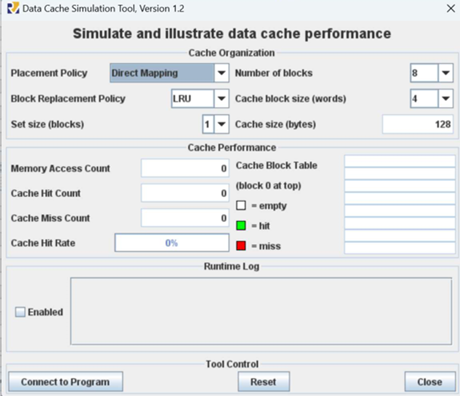
*for (row = 0; row < 16; row++)*

*for (col = 0; col < 16; col++)*

*data[row][col] = value++;*

1. Compile the program.
2. From the **Tools** menu, select **Data Cache Simulator**. A new frame will appear in

the middle of the screen.



This is a RARS Tool that will simulate the use and performance of cache memory when the underlying RISC-V program executes. Notice its three major sections:

* Cache Organization: You can use the combo boxes to specify how the cache

will be configured for this run. Feel free to explore the different settings, but

the default is fine for now.

* Cache Performance: With each memory access during program execution,

the simulator will determine whether that access can be satisfied from

cache and update the performance display accordingly.

* Tool Control: These buttons perform generic control functions as described

by their labels

1. Click the tool's **Connect to Program** button. This causes the tool to register as an observer of RISC-V memory and thus respond during program execution.
2. Back in RARS, adjust the **Run Speed slider** to 30 instructions per second. It is located at the right side of the toolbar. This slows execution so you can watch the Cache Performance animation.
3. In RARS, run the program using the **Run** toolbar button , the menu item or keyboard shortcut. Watch the Cache Performance animate as it is updated with every access to RISC-V memory.
4. What was the final cache hit rate? ***75%***

With each miss, a block of 4 words are written into the cache. In a row-major traversal, matrix elements are accessed in the same order they are stored in memory. Thus, each cache miss is followed by 3 hits as the next 3 elements are found in the same cache block. This is followed by another miss when Direct Mapping maps to the next cache block, and the patterns repeats itself. So, 3 of every 4 memory accesses will be resolved in cache.

1. Given that explanation, *what do you predict the hit rate will be if the block size is increased from 4 words to 8 words?* ***87.5% ~ 88%***. Decreased from 4 words to 2 words? ***50%***

**Explain**:

* With block size = 8 words:

+ When the first access occurs (MISS), the cache loads 8 words into a block

+ This means we now have 8 consecutive elements in the cache

* Pattern of access:

+ 1st access: MISS (loads 8 elements)

+ 2nd to 8th access: HIT (because all these elements are already in the cache)

+ So in 8 accesses, we get: 1 MISS, 7 HITs

* Calculating hit rate:

+ Number of hits / Total number of accesses

+ 7 hits / 8 accesses

+ 7/8 = 0.875 = 87.5% ≈ 88%

* Similarly for block size =4

1. Verify your predictions by modifying the block size and re-running the program from step 6.

*NOTE: when you modify the Cache Organization, the performance values are*

*automatically reset (you can always use the tool's* ***Reset*** *button).*

*NOTE: You must* ***reset*** *the RISC-V program before you can re-run it.*

*NOTE: Feel free to adjust the* ***Run Speed slider*** *to maximum speed anytime you*

*want.*

1. Repeat steps 2 through 10 for program **column-major.asm**. This program will traverse a 16 by 16 element integer matrix in column-major order, assigning elements the values 0 through 255 in order. It performs the following algorithm:

*for (col = 0; col < 16; col++)*

*for (row = 0; row < 16; row++)*

*data[row][col] = value++;*

NOTE: You can leave the Cache Simulator in place, move it out of the way, or close it. It will not interfere with the actions needed to open, assemble, or run this new program and will remain connected to RISC-V memory. If you do not close the tool, then skip steps 3 and 4.

1. What was the cache performance for this program? ***0%.***

The problem is the memory locations are now accessed not sequentially as before, but each access 2 is 16 words beyond the previous one (circularly). With the settings we've used, no two consecutive memory accesses occur in the same block, so every access is a miss.

1. Change the block size to 16. Note this will reset the tool.
2. Create a second instance of the Cache Simulator by once again selecting **Data Cache Simulator** from the **Tools** menu. Adjust the two frames so you can view both at the same time. Connect the new tool instance to RISC-V, change its block size to 16 and change its number of blocks to 16.
3. Re-run the program. What is the cache performance of the original tool instance? ***0%.*** Block size 16 didn't help because there was still only one access to each block, the initial miss, before that block was replaced with a new one. What is the cache performance of the second tool instance? ***93.7%.*** At this point, the entire matrix will fit into cache and so once a block is read in it is never replaced. Only the first access to a block results in a miss.

**Assignment 2:**

1. Open the program **row-major.asm** from the **Lab12** folder.

2. Compile the program.

3. From the **Tools** menu, select **Memory Reference Visualization**. A new frame will

appear in the middle of the screen.

A screenshot of a computer

Description automatically generated

This tool will paint a grid unit each time the corresponding MIPS memory word is referenced. The base address, the first static data segment (.data directive) word, corresponds to the upper-left grid unit. Address correspondence continues in

rowmajor order (left to right, then next row down).

The color depends on the number of times the word has been referenced. Black is 0, blue is 1, green is 2, yellow is 3 and 4, orange is 5 through 9, red is 10 or higher.

View the scale using the tool’s slider control. You can change the color (but not the reference count) by clicking on the color patch.

4. Click the tool's **Connect to Program** button. This causes the tool to register as an

observer of RISC-V memory and thus respond during program execution.

5. Back in RARS, adjust the **Run Speed slider** to 30 instructions per second.

6. Run the program. Watch the tool animate as it is updated with every access to RISC

V memory. Feel free to stop the program at any time.

7. Hopefully you observed that the animation sequence corresponded to the expected

memory access sequence of the **row-major.asm** program. If you have trouble seeing

the blue, reset the tool, move the slider to position 1, change the color to something

brighter, and re-run.

8. Repeat steps 2 through 7, for **column-major.asm**. You should observe that the

animation sequence corresponded to the expected memory access sequence of this

program.

A screenshot of a computer

Description automatically generated

9. Repeat again for **fibonacci.asm** to observe the animated pattern of memory

references. Adjust the run speed and re-run if necessary.

A screenshot of a computer

Description automatically generated

* **Explain:**

Red: 10+ accesses (heavily used in calculations)

Yellow: 3-4 accesses (write+read+calculate+print)

Green: 2 accesses (write+print only)

Blue: 1 access (initial write only)

Black: 0 accesses (unused memory)