ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ

Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри_____ Олена ОЛЬХОВСЬКА
(підпис)

«___»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ НАВЧАЛЬНОГО ІНСТРУМЕНТУ З ТЕМИ «АРИФМЕТИЧНІ КОМАНДИ МОВИ ASSEMBLER» ДИСЦИПЛІНИ «АРХІТЕКТУРА ОБЧИСЛЮВАЛЬНИХ СИСТЕМ»

зі спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерні науки»

ступеня магістр

Виконавець роботи Авраменко Євгеній Ігорович
_____ «___»_____2023 р.
(підпис)

Науковий керівник зав. каф., к.ф.-м.н. Ольховська Олена Володимирівна
_____ «___»_____2023 р.
(підпис)

Рецензент _____

ПОЛТАВА 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
Assembler	мова програмування низького рівня для програмованої обчислювальної системи, в якій існує сувора відповідність між операторами мови та машинними командами
JSX	розширення синтаксису для JavaScript
Компоненти	будівельні блоки інтерфейсу користувача
Функціональні компоненти	функції JavaScript, які приймають реквізити як параметри і повертають

ВСТУП

У сучасному світі інформаційних технологій, де обчислювальні системи відіграють важливу роль у всіх сферах життя, розуміння архітектури комп'ютерів та операційних принципів є критично важливим для професійного зростання та успіху в галузі програмування, системного адміністрування та інженерії програмного забезпечення. Практичні навички роботи з низькорівневими операційними системами та мовами програмування, зокрема мовою Assembler, відіграють ключову роль у розробці швидкодіючих, ефективних та надійних програм.

Тема «Дослідження алгоритмів та методів реалізації тренажеру з теми «Арифметичні команди мови Assembler»» є актуальною, оскільки надає можливість розглянути глибокі аспекти навчання мові Assembler через арифметичні операції.

Ця робота має велике значення для студентів, які прагнуть зрозуміти, як працюють обчислювальні системи зсередини, а також для практикуючих фахівців, які бажають покращити свої навички у розробці ефективного програмного забезпечення. Результати дослідження можуть бути використані у навчальних процесах, а також у реальних проектах з розробки програмного забезпечення та системного програмування.

Метою роботи є дослідження алгоритмів та методів реалізації тренажеру, який дозволить ефективно навчати та вдосконалювати навички використання арифметичних команд мови Assembler. Розробка такого тренажеру дозволить студентам та фахівцям отримати практичний досвід роботи з мовою низького рівня, розширити свої знання у галузі архітектури обчислювальних систем та підвищити рівень володіння програмуванням.

Об’єктом розробки є процес вивчення та оптимізації навчання арифметичним командам мови Assembler шляхом розробки тренажеру.

Предметом розробки є алгоритми та методи реалізації тренажеру, які допомагають студентам та фахівцям ефективно навчатися та розвивати навички використання арифметичних команд мови Assembler.

Перелік використаних методів: засіб для створення, редагування та зневадження сучасних вебзастосунків і програм для хмарних систем Visual Studio Code, відкрита JavaScript бібліотека для створення інтерфейсів користувача React, середовище розробки Create React App.

Робота складається з чотирьох розділів. У першому розділі описано вибір підходу до реалізації тренажеру, визначення завдань та алгоритмів для реалізації в тренажері. У другому розділі розглянуто мову Assembler та арифметичні команди, визначення вимог до тренажеру, аналіз існуючих методів реалізації тренажерів, переваги та недоліки розглянутих рішень. У третьому розділі вказано алгоритмізацію задачі за темою роботи, розробку блок-схеми та архітектурної структури тренажеру. У четвертому розділі описано процес програмної реалізації тренажеру, необхідну користувачу інструкцію, тестування тренажеру арифметичним командам мови assembler.

Обсяг пояснювальної записки: 86 стор., в т.ч. основна частина - 60 стор., джерела - 9 назв.

1. ПОСТАНОВКА ЗАДАЧІ

1.1. Вибір підходу до реалізації тренажеру

При розробці тренажеру для навчання арифметичним командам мови Assembler важливо вибрати відповідні алгоритми та методи реалізації, що забезпечать ефективну та зручну роботу системи. Для визначення критеріїв вибору алгоритмів можна враховувати наступні аспекти:

- час виконання;

Алгоритми повинні забезпечувати швидке виконання арифметичних операцій, щоб користувачі могли швидко отримувати результати.

- використання пам'яті;

Алгоритми мають бути ефективними з точки зору використання пам'яті, щоб забезпечити оптимальну продуктивність та швидкодію.

- точність результатів;

Алгоритми повинні гарантувати правильність обчислень та уникнення помилок, пов'язаних з округленням або переповненням.

- сумісність з різними архітектурами;

Вибрані алгоритми повинні бути адаптовані до різних архітектур та можливостей обчислювальних систем.

- легкість реалізації та обслуговування;

Алгоритми повинні бути зрозумілими для розробників та легко піддаються реалізації та підтримці.

- відповідність навчальному контексту;

Вибрані алгоритми повинні бути підходящими для навчального контексту та допомагати вивченню арифметичних команд.

- масштабованість;

Алгоритми мають бути готовими до розширення та масштабування для підтримки різних функцій та можливостей.

- оптимізація для навчання.

Вибрані алгоритми повинні бути націлені на забезпечення ефективного навчання та тренування користувачів.

З урахуванням цих критеріїв можна вибрати оптимальні алгоритми та методи реалізації, які найкраще відповідають цілям та вимогам до розроблюваного тренажеру.

При розробці тренажеру можна використовувати різні підходи до реалізації системи. Кожен підхід має свої переваги та обмеження, які варто враховувати під час вибору оптимального рішення:

Веб-додаток:

- Переваги: Висока доступність через браузер, не вимагає встановлення додаткового ПЗ, можливість навчання з будь-якого пристрою.
- Недоліки: Обмежені можливості взаємодії з операційною системою, може бути вразливим до збоїв мережі.

Локальний додаток:

- Переваги: Доступ до операційної системи, можливість реалізації більш потужних функцій, незалежність від доступу до Інтернету.
- Недоліки: Вимагає встановлення додаткового програмного забезпечення, може бути обмежена сумісність з різними ОС.

Мобільний додаток:

- Переваги: Мобільність та доступність на різних пристроях, можливість використання сенсорних можливостей для взаємодії.
- Недоліки: Обмежені можливості в порівнянні з десктопним додатком, потребує розробки для різних платформ.

Хмарний сервіс:

- Переваги: Зручність доступу через Інтернет, можливість спільної роботи для багатьох користувачів.
- Недоліки: Залежність від доступу до мережі, може вимагати підписки або оплати.

Автономний додаток:

- Переваги: Робота без доступу до мережі, незалежність від сторонніх сервісів.
- Недоліки: Потребує встановлення, може бути важким для оновлення та підтримки.

При виборі підходу до реалізації тренажеру слід враховувати конкретні потреби користувачів, цілі проекту, доступні ресурси та технічні можливості. Важливо обрати підхід, який найкраще відповідає вимогам проекту та забезпечить комфортне та ефективне навчання арифметичним командам мови Assembler.

1.2. Визначення завдань та алгоритмів для реалізації в тренажері

Завдання дослідження:

- провести детальний аналіз мови Assembler, визначити основні арифметичні команди, їх синтаксис та специфіку виконання;
- визначити функціональні та нефункціональні вимоги до тренажеру, які забезпечать ефективне навчання арифметичних команд;
- провести огляд наявних тренажерів для навчання Assembler, з'ясувати їх переваги та недоліки;
- провести дослідження різних підходів до реалізації тренажеру та вибрати найбільш підходящі алгоритми;

- розробити архітектурну структуру тренажеру, реалізувати основні компоненти;
- підбити підсумки дослідження, оцінити досягнуті результати, висловити висновки.

Вибір конкретних алгоритмів має велике значення, оскільки він визначає продуктивність, точність та зручність використання системи. Обґрунтування вибору алгоритмів включає в себе аналіз їх переваг та недоліків, а також специфіку вимог до розроблюваного тренажеру. Ось кілька обґрунтувань для вибору конкретних алгоритмів:

1. Алгоритм додавання і віднімання.

Обґрунтування: Додавання і віднімання є базовими арифметичними операціями, які важливо вивчати в першу чергу. Вибір оптимального алгоритму для цих операцій допоможе навчати користувачів швидкому та точному виконанню цих операцій.

2. Алгоритм множення.

Обґрунтування: Множення є більш складною операцією, і вибір алгоритму для цієї операції має вплинути на швидкість та точність обчислень. Оскільки множення може використовуватися для реалізації інших операцій (наприклад, піднесення до степеня), важливо вибрати оптимальний алгоритм.

3. Алгоритм ділення.

Обґрунтування: Ділення є ще більш складною операцією, і правильний вибір алгоритму може вплинути на точність та ефективність цієї операції. Особливо важливо вибрати алгоритм, який забезпечує коректність обчислень навіть у випадках з різними дільниками.

4. Оптимізація для навчання.

Обґрунтування: Оскільки основна мета тренажеру - навчання користувачів, важливо вибрати алгоритми, які допоможуть зрозуміти особливості асемблерних команд та їх вплив на результати обчислень.

Вибір алгоритмів, які можна ефективно відобразити та візуалізувати для користувачів, покращить якість навчання.

5. Ефективність в різних архітектурах.

Обґрунтування: Оскільки тренажер може використовуватися для навчання на різних архітектурах, важливо вибрати алгоритми, які підходять для різних типів обчислювальних систем. Алгоритми, що можуть бути адаптовані для різних архітектур, забезпечать універсальність тренажеру.

Підбір алгоритмів повинен бути здійснений з урахуванням вищевказаних обґрунтувань, щоб забезпечити оптимальне навчання користувачів та високу якість роботи тренажеру.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд мови Assembler та арифметичних команд

Мова Assembler є мовою низького рівня, призначеною для програмування на рівні апаратного забезпечення комп'ютера. Ця мова дозволяє розробникам писати програми, які безпосередньо взаємодіють з апаратурою, використовуючи асемблерні інструкції, що відповідають конкретним операціям на процесорі. Основні поняття та принципи мови Assembler включають:

- інструкції;
- регістри;
- операнди;
- директиви;
- макроси;
- асемблювання;
- адресація;
- структура програми.

Мова Assembler використовує набір інструкцій, які відображають окремі операції на апаратурі. Ці інструкції включають операції арифметики, логіки, зчитування та запису пам'яті, переходи та інші операції [1].

Регістри є невеликими швидкодіючими областями пам'яті прямо на процесорі, які використовуються для зберігання тимчасових даних та керування обчисленнями. В мові Assembler регістри мають власні імена (наприклад, AX, BX на архітектурі x86).

Операнди це дані, з якими виконуються операції. Вони можуть бути числами, регістрами, адресами пам'яті або константами.

Директиви асемблера це спеціальні інструкції, які не є операціями на апаратурі, але контролюють процес асемблювання. Наприклад, директива для виділення пам'яті під змінну.

Макроси дозволяють створювати повторювані фрагменти коду, що спрощує написання та збереження часу [2].

Процес перетворення коду мови Assembler у виконуваний код або бінарний файл називається асемблюванням. Після асемблювання код може бути виконаний на цільовій платформі.

Адресація визначає спосіб звернення до операнду, який може бути регістром або пам'яттю. Існують різні режими адресації, такі як безпосередня, непосередня адресація, адресація за вказівником та інші.

Програма на мові Assembler складається з послідовності інструкцій та директив, які виконують певні дії на обчислювальній системі.

Розуміння цих основних понять та принципів допомагає розробникам писати ефективний та оптимізований код на мові Assembler, забезпечуючи більш глибоке розуміння принципів роботи апаратного забезпечення [3].

Арифметичні команди в мові Assembler дозволяють виконувати різні математичні операції над даними, що знаходяться у регістрах або пам'яті. Ці команди розширюють можливості обчислень та операцій з даними (табл. 1.1, 1.2) [4].

Таблиця 1.1 – Арифметичні команди в мові Assembler

Мнемоніка	Формат	Коментар
Команди додавання		
ADD	ADD приймач, джерело	Додати
ADC	ADC приймач, джерело	Додати перенос
INC	INC приймач	Збільшити на одиницю
Команди віднімання		
SUB	SUB приймач, джерело	Відняти
SBB	SBB приймач, джерело	Відняти із запозиченням
DEC	DEC приймач	Зменшити на одиницю
NEG	NEG приймач	Обернути знак
CMP	CMP приймач, джерело	Порівняти

Таблиця 1.2 – Продовження таблиці Арифметичні команди в мові Assembler

Мнемоніка	Формат	Коментар
Команди множення		
MUL	MUL джерело	Перемножити без знаку
IMUL	IMUL джерело	Перемножити зі знаком
Команди ділення		
DIV	DIV джерело	Ділити без знака
IDIV	IDIV джерело	Ділити зі знаком
Команди розширення знаку		
CBW	CBW	Перетворити байт в слово
CWD	CWD	Перетворити слово в подвійне слово

Розглянемо основні види арифметичних команд:

1. Додавання (ADD): Ця команда додає значення двох операндів та зберігає результат у вказаному регістрі або місці пам'яті. Вона використовується для виконання операції додавання.

2. Віднімання (SUB): Команда SUB віднімає значення другого операнда від першого та зберігає результат. Вона застосовується для віднімання одного числа від іншого.

3. Множення (MUL): Команда MUL виконує операцію множення двох значень та зберігає результат.

4. Ділення (DIV): DIV виконує операцію цілочисельного ділення першого операнда на другий та зберігає результат ділення та остачу.

5. Інкремент (INC) та декремент (DEC): Ці команди збільшують або зменшують значення операнда на одиницю. Вони часто використовуються для реалізації циклів або змін змінних.

6. Логічні операції: Деякі арифметичні команди включають операції порівняння (CMP), які встановлюють прапорці стану про результат порівняння двох значень. Інші операції включають побітові операції І (AND), АБО (OR) та виключне АБО (XOR), які виконують бітові операції над значеннями.

7. Переведення типів (CONVERT): Деякі команди дозволяють конвертувати значення з одного типу даних в інший, наприклад, з цілого числа в плаваючу точку.

Ця класифікація арифметичних команд допомагає розуміти різноманітні можливості, які надає мова Assembler для обчислень та роботи з даними на низькорівневому рівні.

Арифметичні операції мови Assembler є основними будівельними блоками для виконання різних обчислень на апаратному рівні. Вони впливають на роботу обчислювальної системи на різних рівнях, включаючи апаратні ресурси та швидкодію виконання [5]. Основні аспекти впливу арифметичних операцій на обчислювальну систему включають:

- використання апаратних ресурсів;

Виконання арифметичних операцій вимагає використання апаратних ресурсів, таких як арифметично-логічна одиниця (ALU) та регістри. Кожна операція потребує часу для виконання, а також може використовувати різні апаратні ресурси.

- вплив на продуктивність;

Використання арифметичних операцій може вплинути на продуктивність системи. Ефективність операцій важлива для отримання швидких результатів та оптимальної роботи програм.

- прапорці стану;

Під час виконання арифметичних операцій встановлюються прапорці стану, які вказують на певні події, такі як переповнення або нульовий результат. Обробка цих прапорців може вплинути на подальші дії програми.

- оптимізація коду;

Розробники можуть впливати на оптимізацію коду шляхом вибору певних арифметичних операцій. Вибір операцій може вплинути на розмір та швидкодію програми.

- використання регістрів;

Арифметичні операції виконуються з використанням регістрів, тому важливо ефективно управляти цими ресурсами для підтримання високої продуктивності.

- потенційні помилки;

Неправильне використання арифметичних операцій може призвести до помилок, таких як переповнення або недійсні результати, які можуть вплинути на коректність програми.

- використання пам'яті;

Деякі арифметичні операції включають доступ до пам'яті для зчитування або запису операндів. Це може вплинути на швидкодію виконання операцій через затримки доступу до пам'яті.

Враховуючи ці аспекти, потрібно ретельно вибирати та оптимізовувати використання арифметичних операцій в програмах для досягнення найкращої продуктивності та коректності.

2.2. Визначення вимог до тренажеру

Функціональні вимоги визначають основні функції та можливості, які повинен мати розроблюваний тренажер для навчання арифметичним командам мови Assembler. Ці вимоги визначають функціональну поведінку системи та допомагають забезпечити ефективну та корисну реалізацію тренажеру [6]. Серед функціональних вимог є:

- Тренажер повинен надавати візуальну можливість вивчення арифметичних команд шляхом графічного відображення виконання команд на екрані.
- Тренажер повинен надавати можливість тренування всіх типів арифметичних команд (додавання, віднімання, множення тощо).

