Enhancing High-Level Language Concept Comprehension through a Notional Machine Approach of Assembly Language Education

Dr. SAGNIK NATH, University of California, Santa Cruz

Sagnik Nath received his PhD in Electrical Engineering from Renssalaer Polytechnic in 2020 and his B.E. in Electronics and Communications Engineering in 2015 from Indian Institute of Engineering Science and Technology, Shibpur, India. He is currently a Teaching Professor at the Computer Science and Engineering division of Baskin Engineering at UC Santa Cruz. His research interests include incorporating DEI in engineering education, computer architecture and VLSI design.

Enhancing High-Level Language Concept Comprehension through a Notional Machine Approach of Assembly Language Education

Abstract

In computer science curricula, the inclusion of assembly language programming is commonplace regardless of students' focus on computer science (CS) or computer engineering (CE) majors. The key objective within our university's(University of California, Santa Cruz) foundational "Computer Systems and Organization" (CSE12) course is dual-fold: to cultivate a deep understanding of machine architecture's programmer model and to seamlessly prepare both CS and CE undergraduates for the advanced concepts in the subsequent "Computer Architecture" (CSE120) course.

Current literature also highlights the merits of teaching assembly language, positing that it enables proficiency in higher-level programming languages. However, transitioning from an introductory high-level language-based programming to assembly language can be jarring due to the stark contrast in instruction paradigms. In response, we have reimagined assembly language assignments within CSE12. Leveraging the open-source RARS assembly language runtime emulator, we have reshaped lab exercises to better emulate scenarios akin to those encountered in high-level language programming. This transforms RARS into a Notional Machine that bridges the gap between high-level language abstractions and low-level assembly implementation. Gathering quantitative and qualitative feedback from course surveys, our study reaffirms the effectiveness of this pedagogical strategy. Preliminary data suggest that students not only gain a deeper comprehension of machine architecture but also exhibit improved readiness for subsequent courses, underscoring the notional machine's role in facilitating a smoother transition between programming paradigms. This abstract encapsulates our ongoing efforts to refine computer science curricula, highlighting the promising impact of RARS in enriching students' educational experiences.

1 Introduction

In computer science and computer engineering curricula, assembly language programming holds a significant position. Its inclusion not only offers practical applications but also plays a pivotal role in laying the foundation for understanding machine operations in-depth. At our university, the CSE12 course, 'Computer Systems and Organization', serves as a gateway for both CS and CE majors into the world of assembly language programming, emphasizing the RISC-V architecture and specifically the RV64I variant. This course is designed to equip students with the necessary skills and knowledge to transition seamlessly to the advanced 'Computer Architecture' course, CSE120, in their junior year. However, the leap from high-level language programming to assembly language presents cognitive challenges. Numerous students in the CSE12 course grapple with reconciling the paradigms of high-level language programming with that of assembly. These challenges and the resulting student frustrations prompted a reconsideration of the assembly language assignments within the course. This paper discusses our specific approach to addressing this pedagogical challenge.

This paper is organized as follows: Section 2 outlines the background and relevant works. Section 3 details our Notional Machine approach in CSE12 in W23 and S23 quarters. In Section 4, we present our data collection and feedback from W23, S23. Section 5 details our Notional Machine approach in F23 quarter and Section 6 provides the data collection and feedback from that quarter. Section 7 delves into discussion and analysis, while Section 8 concludes our findings and future directions.

2 Background and Related Work

In computer science, assembly language programming extends beyond its immediate practicalities, such as direct hardware manipulation and optimized performance. When students juxtapose high-level programming with assembly, they deepen their understanding of how abstract data types, like linked lists or queues, materialize at the machine level. For instance, the direct interaction with memory addresses in assembly language provides a tangible grasp of pointers, often considered abstract in high-level languages [1]. Assembly language also exposes students to crucial engineering trade-offs, such as balancing code efficiency with maintainability [1]. Additionally, its understanding is indispensable for advanced topics, including compiler construction, operating systems, and computer design [2].

However, the transition from high-level languages to assembly isn't always smooth. A challenge that repeatedly emerges in courses like CSE12 at our university is the cognitive dissonance reported by students. The gulf between high-level programming concepts and assembly paradigms often results in frustrations and doubts about the necessity of studying assembly programming in the first place. This sentiment isn't just confined to our institution; many educational settings globally have noted similar observations [3], [4], [5]. Many students entering CSE12 have only had prior programming experience in an introductory Python course (CSE20) that was taught at a very high level of abstraction. Furthermore, students anchored in high-level language (HLL) paradigms frequently misjudge the complexity underlying the conveniences offered by the higher abstraction [6]. Confronted by the intricate details of the ISA, students might adopt a fragmented learning strategy, focusing more on individual instructions rather than comprehending an integrated machine model. This shallow approach directly contrasts with the primary objectives of assembly language courses [6].

Several attempts have been reported in the literature to enable a smoother transition from HLL into assembly. For example, [3] proposes a unique classification system for assembly instructions in a small microcontroller, presented in reference tables based on functionality, allowing students to easily match actions with appropriate mnemonics. Instead of first attempting complex projects, students replicate core high-level programming constructs in assembly, such as flow control and modular techniques to familiarize themselves with a one-to-one mapping between HLL and assembly. [4] suggests a blend of assembly with C, C++, or Java, leveraging standard tools and compilers rather than specialized, course-supplied interfaces as a bridge between high-level and low-level abstraction. [6] proposes using the assembly course exercises as the foundation to acquaint students with the models of computer systems, as relevant to operating systems or the runtime environment. [7] advocates the inclusion of debugging exercises, debugging logs, development logs/memos, and collaborative assignments to foster familiarity with assembly language.

