Understanding Learning Difficulties in Assembly Language among Mechanical Engineering Technology Students

Zairulazha Zainal[1]*, Shamsul Faisal Mohd Hussein[1], Md Fahmi Abd Samad @ Mahmood[1], Nor Salim Muhammad[1], Shamsul Anuar Shamsudin[1], Saleha Mohamad Saleh[2]

[1]Fakulti Teknologi dan Kejuruteraan Mekanikal, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

[2]Fakulti Teknologi dan Kejuruteraan Elektrik, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

***Corresponding Author**

DOI: https://doi.org/10.47772/IJRISS.2026.1026EDU0460

Received: 15 July 2026; Accepted: 20 July 2026; Published: 29 July 2026

ABSTRACT

Assembly language plays a fundamental role in microprocessor and embedded systems education by enabling students to understand how software instructions interact with processor registers, memory, status flags, and hardware-level operations. Despite its importance, the subject is often challenging for students outside computer science, particularly those in Mechanical Engineering Technology, because it requires reasoning at a much lower level of abstraction than is typical in engineering programming courses. This study examined the learning difficulties perceived by Mechanical Engineering Technology students when studying assembly language. A descriptive mixed-methods survey was administered to 30 undergraduate students enrolled in a microprocessor-related course. The instrument comprised Likert-scale items assessing perceived difficulty, confidence, and operational understanding, together with an open-ended question that invited students to describe the aspects of assembly language they found most difficult and explain their reasons. Descriptive findings indicated that students experienced the greatest difficulty in understanding data movement between registers and memory, followed by difficulties with addressing modes and the abstract nature of assembly instructions. Thematic analysis of the qualitative responses identified five recurring challenges: register-memory reasoning, addressing modes and data-source reasoning, execution tracing and prediction, low-level abstraction and visualisation, and confusion related to learning tools. These findings indicate that students' learning difficulties extend beyond syntax and memorisation, reflecting underdeveloped mental models of processor execution and a limited ability to visualise runtime behaviour. The study contributes empirical evidence from the context of engineering technology education and highlights the need for instructional approaches that explicitly support processor-state tracing, register-memory mapping, and progressive visualisation of addressing modes.

Keywords: assembly language, engineering technology education, learning difficulties, microprocessor education, STEM education

INTRODUCTION

Assembly language is widely included in microprocessor, embedded systems, and computer architecture courses because it establishes a direct relationship between software instructions and hardware operation. In contrast to high-level programming languages, assembly language requires learners to understand how individual instructions influence processor registers, memory locations, status flags, and program execution at a lower hardware level. For engineering technology students, this knowledge is particularly relevant as it forms a foundation for understanding embedded control systems, mechatronics applications, microcontrollers, automation, and hardware-related troubleshooting.

Nevertheless, learning assembly language is frequently perceived as challenging, particularly among students

whose primary discipline is not computer science. For instance, Mechanical Engineering Technology students may encounter assembly language as part of a microprocessor course, although their academic background is generally centred on mechanical systems, manufacturing processes, control applications, and practical engineering tasks. Consequently, many of these students may have limited prior experience with programming concepts, memory operations, binary data representation, and models of processor execution.

Research in programming education has consistently shown that novice programmers encounter difficulties in managing multiple aspects of learning simultaneously, including syntax, semantics, problem-solving approaches, and understanding how programs are executed [1], [2], [3], [4]. These challenges can become more pronounced in assembly language, where learners are required to reason about program execution at a detailed level. Students must track changes in registers, determine data locations in memory, interpret addressing mechanisms, identify changes in processor flags, and predict the outcome of each instruction. Therefore, learning assembly language involves more than code construction; it requires the development of an accurate internal representation of processor behaviour.

The concept of the "notional machine" provides a useful perspective for understanding this learning difficulty. Sorva [5] defined the notional machine as an abstract representation of a computer that learners construct to interpret program execution. Without a sufficiently developed notional machine, students may focus on the visible structure of code while failing to understand the underlying execution process. This challenge is particularly significant in assembly language because its notional machine closely reflects actual hardware operations. Unlike high-level programming languages, where many hardware details are abstracted away, assembly language exposes registers, memory, flags, and addressing modes as central components of the learning process.

Within Mechanical Engineering Technology education, this situation represents an important instructional consideration. Students are generally introduced to assembly language not with the intention of becoming professional assembly programmers, but to develop a deeper understanding of microprocessor-based systems applied in engineering contexts. Therefore, identifying the specific sources of learning difficulty is essential for instructors seeking to develop teaching approaches that better support students' understanding of low-level computational processes.