- Тренажер повинен давати змогу користувачеві вибирати значення операндів для виконання вибраної арифметичної команди.
- Після введення операндів, тренажер має показувати результат.
- Система повинна автоматично перевіряти правильність відповіді, яку надав користувач, та надавати зворотний зв'язок.
- Тренажер може містити різні рівні складності, від базових до більш складних арифметичних команд, що дозволить користувачам підібрати підходящий рівень навчання.
- Система повинна відстежувати результати користувача, надавати статистику виконання завдань та оцінювати прогрес у вивченні арифметичних команд.
- Користувач повинен мати можливість повторювати вправи для покращення навичок.

Ці функціональні вимоги допоможуть створити тренажер, який буде ефективно вдосконалювати навички використання арифметичних команд мови Assembler, забезпечуючи зручний та корисний інструмент для навчання та тренування.

Нефункціональні вимоги описують характеристики та властивості системи, які не стосуються безпосередньо її функціональності, але впливають на її якість, продуктивність, безпеку та інші аспекти. Дотримання цих вимог допомагає створити ефективний та відповідний тренажер для навчання арифметичним командам мови Assembler [6]. Деякі нефункціональні вимоги можуть включати:

- Тренажер повинен забезпечувати достатню швидкість відгуку та обробки даних, щоб користувачі могли ефективно вчитися та вдосконалювати навички.
- Вимоги до використання пам'яті та процесорних ресурсів повинні бути обмежені, щоб забезпечити ефективну роботу системи навіть на обмежених платформах.

- Користувацький інтерфейс повинен бути зрозумілим, зручним та інтуїтивно зрозумілим для різних категорій користувачів.
- Тренажер повинен бути сумісним з різними операційними системами та пристроями.
- Тренажер повинен працювати стабільно та без відмов, забезпечуючи надійну роботу для користувачів.
- Система повинна надавати підтримку користувачам.
- Можливість перекладу інтерфейсу на різні мови для забезпечення комфорту користувачів.
- Тренажер повинен піддаватися тестуванню для перевірки її функціональності та відповідності вимогам.

Врахування цих нефункціональних вимог допоможе створити тренажер, який буде не лише ефективним у виконанні своїх завдань, але й відповідатиме стандартам якості та задовольнить потреби користувачів.

Сценарії використання описують типові ситуації, в яких користувачі можуть використовувати тренажер для навчання арифметичним командам мови Assembler. Ці сценарії допомагають зрозуміти, як система буде взаємодіяти з користувачами та як вона буде вирішувати їх завдання. Ось декілька прикладів сценаріїв використання:

1. Вивчення арифметичних операцій: Користувач, який тільки починає вивчати мову Assembler, відкриває тренажер та обирає опцію вивчення арифметичних операцій. Він вибирає тип команди (наприклад, додавання) та відповідні операнди. Тренажер відображає виконання команди, показує результат і надає зворотний зв'язок щодо правильності виконання.

2. Тренування навичок: Досвідчений розробник, який має певний досвід роботи з мовою Assembler, використовує тренажер для покращення своїх навичок у роботі з арифметичними командами. Він вибирає рівень складності, який відповідає його рівню знань, і повторює вправи з різними типами команд.

3. Вдосконалення швидкості: Студент, який готується до іспиту з архітектури обчислювальних систем, використовує тренажер для покращення швидкості та точності виконання арифметичних операцій. Він обирає режим "відповідь на час" та намагається виконати якомога більше команд за обмежений час.

4. Навчання в аудиторії: Викладач архітектури обчислювальних систем використовує тренажер для демонстрації роботи арифметичних команд студентам на лекції. Він створює певний сценарій виконання команд, показує їх виконання на екрані та роз'яснює результати.

5. Навчання на практичних заняттях: Студенти, які беруть участь у практичних заняттях з мови Assembler, використовують тренажер для виконання практичних завдань з арифметичними командами. Вони відтворюють реальні ситуації використання мови Assembler та практикуються у роботі з операціями.

Ці сценарії використання відображають різноманітність використання тренажеру для навчання арифметичним командам мови Assembler різними категоріями користувачів у різних контекстах.

2.3. Аналіз існуючих методів реалізації тренажерів

На сьогоднішній день існують різноманітні тренажери та навчальні ресурси, спрямовані на вивчення мови Assembler та включення до практики арифметичних команд. Ось кілька прикладів таких тренажерів:

Simple 8-bit Assembler Simulator (https://schweigi.github.io/assembler-simulator/) – цей онлайн-тренажер дозволяє користувачам візуально вивчати та відтворювати роботу асемблерних команд. Він надає можливість вибору різних типів команд та вводу власних даних для виконання операцій (рис. 2.1).

Також аналогічний функціонал представляють Assembler Simulator (https://exuanbo.xyz/assembler-simulator/) і 16-bit Assembler Simulator (https://parraman.github.io/asm-simulator/).

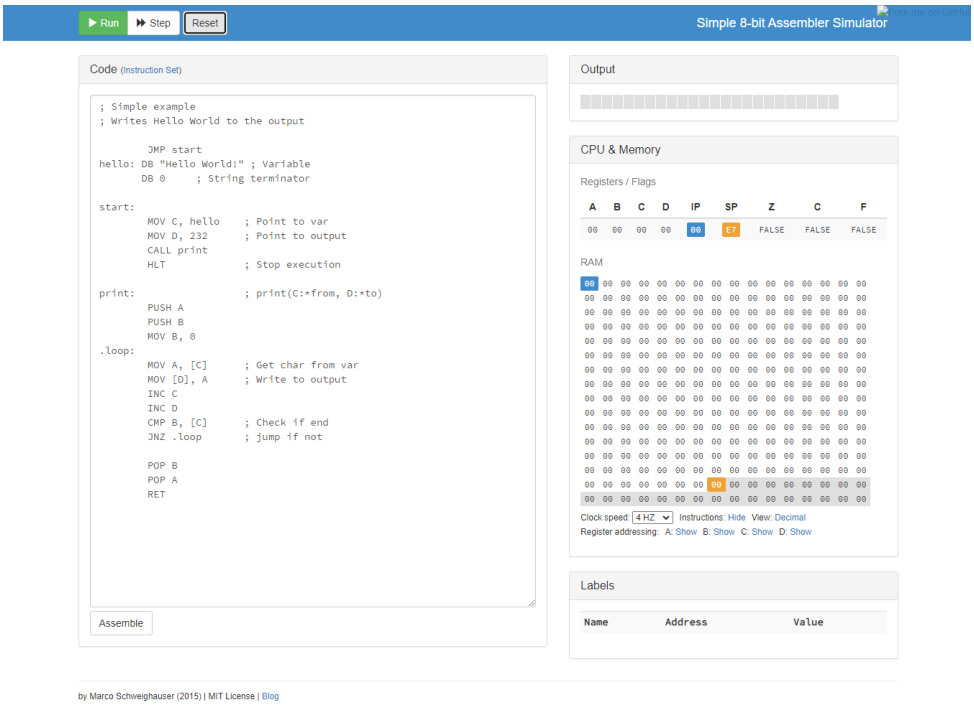

Рисунок 2.1 – Simple 8-bit Assembler Simulator

Emu8086 – є інтегрованим середовищем для вивчення асемблера, спеціально призначеним для платформи Intel 8086. Воно надає можливість вивчати роботу арифметичних команд на даній архітектурі (рис. 2.2).

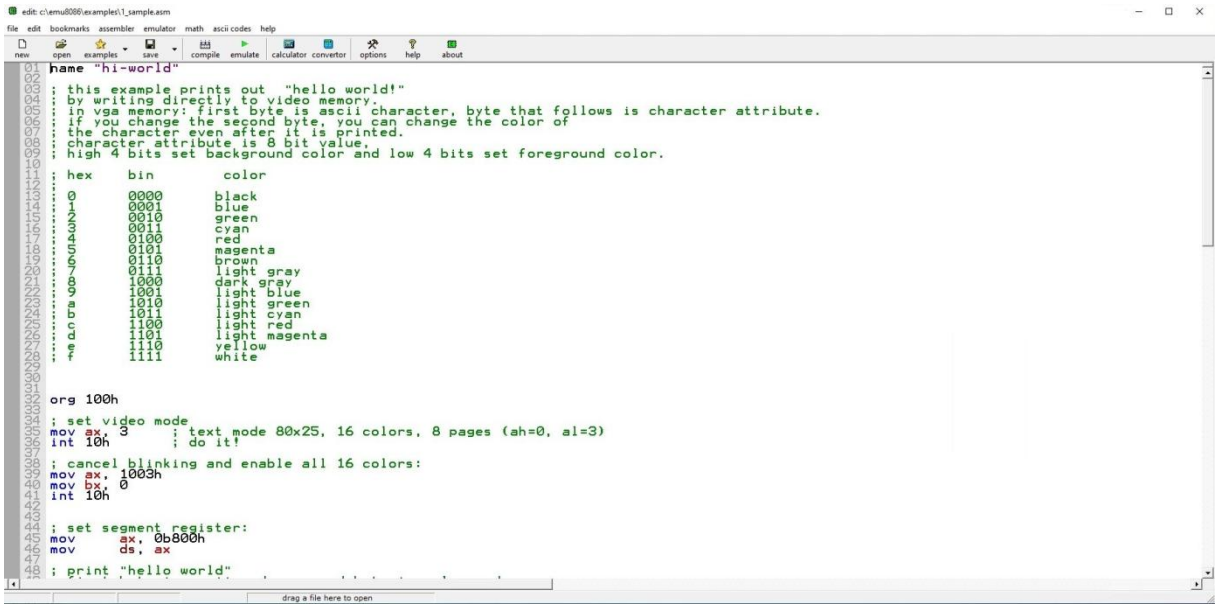

Рисунок 2.2 – Emu8086

QtSPIM – це інтерактивне середовище для вивчення мови MIPS Assembly. Воно дозволяє виконувати арифметичні операції та інші команди на MIPS архітектурі (рис. 2.3).

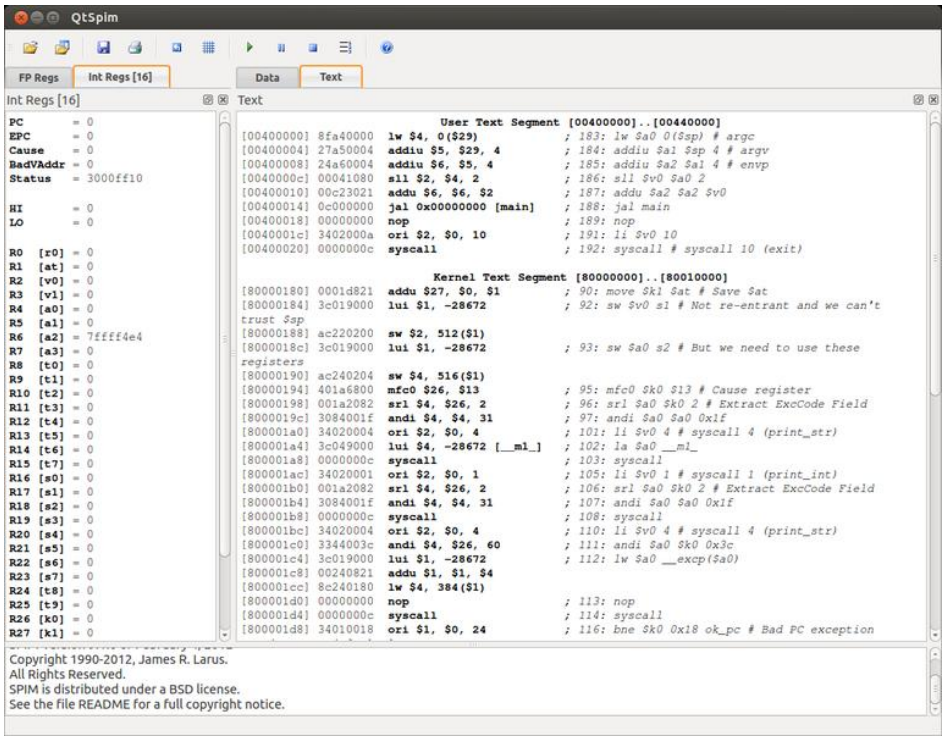

Рисунок 2.3 – QtSPIM

MARS (MIPS Assembler and Runtime Simulator) – цей тренажер розроблений для вивчення асемблера на базі MIPS архітектури. Він надає інтерактивне середовище для виконання арифметичних команд та спостереження за роботою програм (рис. 2.4).

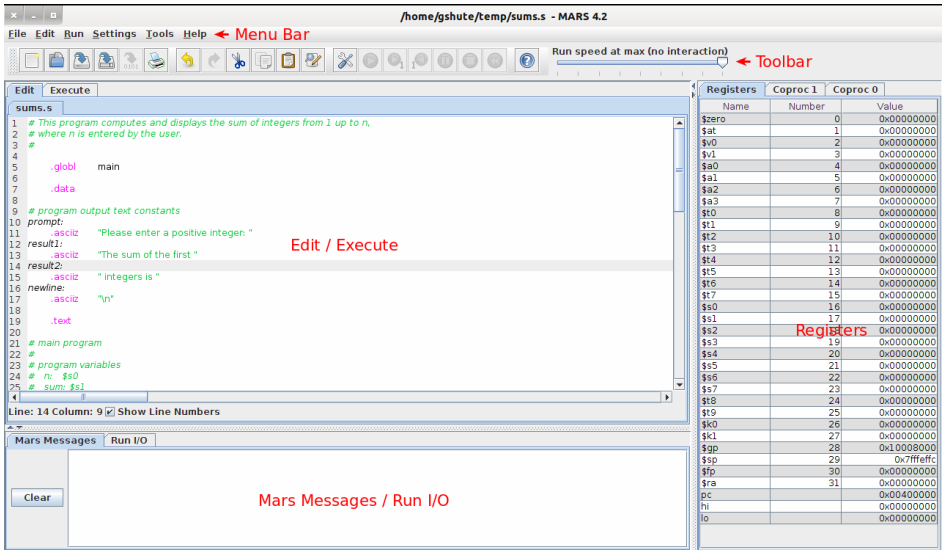

Рисунок 2.4 – MARS

RARS (RISC-V Assembler and Runtime Simulator) – схожий до MARS тренажер, але призначений для мови RISC-V Assembly. Він дозволяє вивчати арифметичні операції та інші команди на RISC-V архітектурі (рис. 2.5).

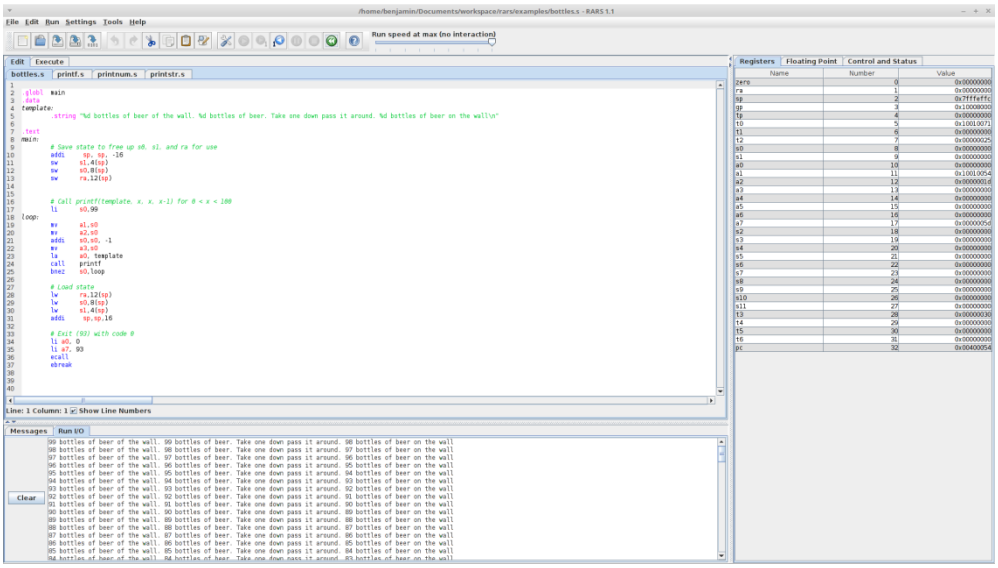

Рисунок 2.5 – RARS

Ці тренажери варто вказати як приклади наявних ресурсів, спрямованих на навчання асемблеру та включення арифметичних команд. Кожен з них має свої особливості та призначення, і їх можна використовувати для вивчення та тренування залежно від потреб користувача.

