ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE

Paper ID #37722

Analyzing the use of auto-graded labs with a built-in simulator to learn assembly programming

Chi Yan Leung

Chelsea Gordon

Research Lead at zyBooks

Efthymia Kazakou

Efthymia Kazakou is Sr. Assessments manager at zyBooks, a startup spun-off from UC Riverside and acquired by Wiley. zyBooks develops interactive, web-native learning materials for STEM courses. Efthymia oversees the development and maintenance of all zyBooks content resources used for assessment purposes.

Yamuna Rajasekhar

Yamuna Rajasekhar is a senior manager of Content at zyBooks, a Wiley Brand. She is a senior author and contributor to various zyBooks titles. She formerly was an assistant professor of Electrical and Computer Engineering at Miami University. She received her M.S. and Ph.D. in Electrical and Computer Engineering from UNC Charlotte.

Analyzing the use of auto-graded labs with a built-in simulator to learn assembly programming

Abstract

Our Computer Organization and Design (COD) online interactive textbook, adapted from a leading computer organization and systems title in 2015, introduced an embedded simulator allowing students to practice assembly programming with MIPS, ARM, or RISC-V within the textbook. In addition to our built-in simulator, in summer 2021, we developed a new auto-graded lab environment to support MIPS with 10 new lab assignments that have been used in 6 different courses.

To evaluate both the platform and the content of the labs, we analyzed these labs attempted by 28-84 students across all 6 courses. We summarize the average completion rate and the average time spent on each lab. Our analysis also shows how using the simulator impacts student struggle on homework assignments embedded in the textbook. Finally, we share our best practices for authoring similar auto-graded assembly problems.

Introduction

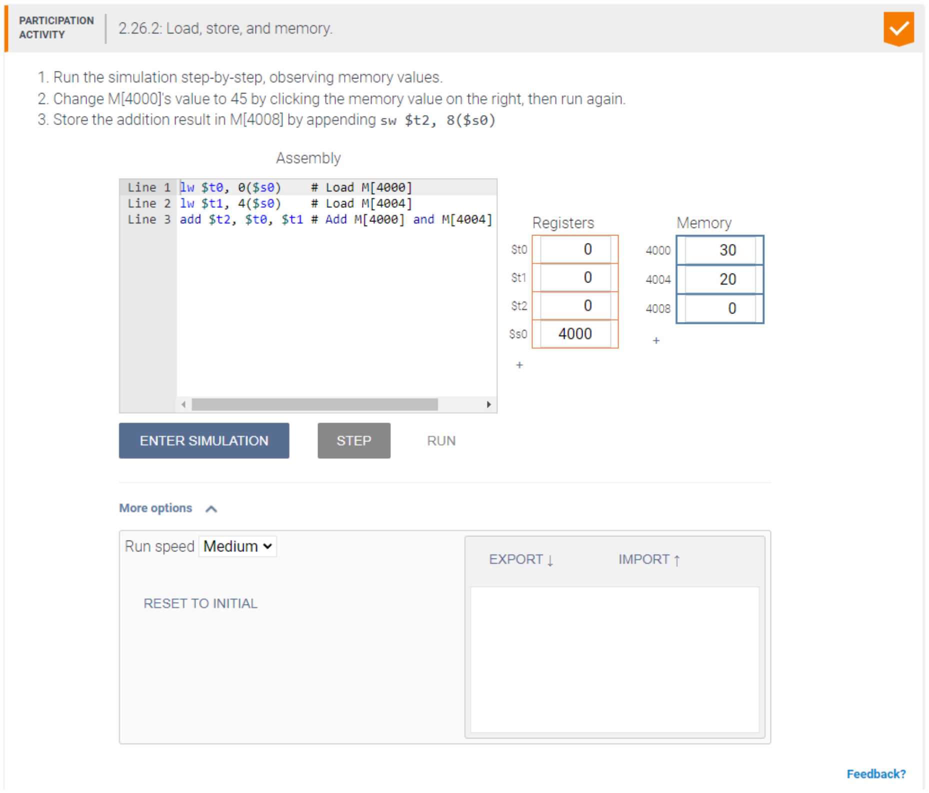
Computer organization courses often come with a programming component that is close to the hardware used in the course [1],[2]. Programming using assembly language can be unintuitive as there are no easy keywords and lower level abstractions that the simulator does. This often makes programming in an assembly language challenging to many students, especially students that are used to programming in languages like Java or Python. Additionally, students have to download and use a new integrated development environment (IDE) that maps to the hardware. This extra effort to write and debug in different environments introduces a mental overhead for students.

In this paper, we present an innovative labs solution for Computer Organization courses. The suite of labs presented are embedded in a web-native, interactive textbook. The textbook itself is an adaptation of a well known textbook in the field. The labs support assembly programming languages based on three instruction set architectures: MIPS (Microprocessor without Interlocked Pipelined Stages), RISC-V (an open standard architecture based on Reduced Instruction Set Computer), and ARM (Advanced RISC Machine). Additionally, the key innovation is a built-in simulator that enables the students to step through the program execution and gain insight into how the registers and memory interact with the program during execution. We have developed a suite of 10 sample labs to accompany the Computer Organization and Design (MIPS/ARM/RISC-V) interactive textbook and the Introduction to Computer Systems and Assembly Programming interactive textbook. The labs range in difficulty from writing simple instructions of arithmetic expressions and memory access to implementing complex procedures.

Simulator Environment

The built-in simulator allows students to practice assembly programming while reading the text, offering the opportunity for students to practice immediately as they read the text and also reducing the overhead of switching environments.