While the literature offers a myriad of strategies to familiarize students with the transition from high-level languages to assembly, our approach must cater to the unique pedagogical challenges posed by the structure of our university-specific curriculum. CSE12, a 7-credit course running over a condensed 2.5-month quarter, stands as the primary touchpoint for assembly language within our Computer Science (CS) and Computer Engineering (CE) tracks. It serves as a crucial bridge, introducing students to the foundational intricacies of computer systems, from digital logic and computer architecture to the mechanics of compiling and the symbiosis of hardware and software. The course first delves separately into digital logic and foundational hardware design before jumping into computer organization through assembly language coding. Given the emphasis on C in the subsequent follow up course CSE13, including embedded systems, and the reality that most students won't grapple with assembly language post-CSE12, there's a pressing need to reconceptualize how we introduce these concepts. Moreover, diving deep into advanced topics pertinent to operating systems or runtime environments might dilute the core objective of this course. Contemporary operating systems, operating at a much higher level of abstraction, often decouple students from the raw machinations of computer processes.

This paper introduces a paradigm shift, suggesting a novel perspective on assembly language programming as a vehicle to impart foundational computer science concepts. Recognizing the primary exposure of many students to programming through abstract languages like Python, we posit that diving directly into assembly might be too overwhelming for students. Instead, leveraging the timeless principles of the von Neumann architecture, we propose a 'notional machine' [8] approach. This didactic tool will act as an intermediary, conceptual bridge: a model of computation that distills the essence of computing to its most basic form, i.e., the orchestrated transaction of data between CPU registers and memory. Through this lens, students are enabled to perceive the universality of computational processes, understanding that the various algorithms and applications they encounter are various manifestations of this fundamental sequence of data transaction. By presenting assembly language through this notional machine approach, we acknowledge that it is but another perspective to programming concepts that students learn from HLL. While HLLs like Python are powerful tools to teach core computing concepts to beginners, assembly language through a notional machine approach can provide another crucial dimensionality to a novice's holistic learning of computing. This approach can help bridge the gap between the abstract and concrete, allowing such students to appreciate the intricate interplay between high-level abstractions and low-level implementations. It's important to note that diverse teaching methods, alongside the notional machine approach, can collectively contribute to a richer and more comprehensive understanding of computing, ultimately preparing students for the multifaceted challenges of the computer science landscape.

In the literature on computing education, the central hurdle of guiding novice programmers to determine 'where to look' within the fabric of code has driven educators to investigate inventive pedagogical approaches, including the utilization of 'notional machines' (NMs)[9]. Originating in the work by Gibson [8], notional machines offer a solution to the complexity inherent in learning programming. A notional machine functions as a representation or analogy that directs attention to key aspects of code, making visible otherwise concealed behaviors. In essence, notional machines are designed to illuminate what may remain obscured, guiding learners toward the essential variables, constructs, and interactions that underpin the logic of a program [9]. Building upon the foundation laid by Gibson's work [8] and the insights provided in [9], we propose an innovative approach that leverages the concept of a notional machine to bridge the cognitive gap between high-level programming and assembly language paradigms. By distilling the complexities of the von Neumann architecture into a comprehensible framework that mirrors the orchestrated data transactions between CPU registers and memory, we aim to provide students with a profound understanding of the fundamental principles underlying computational processes.

3 Methodology for the Winter 2023 and Spring 2023 quarters

For the lab assignments, the course Computer Systems and Organization (CSE12) uses Digital[10] for digital design and RARS [11] for assembly language coding. Starting from the winter 23 quarter, the approach to teaching computer organization underwent a significant change. It structured assembly coding comprehension on a notional machine framework to impart the fundamental concept of programming. In transitioning from digital design to assembly language coding mid-quarter, students are first introduced to the basic Von Neumann architecture [Figure 1], encompassing main memory, CPU with registers, and I/O. This forms the core visualization of the Notional Machine (NM). We juxtapose this with an image in the lecture: the CPU module engaged in a cash transaction (symbolizing data) with the Memory module [Figure 2]. This serves as a visual metaphor conveying that programming condenses to guided data transactions between CPU registers and memory, guided by program code within the same memory (a key Von Neumann feature). We explicitly inform students that this new programming perspective employs the "notional machine" concept. We are now poised to introduce the assembly labs, closely paralleling our lecture content. Guided by the foundational principles of our developed notional machine in CSE12, we have thoughtfully structured the educational outcomes stemming from the lab experience. These outcomes are grounded in the four definitional characteristics of a notional machine as outlined in [9],

Pedagogical Purpose: NMs aid student learning in computational concepts by simplifying actual concepts or skills.

Function: NMs reveal insights about programming, computers, or computation, highlighting aspects not evident in student-used artifacts.

Focus: NMs concentrate on specific aspects of program behavior and the role of computers in development, execution, and storage.

Representation: NMs use selective representation to emphasize certain focus aspects while possibly omitting others.

Prior to beginning the assembly labs, we thoroughly acquainted the students in our lab sections with the workings of RARS including supported instructions, system calls, assembler directives, compile time and runtime placement of data in memory, and runtime breakpoints.

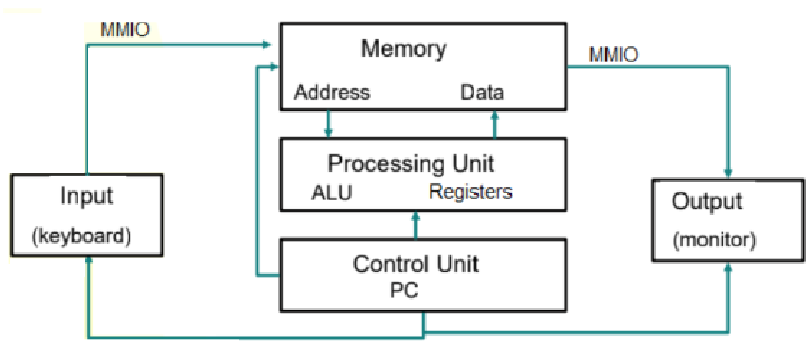


Figure 1: *The basic Von Neumann architecture which forms the crux of the Notional Machine as the conceptual model for programming.*