Аналіз існуючих тренажерів для вивчення мови Assembler та арифметичних команд допомагає зрозуміти, які аспекти їх розробки були успішно враховані, а які можуть бути поліпшені.

2.4. Переваги та недоліки розглянутих рішень

Аналіз переваг та недоліків існуючих рішень допомагає визначити ключові аспекти, які потрібно врахувати при розробці нового тренажеру для арифметичних команд мови Assembler.

Переваги існуючих рішень:

- Більшість існуючих тренажерів надають візуальну можливість спостереження за виконанням арифметичних команд, що полегшує розуміння принципів роботи.
- Багато рішень дозволяють користувачам взаємодіяти з асемблерним кодом, вводити дані, обирати команди та бачити результати в реальному часі.
- Тренажери, спеціалізовані на конкретних архітектурах (наприклад, MIPS або RISC-V), дозволяють вивчати арифметичні команди на реальних платформах.
- Деякі тренажери пропонують різні рівні складності, що дозволяє користувачам вибирати вправи, відповідні їхнім навичкам.

Недоліки існуючих рішень:

- Більшість тренажерів спеціалізовані на певних архітектурах, що обмежує їхню універсальність та застосовність.
- Деякі тренажери можуть бути менш інтерактивними, що обмежує можливості користувачів взаємодіяти з асемблерним кодом.
- Деякі рішення можуть не надавати докладної інформації про результати виконання команд, що ускладнює розуміння результатів.
- Деякі тренажери можуть бути платними або недоступними для всіх користувачів.
- Деякі рішення можуть обмежувати можливості користувачів змінювати та вводити свій асемблерний код.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Алгоритмізація задачі за темою роботи

Запропонований алгоритм роботи тренажеру для навчання арифметичним командам мови Assembler має наступний вигляд:

1. Запуск тренажеру:
   - Користувач запускає тренажер на обраній платформі.
2. Відображення інтерфейсу:
   - Тренажер відображає користувачеві головний екран з вибором режиму навчання, тестування або завдання.
3. Вибір режиму:
   - Користувач обирає бажаний режим роботи тренажеру: навчання, тестування або завдання.
4. Навчання:
   - У режимі навчання користувач має доступ до навчальних матеріалів, прикладів та пояснень з арифметичних команд.
5. Тестування:
   - У режимі тестування користувач проходить серію тестових завдань з арифметичних команд.
   - Він вводить вибирає правильні варіанти відповідей.
   - Кожне завдання має час виконання, щоб виміряти швидкість та точність користувача.
6. Завдання:
   - Користувач отримує завдання, включаючи операції та вхідні дані.
   - Він вводить свої розрахунки або вибирає правильні варіанти відповідей.

- Кожне завдання має час виконання, щоб виміряти швидкість та точність користувача.

7. Перевірка результатів:

- Система перевіряє правильність відповідей користувача та робить підсумок його результатів.

8. Надання зворотного зв'язку:

- Після виконання кожного завдання користувач отримує зворотний зв'язок про те, чи була відповідь правильною.

9. Оцінка та підсумок:

- В кінці навчання або тестування користувач отримує оцінку своїх знань та навичок з арифметичних команд:

Якщо правильних відповідей більше 80%: "Відмінно! Ви добре розумієте арифметичні команди Assembler. Ваш рівень знань вражає!"

Якщо правильних відповідей від 60% до 80%: "Добре зроблено! Ви маєте гарні знання арифметичних команд Assembler. Продовжуйте вивчення для досягнення ще кращих результатів."

Якщо правильних відповідей менше 60%: "Є деяке місце для покращення. Рекомендую повторити матеріал та робити більше практики."

- Також можливі рекомендації щодо подальшого навчання чи практики.

Якщо вам подобається вивчати Assembler, рекомендуємо читати більше літератури та ресурсів з Assembler, виконувати більше практичних завдань та вправ. Якщо ви шукаєте нові виклики, слід розглянути вивчення інших мов програмування або архітектур процесорів.

10.Завершення роботи:

- Користувач може вийти з тренажеру.

Тестові питання допоможуть користувачам перевірити та поглибити свої знання про основні арифметичні команди мови Assembler. Запропоновано наступний алгоритм тестування:

Крок 1. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції додавання у мові Assembler?

а) ADD (Правильна відповідь)

б) SUB

в) MUL

г) DIV

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 2. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції віднімання у мові Assembler?

а) ADD

б) SUB (Правильна відповідь)

в) MUL

г) DIV

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 3. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції множення у мові Assembler?

а) ADD

б) SUB

в) MUL (Правильна відповідь)

г) DIV

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 4. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції ділення у мові Assembler?

а) ADD

б) SUB

в) MUL

г) DIV (Правильна відповідь)

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 5. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції збільшення на одиницю у мові Assembler?

а) NEG

б) INC (Правильна відповідь)

в) DEC

г) CMP

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 6. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції зменшення на одиницю у мові Assembler?

а) NEG

б) INC

в) DEC (Правильна відповідь)

г) CMP

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 7. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції обернути знак у мові Assembler?

а) NEG (Правильна відповідь)

б) INC

в) DEC

г) CMP

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 8. Відображається питання та варіанти відповіді: Яка команда використовується для виконання арифметичної операції порівняння у мові Assembler?

а) NEG

б) INC

в) DEC

г) CMP (Правильна відповідь)

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 9. Відображається питання та варіанти відповіді: Які арифметичні команди дозволяють виконувати операції з числами у мові Assembler?

а) ADD, SUB, MUL, DIV (Правильна відповідь)

б) INC, DEC, CMP, JMP

в) PUSH, POP, CALL, RET

г) MOV, CMP, JMP, JNZ

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 10. Відображається питання та варіанти відповіді: Яке значення буде збережено в регістрі ax після виконання цих інструкцій?

Інструкції:

mov ax, 10

add ax, 5

а) 10

б) 15 (Правильна відповідь)

в) 5

г) 20

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 11. Відображається питання та варіанти відповіді: Яке значення буде збережено в регістрі bx після виконання цих інструкцій?

Інструкції:

mov bx, 25

sub bx, 15

а) 10 (Правильна відповідь)

б) 15

в) 25

г) 5

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 12. Відображається питання та варіанти відповіді: Яке значення буде збережено в регістрі ax після виконання цих інструкцій?

Інструкції:

mov ax, 8

mov bx, 3

mul bx

а) 8

б) 24 (Правильна відповідь)

в) 3

г) 64

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 13. Відображається питання та варіанти відповіді: Яке значення буде збережено в регістрі ax після виконання цих інструкцій?

Інструкції:

mov ax, 50

mov bx, 10

div bx

а) 5 (Правильна відповідь)

б) 0

в) 10

г) 50

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне питання.

Крок 14. Відображається питання та варіанти відповіді: Яке значення буде збережено в регістрі dx після виконання цих інструкцій?

Інструкції:

mov ax, 20

mov bx, 3

div bx

а) 1

б) 2 (Правильна відповідь)

в) 0

г) 3

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то завершується виконання тестів і виводиться оцінка та підсумок.

Завдання дозволять перевірити розуміння та навички з використання арифметичних команд мови Assembler у користувачів тренажеру. Алгоритм виконання завдань:

Крок 1. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Скласти два числа a, b.

var a,b:integer;
sum:integer;
begin
  writeln('Введіть a і b:');
```

```
  readln(a);
  readln(b);
asm
    ...
  end;
  writeln('Сума:',sum);
end.[4]
```

Правильний порядок команд:

```
    mov ax,a
    mov bx,b
    add ax,bx
    mov sum,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 2. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Скласти два числа. Одне число має тип integer, друге число – тип
byte.

var a:integer;
b:byte;
sum:integer;
begin
  write('Введіть a (значення від -32768 до 32768):');
  readln(a);
  write('Введіть b (значення від 0 до 255):');
  readln(b);
  asm
    ...
  end;
  writeln('Сума:',sum);
end.[4]
```

Правильний порядок команд:

```
    mov ax,a
    mov bl,b
    mov bh,0
```

```
    add ax,bx
    mov sum,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 3. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Скласти три числа. Числа мають тип integer.

var a,b,c:integer;
sum:integer;
w:word;

begin
  write('Введіть a (значення від -32768 до 32768):');
  readln(a);
  write('Введіть b (значення від -32768 до 32768):');
  readln(b);
  write('Введіть c (значення від -32768 до 32768):');
  readln(c);

  asm
    ...
  end;

  writeln('Сума:',w);
end. [4]
```

Правильний порядок команд:

```
    mov ax,a
    mov bx,b
    add ax,bx
    mov bx,c
    add ax,bx
    mov w,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 4. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
До числа A (тип Integer) введеному з клавіатури додати число 100.

var a:integer;
begin
  write('Введіть a (значення від -32768 до 32768):');
  readln(a);

  asm
    ...
  end;

  writeln('A=',a);
end.[4]
```

Правильний порядок команд:

```
    mov ax,a
    add ax,100
    mov a,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 5. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Від числа введеного з клавіатури відняти число 10 за допомогою ко
манди додавання.

var a:integer;
begin
  write('Введіть a (значення від -32768 до 32768):');
  readln(a);
  asm
    ...
  end;
  writeln('A=',a);
end. [4]
```

Правильний порядок команд:

```
mov ax,a
add ax,-10
mov a,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 6. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Відняти від a значення b. Результат збільшити на 1.

var a,b:integer;
sum:integer;
begin
  writeln('Введіть a і b:');
  readln(a);
  readln(b);
  asm
    ...
  end;
  writeln('Різниця:',sum);
end.
```

Правильний порядок команд:

```
mov ax,a
mov bx,b
sub ax,bx
inc ax
mov sum,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 7. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Помножити значення a на b. Результат зменшити на 1.

var a,b:integer;
sum:integer;
begin
  writeln('Введіть a і b:');
  readln(a);
  readln(b);
  asm
    ...
  end;
  writeln('Результат:',sum);
end.
```

Правильний порядок команд:

```
    mov ax,a
    mov bx,b
    mul bx
    dec ax
    mov sum,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 8. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Розділити значення a на b.
var a,b:integer;
sum:integer;
begin
  writeln('Введіть a і b:');
  readln(a);
  readln(b);
  asm
    ...
  end;
  writeln('Результат:',sum);
end.
```

Правильний порядок команд:

```
    mov ax,a
    mov bx,b
    div bx
    mov sum,ax
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то виводиться наступне завдання.

Крок 9. Відображається приклад програми на використання арифметичних команд і завдання: Встановіть на місці … правильний порядок виконання команд.

```
Розділити значення a на b. Знайти залишок від ділення.
var a,b:integer;
sum:integer;
begin
  writeln('Введіть a і b:');
  readln(a);
  readln(b);
  asm
    ...
  end;
  writeln('Результат:',sum);
end.
```

Правильний порядок команд:

```
    mov ax,a
    mov bx,b
    div bx
    mov sum,dx
```

Відображається повідомлення про те, чи правильна відповідь. Якщо правильна, то завершується виконання завдань і виводиться оцінка та підсумок.

3.2. Розробка блок-схеми, яка підлягає програмуванню

Загальна блок-схема алгоритму роботи тренажеру складається з таких блоків (рис. 3.1):

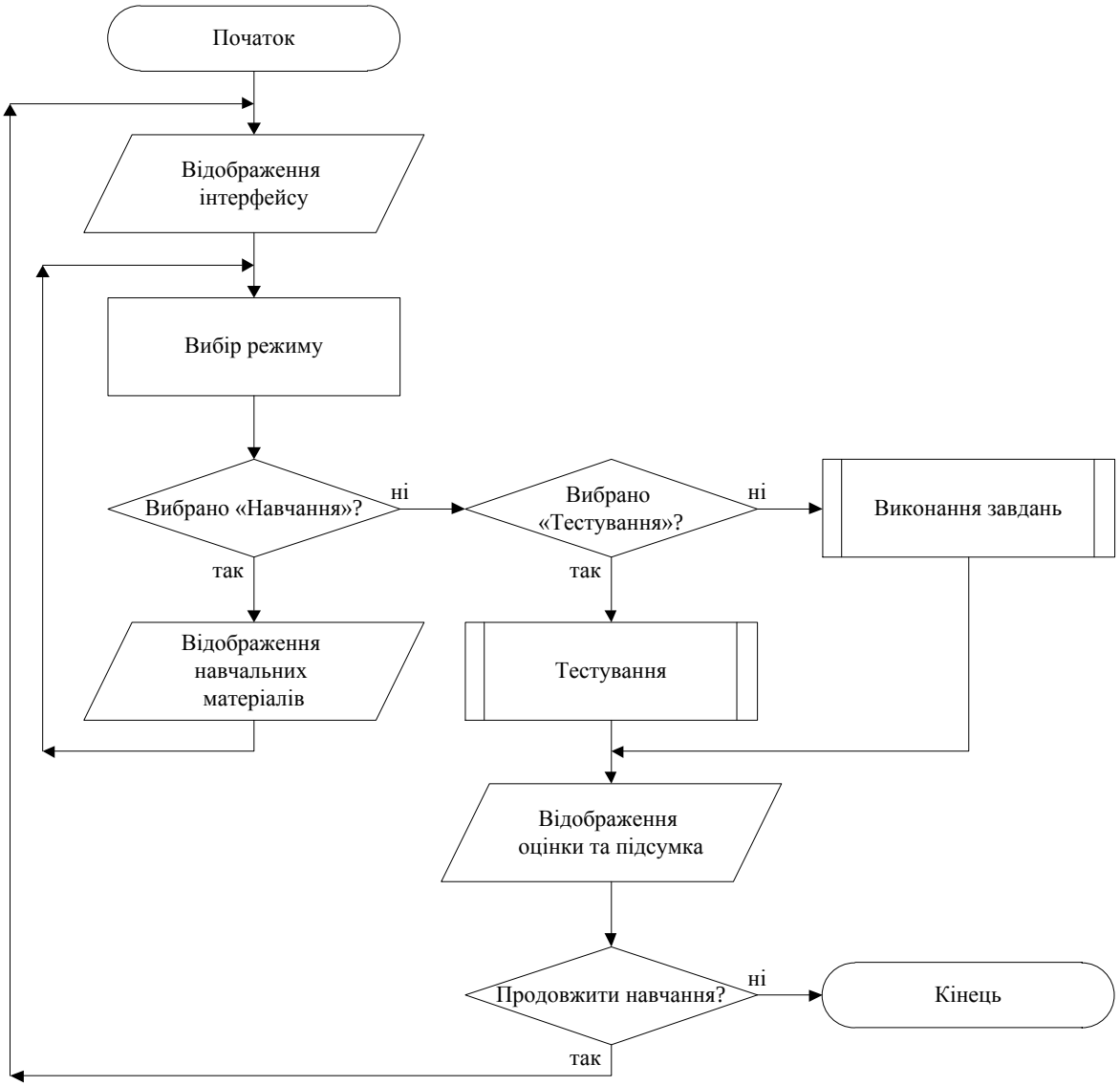

Рисунок 3.1 – Блок-схема алгоритму роботи тренажеру

- Головний блок: Старт програми.
- Вибір режиму: Користувач обирає режим навчання або тестування. Якщо вибрано режим навчання, перехід до блоку навчання. Якщо вибрано режим тестування, перехід до блоку тестування.

- Блок навчання: Показ навчальних матеріалів та прикладів з арифметичних команд.
- Блок тестування: Показ питань з вибором однієї відповіді на використання основних видів арифметичних команд. Користувач вибирає варіант відповіді та натискає кнопку "Далі". Після вибору варіанта відповіді система перевіряє правильність і надає зворотний зв'язок.
- Блок виконання завдань: Показ прикладу програми на використання арифметичних команд і завдання. Користувач може виконувати практичні завдання та отримувати зворотний зв'язок.
- Збір результатів: Підрахунок загального балу користувача на основі вірних відповідей.
- Завершення роботи.

Блоки тестування і виконання завдань мають свої блок-схеми роботи (рис. 3.2, 3.3).