Our simulator supports 38 commonly used MIPS instructions, covering arithmetic, memory access, logics, and control flow operations. To help students focus on learning the basic behavior of assembly programs, we removed features commonly found in third-party simulators including system calls for input and output, data declarations, and memory addressing. Since this course is an introduction to assembly programming for students, we designed an environment that is simple so the students can focus on learning assembly programming. Registers and memory locations are initialized within our simulator before a program executes. Only the registers and memory locations used by the program are displayed. Addresses of memory locations can be any 32-bit unsigned integer values. As a result, students can perform memory operations without using long and difficult-to-remember memory addresses. Furthermore, students can step through a program's execution and observe the flow of the program and the interactions between the program and the storage used for the registers or memory. Students can also run the program simulations sequentially at three different speeds, allowing the students to adjust their learning pace. An example of the simulator is shown in Figure 1.


PARTICIPATION ACTIVITY | 2.26.2: Load, store, and memory.

1. Run the simulation step-by-step, observing memory values.
2. Change M[4000]'s value to 45 by clicking the memory value on the right, then run again.
3. Store the addition result in M[4008] by appending `sw $t2, 8($s0)`

Assembly

```
lw $t0, 0($s0)     # Load M[4000]
lw $t1, 4($s0)     # Load M[4004]
add $t2, $t0, $t1 # Add M[4000] and M[4004]
```

Registers

$t0	0
$t1	0
$t2	0
$s0	4000

Memory

4000	30
4004	20
4008	0

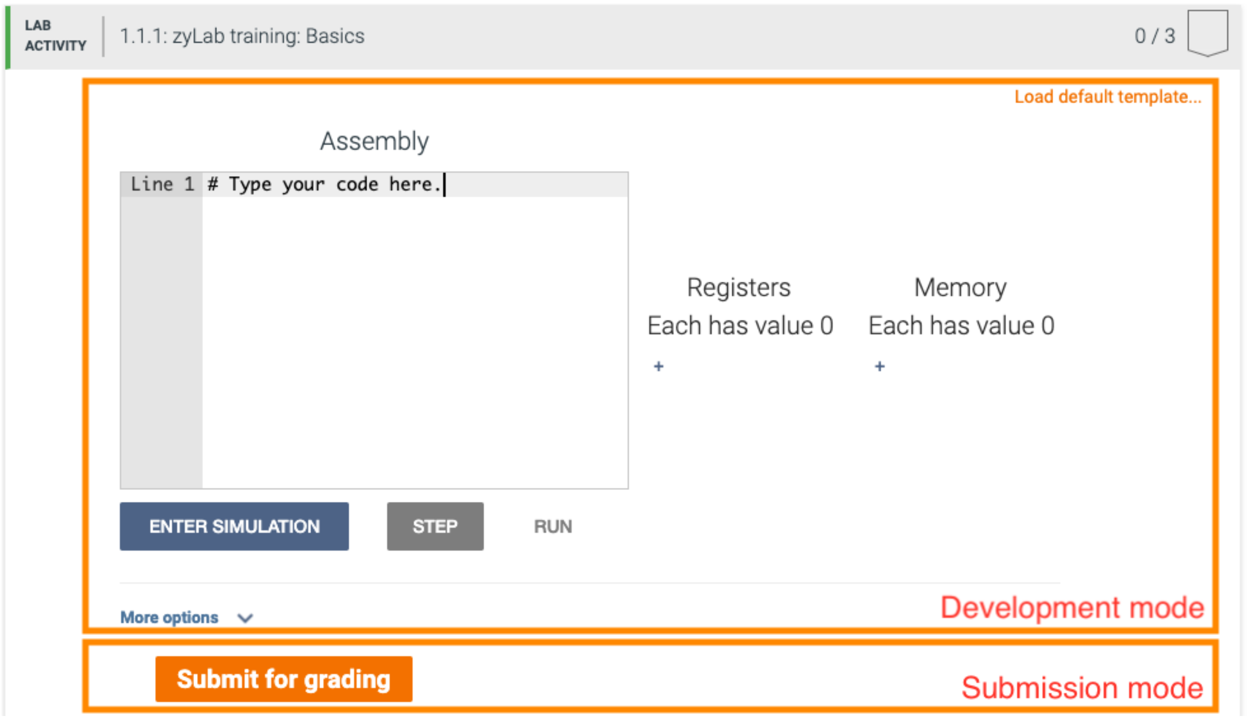
Run speed: Medium

Figure 1: The simulator environment with the programming window, the memory, and registers.

Innovative lab environment

While the simulator in the book adds an advantage to aid student learning and provides a built-in practice environment, lack of grading means that students have no feedback at all on their program. Our lab environment combines our simulator with an auto-grader that offers students immediate feedback, and thus improves learning and reduces student frustration [3]. When combined with our MIPS simulator, our labs platform creates an environment for students to practice and assess their assembly programming skills. This integration also saves students from performing extra logins or unnecessary file transfers. In addition, this autograding feature provides an advantage to instructors because students are able to get meaningful feedback as they work on their labs leading to less dependency on the instructor.

As presented in Figure 2, our lab environment offers two different modes: development and submission. In the development mode, students implement their programs in the coding window of a built-in simulator. With the assistance of the simulator, students can step through a program's execution and observe their program's behavior.


LAB ACTIVITY | 1.1.1: zyLab training: Basics 0 / 3

Load default template...

