

MARS Tutorial

April 2007

Dr. Pete Sanderson, Otterbein College
Dr. Ken Vollmar, Missouri State University



Updated by JG for UTK COSC 130 March 25, 2020

MARS is a software simulator for the MIPS assembly language intended for educational use. This tutorial was written by the developers for MARS Release 3.2.1. As a student in COSC 130, you will be using Release 4.5.0 but the topics covered are the same.

As of Release 4.0, MARS assembles and simulates 155 basic instructions of the MIPS-32 instruction set, approximately 370 pseudo-instructions or instruction variations, plus syscall functions for console and file I/O and more. As you can imagine, it would take a long time to explore all these features. We will only touch upon a limited subset and leave the rest for you to explore on your own or, perhaps, in junior or senior level courses.

Part 1 : Basic MARS Functionality

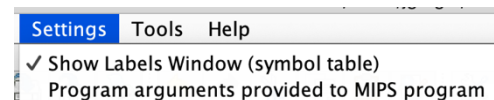
The example program is `fibonacciNumbers.asm` which computes CS student's favorite number sequence: $F_0=0$, $F_1=1$, followed $F_i=F_{i-2}+F_{i-1}$ which produces 0 1 1 2 3 5 8 ...

1. Launch MARS. On most computers, you click or double-click the icon. You may have to first allow the program to run. If you get an error message in that regard, Google it to find the solution which depends on the operating system.
2. Use the Open icon  or the File/Open option from the dropdown menubar to load `fibonacciNumbers.asm`. You are now in the Edit tab where you can explore and change the program. Stick with exploring for now.
3. Assemble the program using the icon  You are now in the Execute tab where you can run the program and explore register and memory content.
4. Identify the text segment at the top of the Execute tab. Here you see a listing of the source code plus the assembled binary code (a.k.a. machine code) and the address where each instruction ended up. Notice that pseudo-instruction `la` was translated into two so-called basic instructions, namely, `lui` and `ori`, which were then assembled.
5. Identify the data segment at the bottom of the Execute tab. Here you see the memory. You can toggle the display format between decimal and hexadecimal or even ASCII. Select decimal for now. Since the program hasn't executed yet, the `.data` segment starting at `0x10010000` has 25 words that are 0 (the `F` array), a word initialized to 25 (array size `N`), followed by a sequence of large integer values. Toggle to hexadecimal.

Still cannot decipher those large integer values? Toggle ASCII on and you will see the TXT1 text string, only it's in little-endian order. Luckily, this is only confusing to humans, not the MIPS computer.

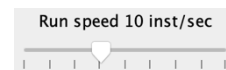
6. Display the .text segment in hexadecimal. Compare with the machine code in the .text segment. Same data. Don't worry about the other segments.
7. Use the Settings menu to configure the MARS displays. The settings will be retained for the next MARS session.

- The Labels display contains the addresses of the assembly code statements with a label, but the default is to *not* show this display. Select the checkbox from the Settings menu, then identify the listed data and text addresses in the data segment.
- Select your preference for allowing pseudo-instructions (programmer-friendly instruction substitutions and shorthand).
- Select your preference for assembling *only one* file, or *many* files together (all the files in the current folder). This feature is useful for subroutines contained in separate files, etc. We will keep each program in one file for simplicity.
- Select the startup address and value display format (decimal vs hexadecimal).







8. Locate the Registers display, which shows the 32 common MIPS registers. Other tabs in the Registers display show the floating-point registers (Coproc 1) and status codes (Coproc 0).

9. Change the Run Speed to 10 instructions per second. This allows you to “watch the action” instead of the program finishing directly.



10. Choose how you will execute the program:

- The  icon runs the program to completion. Using this icon, you should observe the yellow highlight showing the program's progress and the values of the Fibonacci sequence appearing in the Data Segment display.
- The  icon resets the program and simulator to initial values. Memory contents are those specified within the program, and register contents are generally zero.
- The  icon is “single-step.” Its complement is , “single-step backwards” (undoes each operation).




11. Observe the output of the program in the Run I/O display window:

```
F: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 ...
-- program is finished running --
```

12. Modify the contents of memory (modifying a register value is exactly the same).

- Set a breakpoint at the first instruction of the subroutine which prints results. Use the checkbox at the left of the instruction whose address is 0x00400068.