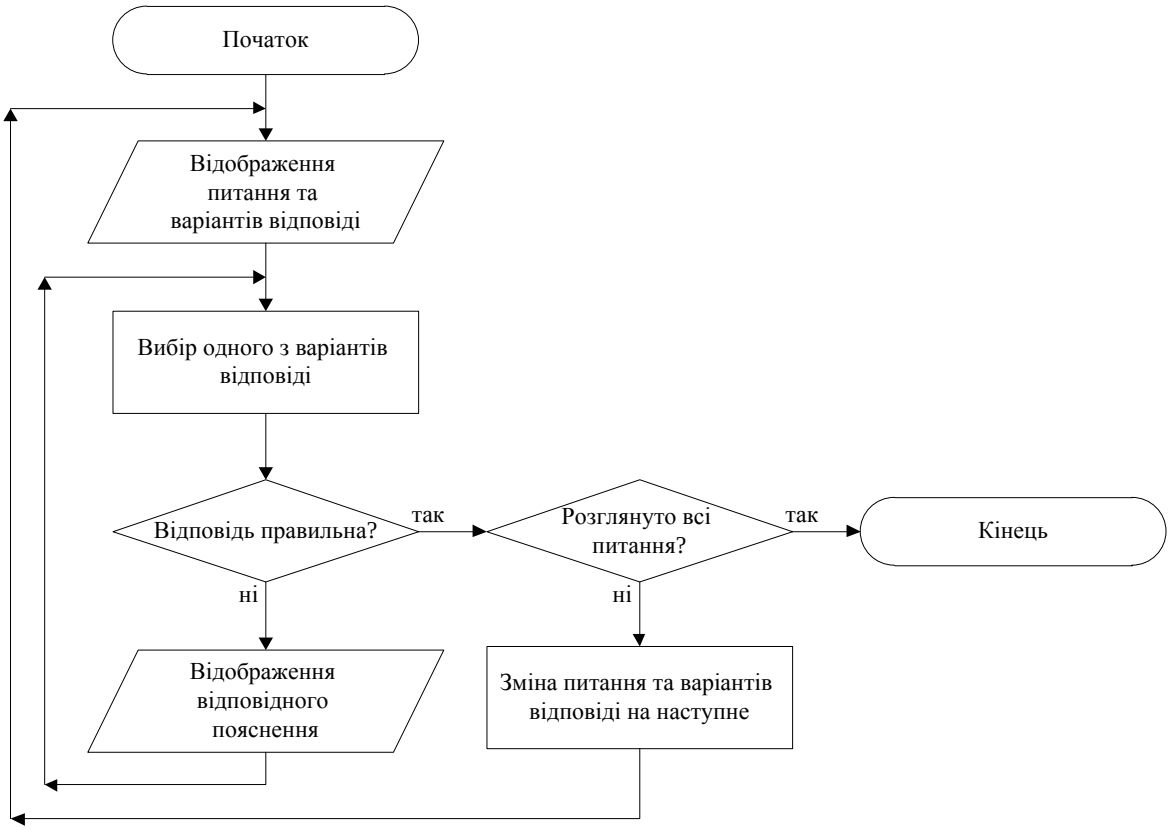

Рисунок 3.2 – Блок-схема блоку тестування

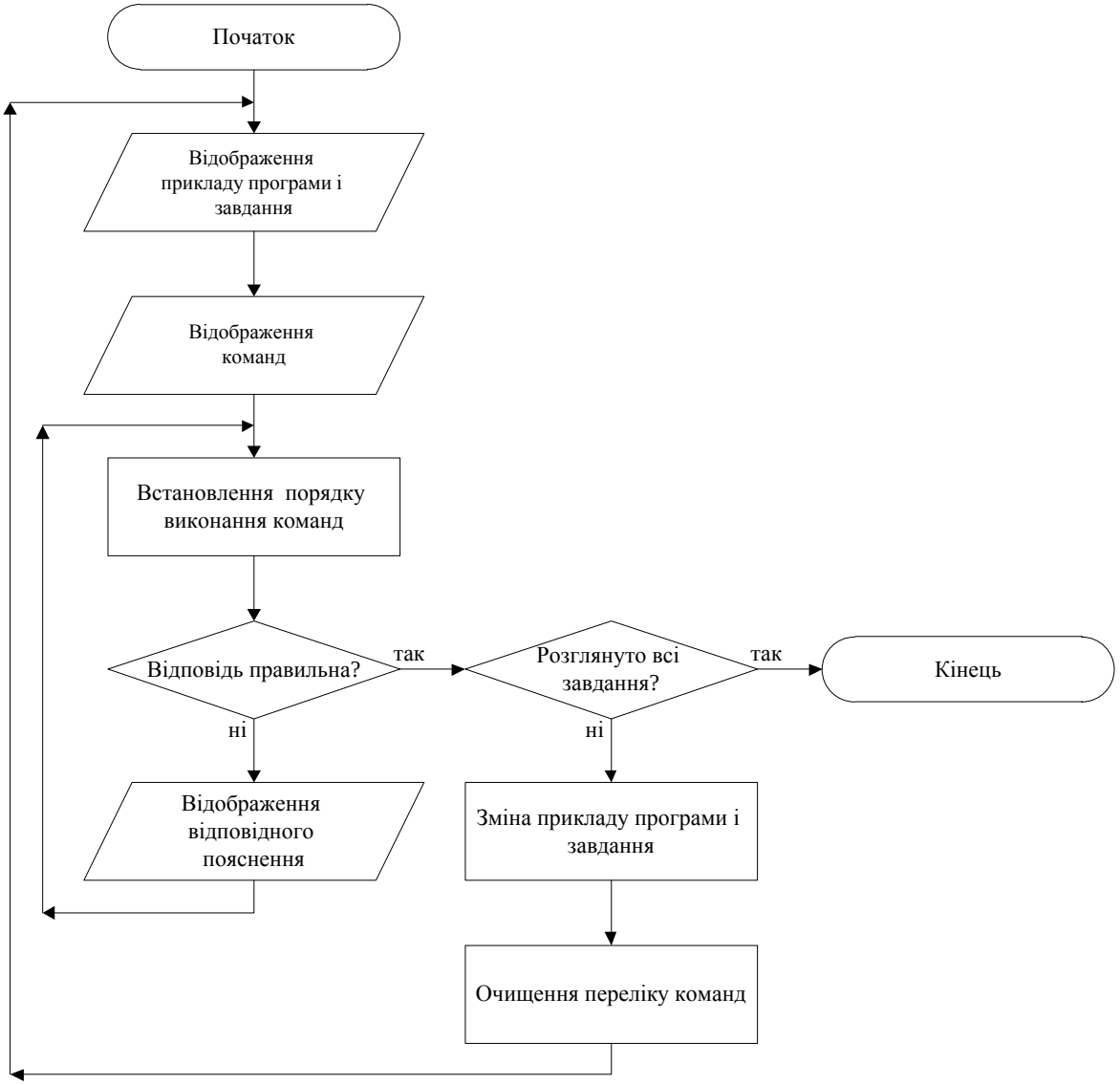

Рисунок 3.3 – Блок-схема блоку виконання завдань

Спочатку відображається питання і варіанти відповіді чи завдання і приклад програми. Після цього від користувача потребуються дії: вибір одного з варіантів, встановлення порядку команд.

При неправильній відповіді відображається пояснення. При правильній – здійснюється перевірка на кількість виконаних кроків. Якщо ще залишились доступні кроки, то виводиться наступне питання чи завдання.

3.3. Розробка архітектурної структури тренажеру

Архітектурна структура тренажеру є важливою складовою його розробки, оскільки вона визначає організацію компонентів, модулів та взаємозв'язки між ними. Враховуючи специфіку тренажеру для навчання арифметичним командам мови Assembler, архітектурна структура може включати наступні компоненти:

1. Користувацький інтерфейс:

- Інтерактивна частина тренажеру, через яку користувачі будуть взаємодіяти з системою (рис. 3.4).
- Включає графічний інтерфейс, вибір режиму, вибір команд (рис. 3.5).
- Забезпечує можливість спостереження за результатами.

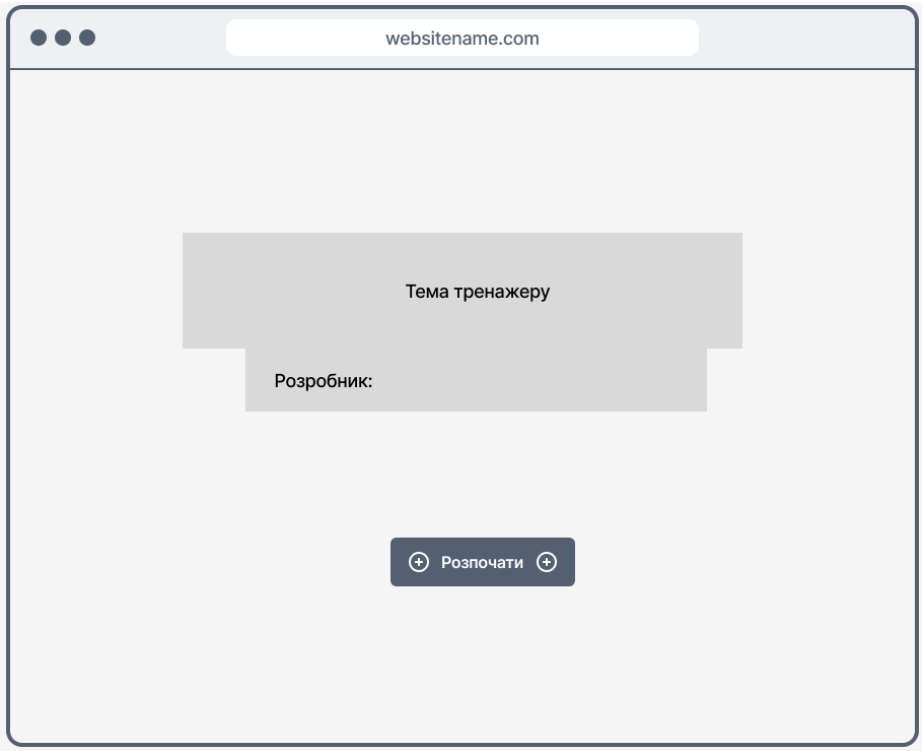

Рисунок 3.4 – Інтерфейс головної сторінки

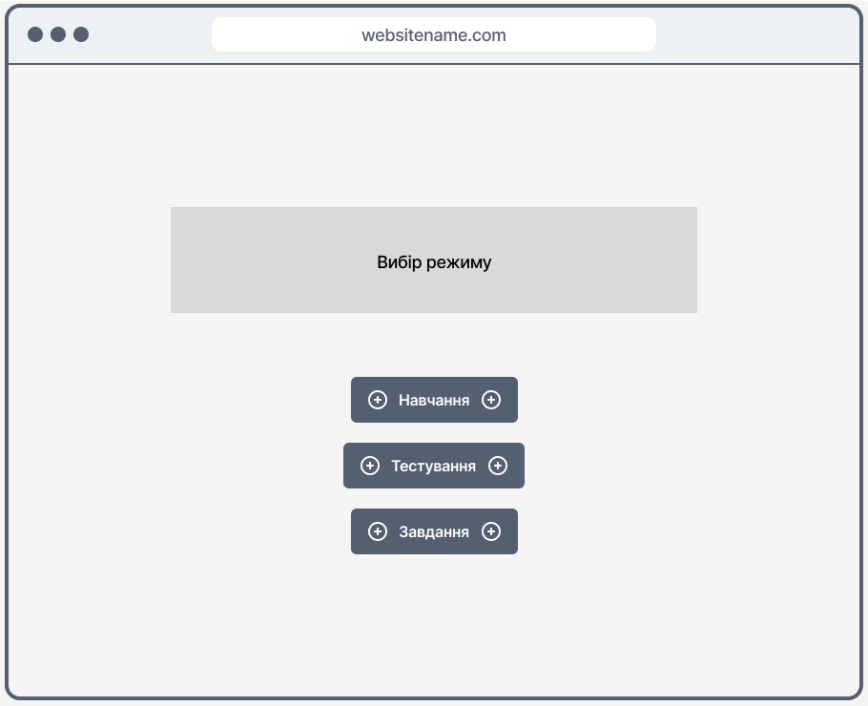

Рисунок 3.5 – Інтерфейс вибору режиму

2. Модуль візуалізації:
   - Відображає кроки тестування і виконання завдань (рис. 3.6).
   - Допомагає користувачам при роботі з тестуванням і виконанням завдань.

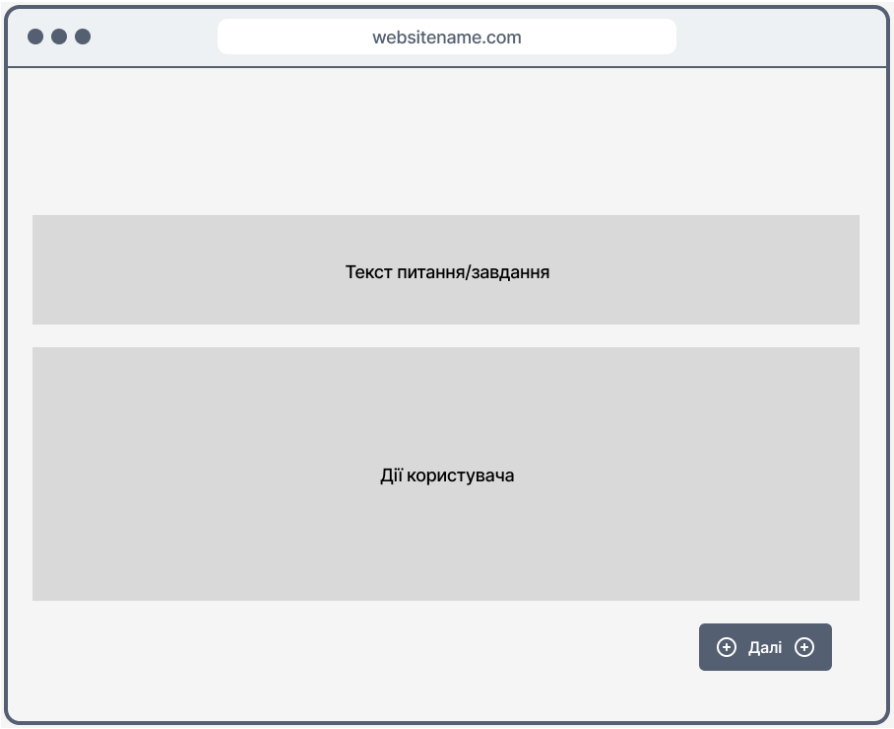

Рисунок 3.6 – Інтерфейс кроків тестування і виконання завдань

3. Система зберігання прогресу користувача:

• Дозволяє користувачам переходити до наступних кроків, виводить результати роботи у тренажері (рис. 3.7).

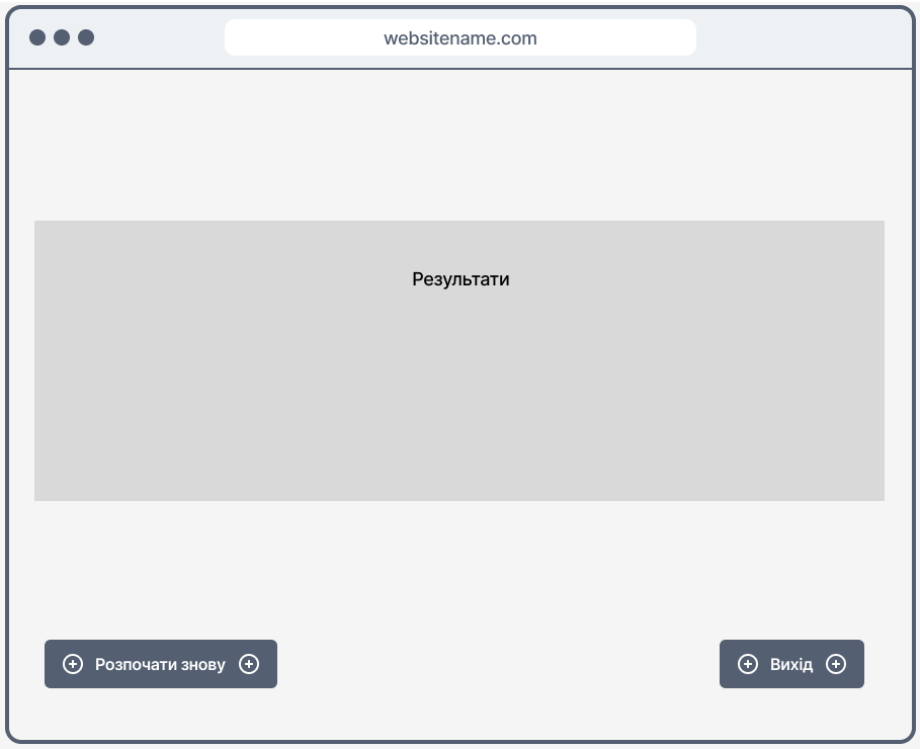

Рисунок 3.7 – Інтерфейс результатів

4. Модуль навчання:

• Надає навчальний матеріал, інструкції, приклади та завдання для виконання (рис. 3.8).