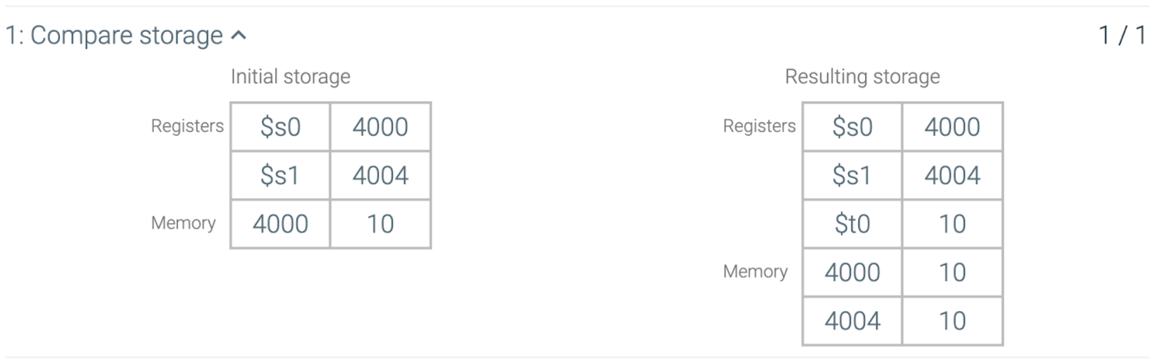
Assembly

```
Line 1 # Type your code here.
```

Registers	Memory
Each has value 0	Each has value 0

Figure 2: Development and submission mode of our lab environment.

Each execution in the simulator is not graded, and students can make as many changes as necessary to ensure their programs behave correctly. Once students are ready to submit their programs, they click the submit button to initiate the submission process. In submission mode, an auto-grader tests a student's program against a set of test cases. Each test case compares the values of a specific group of registers or memory locations, generated by the program. A score is awarded when a test has passed (Figure 3). If a test fails, the auto-grader indicates the incorrect results generated by the student's program (Figure 4).


1: Compare storage 1 / 1

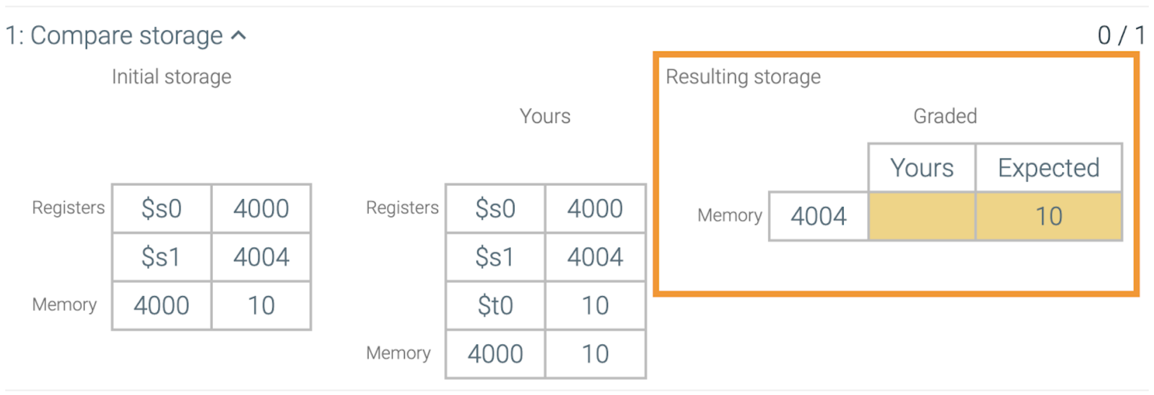
Initial storage

Registers	$s0	4000
	$s1	4004
Memory	4000	10

Resulting storage

Registers	$s0	4000
	$s1	4004
	$t0	10
Memory	4000	10
	4004	10

Figure 3: Lab execution results when all tests pass.