This study examines learning difficulties associated with assembly language among Mechanical Engineering Technology students. Rather than evaluating a specific simulator or instructional intervention, the study focuses on identifying the aspects of assembly language that students perceive as challenging and exploring the reasons underlying these difficulties. Accordingly, the study addresses the following research questions:

1. What aspects of assembly language do Mechanical Engineering Technology students perceive as most difficult?
2. How confident are students in reading assembly language code and predicting instruction execution?
3. What themes emerge from students' open-ended explanations regarding their assembly language learning difficulties?

The study contributes three important insights to engineering technology education. First, it provides empirical evidence from a Mechanical Engineering Technology student cohort, which remains relatively underrepresented in programming education research compared with computer science populations. Second, it identifies specific conceptual challenges in assembly language learning, particularly those related to register-memory reasoning, addressing modes, and execution tracing. Third, it offers practical implications for microprocessor educators seeking to design instructional approaches that better support non-computer science engineering students in developing an understanding of low-level program execution.

LITERATURE REVIEW

Difficulties in Novice Programming Learning

Learning to program is widely recognised as a challenging process for novice learners because they must acquire multiple forms of knowledge simultaneously. Du Boulay [1] proposed that beginners need to develop

an understanding of several domains, including the orientation of programming, the notional machine, notation, programming structures, and pragmatics. This perspective highlights that programming difficulties extend beyond learning syntax; students must also develop an understanding of how instructions are interpreted and executed by the underlying computational system.

A review by Robins, Rountree, and Rountree [2] highlighted the differences between novice and expert programmers in organising and applying programming knowledge. While experts are able to structure knowledge into meaningful conceptual units, novices often rely on surface-level features when interpreting programs. Similarly, Lahtinen, Ala-Mutka, and Järvinen [3] reported that novice programmers commonly experience difficulties in understanding programming concepts, designing solutions, and transferring knowledge to practical tasks. Lister et al. [4] further demonstrated that many novice programmers face challenges in reading and tracing program code, even after completing introductory programming courses.

These findings are particularly relevant to assembly language education because assembly programming places greater emphasis on execution tracing and understanding system states. Unlike high-level programming, where many hardware operations are abstracted away, assembly language requires learners to continuously reason about changes in registers, memory locations, and processor flags after each instruction. Therefore, assembly language represents a learning context in which programming requires not only code construction but also detailed reasoning about how computation unfolds at the machine level.

The Notional Machine and Mental Models of Execution

The concept of the notional machine provides an important theoretical perspective for explaining why assembly language can be difficult for novice learners. Sorva [5] proposed that students require a mental model of the computer system to interpret and predict program execution. Without a sufficiently developed notional machine, learners may memorise instruction patterns while lacking an understanding of the underlying runtime behaviour. This challenge is particularly evident in assembly language because students must reason directly about changes occurring at the processor level.

Research on program visualisation further supports the importance of making execution processes visible to learners. Sorva, Karavirta, and Malmi [6] reviewed program visualisation systems and found that visual representations can assist beginners in understanding runtime behaviour, although their effectiveness depends on appropriate instructional design and learner engagement. In assembly language learning, visualisation may help students observe changes in registers, memory, and flags. Nevertheless, visual representations alone may be insufficient if learners lack the conceptual knowledge required to interpret processor states and identify relevant execution patterns.

From a constructivist perspective, learning computer science involves constructing a meaningful understanding of computational processes rather than merely reproducing syntax or definitions [7]. Therefore, assembly language learning should be examined in terms of how students develop mental models of processor operation and connect individual instructions with system-level behaviour.

Previous studies have shown that programming difficulties are influenced by multiple interacting factors, including conceptual understanding, mental models, problem-solving strategies, programming environments, and prior knowledge [8], [9]. These findings suggest that assembly language learning difficulties should not be viewed solely as problems of syntax or memorisation, but as a combination of conceptual, procedural, and contextual challenges.

Assembly Language as Troublesome Knowledge

Assembly language can also be examined through the perspective of threshold concepts and troublesome knowledge. Meyer and Land [10], [11] described threshold concepts as ideas that are transformative but often difficult to acquire because they require learners to reorganise their existing ways of thinking. In computer science education, threshold concepts have been discussed in relation to areas such as abstraction, object-oriented programming, pointers, and memory-related reasoning [12], [13], [14]. Recent research has further

suggested that understanding the role of main memory in program execution may represent a threshold concept in computer science learning [15].

Within assembly language learning, several concepts may represent troublesome knowledge for students without a strong computer science background. For example, addressing modes require learners to differentiate between values, addresses, registers, and memory locations. Indirect addressing can be particularly challenging because students must understand that an instruction may refer to an address that subsequently points to another location. Similarly, register-memory interaction requires learners to reason about where data is stored and how it moves through the processor rather than simply interpreting instructions as sequential commands.

