ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ БІЗНЕСУ ТА СУЧАСНИХ ТЕХНОЛОГІЙ

ФОРМА НАВЧАННЯ ДЕННА
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____О.О. Ємець
(підпис)

«_____»_____2021 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДО ДИПЛОМНОЇ РОБОТИ

на тему

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ТРЕНАЖЕРУ З ТЕМИ «АРИФМЕТИЧНІ КОМАНДИ МОВИ ASSEMBLER» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «АРХІТЕКТУРА ОБЧИСЛЮВАЛЬНИХ СИСТЕМ»

зі спеціальності 122 «Комп'ютерні науки»

Виконавець роботи Пильник Сергій Олександрович
_____ «___»_____2021р.
(підпис)
Науковий керівник доц., каф.-мат. наук, Чілікіна Т.В.
_____ «___»_____2021р.
(підпис)

ПОЛТАВА 2021р.

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

ЗАТВЕРДЖУЮ

Завідувач кафедри_____О.О. Ємець

« 8 **»** вересня 2020р.

Завдання та календарний графік
виконання дипломної роботи

Студент(ка) спеціальності 122 «Комп'ютерні науки»
Прізвище, ім'я, по батькові Пильник Сергій Олександрович

1. Тема **Алгоритмізація та програмна реалізація тренажеру з теми «Арифметичні команди мови Assembler » дистанційного навчального курсу «Архітектура обчислювальних систем»** затверджена наказом ректора № 121-Н від « 1 » вересня 2020 р.

Термін подання студентом бакалаврської роботи « 20 » травня 2021 р.

2. Вихідні дані до дипломної роботи: публікації з теми «Навчальні тренажери» в дистанційних курсах з комп'ютерних наук.

3. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП

1 Постановка задачі

1.1 Змістовна постановка задачі

2 Інформаційний огляд

2.1. Огляд робіт зі схожим завданням

2.2. Позитивні аспекти оглянутих робіт

2.3. Вади розробок з оглянутих робіт

3 Теоретична частина

3.1 Загальні відомості про арифметичні команди мови Assembler

3.2 Алгоритм роботи тренажера

3.3 Блок-схема програми-тренажера

4 Практична частина

4.1 Опис програмного забезпечення

4.2 Інструкція користувача

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТОК А

4. Перелік графічного матеріалу: 3 аркуші блок-схем, інші необхідні ілюстрації.

5. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1. Постанова задачі	Чілікіна Т.В.	8.09.20	8.09.20
2. Інформаційний огляд	Чілікіна Т.В.	8.09.20	8.09.20
3. Теоретична частина	Чілікіна Т.В.	8.09.20	8.09.20
4. Практична реалізація	Чілікіна Т.В.	8.09.20	8.09.20

6. Календарний графік виконання бакалаврської роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ	10.05.21	
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику	15.09.20	
3. Постановка задачі	1.10.20	
4. Інформаційний огляд джерел бібліотек та інтернету	2.11.20	
5. Теоретична частина	1.02.21	
6. Практична частина	17.05.21	
7. Закінчення оформлення	21.05.21	
8. Доповідь студента на кафедрі	28.05.21	
9. Доробка (за необхідністю), рецензування	14.06.21	

Дата видачі завдання « 8 » __вересня__ 2020 р.

Студент Пильник Сергій Олександрович

Науковий керівник _____ доц., каф.-мат. наук, Чілікіна Т.В.
(підпис)

Результати захисту бакалаврської роботи

Дипломна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2021 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Записка: 46 стор., в т.ч. основна частина 41 стор., джерел - 10.

Предмет розробки – предметом розробки є програма-тренажер для навчання темі «Арифметичні команди мови Асемблер». Для її створення було використання платформу Unity 2020, середовище розробки MS Visual Studio.

Мета роботи – створити програму-тренажер для студентів на тему «Арифметичні команди мови Асемблер» з дисципліни «Архітектура обчислювальних систем».

Методи роботи – бакалаврську роботу було розроблено за допомогою підтримки платформи Unity 2020, та методичних вказівок до практичних і лабораторних робіт з дисциплін «Комп'ютерні технології та програмування», «Мови програмування». Для створення програми було використано мову програмування C# та IDE MS Visual Studio.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, символів, скорочень
ДК	Дистанційний курс.
ДН	Дистанційне навчання.
Assembler	Мова програмування.
Unity	Платформа для програмної реалізації.
IDE MS Visual Studio	Середовище для написання програмного коду.