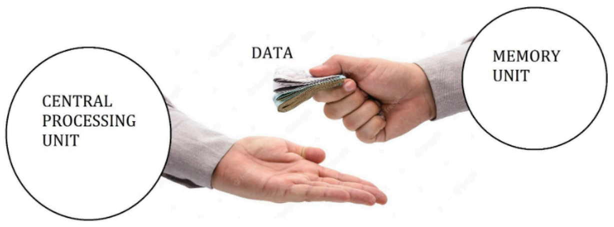
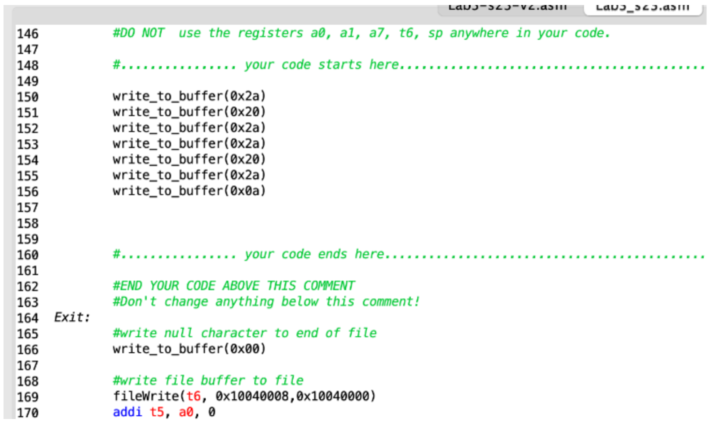


Figure 2: *A visual metaphor provided to students to emphasize programming as the guided data transaction between CPU registers and memory.*

In the following subsections, we detail our notional machine approach in designing lab assignments for CSE12. The initial lab focuses on creating text file character patterns, similar to early exercises in High-Level Language (HLL) courses. Subsequent labs progress to more complex functions, akin to typical introductory HLL coding tasks. Lab assignment 3.1, a staple in all CSE12 iterations, introduces assembly language (Lab 3). Labs outlined in sections 3.2, 3.3, and 3.4 correspond to Lab 4 in the W23, S23, and F23 quarters, respectively.

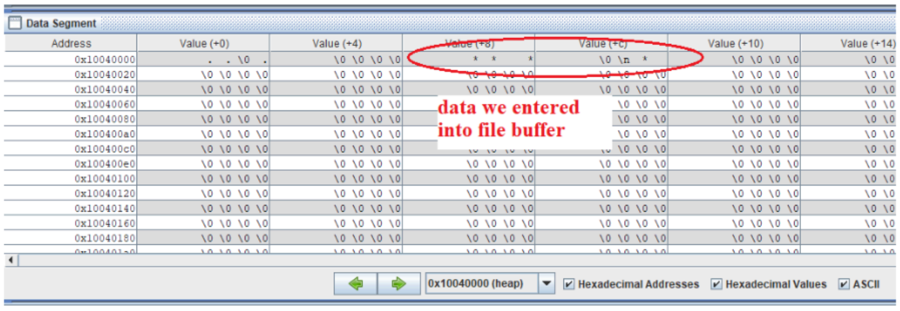
3.1 Introduction to Assembly Language through the Lens of the Notional Machine

In the initial assembly coding lab, students create character patterns from user input using nested loops, written to a text file, in line with notional machine characteristics from [9]. Pedagogically, the lab introduces the fundamental Von Neumann architecture (CPU, registers, memory), the basis of our notional machine, illustrating computing as data transfers between registers and memory. A visual metaphor of cash transactions between CPU and memory helps students conceptualize programming as controlled data exchanges, reducing cognitive load in the RARS environment. The lab focuses on conditional loops and system calls for I/O, highlighting data placement in memory—a critical programming aspect. This aligns with the concept of data transactions in our notional machine. To reinforce these principles, the lab provides detailed examples and instructions.Instructor provided macros like 'write_to_buffer' and 'fileWrite' [Figure 3] demonstrate data transactions, aiding in visualizing register-memory interactions. Their use in representing data placement is shown in examples [Figure 4].



```
	#DO NOT  use the registers a0, a1, a7, t6, sp anywhere in your code.

	#............... your code starts here.........................................

	write_to_buffer(0x2a)
	write_to_buffer(0x20)
	write_to_buffer(0x2a)
	write_to_buffer(0x2a)
	write_to_buffer(0x20)
	write_to_buffer(0x2a)
	write_to_buffer(0x0a)



	#............... your code ends here..........................................

	#END YOUR CODE ABOVE THIS COMMENT
	#Don't change anything below this comment!
Exit:
	#write null character to end of file
	write_to_buffer(0x00)

	#write file buffer to file
	fileWrite(t6, 0x10040008,0x10040000)
	addi t5, a0, 0
```

Figure 3: *Starter code illustrating the use of scaffolded macros for data manipulation at a designated memory location.*



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10040000	. . \0 .	\0 \0 \0 \0	* * *	\0 \n *	\0 \0 \0 \0	\0 \0
0x10040020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040060	\0 \0 \0 \0	\0 \0 \0 \0	[illegible]	\0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040080	\0 \0 \0 \0	\0 \0 \0 \0	[illegible]	\0 \0 \0	\0 \0 \0 \0	\0 \0
0x100400a0	\0 \0 \0 \0	\0 \0 \0 \0	[illegible]	\0 \0 \0	\0 \0 \0 \0	\0 \0
0x100400c0	\0 \0 \0 \0	\0 \0 \0 \0	[illegible]	\0 \0 \0	\0 \0 \0 \0	\0 \0
0x100400e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040120	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040140	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040160	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0
0x10040180	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0

Figure 4: *Visual depiction of data placement in the memory in the Data Segment of RARS after program execution.*

3.2 Developing a Paint Application

In the lab, students develop a simple "Paint" application on an emulated RISC-V system, akin to initial graphical projects in HLL courses. This application uses keyboard inputs to create Bitmap display patterns, with color addresses representing pixels. The task involves setting a starting pixel and using keyboard commands for drawing, reflecting basic HLL logic operations and control flows. This lab's structure is in line with notional machine principles as described in [9], sharing pedagogical purposes with lab 3.1. With regards to the Focus aspect, the lab's main goal is teaching assembly functions aligned with the RISC-V calling convention, emphasizing caller/callee saved registers. This parallels nested function calls in HLLs, fostering an understanding of data placement in memory—a key concept in both assembly and HLLs. Regarding representation, the lab provides clear function call examples, mirroring introductory HLL methods. It includes scaffolded code, such as a polling function in the RARS RISC-V emulator for user input, with comprehensive comments. This structure mirrors HLL environments, easing students' transition to these languages. Students engage more in understanding code relationships and less in writing extensive code, focusing on converting pixel coordinates to memory addresses and cursor movement based on keystrokes [Figure 5]. This method, central to our notional machine approach,gives students a perspective for future HLL courses.