☒ 0x00400068 0x00804020 add \$8,\$4,\$0 71: PRINT: add \$t0, \$a0, \$0

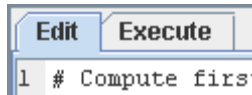
- Reset  and re-run  the program, which now stops at the breakpoint.
- Double-click in one of the memory locations containing the computed Fibonacci numbers. The cell will be highlighted and will accept keyboard entry, similar to a spreadsheet. Enter some noticeably different value, and use the Enter key or click outside the cell to indicate that the change is complete.
- Click  to continue from the breakpoint. The program output includes your entered value instead of the computed Fibonacci number.

13. Use the Help icon , then click first on the MIPS tab followed by the MARS tab.

- MIPS provides helpful information on basic instructions, pseudo-instructions, assembler directives, and syscalls. You can ignore the exceptions and macros unless really want to go deep. You will eventually become familiar with these.
- MARS provides helpful information on the IDE, debugging, and settings. You can ignore the other four tabs.

14. Modify the program to prompt the user for the last Fibonacci number to be computed.




- Select the Edit tab in the upper right to return to the program editor.



- The MIPS comment symbol is #. All characters following # are ignored.
- Un-comment lines 22-35. The newly exposed program fragment will prompt the user for the last index of the Fibonacci sequence to be considered. In order to not go out-of-bounds, this index must be appropriate for the allocated array.
- Try assembling the program. What error message do you see in the Messages Window? Hint: Instead of scrolling up and down, you can make this window bigger or smaller using the small arrows on the left.
- Determine the **syscall** parameter needed to perform “Read Integer”. Open Help



Goto to the Syscall tab and look for “Read Integer Service”. Replace the offending question mark on line 20 with the appropriate integer and assemble.

- Reset  and re-run  the program. The program will stop at the breakpoint from before (set it again, if it got cleared). Continue and finish with .

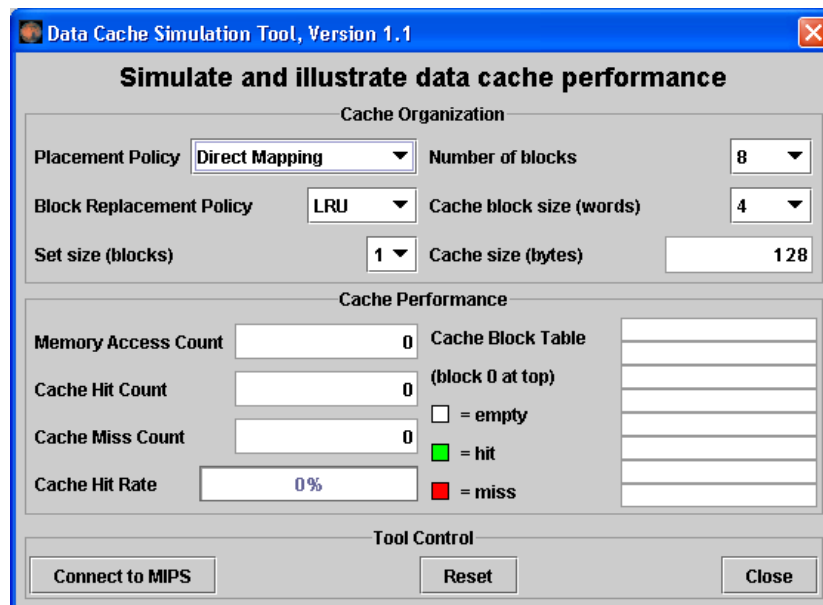
Part 2 : MARS Tools

MARS Tools Activity 1 : Running the Data Cache Simulator tool

1. Close all MIPS programs that are currently open.
2. Open the program **rowMajor.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. That is, the program implements the following algorithm:


```
for (row = 0; row < 16; row++)  
    for (col = 0; col < 16; col++)  
        data[row][col] = value++;
```

3. Assemble the program.
4. From the Tools menu, select Data Cache Simulator. A new frame will appear in the middle of the screen.




This MARS Tool will simulate the use and performance of cache memory when the underlying MIPS 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 or not 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.
5. Click the tool's Connect to MIPS button. This causes the tool to register as an observer of MIPS memory and thus respond during program execution.
 6. Back in MARS, 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.
 7. In MARS, run the program using the Run toolbar button . Watch the Cache Performance animate as it is updated with every access to MIPS memory.
 8. *What was the final cache hit rate?* _____. 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.
 9. *What do you predict the hit rate will be if the block size is increased from 4 words to 8 words vs decreased from 4 words to 2 words?* _____. Verify your predictions by modifying the block size and re-running the program from step 7.