ВСТУП

В умовах сучасності важливо швидко навчатися чомусь новому за будь-яких умов та можливостей, тому важливо мати доступ до навчального матеріалу в зручному варіанті, такому як телефон чи просто в браузері, або завантажити собі на персональний комп'ютер. Для цього були сформовані зручні прогарами для дистанційного навчання такі як: онлайн лекції, тренінги, «програми-тренажери», та інші види.

Зосередимося на «програмах-тренажерах». Саме вони, а допомогою свого функціоналу вважаються одними із найзручніших та швидких способами для самостійного та дистанційного навчання.

До їх функціоналу відносяться:

1. Робота з навчальним матеріалом за допомогою широкого спектру пристроїв (комп'ютери, телефони, тощо);
2. Можливість отримати дистанційну консультацію стосовну матеріалу тренажера;
3. Швидке оновлення бібліотек навчальних матеріалів;
4. Подання навчального матеріалу в будь-якому вигляді (реферат, книга, відео-урок, гра та ін.);
5. Доступ до веб-бібліотек;
6. Можливість розвивання навчальних інтернет-ресурсів;
7. Доступність для осіб з фізичними вадами;
8. Можливість розвивати педагогічні технології та навчання їх викладачів;
9. Підвищення ефективності навчання завдяки гнучкого графіку, та загальної зручності;

Тренажер може використовуватися у будь-якому місці та часі, а в випадку якщо в програми в наявності усі навчальні матеріали то можна користуватись і без доступу до мережі Інтернет.

Мета роботи – створити програму-тренажер для студентів на тему «Арифметичні команди мови Асемблер» з дисципліни «Архітектура обчислювальних систем».

Об’єкт роботи – створення програмі-тренажеру для системи дистанційного навчання.

Предмет роботи – предметом розробки є програма-тренажер для навчання темі «Арифметичні команди мови Асемблер». Для її створення було використання платформу Unity 2020, середовище розробки MS Visual Studio.

Методи роботи – бакалаврську роботу було розроблено за допомогою підтримки платформи Unity 2020, та методичних вказівок до практичних і лабораторних робіт з дисциплін «Комп’ютерні технології та програмування», «Мови програмування». Для створення програми було використано мову програмування C# та IDE MS Visual Studio.

Під час пошуку в репозиторії dspace та в мережі інтернет не було знайдено схожих на цю тему тренажерів, а їх існування значним чином підвищить рівень засвоєння даної теми, в цьому і полягає актуальність даної роботи і необхідність створити тренажер з теми «Арифметичні команди мови Assembler» дистанційного навчального курсу.

Структура пояснювальної записки до бакалаврської записки:

- Титульний аркуш;
- Завдання до бакалаврського проекту;
- Реферат, що містить предмет, мету, методи, анотацію результатів ключові слова, словосполучення;
- Зміст;
- Вступ;
- Суть роботи;
- Висновки;
- Рекомендації;
- Список використаних джерел;

Обсяг пояснювальної записки: 46 стор., в т.ч. основна частина 41 стор., джерел - 10.

1 Постановка задачі

1.1 Змістовна постановка задачі

Головною задачею бакалаврської роботи є програмна реалізація тренажера з теми «Арифметичні команди мови Асемблер» дисципліни «Архітектура обчислювальних систем».

Дана програма буде створена у середовищі розробки MS Visual Studio, з використанням мови програмування C#.

Головною задачею програми у вигляді тренажеру буде навчання студентів «Арифметичні команди мови Асемблер»

Основними завданнями даної бакалаврської роботи є:

- Вибір мови програмування та програмної реалізації для зручної роботи з дистанційним курсом;
- Складання основного алгоритму роботи тренажера;
- Складання блок-схеми алгоритму роботи програми;
- Програмна реалізація тренажеру;
- Перевірка та тестування програми;

Основними вимогами до програмного забезпечення є:

- Під час навчання користувач повинен мати доступ до завдань практичного вигляду з будь-якого робочого місця;
- Під час вибору відповіді в практичній частині необхідно реалізувати механізм для перевірки даних на помилку, та вивести повідомлення згідно до проблеми;
- При вводі неправильної відповіді реалізувати повернення до теоретичного завдання, де буде описано типове завдання;
- Роботу тренажеру необхідно поділити на блоки для зручності;
- Навігацію між блоками необхідно виконати через відповідні меню.