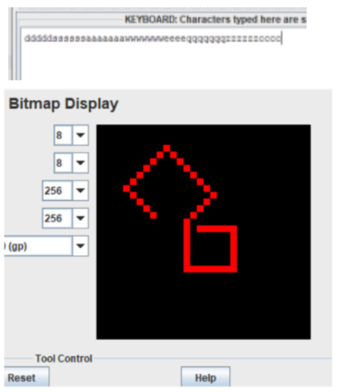
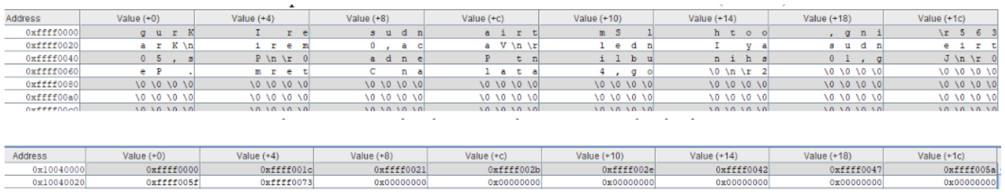

KEYBOARD: Characters typed here are s
dddddaaaaaaaaaaaaaawwwwwweeeeqqqqqqqzzzzzzzcccc
Bitmap Display
8
8
256
256
(gp)
Tool Control
Reset
Help

Figure 5: *Generating a pattern on the Bitmap Display through the virtual MMIO keyboard.*

3.3 CSV File Analysis

This lab involves processing a CSV file ('data.csv') that details annual returns from an investment portfolio, with columns for stock names and returns. In the RARS environment, students perform tasks like determining file size, calculating total income from stocks, and identifying stocks with highest and lowest returns. Similar to the Paint lab, the main educational goal is mastering the RISC-V function calling convention and handling CSV data in memory. This approach links assembly with higher-level language tasks, echoing foundational logic and control flows in HLLs and aligning with notional machine principles as in [9], with pedagogical aims akin to labs 3.1 and 3.2. The focus of this lab as NM is to teach assembly functions following RISC-V conventions and pointer referencing/dereferencing, crucial for transitioning to languages like C. Students analyze CSV file columns, using reference pointers for data alignment, linking assembly concepts with high-level programming paradigms and laying groundwork for advanced software engineering tasks.
In terms of NM representation, the lab is an interactive learning experience. Using resources like function call examples and provided macros to allocate file records, students learn data placement in memory. Visualizing data from the CSV file in RISC-V memory, with pointers referencing each entry [Figure 6], underscores the notional machine's focus on data transactions between registers and memory, enhancing understanding for high-level language concepts. This approach underpins the philosophy of using assembly education to bolster high-level language skills, mainly pointers.



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0xffff0000	g u r K	I r e	s u d n	a i r t	m S l	h t o o	, g n i	\r 5 6 3
0xffff0020	a r K \n	i r e m	0 , a c	a V \n \r	l e d n	I y a	s u d n	e i r t
0xffff0040	0 5 , s	P \n \r 0	a d n e	P t n	i l b u	n i h s	0 l , g	J \n \r 0
0xffff0060	e P .	m r e t	C n a	l a t a	4 , g o	\0 \n \r 2	\0 \0 \0 \0	\0 \0 \0 \0
0xffff0080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0xffff00a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0xffff00c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10040000	0xffff0000	0xffff001c	0xffff0021	0xffff002b	0xffff002e	0xffff0042	0xffff0047	0xffff005a
0x10040020	0xffff005f	0xffff0073	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figure 6: *(Top)Visual depiction of csv file data placement in the memory in the RARS Data Segment and (Bottom)pointer reference table to column data after program execution.*

4 Survey and Results from W23 and S23 quarters

To assess the impact of our notional machine approach in "Computer Systems and Organization" (CSE12), course exit surveys were initially undertaken at the end of W23 and S23 quarters.

W23 Quarter Feedback:

Statement 1: "A background in C/C++ would have enhanced my assembly language comprehension."
Agree: 82%
Disagree: 17%

Statement 2: "Assembly Labs provided a comprehensive view of programming, emphasizing the low-level essence of assembly."

Strongly/Agree: 70%
Somewhat Agree: 21%
Somewhat/Strongly Disagree: 5%

S23 Quarter Feedback:

Statement 1: (Similar to W23)
Agree: 75%
Disagree: 24%

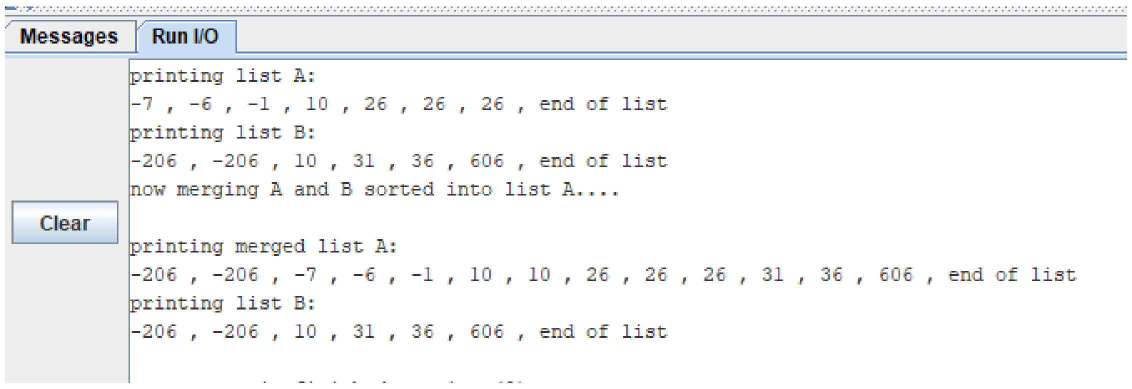
Statement 2: "Labs in CSE12 effectively embodied the notional machine, illustrating computation as guided data interplay between CPU and memory."
Strongly/Agree: 69%
Somewhat Agree: 25%
Somewhat/Strongly Disagree: 5%
Anecdotal Feedback:

Feedback from Student Experience of Teaching surveys was highly favorable. From W23, comments highlighted the pivotal nature of labs, their clear documentation, and the enrichment of learning. In S23, remarks accentuated the labs' facilitation of hands-on understanding and their relevance to real-world applications. Notably, students moving from CSE12 in W23 to Computer Architecture (CSE120) in S23 expressed better readiness and underscored the advantages of the RISC-V notional machine in helping them be better prepared for subsequent ISA and microarchitecture topics in CSE120.

5 Enhanced NM based Learning in F23 Quarter: Merging Sorted Linked Lists

Following the implementation of "Computer Systems and Organization" (CSE12) in the W23 and S23 quarters, the course was further refined and offered again in the F23 quarter. Building upon the previous experiences and feedback, this iteration included an enhanced evaluative framework to more rigorously assess the impact of our notional machine approach on student learning outcomes. This new Lab4 focused on the practical application of merging sorted linked lists in a RISC V assembly environment. The core objective was for students to gain proficiency in implementing functions in RISC V assembly, adhering to RISC-V register conventions, and understanding the architecture and operations of linked lists.

Students were tasked with writing code for three assembly files newNode.asm, print_list.asm, insertSorted.asm, and mergeLinkedLists.asm. These files, initially incomplete, were to be filled in and submitted. The lab involved creating two distinct linked lists, A and B, with specific sequences of keys, and then merging these lists using the mergeLinkedLists function (Figure 7). Scaffolded code was provided to students to help start their specific code contributions. The course TAs and tutors were provided solutions beforehand and instructed to assist students in arriving at the pseudocode solutions . If the student themself was unable to arrive at the pseudocode, it would be revealed and taught to them. The lab's educational outcome aimed at not just teaching how to write assembly code for functions but also at providing a deep understanding of linked lists within the Von Neumann architecture memory.



```
printing list A:
-7 , -6 , -1 , 10 , 26 , 26 , 26 , end of list
printing list B:
-206 , -206 , 10 , 31 , 36 , 606 , end of list
now merging A and B sorted into list A....

printing merged list A:
-206 , -206 , -7 , -6 , -1 , 10 , 10 , 26 , 26 , 26 , 31 , 36 , 606 , end of list
printing list B:
-206 , -206 , 10 , 31 , 36 , 606 , end of list
```

Figure 7: *Output of testbench assembly file when it was compiled and executed with correct student code to merge sorted linked lists.*

To effectively bridge the gap from arrays to linked lists, the lab introduced the concept of the Linked List by using an engaging and narrative-driven method utilizing cartoon characters (Figure 8) to facilitate students' transition from the static nature of arrays to the dynamic structure of linked lists. This narrative approach not only highlighted the adaptability, ease of modifications, and efficiency of linked lists but also aligned with the Notional Machine principles by simplifying complex concepts into relatable metaphors. Each cartoon character, hailing from a unique universe with a given key value and represented by a node in the linked list, served as a vivid illustration of the linked list's operational mechanics.

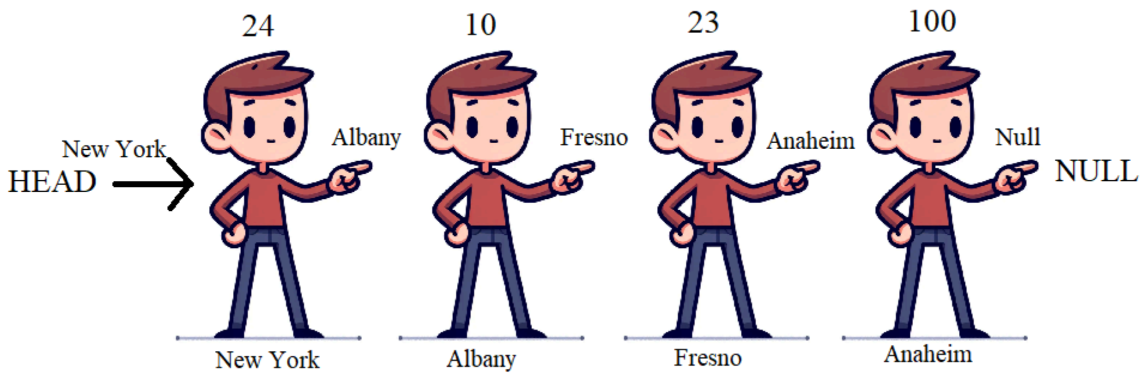


Figure 8: *Cartoon depiction of the Linked List with key pattern 24→10→23→100. The location of these keys in the visual metaphor are no longer actual memory locations but shown as actual cities inhabited by a cartoon with a specific key. A cartoon living in a city points to the city location of the next cartoon in the list.*