1: Compare storage

0 / 1

Initial storage

Registers

$s0	4000
$s1	4004

Memory

4000	10

Yours

Registers

$s0	4000
$s1	4004
$t0	10

Memory

4000	10

Resulting storage

Graded

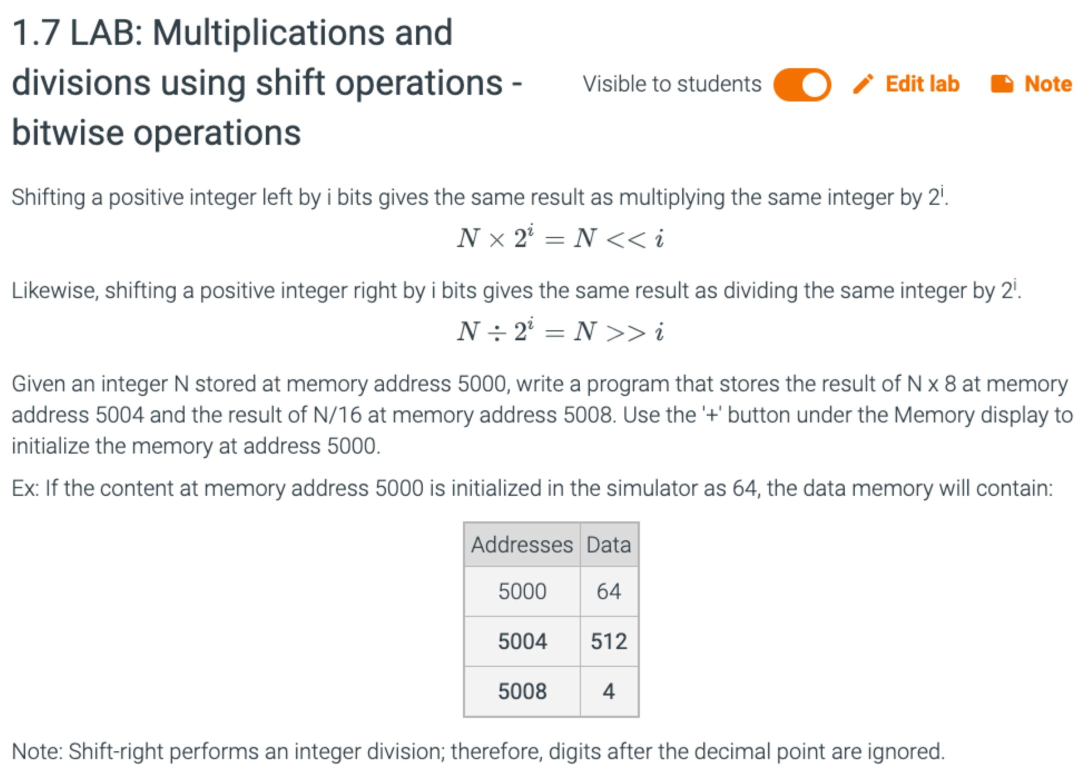
Memory	Yours	Expected
4004		10

Figure 4: Lab execution results when some tests fail.

Labs and Methods

We have created ten new labs to help students understand the basic constructs of MIPS assembly programs. Besides a training lab that introduces students to our lab environment, the other nine labs cover the concepts such that each falls into one of five categories: arithmetic and logical operations, memory access, conditional branching, array operations, and procedures.

1) In the category of arithmetic and logical operations, three labs are designed to assess a student's ability to apply MIPS instructions on additions, subtractions, multiplications, and bitwise logical shifts. (Arithmetic expression - add/sub, Arithmetic expression - add/sub/mult, and Multiplications and divisions using shift operations - bitwise operations)



1.7 LAB: Multiplications and divisions using shift operations - bitwise operations

Shifting a positive integer left by i bits gives the same result as multiplying the same integer by 2^i.

$$N \times 2^i = N << i$$

Likewise, shifting a positive integer right by i bits gives the same result as dividing the same integer by 2^i.

$$N \div 2^i = N >> i$$

Given an integer N stored at memory address 5000, write a program that stores the result of N x 8 at memory address 5004 and the result of N/16 at memory address 5008. Use the '+' button under the Memory display to initialize the memory at address 5000.

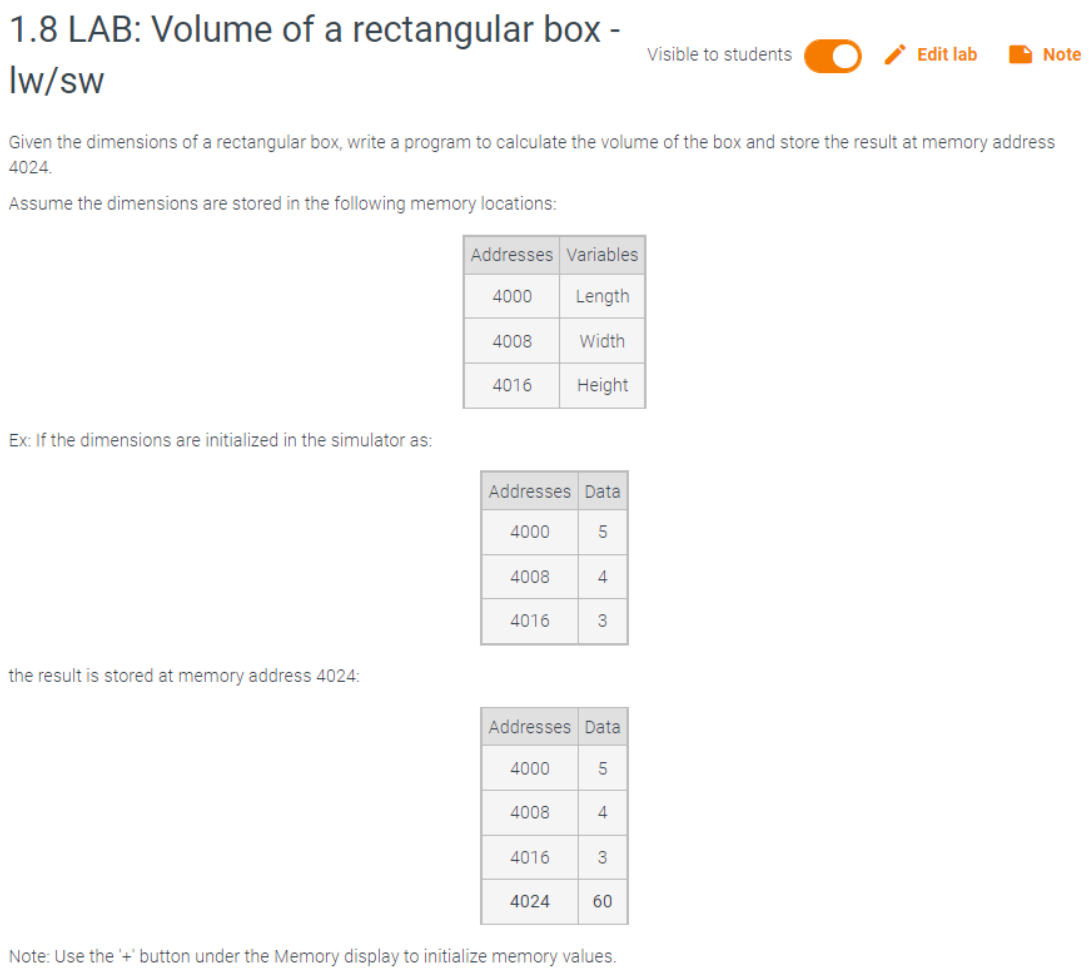
Ex: If the content at memory address 5000 is initialized in the simulator as 64, the data memory will contain:

Addresses	Data
5000	64
5004	**512**
5008	**4**

Note: Shift-right performs an integer division; therefore, digits after the decimal point are ignored.