These concepts may become troublesome because they involve low-level and invisible processes that are not directly observable. A small change in an instruction, register state, or memory address may produce a different execution outcome. For Mechanical Engineering Technology students, this challenge may be amplified because assembly language requires reasoning about abstract processor activities rather than the physical and graphical representations commonly encountered in mechanical engineering domains. Instead of observing motion, force, or mechanical components directly, students must construct mental representations of invisible changes occurring within the processor.

Cognitive Load in Low-Level Programming

Cognitive Load Theory provides a useful explanation for why assembly language may be particularly challenging for students with limited prior programming experience. Sweller [16] proposed that learning is constrained by the capacity of working memory, especially when learners attempt to solve problems without sufficiently developed schemas. Paas, Renkl, and Sweller [17] further emphasised that effective instructional design should consider both task complexity and learner expertise. In assembly language learning, novices are required to coordinate multiple interacting elements, including instruction syntax, register operations, memory addresses, processor flags, data movement, and execution sequences. Without appropriate schemas to organise these elements, learners may experience high intrinsic cognitive load during problem solving.

The cognitive demands of assembly language are further influenced by the need to understand processes that are not directly observable. Mayer's cognitive theory of multimedia learning [18] suggests that learning can be supported when verbal and visual representations are meaningfully integrated. This perspective is relevant to assembly language because processor states, memory changes, and instruction effects are largely invisible to learners. However, visual representations or simulations must be carefully designed because presenting excessive processor information simultaneously may increase rather than reduce cognitive load. Therefore, instructional support should guide learners' attention toward specific execution changes, such as identifying which register is modified, the source and destination of data movement, and the effect of an instruction on processor flags.

METHODOLOGY

Study Design

This study employed a descriptive mixed-methods research design to examine the learning difficulties experienced by Mechanical Engineering Technology students in learning assembly language. The quantitative component was used to identify students' perceived level of difficulty, confidence, and operational understanding through Likert-type items. The qualitative component was used to explore students' own explanations of the most difficult aspects of assembly language learning.

A mixed-methods approach was considered appropriate because students' learning difficulties in assembly language may not be fully captured through numerical responses alone. While Likert-type items can indicate the general level of perceived difficulty, open-ended responses provide deeper insight into the specific concepts or learning processes that students find challenging. Therefore, the study combined descriptive statistical analysis with thematic analysis to provide a more complete understanding of students' difficulties.

The study was exploratory and descriptive in nature. It did not aim to test the effectiveness of a specific teaching intervention or simulator. Instead, the main purpose was to identify and describe the learning difficulties that emerged from students' perceptions and written reflections.

Participants

The participants were 30 undergraduate students enrolled in a Mechanical Engineering Technology programme and taking a course that included microprocessor and assembly language content. The majority were Semester 3 students (29 students, 96.7%), while one participant was from Semester 4 (3.3%). Table 1 presents the demographic profile of the respondents.

Table 1. Respondent Profile

Category	Frequency	Percentage
Mechanical Engineering Technology	30	100.0%
Semester 3	29	96.7%
Semester 4	1	3.3%
No previous programming experience	22	73.3%
Basic C / C++ experience	5	16.7%
Learned assembly before	2	6.7%
Java experience	1	3.3%

The participant profile indicates that most students entered the course with limited prior programming experience. This characteristic is relevant because assembly language requires learners to reason about low-level execution processes involving registers, memory locations, addressing modes, and processor flags. For novice learners, limited programming experience may increase the difficulty of constructing appropriate mental models for interpreting processor-level operations.

Survey Instrument

The survey instrument consisted of three sections. The first section collected demographic and academic background information, including programme of study, semester of study, and previous programming experience. The second section comprised eight Likert-type items measured using a five-point Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree). The third section included one open-ended question that asked students to identify the most difficult aspect of assembly language learning and explain the reasons for their difficulty.

The Likert-type items were organised into two conceptual groups. The first group assessed students' perceived learning difficulties related to assembly language concepts, including register-memory interaction, abstraction, execution tracing, and addressing modes:

1. I find assembly language difficult to understand.
2. I struggle to understand how data moves between registers and memory.
3. Assembly language instructions are too abstract for me.
4. I find it difficult to follow the step-by-step execution of assembly programs.

5. Understanding addressing modes in assembly language is challenging.

The second group assessed students' confidence and perceived understanding of assembly instruction execution:

1. I feel confident when reading assembly language code.
2. I understand how an assembly instruction affects registers and flags.
3. I can predict what will happen after an instruction is executed.