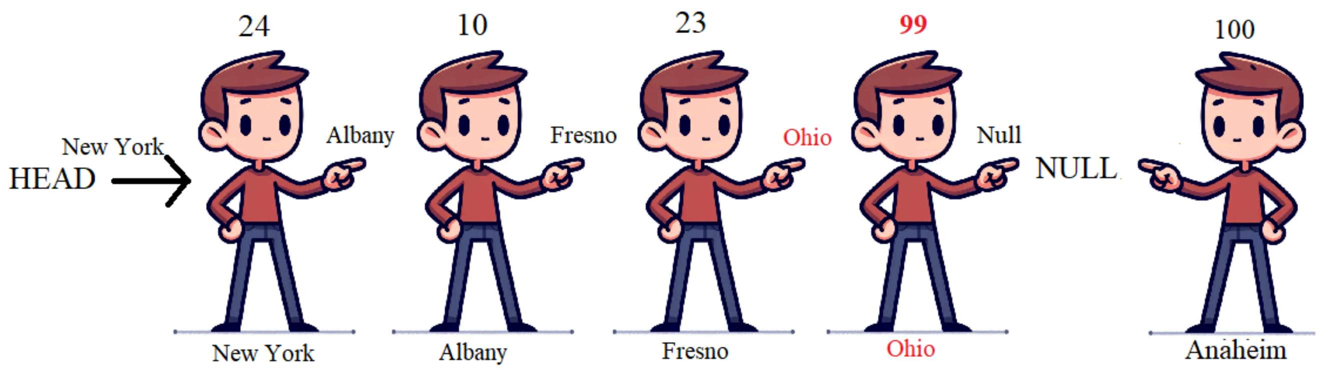


Figure 9: *Cartoon depiction of deletion of node from Linked List and a new node insertion. List 24→10→23→100 now reads as 24→10→23→99. Note that cartoon with key 100 is not shown as physically deleted but simply now existing as a "floating" node*

After familiarizing students with this cartoon precedent, the actual tangible implications of a Linked List residing in the heap memory section of a bare RISCV machine is demonstrated as a contrast. For Lab4, each node was composed of 8 bytes, 4 bytes to hold the key and the next 4 bytes to hold the next node memory address. HEAD node was the only node having 4 bytes, as shown in Figure 10.

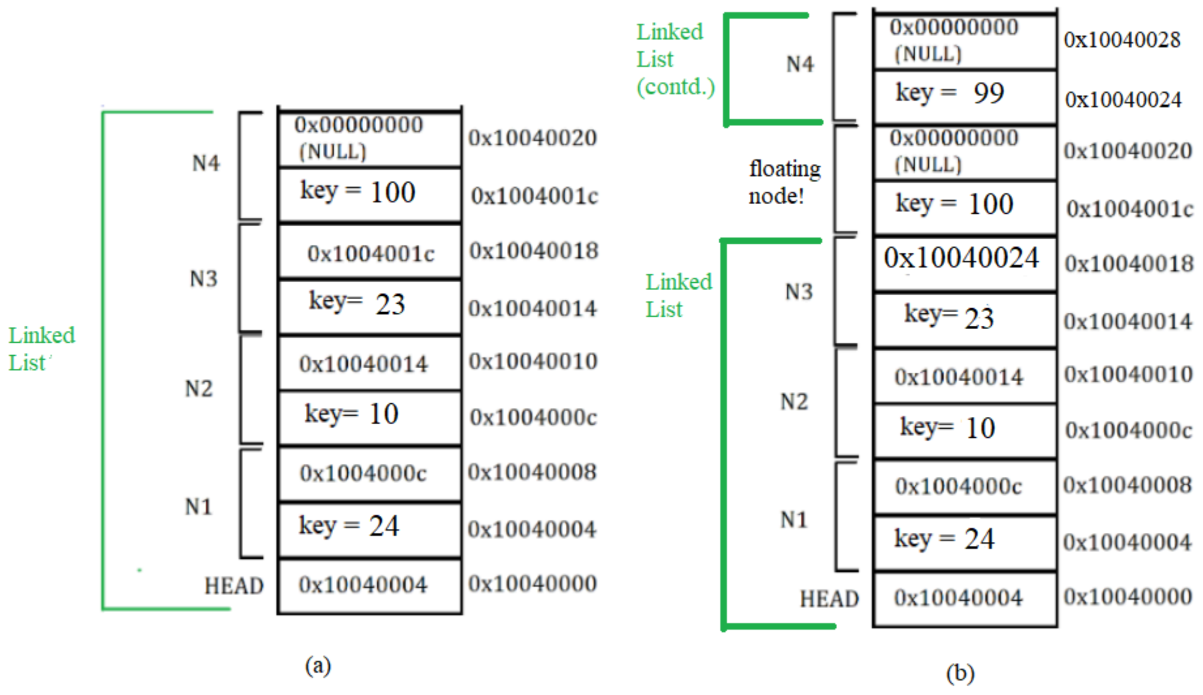


Figure 10: *(a) List 24→10→23→100 represented in RISC V heap memory. (b)List changed to 24→10→23→99 by adding a new node and adjusting the pointer reference of node N3(key=23).*

This lab's pedagogical purpose was multi-faceted: firstly, to reinforce understanding of the Von Neumann architecture and its implications for data structures within memory, and secondly, to contextualize the abstract concept of memory management in assembly language. By engaging with the cartoon narrative, students could grasp the practicalities of memory allocation and data linkage in a more tangible manner, resonating with the 'Representation' characteristic of the Notional Machine. Furthermore, the lab focused on the application of RISC-V function calling conventions and pointer operations, essential for understanding both low-level assembly language and high-level programming concepts. This focus catered to the 'Drawing Attention' aspect of the Notional Machine, guiding students to appreciate the intricacies of pointer referencing and dereferencing—a foundational skill in computer programming.

By integrating a hands-on programming task with an imaginative storyline, the lab sought to demystify the complexities of assembly language and data structures. This approach not only facilitated a deeper understanding of the material but also aimed to enhance students' readiness for more advanced topics in computer science, aligning with the overarching goals of the Notional Machine framework.

The lab's structure and narrative, therefore, served to make abstract concepts accessible, preparing students for future challenges in the realm of software development and high-level language comprehension. This method of instruction underscores the effective application of Notional Machine principles in teaching complex computational concepts through relatable and engaging contexts.

