

HW 5 (MIPS Assembly) Report

Task 1:

The screenshot displays the JSPlim online MIPS simulator interface. At the top, a browser tab shows the URL `shawnzhong.github.io/JsSpin/`. The interface is divided into several panels:

- Regs:** Shows the state of special and general registers. Special registers include PC (00400034), EPC (00000000), Cause (00000000), BadVAddr (00000000), Status (3000ff10), HI (00000000), and LO (00000000). General registers R0 through R31 are all set to 00000000.
- Text Segment:** Displays the assembly code for the program. The code includes instructions for loading arguments, setting up the environment, and making a system call to print the string "hello_world". The instruction `00400034: syscall` is highlighted in yellow.
- Data Segment:** Shows memory addresses and their contents. The User Data Segment and Kernel Data Segment are both empty.
- User Stack:** Shows the stack frame for the user program, with addresses ranging from 7fffffff to 414e474f.
- Execution speed:** A slider control is set to the minimum speed.
- Output:** The text "Hello, World" is displayed in the output window.
- Log:** The log shows the execution speed and the completion of the program.

For task 1 I downloaded a QtSpim version compatible with Mac OSX. This version did not work particularly well so I decided to use the online simulator. For task 2 I made a practice “Hello World” in MIPS. The results of the successful operation of the JSPIM simulator and my “Hello World” program can be seen in the screenshot above as well as my .asm file.

Task 2:

The screenshot displays the JsPsim MIPS simulator interface. At the top, a browser window shows the URL `shawnzhong.github.io/JsPsim/`. Below the browser, a navigation bar includes links to 'Feed | LinkedIn', 'LeetCode - The W...', 'Certificate Offerin...', 'Home - Workday', 'Knowledge Base - ...', 'CSD - Adam - Jira', 'Update a Person', and 'https://webutil.ws...'. The main interface is divided into several panels:

- Regs:** A panel on the left showing the state of special and general registers. Special registers include PC (00400020), EPC (00000000), Cause (00000000), BadAddr (00000000), Status (3000ff10), HI (00000000), and LO (00000000). General registers R0 through R31 are listed with their current values, mostly zero.
- Text Segment:** A central panel showing MIPS assembly code. The code includes instructions like `lw $4, 0($29)`, `addiu $5, $29, 4`, `addiu $6, $5, 4`, `sll $2, $4, 2`, `addu $6, $6, $2`, `jal 0x00400024 [main]`, `nop`, `ori $2, $0, 10`, `syscall`, `lui $8, 4097 [value]`, `lw $9, 0($8)`, `lw $10, 4($8)`, `add $11, $9, $10`, `sw $11, 0($8)`, `addu $4, $0, $11`, `ori $2, $0, 1`, `syscall`, and `jr $31`. Comments on the right explain the purpose of each instruction, such as loading arguments, setting environment pointers, and making a system call to print the value in register \$4.
- Data Segment:** A panel on the right showing memory addresses and their contents. It includes 'User Data Segment' and 'Kernel Data Segment'.
- User Stack:** A panel at the bottom right showing the stack memory layout.

At the bottom of the Text Segment panel, there is an 'Execution speed' slider and buttons for 'Play', 'Step', and 'Reset'. Below this, an 'Output' panel shows the number '48', and a 'Log' panel shows the message 'Based on SPIM Version 9.1.20 of August 29, 2017 by James Larus. Execution finished'.

For task two I utilized MIPS code to add two numbers in memory and store the result back into memory. I added extra MIPS instructions to print the result from memory to the screen to check my work.

Task 3:

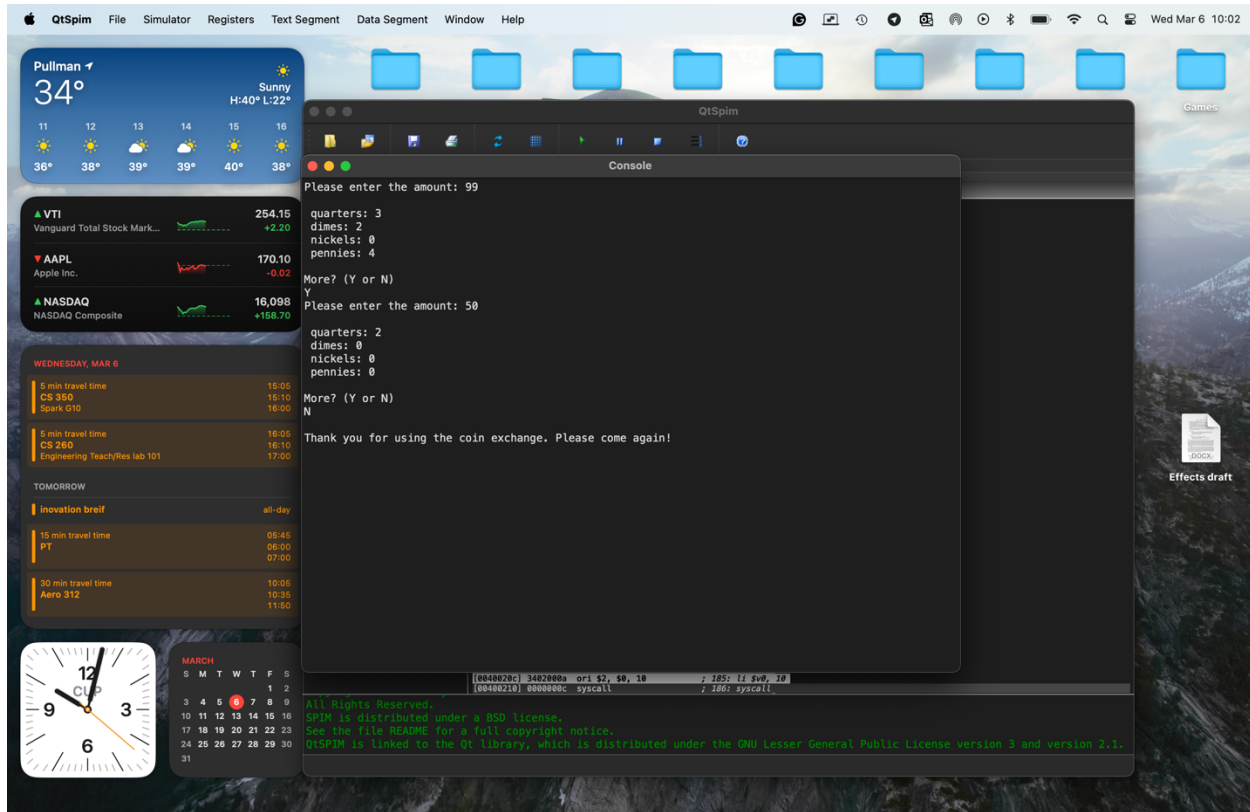
The screenshot displays the JSPTM online MIPS simulator interface. The browser tabs at the top include "Assignment 5 - MIPS", "JsPtm - Online MIPS", "Fiber Lasers: Everyth...", "LLAB 08 GLPs - Go...", "GLP 3A & 3B - Goog...", "LLAB 08 TP - Goog...", and a plus sign for more tabs. The address bar shows "shawnzhong.github.io/JsPtm/". The page has a navigation bar with links to "Feed | LinkedIn", "LeetCode - The W...", "Certificate Offerin...", "Home - Workday", "Knowledge Base - ...", "CSD - Adam - Jira", "Update a Person", "https://webutil.ws...", "Third Party Author...", and "All Bookmarks".

The main interface is divided into several sections:

- Regs:** A sidebar on the left showing the state of registers. It includes "Special Registers" (PC, EPC, Cause, BadVAddr, Status, HI, LO) and "General Registers" (R0-R31). The PC register is highlighted in yellow.
- Text Segment:** The central area displaying the MIPS assembly code for a Fibonacci program. The code includes comments and instructions like `addu $0, $0, $4`, `jal 0x00400024 [main]`, `nop`, `ori $2, $0, 10`, `syscall`, `lui $1, 4097 [array1]`, `ori $16, $1, 4 [array1]`, `lui $1, 4097 [array2]`, `ori $17, $1, 24 [array2]`, `ori $8, $0, 0`, `lui $1, 4097`, `lw $9, 0($1)`, `ori $10, $0, 100`, `lw $12, 0($17)`, `add $12, $12, $9`, `sw $12, 0($16)`, `addi $16, $16, 4`, `addi $17, $17, 4`, `addi $8, $8, 1`, `slt $1, $8, $10`, `bne $1, $0, -28 [loop-0x00400060]`, `ori $2, $0, 10`, `li $v0, 10`. The code is color-coded: comments are grey, instructions are blue, and immediate values are yellow.
- Data Segment:** A section on the right showing memory addresses and their corresponding values. It includes a "User Data Segment" and a "User Stack".
- Execution speed:** A slider and buttons for "Run", "Step", and "Reset".
- Output:** A text area for the program's output.
- Log:** A text area showing the execution log, including the version (9.1.20) and the author (James Larus).

For task 3 I wrote an assembly program to add a constant to every element of an array. I translated the for loop provided into MIPS with the base array address of array A in \$s0, the base address of array B in \$s1, I associated with \$t0, and C associated with \$t1. I chose a random constant value for c of 10. Results can be seen operating successfully in the JSPTM screenshot above as well as in my included .asm file.

Task 4:



For task four we had to translate the Python code for a very simple teller into MIPS assembly code. I did this by referencing the Python design and directly translating parts into labels which then fit together nicely. The results of input and correct output as well as testing that the upper bound of 99 works can be seen in the screenshot above.

Experience:

My experience with QTSPIM and JsSpim (I used both) was very mixed. While JsSpim worked very well, it did not display text next to input when getting user input making debugging annoying and difficult in certain circumstances. Whereas QTSPIM worked very well for debugging and execution of my code, it kept crashing as the Mac OSX version was not very stable. Overall, it was an interesting yet frustrating experience as I wish there were better simulation tools at my disposal.

From this assignment though I have learned about the intricacies of assembly code and how programming languages work at a lower level. I especially appreciated the work spent on translating Python code and appreciated example code from task 2. I also gained a better perspective on how code works closer to the processor and how instruction sets interact with processor registers.