2 Інформаційний огляд

2.1. Огляд робіт зі схожим завданням

Після пошуку тренажерів схожих за темою було знайдено такі тренажери: Мороз Артур Вадимович тренажер з теми «Оптимізація виробництва столів: програмна реалізація тренажера (моделювання)» [1], Григор'єв Владислав Віталійович тренажер з теми «Побудова математичних моделей лінійних задач планування виробництва» [2], Голубенко Роман Володимирович тренажер з теми «Метод Дальтона-Ллевеліна» [3].

2.2. Позитивні аспекти оглянутих робіт

У роботі Мороза Артура Вадимовича було знайдено такі плюси:

1. Перевірка введеної відповіді;
2. Підказка користувачу при виборі неправильної відповіді;
3. Умова завжди доступна користувачу;
4. Є кнопка повтору тренажера.

У роботі Григор'єва Владислава Віталійовича було знайдено такі плюси:

1. Зручний інтерфейс;
2. Перевірка введеної відповіді;
3. Підказка користувачу при виборі неправильної відповіді.

У роботі Голубенко Романа Володимировича було знайдено такі плюси:

1. Наявність тестових завдань та завдань з полями для вводу;
2. Наявність можливості правильного заповнення полей для вводу;
3. Наявність прикладу з правильним заповненням під час роботи.

2.3. Вади розробок з оглянутих робіт

У роботі Мороза Артура Вадимовича було знайдено такі мінуси:

1. Не зручний дизайн, що заважає роботі;

У роботі Григор'єва Владислава Віталійовича було знайдено такі мінуси:

1. Немає можливості повторити роботу з тренажером без перезавантаження застосунку;

У роботі Голубенко Романа Володимировича було знайдено такі мінуси:

1. Замість кнопки правильного заповнення краще реалізувати підказку з поясненням.

3 Теоретична частина

3.1 Загальні відомості про арифметичні команди мови Assembler

Арифметичні операції дозволяють проводити арифметичні дії в заданих задачах в будь-якій мові програмування, арифметичні ді являються одними з основних дій в програмуванні. Нажаль мова Асемблер являється низькою мовою програмування процесорів, тому це зумовило те що ця мова стала доволі складною та одночасно простою, на приклад в Асемблер для звичайних арифметичних дій у нас є більше десяти команд, для різних дій таких як додавання наприклад є такі команди для додавання як ADD та ACD являются командами для додавання.

В даній бакалаврській роботі було розглянуто основні арифметичні команди мови асемблер.

- **Синтаксис команди ADD**: ADD СУМА, ЧИСЛО. За допомогою цієї команди можливо додати два числа: СУМА и ЧИСЛО додаються, а результат поміщається в СУММУ. Ця команда може змінювати прапори (в залежності від результату - але про це трохи пізніше). ЧИСЛОМ може бути одне з цих: Область пам'яті (MEM) Регістр загального призначення (REG) Безпосереднє значення (наприклад, число) (IMM) СУМОЙ може бути одна із таких: Область пам'яті (MEM) Регістр загального призначення (REG) Тобто ця команда не працює з сегментними регістрами. Комбінація СУМА-ЧИСЛО можуть бути такими:

REG, MEM

MEM, REG

REG, REG

MEM, IMM

REG, IMM

Приклад використання інструкції ADD:

```
MOV AL, 5   ; AL = 5
ADD AL, -3  ; AL = 2
```

А тепер про прапори.

Команда ADD в залежності від результату виконання може змінювати всі основні прапори, а саме:

CF - прапор переносу. Цей прапор може бути змінений, наприклад, при переповненні.

ZF - прапор нуля. Цей прапор буде встановлено, якщо результатом складання буде нуль

SF - прапор знаку. Цей прапор буде встановлено, якщо результатом складання буде негативне число.

OF - прапор переповнення. Цей прапор буде встановлено в разі переповнення (тобто коли результат не поміщається в СУМУ - значення занадто велике).