6 Methodological Framework for Pretest-Posttest Likert Scale Surveys in F23

To deepen our understanding of the course's effectiveness for F23, a pretest-posttest survey based on the Likert scale was conducted among 209 students. The pretest was administered before Lab 3, marking the commencement of assembly programming assignments, while the posttest followed the completion of Lab 4 on Linked Lists, the final lab assignment. This structure remained consistent with the blueprint established in the W23 and S23 quarters. The following section delves into the detailed survey process, detailing the specific pretest and posttest survey questions, and discussing the statistical methodologies used for analysis. A significant aspect of this analysis is the implementation of a paired t-test, which provides a more nuanced and statistically robust evaluation of the course's NM approach effectiveness as opposed to the initial surveys taken in W23 and S23. The findings from the F23 quarter offer insights into the evolving pedagogical impact of the course, enhancing understanding of how students interact with and benefit from the notional machine approach in assembly language learning. Both the pretest and posttest surveys consisted of a blend of Likert scale questions, designed to quantitatively measure students' perceptions and understanding, and open-ended questions aimed at capturing qualitative feedback. The 5 point Likert scale(1=Strongly Disagree to 5=Strongly Agree) portion included questions to gauge familiarity with the Von Neumann architecture, comfort with assembly language concepts, and expectations regarding the practical application of these concepts. The specific pretest and posttest Likert scale questions were as follows:

Pretest Survey Questions:

Pretest_Q1:I am familiar with the basic concepts of the Von Neumann architecture.
Pretest_Q2:I understand the role of memory, CPU, and registers in the Von Neumann model.
Pretest_Q3:I am comfortable with the concept of stored program concepts in computer systems.
Pretest_Q4:I have prior experience or knowledge of assembly language programming.
Pretest_Q5:I can see the relationship between high-level programming languages and assembly language.
Pretest_Q6:I believe understanding assembly language is crucial for a deeper understanding of computer systems.
Pretest_Q7:Learning assembly language will enhance my skills in higher-level programming languages.
Pretest_Q8:I think using a notional machine, like RARS, will aid in understanding the practical applications of the Von Neumann architecture.
Pretest_Q9:I am confident in my ability to apply theoretical concepts of computer architecture in practical assembly language tasks.

Posttest Survey Questions:

Posttest_Q1:My understanding of the Von Neumann architecture has improved after Pretest_Q1:I am familiar with the basic concepts of the Von Neumann architecture.the labs.
Posttest_Q2: I am now more comfortable with assembly language programming.

Posttest_Q3: The labs helped me in making connections between high-level programming languages and assembly language.
Posttest_Q4: I have a better understanding of how theoretical concepts of computer architecture apply in practical tasks.
Posttest_Q5: The use of RARS as a notional machine aided my understanding of the practical applications of the Von Neumann architecture.

To evaluate the effectiveness of the instructional approach and the labs themselves, a paired t-test was planned. The pairs were carefully chosen to match pre-lab expectations with post-lab outcomes, ensuring that each pretest question corresponded with a related posttest question, as detailed in the following pairing scheme:

Paired T-test Reference Scheme:

Pretest_Q1 ↔ Posttest_Q1 : test 1
Pretest_Q2 ↔ Posttest_Q4 : test 2
Pretest_Q3 ↔ Posttest_Q3 : test 3
Pretest_Q4 ↔ Posttest_Q2 : test 4

Pretest_Q5 ↔ Posttest_Q3 : test 5
Pretest_Q6 ↔ Posttest_Q5 : test 6
Pretest_Q7 ↔ Posttest_Q5 : test 7
Pretest_Q8 ↔ Posttest_Q5 : test 8
Pretest_Q9 ↔ Posttest_Q4 : test 9

The selection of paired questions for the pretest-posttest survey was intentional, aiming to directly compare specific concepts and perceptions before and after the lab assignments. This pairing strategy allows for a focused evaluation of students' growth in understanding and confidence related to the Von Neumann architecture and assembly language programming.

The use of the paired t-test in analyzing Likert scale data, despite some contention in statistical discourse, has been prevalent in research. Critics often point out that Likert scales produce ordinal data, which may not fulfill the interval data assumptions of the t-test. However, when Likert scales are used in a way that assumes equal intervals between response options, the t-test has been applied extensively. For instance, [12] discusses the use of the t-test in analyzing Likert scale data, highlighting its widespread acceptance and application despite the ongoing debate. It's acknowledged that the Wilcoxon signed-rank test is often recognized as more appropriate for Likert scale data due to its non-parametric nature, making fewer assumptions about the distribution of the data [13]. However, in this study, we opted for the paired t-test because we dealt with a dependent sample where the same respondents took both the pretest and the posttest. This dependency in the sample is a prerequisite for the paired t-test, which is designed to compare two related groups or conditions. Additionally, it should be noted that Likert scales can be interpreted as interval scales, especially when they are symmetric and balanced. This interpretation allows for mean values to be calculated and differences to be assessed through parametric tests like the paired t-test. Furthermore, the central limit theorem suggests that for large sample sizes, such as 209 student respondents in our case, the distribution of the sample means will approximate a normal distribution, thereby meeting one of the key assumptions of the paired t-test. In light of the above considerations, the decision to use the paired t-test was made with a thorough understanding of the sample characteristics and the research design, and with precedents in existing literature that validate such an approach.

In addition to the Likert scale questions, the survey included open-ended questions to provide insights into students' subjective experiences and nuanced understanding of the material. The open-ended questions from the pretest and posttest surveys sought to explore students' expectations, concerns, and perceived benefits or challenges associated with the labs. While the quantitative data derived from the Likert scale questions were the primary focus for the paired t-test analysis, the qualitative responses from the open-ended questions were also collected. The intention for the qualitative data is to inform future iterations of the course through a thematic analysis, aiming to identify common threads and unique perspectives that can enhance the lab experience. However, due to the extensive nature of the qualitative data, the analysis of these responses is acknowledged as part of the scope for future work.

7 Analysis of Paired T-Test Outcomes from F23

It is promising to witness the predominantly positive direction in student learning and understanding. This section presents a detailed examination of the results, offering a statistical perspective on the efficacy of our teaching methods.

Test 1: Von Neumann Architecture Understanding
T-value: -10.75
P-value: ~1.22e-21
Interpretation: A statistically significant improvement, suggesting enhanced student understanding post-intervention.

Test 2: Memory, CPU, Registers in Von Neumann Model
T-value: -6.49
P-value: ~6.31e-10
Interpretation: Significant positive impact, indicating improved student comprehension after the intervention.