Figure 5: Overview of the multiplications and divisions using shift operations - bitwise operations lab

2) In the category of memory access, a lab is designed to assess a student's ability to retrieve data from the memory and apply arithmetic operations to the retrieved data. (Volume of a rectangular box - lw/sw).



1.8 LAB: Volume of a rectangular box - lw/sw

Given the dimensions of a rectangular box, write a program to calculate the volume of the box and store the result at memory address 4024.

Assume the dimensions are stored in the following memory locations:

Addresses	Variables
4000	Length
4008	Width
4016	Height

Ex: If the dimensions are initialized in the simulator as:

Addresses	Data
4000	5
4008	4
4016	3

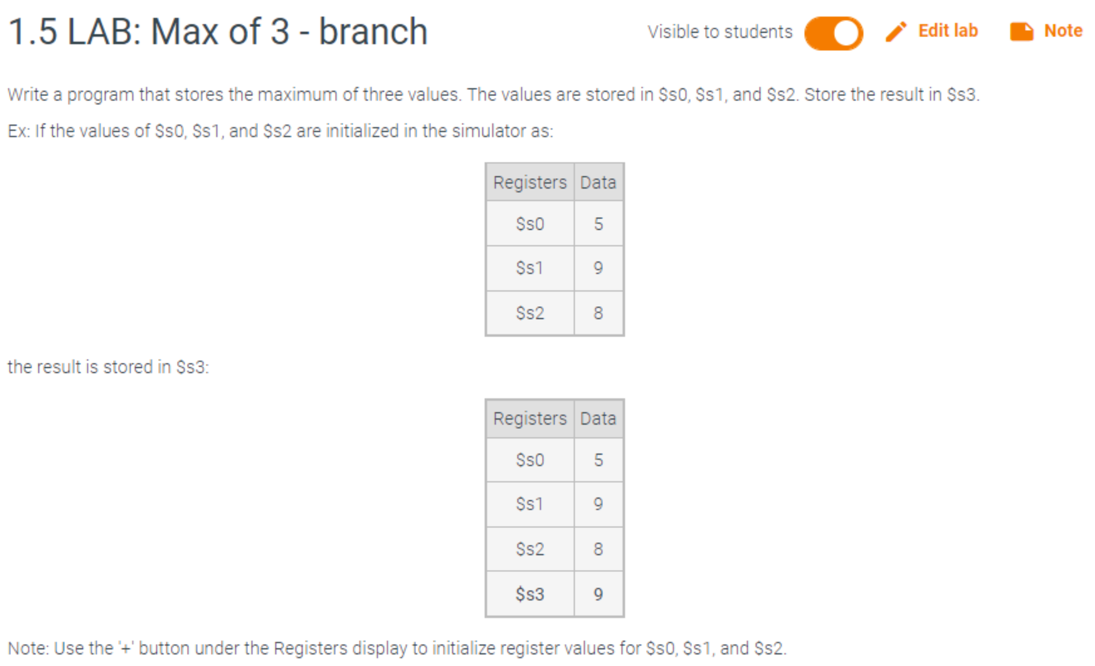
the result is stored at memory address 4024:

Addresses	Data
4000	5
4008	4
4016	3
4024	**60**

Note: Use the '+' button under the Memory display to initialize memory values.

Figure 6: Overview of the Volume of a rectangular box - lw/sw lab

3) In the category of conditional branching, a lab is designed to assess a student's ability to apply the correct sequence of conditional branching instructions to find the maximum values among three numbers. (Max of 3 - slt/branch)


1.5 LAB: Max of 3 - branch

Visible to students Edit lab Note

Write a program that stores the maximum of three values. The values are stored in $s0, $s1, and $s2. Store the result in $s3.

Ex: If the values of $s0, $s1, and $s2 are initialized in the simulator as:

Registers	Data
$s0	5
$s1	9
$s2	8

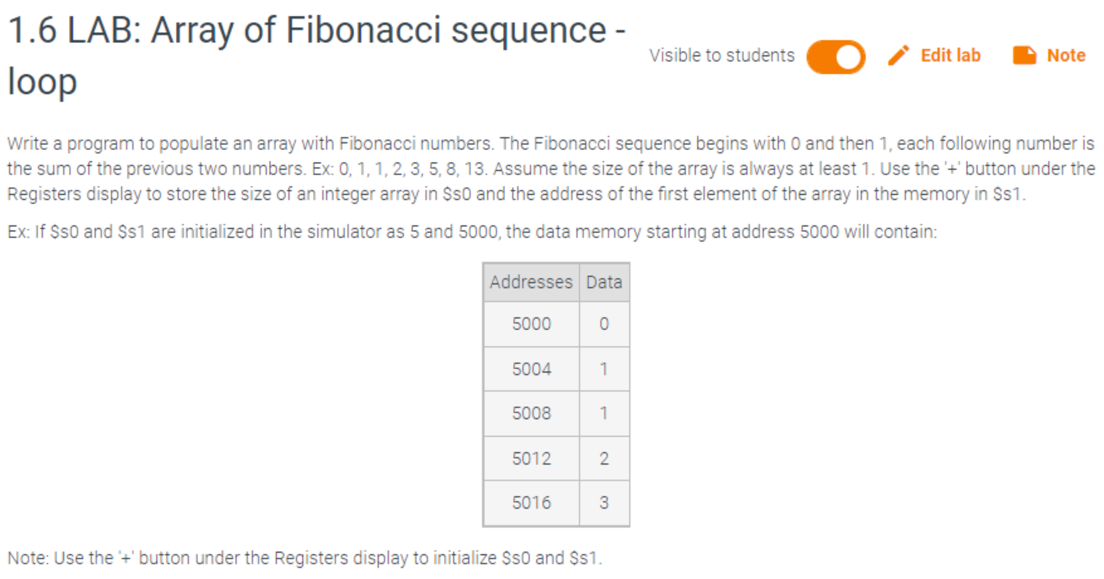
the result is stored in $s3:

Registers	Data
$s0	5
$s1	9
$s2	8
$s3	9

Note: Use the '+' button under the Registers display to initialize register values for $s0, $s1, and $s2.

Figure 7: Overview of the Max of 3 - slt/branch lab

4) In the category of array operations, concepts of memory access and conditional branching are combined, and two labs are designed to assess a student's ability to perform sequential memory access and loop operations. (Array of squares - lw/sw and Array of Fibonacci sequence - loop)


1.6 LAB: Array of Fibonacci sequence - loop

Visible to students Edit lab Note

Write a program to populate an array with Fibonacci numbers. The Fibonacci sequence begins with 0 and then 1, each following number is the sum of the previous two numbers. Ex: 0, 1, 1, 2, 3, 5, 8, 13. Assume the size of the array is always at least 1. Use the '+' button under the Registers display to store the size of an integer array in $s0 and the address of the first element of the array in the memory in $s1.

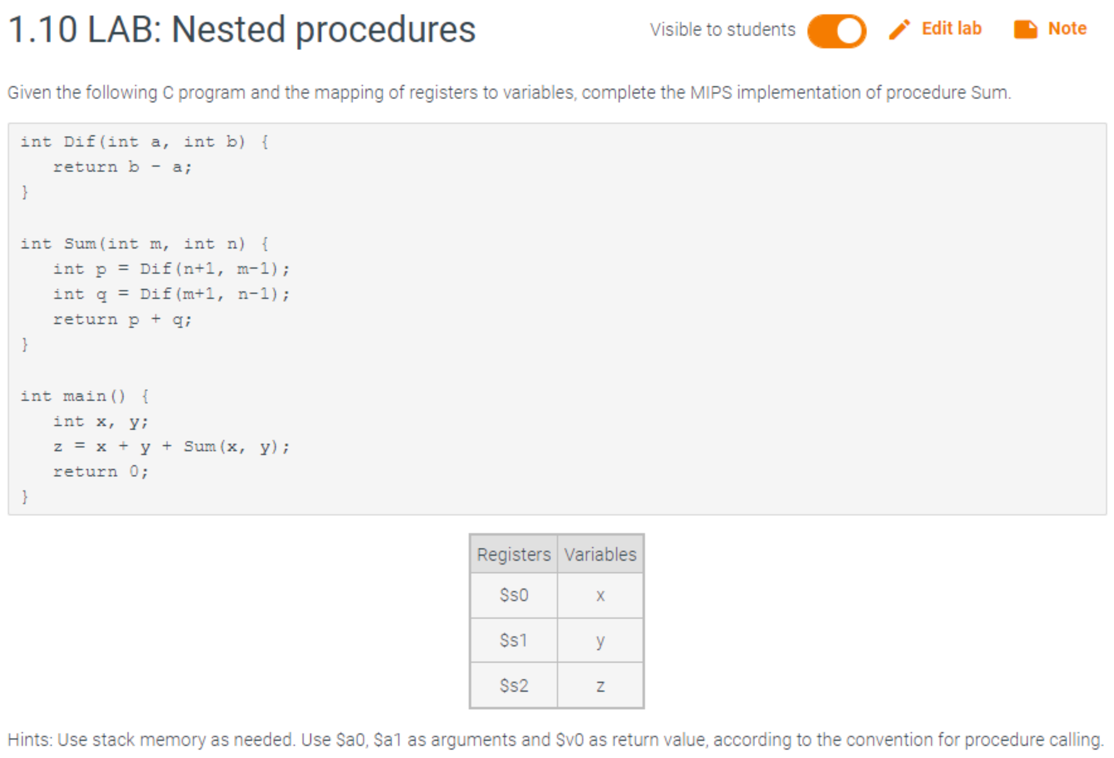
Ex: If $s0 and $s1 are initialized in the simulator as 5 and 5000, the data memory starting at address 5000 will contain:

Addresses	Data
5000	0
5004	1
5008	1
5012	2
5016	3

Note: Use the '+' button under the Registers display to initialize $s0 and $s1.

Figure 8: Overview of the Array of Fibonacci sequence - loop lab

5) In the category of procedures, two labs are designed to assess a student's ability to make procedure calls and implement procedures in a MIPS assembly program. (Procedure calls and Nested procedures)



Given the following C program and the mapping of registers to variables, complete the MIPS implementation of procedure Sum.

```
int Dif(int a, int b) {
   return b - a;
}

int Sum(int m, int n) {
   int p = Dif(n+1, m-1);
   int q = Dif(m+1, n-1);
   return p + q;
}

int main() {
   int x, y;
   z = x + y + Sum(x, y);
   return 0;
}
```

Registers	Variables
$s0	x
$s1	y
$s2	z

Hints: Use stack memory as needed. Use $a0, $a1 as arguments and $v0 as return value, according to the convention for procedure calling.