• Включає можливість відображення теоретичних пояснень до асемблерних команд.

5. Система зберігання матеріалів:

• Забезпечує зберігання теоретичних матеріалів, прикладів коду, пояснень до асемблерних команд та іншого навчального контенту (рис. 3.8).

6. Модуль тестування:

• Реалізує перехід між кроками тестування.

• Включає можливість відображення пояснень при неправильній відповіді (рис. 3.9).

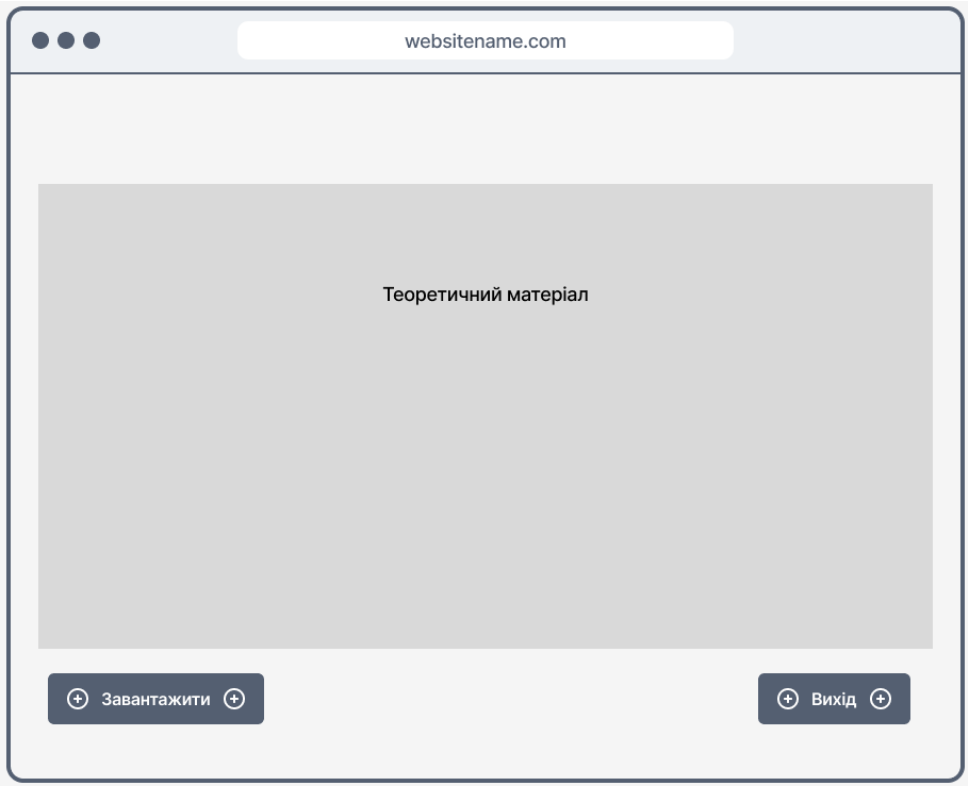

Рисунок 3.8 – Інтерфейс навчальних матеріалів

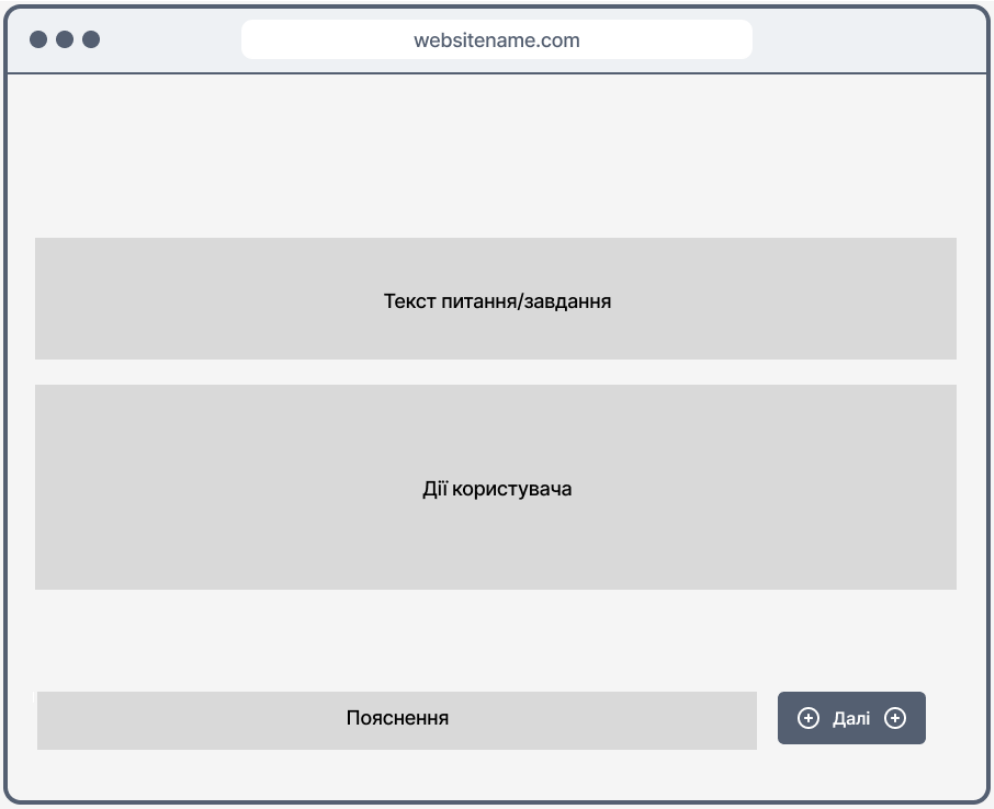

Рисунок 3.9 – Інтерфейс пояснень

7. Модуль виконання завдань:

- Реалізує перехід між кроками виконання завдань.
- Включає можливість відображення пояснень при неправильній відповіді (рис. 3.9).

Архітектурна структура тренажеру повинна бути добре збалансованою та спроектованою з огляду на зручність використання, ефективність та навчальні цілі користувачів.

Вибір та реалізація інтерфейсу користувача є ключовим етапом розробки тренажеру. Інтерфейс повинен бути зручним, інтуїтивно зрозумілим та відповідати потребам користувачів. Залежно від цілей та цільової аудиторії, можна вибрати різні типи інтерфейсу:

Веб-додаток:

- Вибір: Веб-інтерфейс є універсальним та доступним для різних платформ, оскільки може використовуватися через браузер на будь-якому пристрої з доступом до Інтернету.
- Реалізація: Використання HTML, CSS та JavaScript для створення графічного інтерфейсу. Організація взаємодії з сервером для обробки асемблерних команд та зберігання прогресу.

Локальний додаток:

- Вибір: Локальний інтерфейс може бути більш потужним та забезпечувати більшу гнучкість, оскільки він працює безпосередньо на пристрої користувача.
- Реалізація: Використання мов програмування, таких як Python, Java або C#, для створення графічного інтерфейсу. Розробка функцій для обробки асемблерних команд та зберігання даних.

Мобільний додаток:

- Вибір: Мобільний інтерфейс надає можливість навчання на різних мобільних пристроях, що є зручним для користувачів, які часто перебувають в дорозі.

- Реалізація: Використання платформи для розробки мобільних додатків (наприклад, Android Studio для Android або Xcode для iOS) для створення інтерфейсу та функціональності додатку.

При реалізації інтерфейсу користувача варто враховувати наступні аспекти:

- Дизайн: Забезпечення чистого та зрозумілого дизайну інтерфейсу, який не завантажує користувачів зайвою інформацією та деталями.
- Інтуїтивність: Мінімізація кількості кроків для виконання завдань, зрозумілість кнопок та інших елементів інтерфейсу.
- Взаємодія: Достатній рівень взаємодії між користувачем та інтерфейсом, можливість введення, вибору та взаємодії з асемблерними командами.

Реалізація інтерфейсу користувача вимагає компетентності в обраному підході та використовуваних технологіях для забезпечення зручного та ефективного навчання користувачів.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис процесу програмної реалізації тренажеру

При реалізації тренажеру для навчання арифметичним командам мови Assembler важливо детально описати реалізацію основних компонентів, які були визначені в архітектурній структурі.

В кожному компоненті першим кроком імпортуються бібліотеки та інші компоненти, що використовуються [7]. Головним є App.js, який управляє станом додатка та відображає відповідні компоненти в залежності від значення mode. З нього і розпочнемо.

```
import React, { useState } from "react";
import LearningMode from "./LearningMode";
import TestingMode from "./TestingMode";
import TaskMode from "./TaskMode";
import './App.css';
```

Використовується import для завантаження необхідних бібліотек та компонентів LearningMode, TestingMode, TaskMode. Також підключається таблиця стилів App.css (див. Додаток А).

```
function App() {
  const [mode, setMode] = useState(null);

  const startHome = () => setMode(null);
  const startLearning = () => setMode("learning");
  const startTesting = () => setMode("testing");
  const startTask = () => setMode("task");

  const [finishMode, setFinishMode] = useState("");
  const [time, setTime] = useState("");
  const [rightAnswer, setRightAnswer] = useState(0);
  const [wrongAnswer, setWrongAnswer] = useState(0);
  const startResult = (finishMode, time, rightAnswer, wrongAnswer) => {
    setMode("result");
    setFinishMode(finishMode);
    setTime(time);
```

```
  setRightAnswer(rightAnswer);
  setWrongAnswer(wrongAnswer);
 };
```

Створюються різні стани (mode, finishMode, time, rightAnswer, wrongAnswer) і функції для їх зміни (setMode, setFinishMode, setTime, setRightAnswer, setWrongAnswer, startHome, startLearning, startTesting, startTask, startResult) з використанням хуків стану (useState) [8].

```
return (
 <div className="App">
  {mode === null && (
   <MainMenu
    onStartLearning={startLearning}
    onStartTesting={startTesting}
    onStartTask={startTask}
   />
  )}
  {mode === "learning" && <LearningMode onStartHome={startHome} />}
  {mode === "testing" && <TestingMode onStartHome={startResult} />}
  {mode === "task" && <TaskMode onStartHome={startResult} />}
  {mode === "result" && (
   <Result
    mode={finishMode}
    timer={time}
    rigthCount={rightAnswer}
    wrongCount={wrongAnswer}
    onStartHome={startHome}
    onStartTesting={startTesting}
    onStartTask={startTask}
   />
  )}
 </div>
);
```

Для рендерингу компонентів в залежності від mode використовується конструкція умовного рендерингу. Залежно від значення mode відображається відповідний компонент: MainMenu, LearningMode, TestingMode, TaskMode або Result.

```
export default App;
```

Кінець файлу вказує, що App експортується для використання в інших частинах програми.

Компоненти головного меню і виведення результатів реалізовано також в App.js.

MainMenu відображає початковий екран з інформацією про додаток та, при натисканні на кнопку "Розпочати", переходить до відображення іншого набору кнопок для вибору режиму (навчання, тестування, завдання).

```
function MainMenu({ onStartLearning, onStartTesting, onStartTask }) {
 const [changeButton, setChangeButton] = useState(false);

 const startMode = () => {
  setChangeButton(true);
 };
```

Компонент отримує об'єкт props, який містить три функції: onStartLearning, onStartTesting, onStartTask. Ці функції передаються компоненту ззовні [9]. Змінна changeButton визначає, чи потрібно показувати початкову кнопку чи набір кнопок після натискання "Розпочати". Функція startMode встановлює значення changeButton в true, що призведе до зміни відображення кнопок у компоненті.

```
return (
 <div className="main">
  <h1>
   Тренажер з теми "Арифметичні команди мови Assembler" дисципліни
   "Архітектура обчислювальних систем"
  </h1>
  <h2>Розробник: Авраменко Євгеній Ігорович</h2>
  {!changeButton && (
   <div className="column-position">
    <button className="button" onClick={startMode}>Розпочати</button>
   </div>
  )}
  {changeButton && (
   <div className="column-position">
    <button className="button" onClick={onStartLearning}>Навчання</button>
    <button className="button" onClick={onStartTesting}>Тестування</button>
    <button className="button" onClick={onStartTask}>Завдання</button>
   </div>
  )}
 </div>
);
}
```

У цьому фрагменті коду відбувається рендеринг JSX для компонента. В залежності від значення changeButton відображається або одна кнопка "Розпочати", або набір кнопок для навчання, тестування та завдань. Рендеринг також включає текстові елементи, які виводять інформацію про тему та розробника.

Result взаємодіє з користувачем, надаючи звіт про його успішність та рекомендації для подальшого навчання.

```
function Result({ mode, timer, onStartHome, onStartTesting, onStartTask, rigthCount, wrongCount }) {
  let count = rigthCount / (rigthCount + wrongCount);
```

Функція компоненту Result приймає параметри, які передаються в компонент ззовні: mode, timer, onStartHome, onStartTesting, onStartTask, rigthCount, wrongCount. Обчислюється відсоток правильних відповідей. Це використовується для визначення рівня успішності користувача.

```
  return (
    <div className="main">
      <h1>Оцінка та підсумок</h1>
      <p>Затрачений час на виконання:{timer}</p>
      <p>Правильних відповідей: {rigthCount}</p>
      <p>Неправильних відповідей: {wrongCount}</p>
      {count >= 0.8 && <h3>Відмінно! …</h3>}
      {count < 0.8 && count >= 0.6 && <h3>Добре зроблено! …</h3>}
      {count < 0.6 && <h3>Є деяке місце для покращення. ..</h3>}
      <p><strong>Рекомендації для подальшого навчання чи практики:</strong></p>
      <ul>
        <li>Читати більше літератури та ресурсів з Assembler.</li>
        <li>Виконувати більше практичних завдань та вправ.</li>
        <li>Розглянути вивчення інших мов програмування або архітектур процесорів.</li>
      </ul>
      <div className="row-position">
        <button className="button" onClick={mode === "test" ? onStartTesting : onStartTask}>Розпочати знову</button>
        <button className="button" onClick={onStartHome}>Вихід</button>
      </div>
    </div>
  );
```

У цьому фрагменті коду відбувається рендеринг JSX для компонента Result. Компонент виводить інформацію про результати тестування,

включаючи час, кількість правильних та неправильних відповідей. Також виводиться рекомендація в залежності від відсотка правильних відповідей. Кнопки "Розпочати знову" та "Вихід" дозволяють користувачеві перейти знову до тестування чи завдань або повернутися до початкового екрану.

Компонент LearningMode відображає режим навчання, надає можливість завантажити матеріали і вийти до головного меню.

```
import React from "react";
import Theory from './theory.pdf';
```

У цьому розділі використовується import для завантаження бібліотеки React та ресурсу, який є PDF-файлом з ім'ям theory.pdf.

```
function LearningMode({ onStartHome }) {
const downloadFile = () => {
  const pdfUrl = Theory;
  const link = document.createElement("a");
  link.href = pdfUrl;
  link.download = "theory"; // specify the filename
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
};
```

Функція LearningMode отримує параметр onStartHome, який є функцією, переданою ззовні і використовується для повернення користувача до початкового екрану. Функція downloadFile відповідає за завантаження файлу при натисканні на кнопку "Завантажити". Спочатку вона створює новий елемент <a> (посилання), встановлює йому властивості href та download для вказання шляху до файлу та ім'я файлу, додає це посилання до тіла документа, симулює клік по посиланню та видаляє його з тіла документа після завантаження.

```
return (
  <div className="main-theory">
    <h2>Режим навчання</h2>
    {/* Матеріал */}
    <div className="row-position">
      <button className="button" onClick={downloadFile}>Завантажити</button>
      <button className="button" onClick={onStartHome}>Вихід</button>
    </div>
  </div>
```

У цьому фрагменті коду відбувається рендеринг JSX для компоненту LearningMode. Відображається заголовок "Режим навчання", матеріал і дві кнопки: "Завантажити", яка викликає функцію downloadFile, і "Вихід", яка викликає функцію onStartHome.