The open-ended question was:

"*Which part of assembly language do you find most difficult to understand? Why?*"

The open-ended question was included to capture students' explanations of their learning difficulties in their own words. This qualitative component was important because students' challenges may involve specific reasoning processes or conceptual difficulties that are not fully represented through predetermined response options.

Data Preparation

Before analysis, the dataset was examined for completeness and relevance to the study objectives. The final dataset consisted of 30 valid responses from Mechanical Engineering Technology students. The Likert-type responses were converted into numerical codes as shown in Table 2.

Table 2. Coding of Likert-Type Responses

Response	**Code**
Strongly Disagree	1
Disagree	2
Neutral	3
Agree	4
Strongly Agree	5

Descriptive statistics, including response frequencies, percentages, and mean scores, were used to summarise the Likert-type responses. The five perceived difficulty items and three confidence-related items were analysed separately because they represented different dimensions of students' perceptions. The difficulty items reflected perceived challenges in learning assembly language, while the confidence-related items reflected students' perceived ability to read assembly code, understand instruction effects, and predict execution behaviour.

The open-ended responses were reviewed before thematic analysis. Non-substantive responses, including entries such as "-", "/", ".", and "not sure", were separated from meaningful responses. Responses that provided limited detail, such as "all of it", were retained and coded as general or non-specific difficulty. Responses related mainly to simulator usage were not analysed as evidence of simulator effectiveness; instead, they were classified under tool-related confusion because the focus of the study was students' learning difficulties in assembly language.

For thematic analysis, meaningful responses were reviewed to identify recurring ideas and grouped into broader categories representing common learning difficulties.

Reliability of the Likert-Type Items

Internal consistency of the Likert-type items was examined separately for the two item groups using Cronbach's alpha, which is commonly used to assess the consistency of items within a measurement set [19]. The five items representing perceived learning difficulty demonstrated good internal consistency, with a Cronbach's alpha value of 0.89. The three items representing confidence and perceived understanding of instruction execution also demonstrated acceptable internal consistency, with a Cronbach's alpha value of 0.83.

These results indicate that the items within each group showed consistent response patterns within the context of this exploratory study. However, the reliability values were interpreted cautiously due to the relatively small sample size. Therefore, the results were used to support descriptive interpretation of students' perceptions rather than to establish generalisable measurements beyond the study context.

Quantitative Data Analysis

The quantitative data were analysed using descriptive statistics. Frequencies, percentages, mean scores, and standard deviations were calculated for each Likert-type item. Mean scores were used to identify higher ratings of perceived difficulty or confidence, while response frequencies and percentages were examined to describe overall response patterns. The analysis focused on identifying the aspects of assembly language that students perceived as most challenging.

Likert-type responses were interpreted cautiously as indicators of students' perceptions rather than precise interval-level measurements [20]. No inferential statistical tests were conducted because the sample size was limited and the study was not designed to compare groups or examine causal relationships. Instead, the quantitative analysis aimed to describe patterns in students' perceived difficulties and confidence related to assembly language learning.

Qualitative Data Analysis

The open-ended responses were analysed using thematic analysis [21]. The analysis followed an iterative process involving familiarisation with the responses, identification of meaningful units, coding of reported difficulties, development of broader themes, and refinement of themes based on the research questions.

Initially, all responses were read repeatedly to obtain an overall understanding of students' reported difficulties. Non-substantive responses were separated from meaningful responses before coding. Meaningful responses were then coded according to the specific difficulty described by students, and related codes were grouped into broader themes. The themes were subsequently reviewed to ensure consistency and alignment with the focus of the study.

As a single response could describe multiple learning difficulties, overlapping coding was permitted. For example, a response mentioning registers, memory addressing, and instruction execution flow was assigned to more than one theme. This approach was considered appropriate because difficulties in assembly language often involve interconnected concepts rather than isolated problems. The coding process was guided by the codebook shown in Table 3.

Table 3. Qualitative Codebook for Open-Ended Responses

Code / Theme	Description	Inclusion Criteria	Example of Student Response
Register-memory reasoning	Difficulty understanding registers, memory, flags, and data movement	Responses mentioning registers, memory, flags, or movement of data between registers and memory	"Understanding registers, memory, and flags is difficult because many things happen at the same time..."