Figure 9: Overview of the Nested procedures lab

Our goal was to better understand student usage of the nine new auto-graded lab assignments that have been used in 6 different courses.

Metrics

For each of the nine lab assignments, we defined the following metrics:

- Average time spent (minutes): Of students who submitted to the lab, the median amount of time spent by students in the lab.
- Average number of submissions: Of students who submitted to the lab, the median number of total program runs that they made. This includes both submission and simulator runs.

We only included students who spent at least one minute in the lab. The number of students counted for each lab can be seen in table 1.

Lab	Students
Arithmetic expression	45
Arithmetic expression 2	57
Array of squares	82
Max of 3	83
Array of Fibonacci sequence	72
Multiplications and divisions	57
Volume of a rectangular box	84
Procedure calls	28
Nested procedures	34

Table 1: Number of students who completed each lab

Results and Discussion

Figures 10 and 11 show the median number of the combined simulator and submission runs and the total development time spent in minutes for each COD lab. The labs are listed in a chronological order. The time spent data confirms the experience of many instructors teaching this course that learning the implementations of the abstraction structures such as memory organizations, control flow, and procedures in assembly language is more challenging than learning the same concepts in high-level programming languages such as Java and Python. In those languages, students typically spend between 10 and 30 minutes on a typical lab assignment. In the Array of squares lab, students need to create the memory structure of an array, which requires students to keep track of the memory addresses of the elements stored in an array. In the Max of 3 lab, students need to instruct the program which part of the program to go to after a test is performed because if-else clauses are not available in assembly language. In the Array of Fibonacci lab, students need to implement the flow of the program because loop structures are not available in assembly language. In the Nested procedures lab, students need to maintain the addresses of the program instructions so that the program can execute the correct instructions after returning from a procedure call. High-level programming languages often encapsulate these abstractions so a programmer does not need to manage where data or instructions are stored in a physical memory.

The time spent data shown in figure 11 also suggests that our lab ordering helps students learn the concepts in the same category effectively. After completing the Arithmetic expression lab, students took close to half the time on the subsequent and more difficult Arithmetic expression 2 lab. A similar observation can be made between the Array of square lab and the Volume of a rectangular box lab. With thoughtful planning on the difficulties and arrangement of labs, students can learn the more challenging concepts in assembly programming effectively.

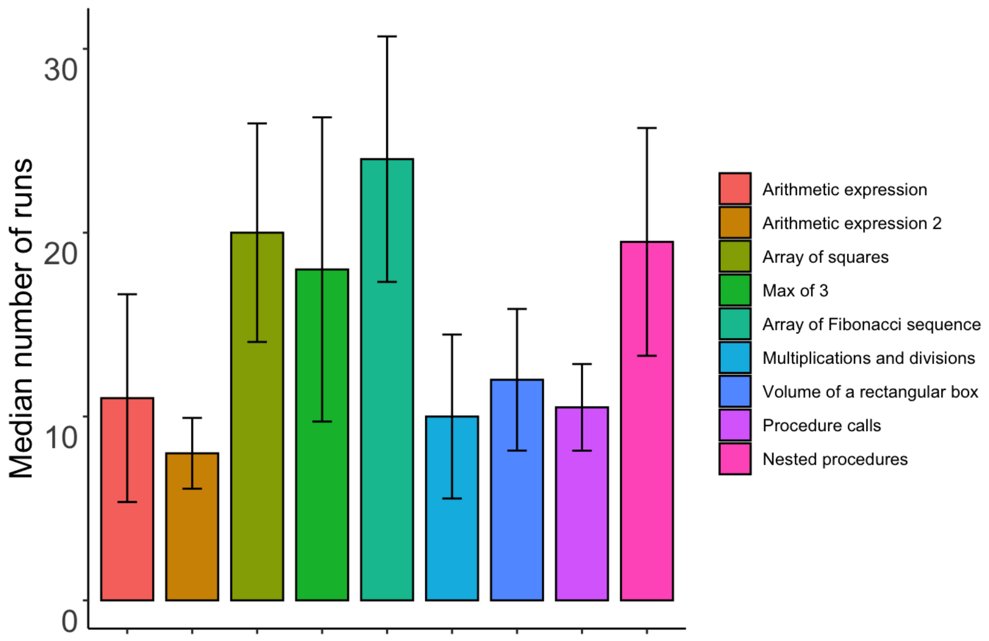


Figure 10: Median number of total program runs per student for each COD lab. Error bars represent the 95% confidence interval.