Оскільки в тестуванні і завданнях використовується таймер, то розглянемо його спочатку. Компонент Timer створює таймер, який можна включити та виключити, а також виводить час у форматованому вигляді.

```
function Timer({ isRunning, onStop }) {
 const [time, setTime] = useState(0);

 const formatTime = (totalMilliseconds) => {
  const minutes = Math.floor(totalMilliseconds / (60 * 100));
  const seconds = Math.floor((totalMilliseconds % (60 * 100)) / 100);

  return `${String(minutes).padStart(2, "0")}:
  ${String(seconds).padStart(2, "0")}`;
 };
```

Функція Timer отримує два параметри: isRunning (булеве значення, яке вказує, чи таймер повинен бути ввімкнений) та onStop (функція, яка буде викликана при зупинці таймера). Використовується хук стану useState для створення змінної time та функції setTime, яка дозволяє змінювати значення стану. Також визначається функція formatTime, яка конвертує час з мілісекунд у рядок у форматі "хвилини:секунди".

```
useEffect(() => {
 let intervalId;

 if (isRunning) {
  intervalId = setInterval(() => {
   setTime((prevTime)  => prevTime + 10);
  }, 100);
 }

 return () => {
  clearInterval(intervalId);
  onStop(formatTime(time));
 };
}, [isRunning, onStop, time]);
```

Використовується хук ефекту useEffect, який спрацьовує при зміні значень isRunning, onStop або time. Усередині ефекту визначається інтервал, який додає 10 мілісекунд до значення time кожні 100 мілісекунд, якщо isRunning встановлено в true. При зупинці таймера викликається функція onStop з передачею форматованого часу.

```
return (
  <div>
    <p>Час: {formatTime(time)}</p>
  </div>
);
```

Компонент повертає JSX, який відображає час у форматованому вигляді.

Розглянемо TestingMode і TaskMode, що відображають режими тестування і завдань відповідно.

```
function TestingMode({ onStartHome }) {
  const questions = [
    {
      question:
        "Яка команда використовується для виконання арифметичної операції додавання у мові Assembler?",
      instructions: "",
      options: ["ADD", "SUB", "MUL", "DIV"],
      correctAnswer: "ADD",
    },
    //Інші питання
  ];
  const [newQuestions, setNewQuestions] = useState([]);

  const [currentQuestion, setCurrentQuestion] = useState(0);
  const [userAnswer, setUserAnswer] = useState("");
  const [feedback, setFeedback] = useState("");
  const [rightAnswer, setRightAnswer] = useState(0);
  const [wrongAnswer, setWrongAnswer] = useState(0);
  const [timerValue, setTimerValue] = useState("");
  const [runTimer, setRunTimer] = useState(true);
```

У цьому розділі використовується хук стану useState для створення змінних стану компоненту. Зокрема, створені змінні для збереження нових питань (newQuestions), номеру поточного питання (currentQuestion),

відповіді користувача (userAnswer), зворотного зв'язку щодо відповіді (feedback), кількості правильних (rightAnswer) і неправильних (wrongAnswer) відповідей, значення таймера (timerValue), стану виконання таймера (runTimer), а також флагу для відображення/приховування зворотного зв'язку (hideFeedback). В змінній question оголошено перелік питань з варіантами відповіді і правильною відповіддю.

```
const handleStopTimer = (time) => {
  setTimerValue(time);
};
```

Функція handleStopTimer викликається, коли таймер зупиняється, та зберігає час у стані timerValue.

```
const handleNext = () => {
  if (userAnswer === questions[currentQuestion].correctAnswer) {
    setFeedback(<h4 className="green-text">Відповідь правильна!</h4>);
    setRightAnswer(rightAnswer + 1);
  } else {
    setFeedback(<h4 className="red-text">Відповідь неправильна!</h4>);
    setWrongAnswer(wrongAnswer + 1);
  }
  if (currentQuestion < questions.length - 1) {
    setUserAnswer("");
    const newQuestion = {
      question: questions[currentQuestion].question,
      instructions: questions[currentQuestion].instructions,
      options: questions[currentQuestion].options,
      userAnswer: userAnswer,
    };
    setNewQuestions([...newQuestions, newQuestion]);
    setCurrentQuestion(currentQuestion + 1);
  } else {
    setRunTimer(false);
    onStartHome("test",timerValue,rightAnswer,wrongAnswer);
  }
};
```

Функція handleNext відповідає за обробку відповідей користувача, виведення зворотного зв'язку, підрахунок правильних та неправильних відповідей, а також перехід до наступного питання чи завершення тесту.

У цьому блоці коду перевіряється, чи відповідь користувача (userAnswer) співпадає з правильною відповіддю (questions[currentQuestion].correctAnswer). В залежності від результату виводиться відповідне повідомлення у стан feedback та збільшується кількість правильних чи неправильних відповідей (rightAnswer або wrongAnswer).

Якщо currentQuestion менше, ніж кількість питань (questions.length - 1), то оновлюється стан для наступного питання. Зокрема, встановлюється порожня відповідь (setUserAnswer("")), створюється об'єкт нового питання (newQuestion) з інформацією про поточне питання та відповідь користувача, додається нове питання до списку newQuestions та оновлюється номер поточного питання (setCurrentQuestion(currentQuestion + 1)).

Якщо currentQuestion вже останнє, то зупиняється таймер (setRunTimer(false)) і викликається функція onStartHome, яка повертає користувача до головного меню та передає результати тестування ("test", timerValue, rightAnswer, wrongAnswer).

```
const div = useRef(null);
useEffect(() => {
 if(div.current) {
  div.current.scrollIntoView({ behavior: "smooth", block: "end" });
 }
}, [currentQuestion]);

const [hideFeedback, setHideFeedback] = useState(true);
useEffect(() => {
 const timeoutId = setTimeout(() => {
  setHideFeedback(false);
 }, 1500);

 return () => {
  clearTimeout(timeoutId);
  setHideFeedback(true);
 }
}, [feedback]);
```

У цьому розділі використовуються хуки useEffect для виконання певних дій при зміні значень currentQuestion та feedback. А саме,

автоматична прокрутка до останнього питання та відображення зворотного зв'язку.

```
return (
 <div className="main"  ref={div}>
 <QuestionList newQuestions={newQuestions} />
  {/* Вміст компонента */}
 </div>
);
```

У цьому фрагменті відбувається рендеринг JSX для компонента TestingMode. В компоненті відображається номер питання, саме питання, опції відповідей, таймер, зворотний зв'язок та кнопка для переходу до наступного питання (див. Додаток А).

```
function QuestionList({ newQuestions }) {
 return (
  <div className="main-test">
   {/* Вміст компонента */}
  </div>
 );
}
```

Цей компонент відображає інформацію про попередні питання та відповіді, які були задані під час тестування.

```
function TaskMode({ onStartHome }) {
 const tasks = [//Завдання];
 const [newTasks, setNewTasks] = useState([]);

 const [currentTask, setCurrentTask] = useState(0);
 const [userCommands, setUserCommands] = useState([]);
 const [feedback, setFeedback] = useState("");
 const [rightAnswer, setRightAnswer] = useState(0);
 const [wrongAnswer, setWrongAnswer] = useState(0);
 const [timerValue, setTimerValue] = useState("");
 const [runTimer, setRunTimer] = useState(true);
 const [isButtonDisabled, setIsButtonDisabled] =
useState(Array(tasks[0].commands.length).fill(false));
```

Відповідно до компоненту TestingMode в TaskMode спочатку оголошуються завдання, кожне з яких містить текстовий опис, фрагмент коду та правильний порядок виконання команд (див. Додаток А). Потім використовуються різні стани та хуки для відстеження поточного стану виконання завдань, введених користувачем команд, та збереження результатів.

Деякі функції обробки подій та ефекти аналогічні, тому їх не розглядаємо.

```
const handleButtonClick = (command, index) => {
  setUserCommands([...userCommands,command]);

  const updatedButtonStates = [...isButtonDisabled];
  updatedButtonStates[index] = true;
  setIsButtonDisabled(updatedButtonStates);
};
```

Цей код допомагає в управлінні станом вибору команд в режимі виконання завдань, запобігаючи повторному вибору та відключаючи вже вибрані кнопки.

```
const shuffleArray = (array) => {
  const shuffledArray = [...array];
  for (let i = shuffledArray.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [shuffledArray[i], shuffledArray[j]] = [shuffledArray[j], shuffledArray[i]];
  }
  return shuffledArray;
};
const [shuffledCommands, setShuffledCommands] = useState(shuffleArray(tasks[0].commands));
```

Після створення функції перемішування масиву, її результат використовується для ініціалізації стану shuffledCommands у компоненті за допомогою хука useState. Таким чином, на початку виконання компонента TaskMode масив команд із першого завдання перемішується, і результат зберігається у стані shuffledCommands. Це дає можливість представляти кнопки з перемішаним порядком команд для кожного завдання.

```
const handleNext = () => {
  if (userCommands.join("") === tasks[currentTask].commands.join("")) {
    setFeedback(<h4 className="green-text">Відповідь правильна!</h4>);
    setRightAnswer(rightAnswer + 1);
  } else {
    setFeedback(<h4 className="red-text">Відповідь неправильна!</h4>);
    setWrongAnswer(wrongAnswer + 1);
  }

  if (currentTask < tasks.length - 1) {
    setUserCommands([]);
    //setFeedback("");
    const newTask = {
      task: tasks[currentTask].task,
```

```
    code: tasks[currentTask].code,
    userAnswer: userCommands.join("\n"),
   };
   setNewTasks([...newTasks, newTask]);
   setCurrentTask(currentTask + 1);
   setShuffledCommands(shuffleArray(tasks[currentTask+1].commands));
   setIsButtonDisabled(Array(tasks[currentTask].commands.length).fill(false));
  } else {
   setFeedback("Ви завершили всі завдання!");
   setRunTimer(false);
   onStartHome("task",timerValue,rightAnswer,wrongAnswer);
  }
 };
```

Функція handleNext відповідає за обробку події "Далі" в режимі виконання завдань.

Визначається, чи відповідь користувача правильна чи ні. Для цього порівнюється рядок, який представляє вибрані команди користувача (userCommands.join("")) з рядком команд для поточного завдання (tasks[currentTask].commands.join("")).

Перевіряється, чи є ще завдання для виконання. Якщо так, тоді налаштовуються стани для переходу до наступного завдання, включаючи очищення вибраних користувачем команд, додавання інформації про поточне завдання до списку вже виконаних, оновлення завдання, яке відображається користувачу, перемішування команд для нового завдання та відновлення стану кнопок. Якщо всі завдання виконані, режим виконання завдань завершується, і викликається функція onStartHome для повернення до головного меню та передачі результатів.

```
return (
 <div className="main"  ref={div}>
  <h2>Режим виконання завдань</h2>
  <TaskList newTasks={newTasks} />
  {/* Вміст компонента */}
 </div>
);
```

У цьому блоці відбувається виведення на екран компонента, що містить усю інформацію про поточне завдання, таймер, зворотний зв'язок та кнопку для переходу до наступного завдання (див. Додаток А).

```
function TaskList({ newTasks }) {
 return (
  <div className="main-task">
   {/* Вміст компонента */}
  </div>
 );
}
```

Компонент TaskList виводить вже виконані завдання у форматі, який дозволяє користувачеві переглядати текст завдань та порівнювати відповіді.

4.2. Необхідна користувачу інструкція

На вкладці в браузері спочатку виводиться головний екран з темою тренажеру та інформацією про розробника (рис. 4.1).

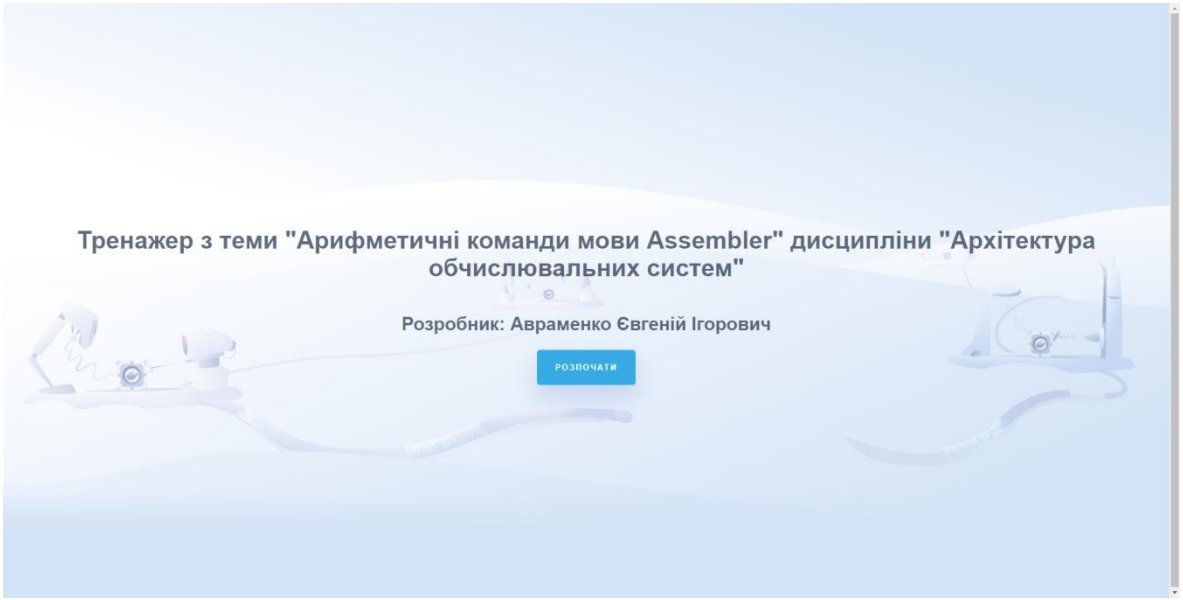

Рисунок 4.1 – Головний екран

Користувачу потрібно натиснути кнопку «Розпочати» для відображення кнопок з вибором режиму (рис. 4.2):

- Навчання;
- Тестування;
- Завдання.

Рисунок 4.2 – Вибір режиму

Щоб переглянути матеріал з даної теми слід вибрати режим «Навчання». При цьому виведеться відповідний вміст (рис. 4.3).

Рисунок 4.3 – Режим навчання

В кінці сторінки є дві кнопки: «Завантажити» і «Вихід» (рис. 4.4). Якщо користувач бажає зберегти цей матеріал з теми, то необхідно натиснути «Завантажити». Для повернення на головний екран служить «Вихід» .

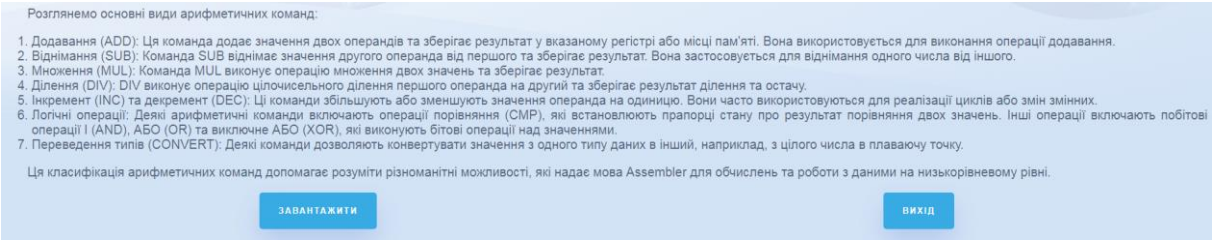

Рисунок 4.4 – Кнопки для завантаження файлу з матеріалами і виходу

Завантаження відбувається автоматично (рис. 4.5). Після цього файл можна знайти в завантаженнях браузера чи пристрою та відкрити його (рис. 4.6).