Addressing modes and data-source reasoning	Difficulty understanding addressing modes or identifying where data comes from	Responses mentioning addressing modes, indirect addressing, pointers, offsets, memory addressing, or address changes	"Addressing modes because it is confusing to know where the data comes from."
Execution tracing and prediction	Difficulty following what happens step by step during program execution	Responses mentioning execution flow, predicting outcomes, or following instruction effects	"How registers, memory addressing, and instruction execution flow work together."
Low-level abstraction and visualisation	Difficulty visualising invisible processor operations or understanding low-level hardware behaviour	Responses mentioning low-level concepts, hardware registers, machine language, or difficulty visualising internal processor processes	"It is hard to visualize what is happening inside the processor."
Tool-related confusion	Difficulty related mainly to assembly simulation, software interface, or interpretation of simulation output	Responses mentioning Sim8085, assembly simulation, or difficulty interpreting register, memory, flag, or program-output displays. Responses referring to software or activities outside the assembly-language component were treated as peripheral and were not included in the interpretation of assembly-language learning difficulties.	"Sim 8085 because the simulation is a bit complicated..."
General or non-specific difficulty	Broad statement that assembly language is difficult without identifying a specific concept	Responses such as "all of it" or "every part is hard"	"I think every part is hard..."
Study habit / preparation issue	Difficulty attributed to lack of revision or preparation	Responses mentioning revision, practice, or preparation rather than a specific concept	"For now don't have just lack of revision."
Non-substantive / no clear difficulty stated	Responses that do not provide analysable information	Blank symbols, "not sure", "none", or unclear entries	"-", "/", ".", "not sure"

This coding approach helped distinguish between conceptual difficulties in assembly language and responses that reflected tool issues, lack of revision, or insufficient information.

Ethical Considerations

Participation in the survey was voluntary. Students were informed that participation was optional, and the survey form stated that responses would be used solely for research purposes. The collected data were analysed and presented in aggregate form. No individual students were identified, and qualitative excerpts were reported anonymously. The study was considered to involve minimal risk because it focused only on students' perceptions of learning difficulties within a classroom-based educational context.

RESULTS

Descriptive Findings from Likert-Type Items

Table 4 presents the descriptive results for the eight Likert-type items. The items were analysed based on response distribution, mean scores, and standard deviations (SD) to identify aspects of assembly language that students perceived as more challenging.

Table 4. Descriptive Results of Students' Perceptions

Item	Disagree	Neutral	Agree	Strongly Agree	Mean	SD
I find assembly language difficult to understand.	3.3%	56.7%	36.7%	3.3%	3.40	0.62
I struggle to understand how data moves between registers and memory.	0.0%	53.3%	43.3%	3.3%	3.50	0.57
Assembly language instructions are too abstract for me.	3.3%	53.3%	40.0%	3.3%	3.43	0.63
I find it difficult to follow the step-by-step execution of assembly programs.	13.3%	53.3%	30.0%	3.3%	3.23	0.73
Understanding addressing modes in assembly language is challenging.	3.3%	53.3%	40.0%	3.3%	3.43	0.63
I feel confident when reading assembly language code.	6.7%	66.7%	23.3%	3.3%	3.23	0.63
I understand how an assembly instruction affects registers and flags.	0.0%	73.3%	23.3%	3.3%	3.30	0.53
I can predict what will happen after an instruction is executed.	20.0%	53.3%	23.3%	3.3%	3.10	0.76

Note: No respondent selected Strongly Disagree for any item.

The highest mean score was recorded for the item "I struggle to understand how data moves between registers and memory" (M = 3.50, SD = 0.57). This indicates that register-memory interaction represented the highest perceived difficulty among the students. The next highest difficulty ratings were observed for "Assembly language instructions are too abstract for me" and "Understanding addressing modes in assembly language is challenging" (both M = 3.43, SD = 0.63).

Among the confidence-related items, the lowest mean score was observed for "I can predict what will happen after an instruction is executed" (M = 3.10, SD = 0.76). This suggests that students reported lower confidence and perceived understanding when predicting instruction outcomes compared with reading assembly code or understanding the effects of instructions on registers and flags.

A notable pattern in the responses was the high proportion of neutral selections across most items. This pattern should not be interpreted simply as the absence of difficulty. In the context of learning abstract technical concepts, neutral responses may reflect uncertainty, partial understanding, or difficulty in evaluating one's own

competence. Therefore, these responses warrant further qualitative examination of students' explanations regarding their assembly language learning difficulties.

Themes from Open-Ended Responses

The open-ended responses provided deeper insight into the specific nature of students' reported difficulties. The responses were coded using the qualitative codebook described in the methodology section. Since individual responses could contain more than one difficulty, overlapping coding was permitted. Therefore, the total number of theme mentions does not equal the number of participants. Table 5 summarises the main themes identified from the meaningful open-ended responses.