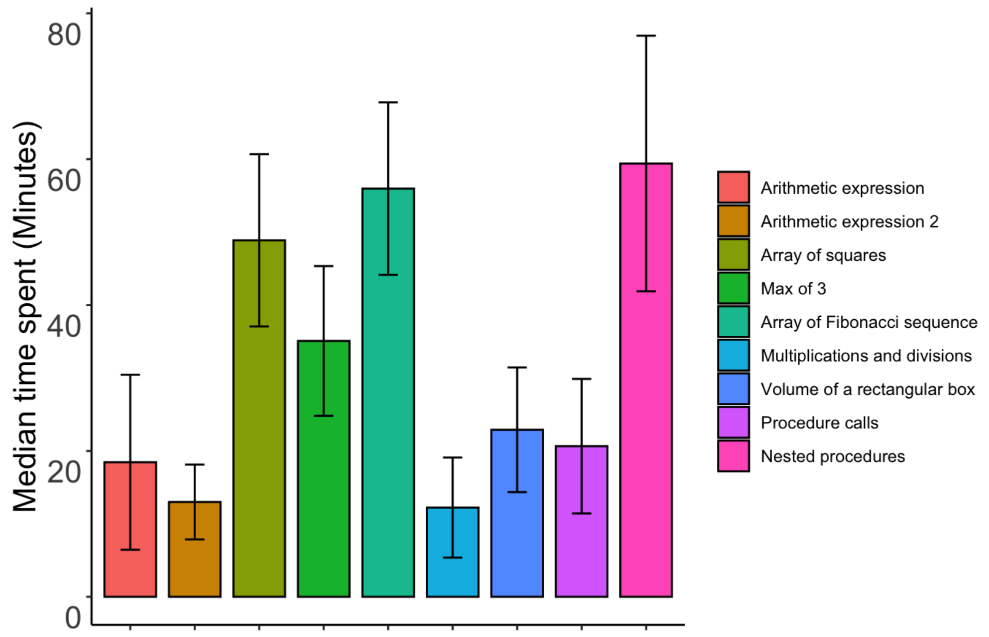


Figure 11: Median number of total minutes spent per student for each COD lab. Error bars represent the 95% confidence interval.

Figure 12 shows the median number of program runs for each COD lab in the development and submission mode. The data suggests that students are utilizing the built-in simulator to develop and troubleshoot their programs. This observation is especially true when the students are doing the more difficult labs. While the number of submission runs remains low, the number of development runs is substantially varied between labs, suggesting that the simulator is helping them learn the concepts that they find difficult.

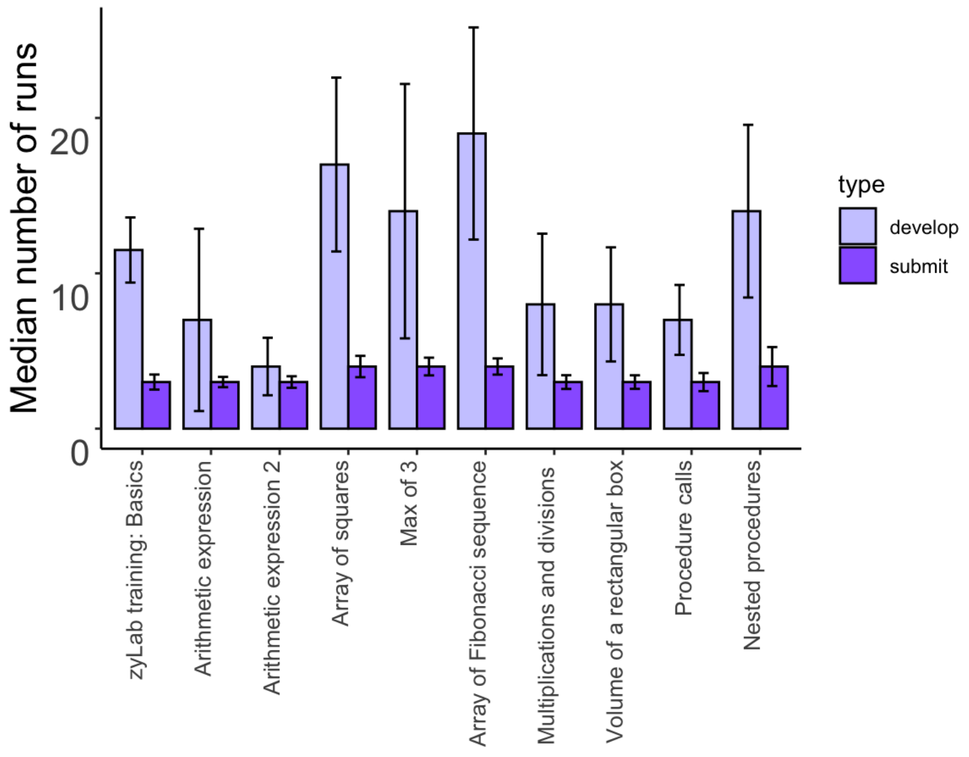


Figure 12: Median number of program runs for each COD lab in the development and submission modes

Conclusion

This paper analyzed the use of nine auto-graded lab activities by 28-84 students across 6 courses. We observed the average time spent and the average number of submissions for each lab assignment. Time spent data aligned with our expectations, indicating that learning assembly programming is difficult, but can be made easier with good interactive simulators.

These results come from early adopters of the new COD labs, and as such is a small sample size. Future work includes collecting more data for the labs analyzed above as our COD books become available to all institutions and not just the 6 universities that beta tested our labs. Also, the ARM and RISC-V versions of the book will be available for evaluations in Spring 2022. In addition to the ARM and RISC-V versions, an introduction to Computer Systems and Assembly Programming (CSAP) version will follow soon after. CSAP covers MIPS assembly programming as well as some basic concepts of computer organization and uses a simplified version of MIPS to make learning assembly programming easier.

References

[1] K. Vollmar and P. Sanderson. "MARS: an education-oriented MIPS assembly language simulator," *In Proceedings of the 37th SIGCSE technical symposium on Computer science education* (pp. 239-243), March 2006.

[2] Y. Hung, "Combining Self-Explaining With Computer Architecture Diagrams to Enhance the Learning of Assembly Language Programming," in *IEEE Transactions on Education*, vol. 55, no. 4, pp. 546-551, Nov. 2012, doi: 10.1109/TE.2012.2196517.

[3] C. L. Gordon, R. Lysecky, and F. Vahid. "The rise of program auto-grading in introductory cs courses: A case study of zylabs," *In 2021 ASEE Virtual Annual Conference.,* July 2021.