Test 3: Stored Program Concept
T-value: -6.51
P-value: ~5.66e-10
Interpretation: Statistically significant results, pointing to better student understanding post-intervention.

Test 4: Assembly Language Programming

T-value: -17.80
P-value: ~1.80e-43
Interpretation: A highly significant improvement, reflecting substantial enhancement in students' skills.

Test 5: High-Level vs. Assembly Language
T-value: -1.34
P-value: ~0.183
Interpretation: No significant change, indicating an area for potential growth and exploration.

Test 6: Assembly Language Importance
T-value: 3.77
P-value: ~0.000214
Interpretation: Significant, but unexpected change in belief about assembly language understanding post-intervention.

Test 7: Enhancing Higher-Level Programming
T-value: 2.44
P-value: ~0.01546
Interpretation: Significant results, but in an unexpected direction, suggesting a mismatch between expectations and outcomes.

Test 8: Practical Application of Von Neumann Architecture
T-value: 0.80
P-value: ~0.424
Interpretation: Inconclusive results, highlighting a need for instructional strategy refinement.

Test 9: Applying Theoretical Concepts
T-value: -8.71
P-value: ~1.05e-15
Interpretation: A very significant improvement, indicating enhanced confidence in applying theoretical concepts in practical tasks.

These outcomes not only validate the effectiveness of certain aspects of our intervention but also illuminate areas requiring further attention and adjustment. The overall trajectory is promising, suggesting a positive impact on student learning, while also guiding us towards targeted improvements in our teaching methodologies.

8 Conclusion

Our exploration of the Notional Machine pedagogical approach in the CSE12 curriculum has progressed significantly since its inception in W23. Initially, as a pilot project, the feedback from the W23 and S23 quarters provided foundational insights into the preliminary impact of this initiative. These initial surveys indicated a promising direction, particularly in how students transitioned from CSE12 to subsequent courses like Computer Architecture (CSE120), suggesting a potential enhancement in their readiness for more advanced topics.

Building upon these initial insights, the results from the F2023 paired t-tests offer a more nuanced understanding of the approach's effectiveness. The significant improvements in key areas, as evidenced by the negative t-values in Tests 1, 2, 3, 4, and 9, underscore the positive impact of the intervention on students' comprehension of complex concepts like the Von Neumann architecture and assembly language programming. This aligns with earlier anecdotal feedback highlighting the labs' role in enriching learning and facilitating hands-on understanding.

However, the mixed results from Tests 5, 6, 7, and 8 highlight areas for further refinement. The lack of significant improvement in some areas, and the unexpected direction of change in others, suggest the need to re-evaluate our strategies, particularly in how we introduce and integrate the notional machine concepts across the curriculum.

The integration of Notional Machine-guided labs in CSE12, including new assignments on Linked Lists and Hash Maps, reflects our ongoing commitment to this pedagogical approach. As we plan to expand these labs to include more complex data structures, it is crucial to balance the depth and breadth of these concepts without overwhelming the students. While the direction in CSE12 has been promising and aligns with our initial observations, continuous refinement of the lab assignments and statistical tests, robust data collection, and feedback-driven iterations are essential. Our journey in enhancing the efficacy of the Notional Machines approach is ongoing, and we remain dedicated to adapting our methods to meet the diverse learning paths of our students and the evolving demands of computer science education.

REFERENCES

[1] Loui, Michael C. "The case for assembly language programming." IEEE Transactions on Education 31, no. 3 (1988): 160-164.

[2]Silvester, Peter P. "Introducing computer structure by machine simulation." IEEE Transactions on Education 33, no. 4 (1990): 326-332.

[3]Bolanakis, Dimosthenis E., Georgios A. Evangelakis, Euripidis Glavas, and Konstantinos T. Kotsis. "A teaching approach for bridging the gap between low‐level and high‐level programming using assembly language learning for small microcontrollers." Computer Applications in Engineering Education 19, no. 3 (2011): 525-537.

[4]K. Buckner, "A non-traditonal approach to an assembly language course." Journal of Computing Sciences in Colleges 22.1 (2006): 179-186.

[5]Imamura, Kosuke. "Assembly language is more than a teaching tool." Journal of Computing Sciences in Colleges 20, no. 2 (2004): 49-54.

[6]Little, R. Rainey, and Mark K. Smotherman. "Assembly language courses in transition." ACM SIGCSE Bulletin 20, no. 1 (1988): 95-99.

[7]Chmiel, Ryan, and Michael C. Loui. "Debugging: from novice to expert." ACM SIGCSE Bulletin 36, no. 1 (2004): 17-21.

[8]Adolph, Karen E., and Kari S. Kretch. "Gibson's theory of perceptual learning." International encyclopedia of the social and behavioral sciences 10 (2015): 127-134.

[9]Fincher, Sally, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict Du Boulay, Matthias Hauswirth, Arto Hellas et al. "Notional machines in computing education: The education of attention." In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, pp. 21-50. 2020.

[10]Hneemann. "Hneemann/Digital: A Digital Logic Designer and Circuit Simulator." GitHub. Accessed August 16, 2023. https://github.com/hneemann/Digital.

[11]TheThirdOne. "TheThirdOne/RARS: RARS -- RISC-V Assembler and Runtime Simulator." GitHub. Accessed August 16, 2023. https://github.com/TheThirdOne/rars.

[12]Mellor, David, and Kathleen A. Moore. "The use of Likert scales with children." Journal of pediatric psychology 39, no. 3 (2014): 369-379.

[13]Meek, Gary E., Ceyhun Ozgur, and Kenneth Dunning. "Comparison of the t vs. Wilcoxon signed-rank test for Likert scale data and small samples." Journal of modern applied statistical methods 6, no. 1 (2007): 10.

[14]Nath, Sagnik, Jennifer Quynn, and Jose Renau. "Developing the ITL framework and committing to inquiry as a method for reducing equity gaps in high-impact, computer science and engineering courses." 2023 ASEE Annual Conference & Exposition. 2023.