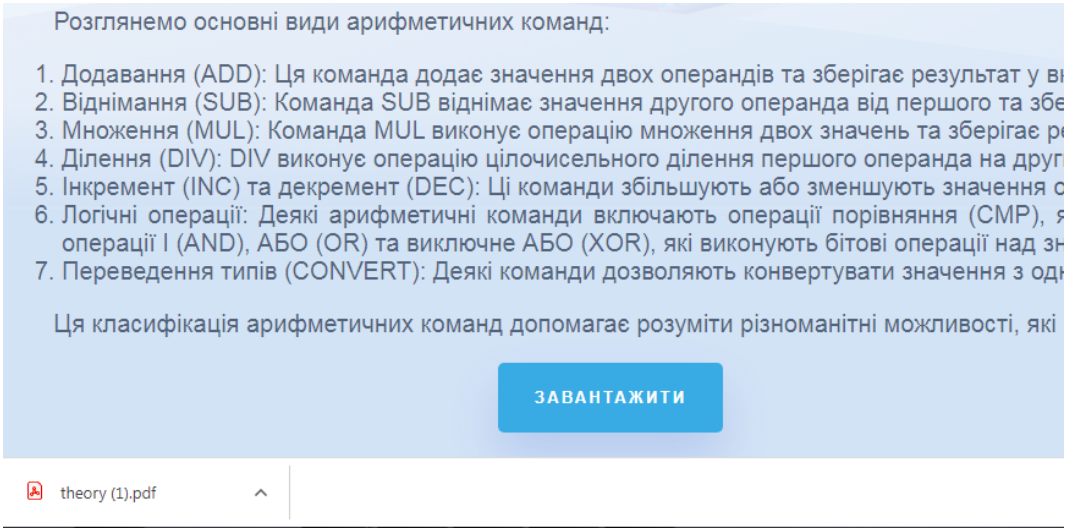

Рисунок 4.5 – Файл з матеріалами завантажено

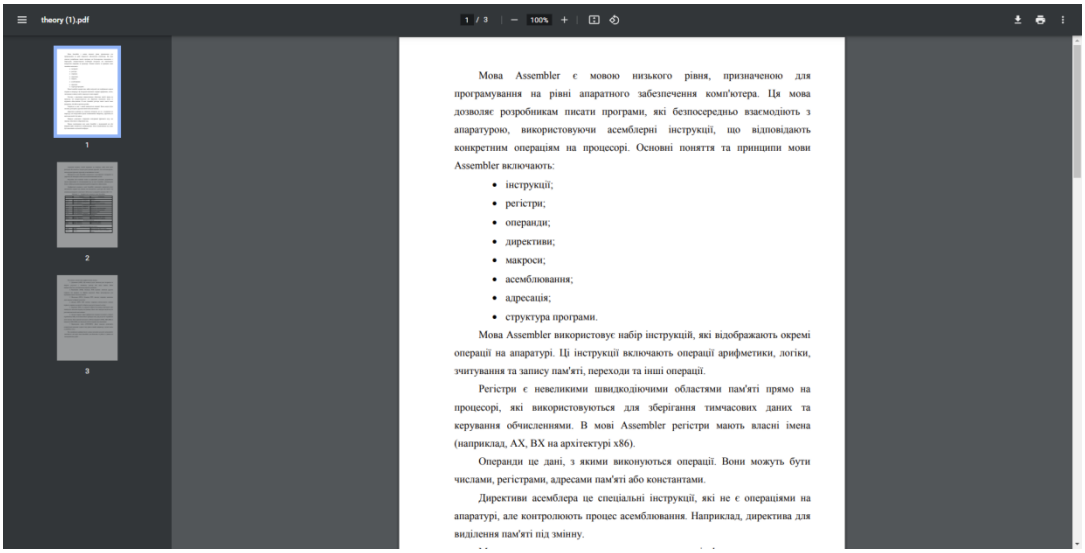

Рисунок 4.6 – Відкритий файл з матеріалами

Щоб перейти до тестування слід вибрати режим «Тестування». При цьому виведеться перше питання і варіанти відповіді до нього. В цей час також запускається таймер з часом (рис. 4.7).

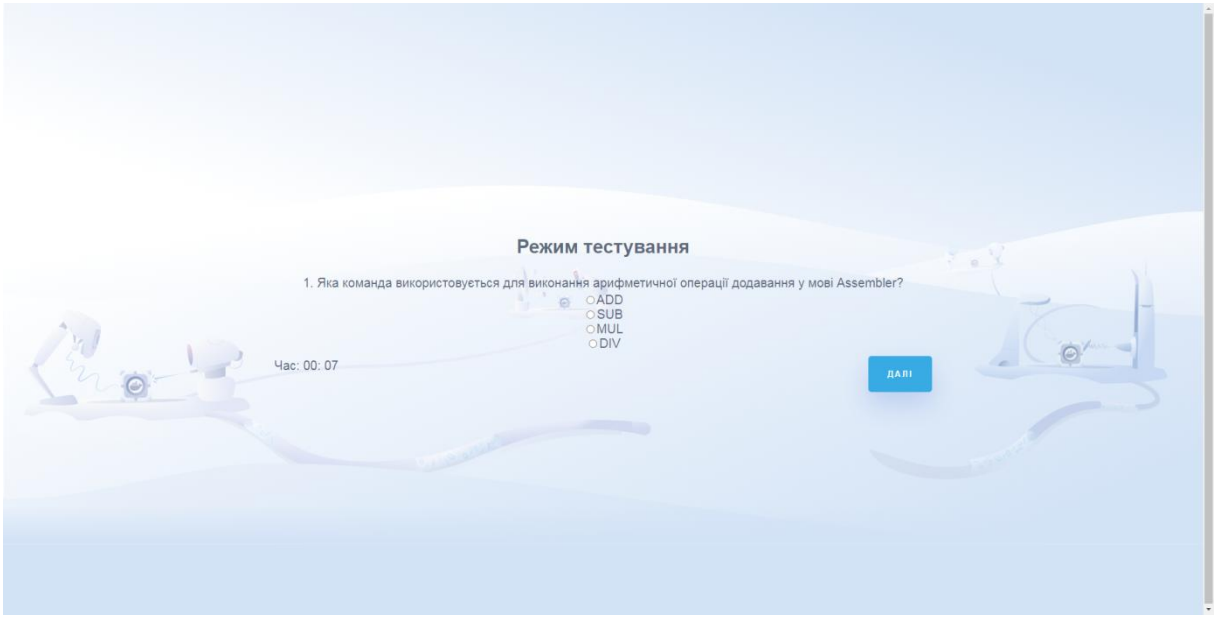

Рисунок 4.7 – Режим тестування

Вибирається один з варіантів і необхідно натиснути кнопку «Далі». Відповідь перевіряється і відображається повідомлення про те, правильна вона чи ні. Також виводиться наступне питання та блокується можливість змінити відповідь на попереднє (рис. 4.8).

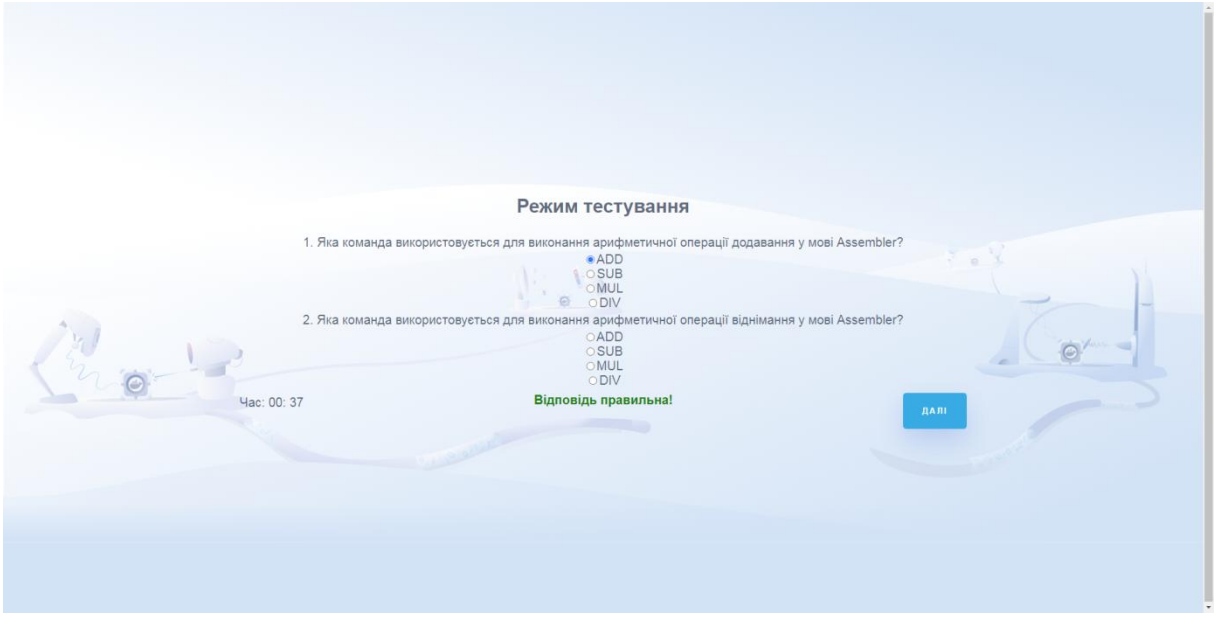

Рисунок 4.8 – Правильна відповідь на питання

Зразок повідомлення при неправильно вибраному варіанті :

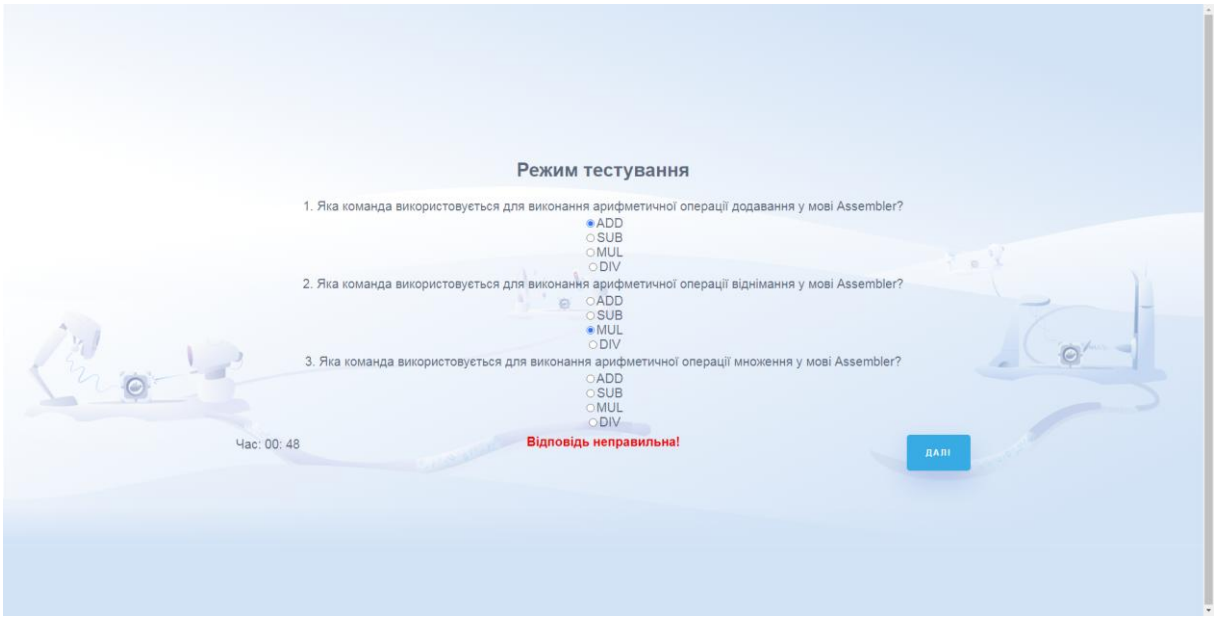

Рисунок 4.9 – Неправильна відповідь на питання

Якщо користувач відповів на всі питання, то відображається «Тестування завершено» (рис. 4.10).

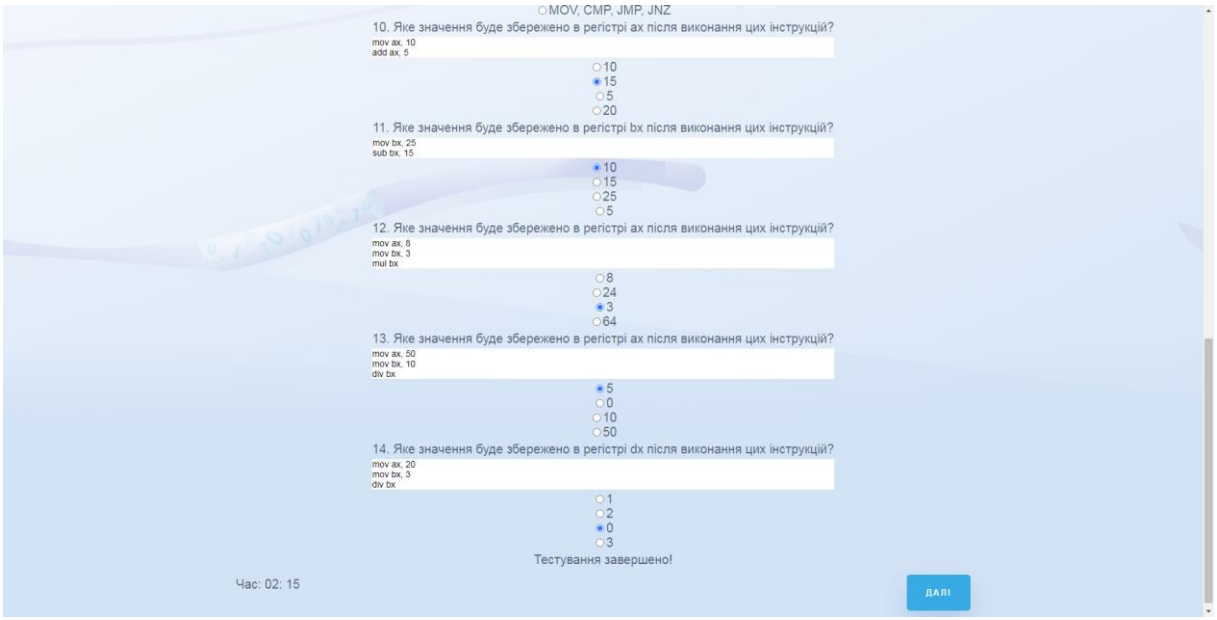

Рисунок 4.10 – Завершення тестування

Для переходу до оцінки та підсумку тестування потрібно натиснути кнопку «Далі». На сторінці виводиться (рис. 4.11):

- Витрачений час;

- Кількість правильних і неправильних відповідей;
- Оцінка користувача;
- Подальші рекомендації.

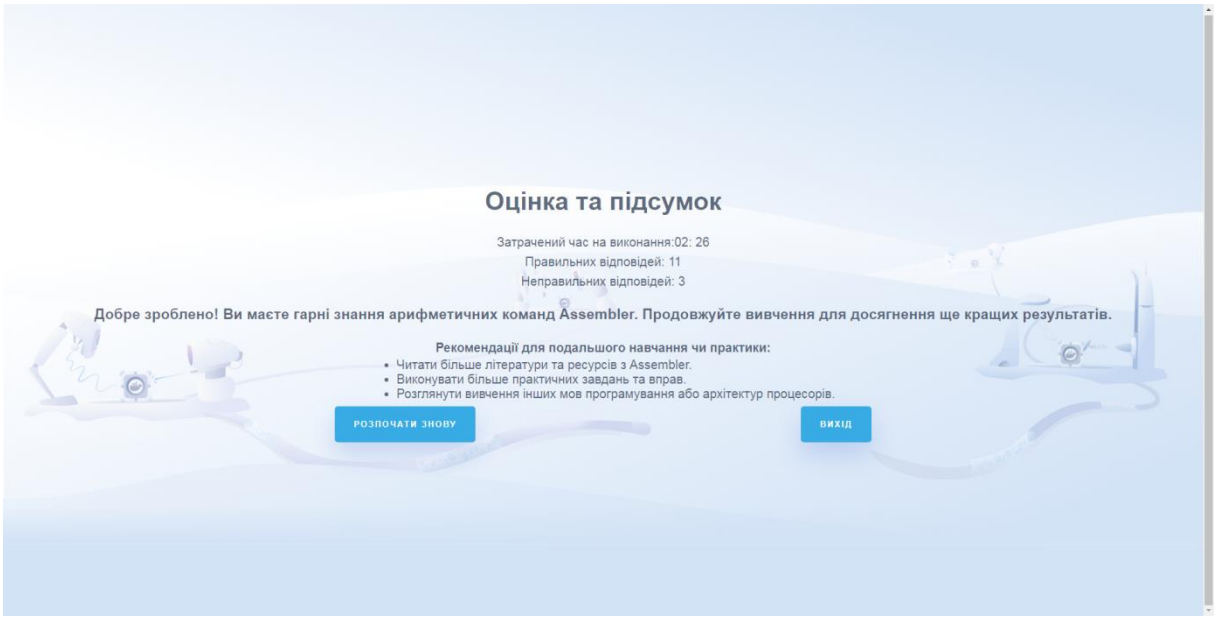

Рисунок 4.11 – Оцінка та підсумок тестування

Щоб заново пройти тестування слід вибрати «Розпочати знову», а щоб повернутися на головний екран – «Вихід».

Для переходу до виконання завдань необхідно вибрати режим «Завдання» (рис. 4.12).