Table 5. Themes Identified from Open-Ended Responses

Theme	Number of Mentions	Explanation
Register-memory reasoning	8	Students struggled to understand registers, memory, flags, and data movement.
Addressing modes and data-source reasoning	8	Students found it difficult to identify where data comes from and how addressing modes work.
Low-level abstraction and visualisation	6	Students found it hard to visualise invisible processor operations.
Execution tracing and prediction	4	Students struggled to follow the step-by-step effect of instructions.
Tool-related confusion	3	Some students referred to simulator or software-related difficulty.
General or non-specific difficulty	2	Some students stated that all parts were difficult without specifying a concept.
Study habit / preparation issue	1	One response attributed the difficulty to lack of revision.
Non-substantive / no clear difficulty stated	10	Some responses did not provide enough information for thematic interpretation.

The most frequently identified learning difficulty themes were register-memory reasoning and addressing modes and data-source reasoning. This finding was consistent with the quantitative results, in which the highest difficulty rating was related to understanding how data moves between registers and memory. The convergence between quantitative and qualitative findings suggests that understanding processor-level data movement was a central challenge in assembly language learning among the students.

Responses classified as general difficulty, study habit/preparation issues, or non-substantive entries were not considered primary learning difficulty themes but were retained to provide a complete representation of students' responses.

Register-Memory Reasoning

Register-memory reasoning was one of the most frequently identified themes in the data. This finding was consistent with the quantitative results, where difficulty in understanding data movement between registers and memory received the highest perceived difficulty rating ($M = 3.50$, $SD = 0.57$). Students repeatedly mentioned registers, memory, flags, and data movement as challenging aspects of assembly language learning. One

student explained that understanding "registers, memory, and flags" was difficult because many processes occur simultaneously. Another student stated that the most difficult part was understanding how "registers, memory addressing, and instruction execution flow work together."

These responses indicate that students were not only struggling with individual terms or instruction names. Instead, the difficulty involved understanding the relationships between different components of processor operation. Students needed to determine where data was stored, how data moved between locations, and what changes occurred after an instruction was executed. This type of reasoning represents a central aspect of assembly language learning and may introduce unfamiliar forms of low-level computational reasoning for students with limited prior exposure to processor-level concepts.

Addressing Modes and Data-Source Reasoning

Addressing modes and data-source reasoning emerged as one of the major difficulty themes in the open-ended responses. This finding was consistent with the quantitative results, where understanding addressing modes was among the highest-rated perceived difficulties ($M = 3.43$, $SD = 0.63$). Students referred to indirect addressing, pointers, offsets, memory addressing, and the difficulty of identifying where data originates. One student stated that addressing modes were difficult because it was confusing to determine where the data came from. Another student explained that indirect addressing was challenging because the concept of an "address of an address" was difficult to visualise.

The responses indicate that students struggled with distinguishing between data values and the locations where data is stored. In assembly language, this distinction is essential because instructions may operate using immediate values, register contents, direct memory locations, or indirect memory references. For novice learners with limited exposure to low-level programming concepts, the relationship between a value and its address may not be immediately intuitive.

Execution Tracing and Prediction

Execution tracing and prediction emerged as an additional difficulty theme in the open-ended responses. Some students reported difficulty understanding how registers, memory addressing, and instruction execution flow interact during program execution. This finding was consistent with the quantitative results, where the item "I can predict what will happen after an instruction is executed" recorded the lowest mean score among the confidence-related items ($M = 3.10$, $SD = 0.76$).

The responses suggest that students may be able to recognise assembly instructions at a surface level but still experience difficulty mentally simulating the changes that occur during execution. In assembly language learning, understanding requires more than reading instruction sequences; students need to track processor states before and after each instruction, including register values, memory contents, and flag changes. This process may place increased demands on working memory and conceptual understanding, particularly for learners who are developing their understanding of low-level processor behaviour.

Low-Level Abstraction and Visualisation

Low-level abstraction and visualisation emerged as another theme in students' reported difficulties. Six responses indicated that assembly language was challenging because processor-level operations were difficult to visualise. One student stated that it was difficult to imagine what was happening inside the processor, while another response referred to the direct manipulation of hardware registers.

These responses indicate that the difficulty was not only related to learning instructions, but also to forming an understanding of invisible processor operations. Assembly language requires learners to reason about internal changes involving registers, memory, and execution states that cannot be directly observed. For Mechanical Engineering Technology students who are more accustomed to physical systems, components, and observable mechanical behaviour, this form of low-level computational reasoning may represent a different type of learning challenge.

Tool-Related Confusion

Three responses referred to software-related difficulties, mainly involving the use of Sim8085 or the interpretation of assembly simulation output. Since the focus of this study was learning difficulty in assembly language rather than simulator effectiveness, these responses were treated as a secondary theme. Responses related primarily to software operation or activities outside the assembly-language component were considered peripheral and were not used as direct evidence of conceptual difficulty in assembly language learning.