NOTE: When you modify the Cache Organization, the performance values are automatically reset. You can also use the tool's Reset button.

NOTE: You have to reset  the MIPS program before you can re-run it.
NOTE: Feel free to adjust the Run Speed slider to maximum speed anytime you want.

10. Repeat steps 1 through 9 for program **columnMajor.asm**. This program traverses 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 MIPS memory. If you do not close the tool, then skip steps 4 and 5.

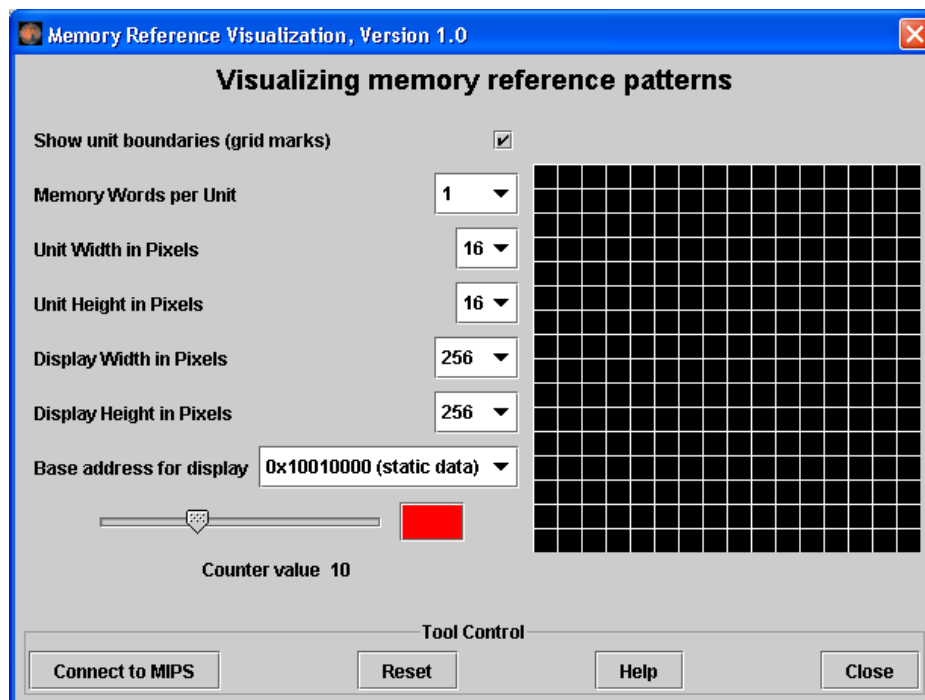
11. *What was the cache performance for this program?* _____. The problem is the memory locations are now accessed not sequentially as before, but each access is

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.

12. Set the block size to 16 and re-run the program. *What is the cache performance now?* _____. This should be no surprise as there was still only one access to each block, the initial miss, before that block was replaced with a new one.
13. Set the number of blocks to 16 and re-run the program. *What is the cache performance now?* _____. Again, this should be no surprise as the entire matrix now fits 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.

MARS Tools Activity 2 : The Memory Reference Visualization tool

1. Open the program **rowMajor.asm** from the `Examples` folder if it is not already open.
2. Assemble the program.
3. From the **Tools** menu, select **Memory Reference Visualization**. A new frame will appear in the middle of the screen.



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 row-major 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 MIPS** button. This causes the tool to register as an observer of MIPS memory and thus respond during program execution.
5. Back in MARS, 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 MIPS 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 **columnMajor.asm**. You should observe that the animation sequence corresponded to the expected memory access sequence of this program.
9. Repeat again for **fibonacciNumbers.asm** to observe the animated pattern of memory references. Adjust the run speed and re-run if necessary.
10. *(Optional)* Create a new instance of the Data Cache Simulator. Move the two frames around so you can see both. Connect the cache simulator to MIPS and reset the Memory Reference Visualization. Re-run the program. This exercise illustrates that two different tools can be used simultaneously.

MARS Tools Activity 3 : The Floating Point Tool

Try this out on your own. Without connecting with MIPS, enter some numbers and see if they are similar to what your Lab 4 code produced.

MARS Tools Activity 4 : The MIPS X-ray Tool

Back to hardware. This tool shows you what the MIPS data path looks like and how instructions are executed. Try this out on a simple program using the single-step mode (see button in the upper left corner).