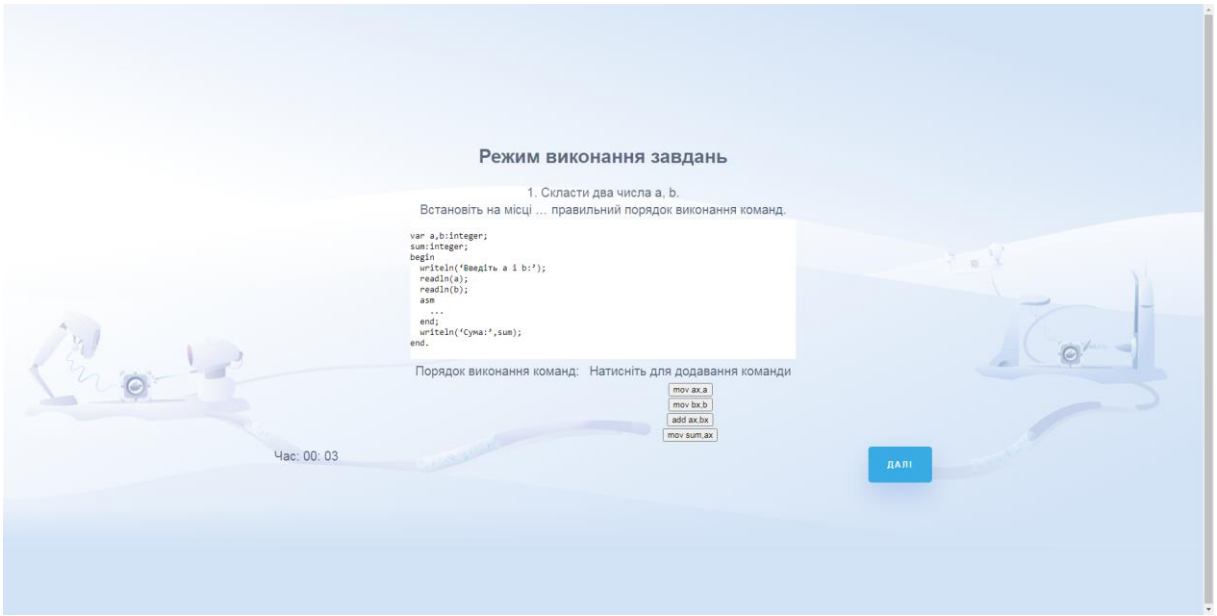

Рисунок 4.12 – Режим виконання завдань

Користувачу вказується завдання і необхідно встановити правильний порядок команд. Для їх додавання потрібно натиснути на відповідну кнопку з командою (рис. 4.13). В такий спосіб слід використати всі надані команди і натиснути «Далі».

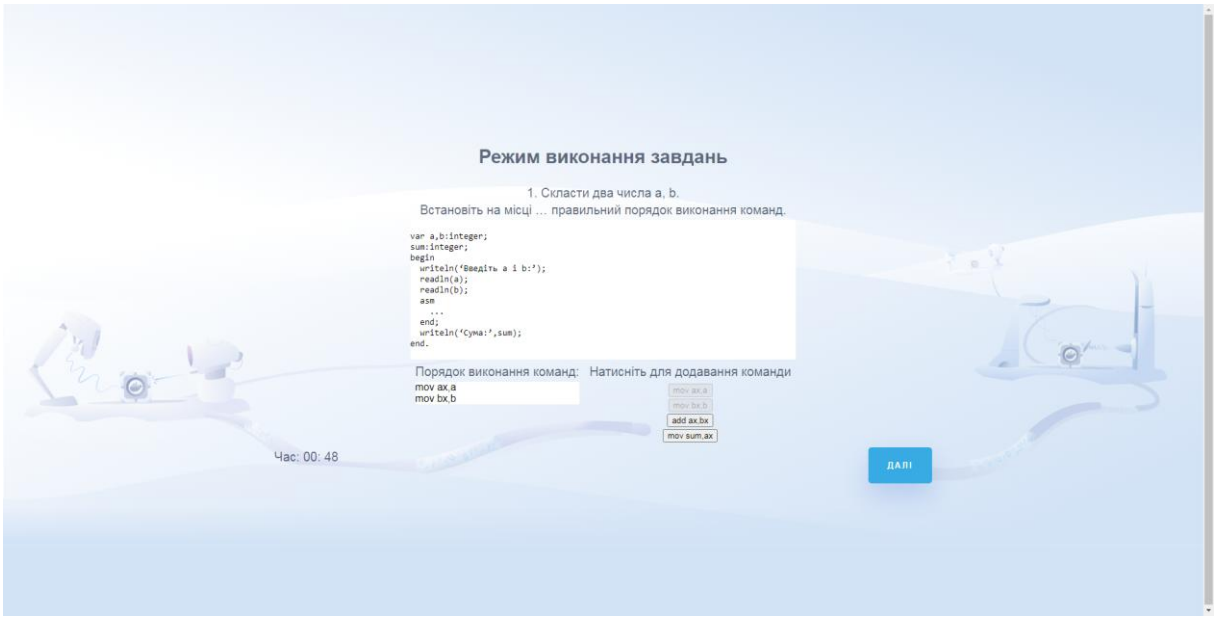

Рисунок 4.13 – Додавання команд до порядку виконання

Встановлений порядок перевіряється і відображається повідомлення про те, правильний він чи ні. Також виводиться наступне завдання та в попереднє (рис. 4.14).

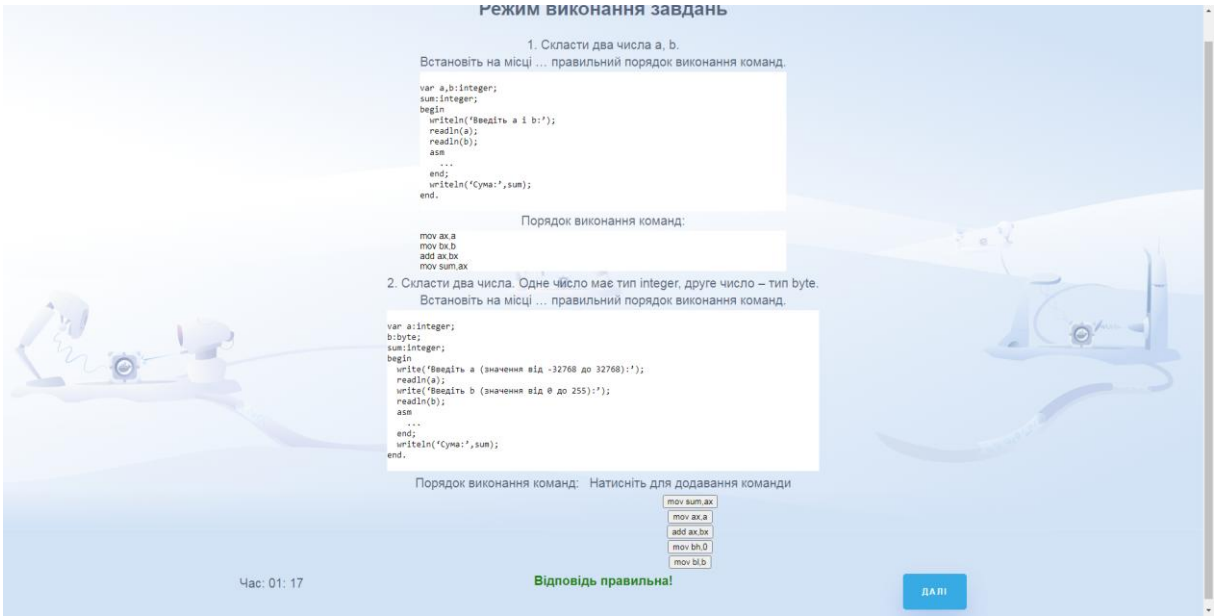

Рисунок 4.14 – Виведення наступного завдання

Якщо користувач відповів на всі завдання, то відображається «Ви завершили всі завдання» (рис. 4.15).

Рисунок 4.15 – Завершення виконання завдань

Для переходу до оцінки та підсумку виконання завдань потрібно натиснути кнопку «Далі». На сторінці виводиться відповідна інформація (рис. 4.16).

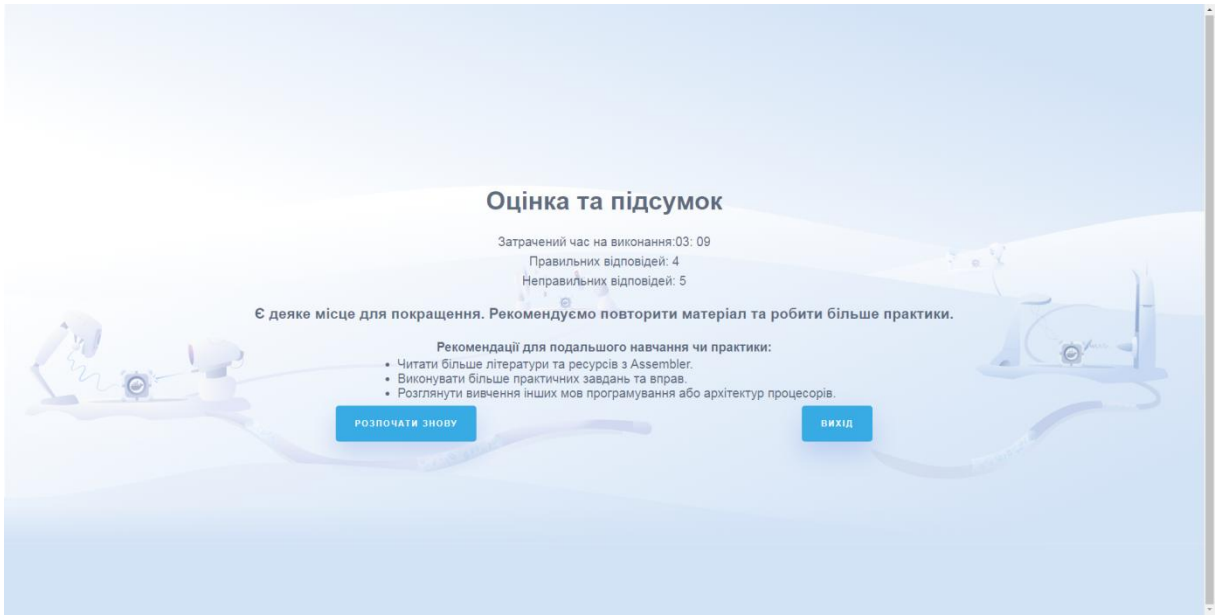

Рисунок 4.16 – Оцінка та підсумок виконання завдань

4.3. Тестування тренажеру арифметичним командам мови Assembler

Після реалізації тренажеру для навчання арифметичним командам мови Assembler важливо провести аналіз результатів, щоб визначити якість продукту, виявити проблеми та прийняти рішення щодо подальших дій.

Аналіз включає наступні кроки:

1. Виявлення помилок: Ретельно проаналізовано результати тестів, щоб виявити всі виявлені помилки, включаючи невідповідності функціональним вимогам, некоректну поведінку та інші проблеми.

2. Класифікація помилок: Помилки розділено на категорії відповідно до їх характеру. Наприклад, це можуть бути проблеми з інтерфейсом, арифметичними операціями, зберіганням даних тощо.

3. Пріоритети помилок: Визначено, які помилки мають найбільший вплив на користувачів або на загальну функціональність тренажеру. Встановлено пріоритети для виправлення помилок.

4. Корекція помилок: Розроблено план виправлення помилок, визначено, які конкретні кроки потрібно вжити для виправлення кожної помилки.

5. Ретестування: Після виправлення помилок проведено ретестування, щоб перевірити, чи були вони успішно виправлені та чи не виникли нові проблеми.

6. Повторний аналіз: Проаналізовано результати повторного тестування та порівняно їх з попередніми результатами. Визначено, чи були виправлені всі виявлені помилки та чи відповідає продукт функціональним вимогам.

7. Оцінка якості: Здійснено оцінку якості тренажеру на основі результатів тестування. Визначено, що він відповідає вимогам та якісно функціонує.

8. Прийняття рішень: На підставі аналізу результатів тестування продукт готовий до впровадження. Вже після подальшого використання тренажеру викладачами і студентами можна буде дізнатися, чи не потребує він додаткових виправлень та покращень.

Аналіз результатів тестування є ключовим етапом перед впровадженням тренажеру. Він допомагає забезпечити якість та надійність продукту, а також забезпечити задоволення користувачів від його використання.

ВИСНОВКИ

У ході розробки тренажеру з вивчення арифметичних команд мови Assembler було проведено значне дослідження алгоритмів та методів реалізації, яке включало в себе аналіз існуючих рішень, визначення вимог до програми, розробку архітектури, вибір технологій, програмування та тестування.

Під час аналізу існуючих тренажерів для вивчення мови Assembler було виявлено, що більшість з них надають можливість вивчати та тестувати загальні поняття мови Assembler, але дещо обмежені в розмаїтості вправ та завдань, особливо у режимі виконання арифметичних команд. Також, багато існуючих рішень мають застарілий інтерфейс та обмежені можливості інтерактивності.

Це дозволило виявити переваги та недоліки різних продуктів.

Основним результатом роботи стало розроблення тренажеру, який дозволяє користувачам не тільки ознайомитися з арифметичними командами мови Assembler, а й практично виконувати їх у режимі навчання та тестування. Під час використання тренажеру користувач може вчитися вказувати правильний порядок команд та перевіряти свої знання.

Важливо відзначити, що даний тренажер може бути корисним для студентів та всіх, хто цікавиться навчанням асемблерної мови.

У підсумку, ця робота підтвердила важливість розробки тренажерів для навчання асемблерної мови програмування, зокрема арифметичних команд, та надала користувачам зручні та ефективні засоби для вивчення даної теми.

У ході реалізації тренажеру з вивчення арифметичних команд мови Assembler було досягнуто поставлені цілі та завдання:

- аналіз та порівняння існуючих рішень;
- розробка алгоритму та блок-схеми для програмування;
- розробка функціоналу для навчання;

- реалізація тестування;
- реалізація практичних завдань;
- інтерактивний інтерфейс;
- тестування розробленого продукту і виправлення помилок.

Режим тестування дозволяє перевірити знання користувача щодо арифметичних команд. Користувач може вибрати правильну відповідь і отримує негайний зворотний зв'язок про правильність вибору.

Режим виконання практичних завдань дозволяє користувачу активно застосовувати знання, виконуючи арифметичні операції у правильному порядку. Після виконання завдання користувач отримує зворотний зв'язок щодо правильності виконання.

Розроблений тренажер має інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам легко навігувати та взаємодіяти з різними режимами навчання та завдань.

Використання тренажеру може позитивно вплинути на розуміння та вміння користувачів виконувати арифметичні команди мови Assembler, що є одним із головних показників успішності навчального процесу.

Розробка тренажеру для вивчення арифметичних команд мови Assembler є лише початком у великій галузі навчання обчислювальних систем та програмування. Дана робота відкриває широкий простір для подальших досліджень та розвитку.

Подальші дослідження можуть спрямовуватися на вдосконалення і розширення функціоналу, вдосконалення інтерактивності, а також дослідження ефективності і практичності таких тренажерів у сучасному навчальному процесі та практиці програмістів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Строкань О. В. Комп’ютерна схемотехніка та архітектура комп’ютерів. / О. В. Строкань, С. М. Прийма, Ю. О. Литвин - Мелітополь: ТДАТУ, 2019. 186 с.
2. Матвієнко М. П. Комп’ютерна логіка [Текст]: навч. Посібник. К.: Видавництво Ліра-К, 2012. 288 с.
3. Інформатика. Комп’ютерна техніка. Комп’ютерні технології: Підручник для ВНЗ / За ред. О.І. Пушкаря. - К.: Академія, 2003. - 704 с.
4. Дистанційний курс «Архітектура обчислювальних систем» [Електронний ресурс] // Сайт дистанційного навчання, Полтавський університет економіки і торгівлі. – Режим доступу: https://el.puet.edu.ua/
5. Волошин, В. В. Основи асемблерної мови: Навчальний посібник. Національний технічний університет України "Київський політехнічний інститут". – 2012
6. Богданов, О. В.(). Основи програмування на асемблері. / Богданов, О. В., Гусєва, Н. В. Видавництво "Літера ЛТД". – 2010
7. React documentation [Електронний ресурс]. – Режим доступу: https://reactjs.org/docs/getting-started.html
8. JavaScript MDN Web Docs. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
9. W3Schools tutorials. [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/

ДОДАТОК А. КОД ПРОГРАМИ