Nevertheless, the presence of tool-related responses indicates that simulator-based learning may introduce additional challenges when students are required to interpret processor representations such as registers, memory, flags, and program output. These responses highlight the need to distinguish between difficulties caused by assembly language concepts and difficulties associated with interpreting learning tools.

DISCUSSION

The findings suggest that Mechanical Engineering Technology students experience assembly language difficulty primarily as a challenge in reasoning about processor states. The highest quantitative difficulty rating was related to understanding how data moves between registers and memory. This finding was supported by the qualitative responses, in which students repeatedly referred to registers, memory, addressing modes, flags, and instruction execution flow as challenging aspects of assembly language learning.

These findings suggest that students' difficulties extend beyond memorising assembly instructions. Instead, the central challenge involves understanding how instructions modify the internal state of the processor. Developing a functional understanding of processor behaviour may therefore be essential for successful assembly language learning. Without such understanding, students may recognise instruction names and syntax but still experience difficulty explaining what occurs during execution.

Addressing modes appear to represent a key conceptual difficulty. Students reported challenges in identifying where data originates, particularly when indirect addressing, pointers, offsets, or memory locations were involved. This suggests that students require a clear distinction between values, addresses, registers, and memory contents. Without this distinction, tracing even simple assembly programs may remain challenging.

The results also highlight the prediction of execution outcomes as an important area of difficulty. The item "I can predict what will happen after an instruction is executed" recorded the lowest mean among the confidence-related items. This is educationally significant because the ability to predict instruction outcomes reflects a deeper understanding of processor behaviour than merely recognising instruction syntax. Students who develop a stronger understanding should be better able to anticipate changes in registers, flags, and memory locations after instruction execution.

Another notable finding was the high proportion of neutral responses across the Likert-type items. These responses may indicate uncertainty or incomplete confidence in students' own understanding. One possible explanation is that students may possess partial understanding when concepts are explained but remain uncertain when required to apply the concepts independently. Therefore, neutral responses may represent an early indication of unstable understanding rather than an absence of difficulty.

Interpretation of the Findings

The findings suggest that students' difficulties in assembly language can be understood as challenges in processor-state reasoning. Rather than viewing assembly language difficulty only as a matter of syntax, memorisation, or unfamiliar instruction names, the results indicate that students struggled to understand how data is stored, moved, accessed, and modified during program execution.

Four related patterns of reasoning difficulty were identified in the findings. First, students experienced difficulty with data-location reasoning, which involves understanding where data is stored and how it moves between registers and memory. Second, students experienced challenges in address-source reasoning, which

involves identifying whether data originates from an immediate value, a register, a direct memory location, or an indirectly addressed location. Third, execution-state reasoning represented another challenge because students needed to predict how instructions would modify registers, memory contents, and flags. Fourth, low-level visualisation required students to imagine processor operations that cannot be directly observed.

This interpretation helps explain why assembly language may be challenging for Mechanical Engineering Technology students. Students from engineering technology backgrounds may often encounter physical, graphical, or system-level representations in their disciplinary learning, whereas assembly language requires reasoning about invisible processor-level operations through symbolic instructions. Therefore, the difficulty should not be attributed simply to the complexity of assembly language, but to the need to coordinate multiple interacting processor-level concepts during execution.

These insights provide a useful basis for developing more supportive assembly language instruction. The findings suggest that instruction should extend beyond introducing instruction syntax and include activities that help students develop clearer representations of registers, memory, addressing modes, flags, and step-by-step execution processes.

Instructional Implications

The findings suggest several instructional implications for microprocessor and assembly language teaching. First, assembly instructions may be taught as processor-state changes rather than as syntax alone. For each instruction, students can be guided to consider questions such as: What is the source of the data? What is the destination? Which register changes? Does memory content change? Are any flags affected? This approach may help students develop a clearer understanding of instruction behaviour during execution.

Second, addressing modes may benefit from progressive introduction. Before learning multiple addressing modes, students should develop a clear understanding of the distinction between a value and an address. Immediate, register, direct, and indirect addressing can then be introduced using simple diagrams and memory maps to support understanding of data location and movement.

Third, the regular use of trace tables may support students in following program execution. By recording register values, memory contents, and flag changes after each instruction, trace tables can provide an external representation of processor-state changes. This may reduce the mental effort required to simulate execution and help students visualise otherwise invisible processes.

Fourth, simulation tools may be more effective when combined with structured observation tasks. Instead of only asking students to execute a program, learning activities can include prompts that ask students to observe changes in specific registers, identify modified memory locations, or explain the addressing mode used. Such guidance may help students interpret simulator output rather than using the simulator only as a program execution environment.

Finally, assembly language examples may be contextualised within engineering technology applications. For Mechanical Engineering Technology students, examples related to sensors, actuators, motor control, automation, and embedded systems may help connect low-level programming concepts with familiar engineering applications.

Limitations

This study has several limitations. First, the sample size was relatively small and limited to one Mechanical Engineering Technology cohort. Therefore, the findings should be interpreted within the study context and should not be generalised to all engineering technology students. Second, the study relied on students' self-reported perceptions, and their actual assembly language performance was not measured. Therefore, the findings represent perceived learning difficulties rather than objectively assessed competence. Third, some open-ended responses were brief or non-substantive, which limited the extent of qualitative interpretation. Fourth, the study did not evaluate the effectiveness of a specific teaching strategy or simulation tool.

Despite these limitations, the study provides useful exploratory evidence for understanding assembly language learning difficulties in engineering technology education. Future studies may involve larger and more diverse samples, diagnostic assessments, interviews, classroom observations, or pre- and post-intervention evaluations of targeted teaching strategies.

CONCLUSION

This study investigated learning difficulties in assembly language among Mechanical Engineering Technology students. The findings indicate that students' main difficulty is not simply remembering assembly instructions, but understanding how instructions modify processor state, particularly registers, memory, addressing modes, flags, and step-by-step execution behaviour.

The quantitative findings showed that the strongest perceived difficulty was related to understanding how data moves between registers and memory. Addressing modes and the abstract nature of assembly instructions were also identified as important areas of difficulty. These findings were supported by the qualitative analysis, which revealed recurring challenges related to register-memory reasoning, addressing modes, execution tracing, low-level abstraction, and interpretation of simulation-related information.

The study identified several conceptual challenges in assembly language learning among Mechanical Engineering Technology students. The findings suggest that assembly language instruction should make processor execution more visible, explicit, and traceable. Instruction should therefore extend beyond teaching instruction syntax and support students in developing a functional mental model of processor operation. For engineering technology students, this approach may help them better connect assembly language concepts with microprocessor behaviour and embedded engineering applications.

ACKNOWLEDGEMENT

The authors thank Universiti Teknikal Malaysia Melaka for its continuous support and encouragement in promoting scholarly publication, including this article.

REFERENCES

1. Du Boulay, B. (1986). Some difficulties of learning to program. Journal of Educational Computing Research, 2(1), 57-73.
2. Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2), 137-172.
3. Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 14-18.
4. Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. In Working group reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '04) (pp. 119-150). Association for Computing Machinery.
5. Sorva, J. (2013). Notional machines and introductory programming education. ACM Transactions on Computing Education, 13(2), Article 8, 1-31.
6. Sorva, J., Karavirta, V., & Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. ACM Transactions on Computing Education, 13(4), Article 15, 1-64.
7. Ben-Ari, M. (2001). Constructivism in Computer Science Education. Journal of Computers in Mathematics and Science Teaching, 20(1), 45-73.
8. Qian, Y., & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. ACM Transactions on Computing Education, 18(1), Article 1, 1-24.

9. Cheah, C. S. (2020). Factors Contributing to the Difficulties in Teaching and Learning of Computer Programming: A Literature Review. Contemporary Educational Technology, 12(2), ep272.
10. Meyer, J. H. F., & Land, R. (2003). Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. In Improving Student Learning: Theory and Practice Ten Years On (pp. 412-424). Oxford Brookes University.
11. Meyer, J. H. F., & Land, R. (2005). Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. Higher Education, 49, 373-388.
12. Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2007). Threshold concepts in computer science: Do they exist and are they useful? ACM SIGCSE Bulletin, 39(1), 504-508.
13. Rountree, J., & Rountree, N. (2009). Issues regarding threshold concepts in computer science. In Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95 (ACE '09) (pp. 139-146). Australian Computer Society.
14. Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2006). Putting threshold concepts into context in computer science education. In Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (pp. 103–107). Association for Computing Machinery.
15. Kyriakou, C., Gogoulou, A., & Grigoriadou, M. (2025). Main memory in program execution: Threshold concept in CS. SN Computer Science, 6, Article 464.
16. Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. Cognitive Science, 12(2), 257-285.
17. Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive Load Theory and Instructional Design: Recent Developments. Educational Psychologist, 38(1), 1-4.
18. Mayer, R. E. (2005). Cognitive Theory of Multimedia Learning. In The Cambridge Handbook of Multimedia Learning (pp. 31-48). Cambridge University Press.
19. Tavakol, M., & Dennick, R. (2011). Making sense of Cronbach's alpha. International Journal of Medical Education, 2, 53-55.
20. Sullivan, G. M., & Artino, A. R. Jr. (2013). Analyzing and Interpreting Data From Likert-Type Scales. Journal of Graduate Medical Education, 5(4), 541-542.
21. Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. Qualitative Research in Psychology, 3(2), 77-101.