SUB *операнд1, операнд2*

Відняти від операнда1 операнд2 і записати результати до операнда1, не змінюючи операнд2. Операнд2 може бути числом, регістром чи змінною, операнд1 –регістром чи змінною:

SUB EAX,X *//Зменшити значення регістра EAX на значення змінної X*

SUB X,2 *//Зменшити значення змінної X на*

2SUB EAX,ECX *//Зменшити значення регістра EAX на значення регістра ECX*

- **Інструкція MUL** в Асемблері виконує множення без знаку.

Зрозуміти роботу команди MUL дещо складніше, ніж це було для команд,

розглянутих раніше. Але, сподіваюся, що я допоможу вам у цьому розібратися.

Отже, синтаксис команди MUL такий:

MUL ЧИСЛО

Виглядає все дуже просто. Однак ця простота оманлива.

Перш ніж розібратися в подробицях роботи цієї інструкції, давайте

подивимося, що може бути ЧИСЛОМ.

ЧИСЛОМ може бути один з наступних:

Область пам'яті (MEM)

Регістр загального призначення (REG)

Ця команда не працює з сегментними регістрами, а також не працює безпосередньо з числами. Тобто ось так

MUL 200; неправильно

робити не можна.

А тепер алгоритм роботи команди MUL:

Якщо ЧИСЛО - це БАЙТ, то AX = AL * ЧИСЛО

Якщо ЧИСЛО - це СЛОВО, то (DX AX) = AX * ЧИСЛО

Ось така трохи важча команда. Хоча складно це з незвички.

Зараз ми розберемо все "по кісточках" і все стане ясно.

Для початку зверніть увагу, що інструкція MUL працює або з регістром AX, або з регістром AL.

Тобто перед виконанням цієї команди нам треба записати в регістр AX або в регістр AL значення, яке буде брати участь в множенні. Зробити це можна, наприклад, за допомогою вже відомої нам команди MOV.

Потім ми виконуємо множення, і отримуємо результат або в регістр AX (якщо ЧИСЛО - це байт), або в пару регістрів DX і AX (якщо ЧИСЛО - це слово). Причому в останньому випадку в регістрі DX буде старше слово, а в регістрі AX - молодше.

А тепер, щоб зовсім все стало зрозуміло, розберемо кілька прикладів - з байтом і словом.

Отже, наприклад, нам треба помножити 150 на 250. Тоді ми робимо так:

MOV AL, 150; Перший множник в регістр AL

MOV BL, 250; Другий множник в регістр BL

MUL BL; Тепер AX = 150 * 250 = 37500

Зверніть увагу, що нам доводиться два рази використовувати команду MOV, так як команда MUL не працює безпосередньо з числами, а тільки з

регістрами загального призначення або з пам'яттю.

Після виконання цього коду в регістрі AX буде результат множення чисел 150 і 250, тобто число 37500 (927C в шістнадцятковій системі).

Тепер спробуємо помножити 10000 на 5000.

MOV AX, 10000; Перший множник в регістр AX

MOV BX, 5000; Другий множник в регістр BX

MUL BX; Тепер (DX AX) = 10000 * 5000 = 50000000

В результаті ми отримали досить велике число, яке, звичайно, не поміститься в слово. Тому для результату використовуються два регістри - DX і AX. У нашому прикладі в регістрі DX, буде число 762 (02FA - в шістнадцятковій системі), а в регістрі AX - число 61568 (F080 - в шістнадцятковій системі). А якщо розглядати їх як одне число (подвійне слово), де в старшому слові 762, а в молодшому - 61568, то це і буде 50000000 (2FAF080 - в шістнадцятковій системі). Якщо не вірите - може перевести все це в двійкове число і перевірити. Тепер про прапори.

Після виконання команди MUL стану прапорів ZF, SF, PF, AF не визначені і можуть бути будь-якими.

А якщо старша секція результату (реєстр AH при множенні

байтів або регістр DX при множенні слів) дорівнює нулю, то

CF = OF = 0

Інакше ці прапори або не рівні, або рівні 1.

В кінці як завжди розповім, чому ця команда асемблера називається MUL. Це скорочення від англійського слова MULTIPLY, яке можна перевести як "помножити, множити".

Інструкція SUB теж досить проста для розуміння. Якщо інструкція ADD може додавати, то команда SUB процесорів сімейства i80x86 використовується для вирахування. Синтаксис команди SUB такий:

- **SUB** РІЗНИЦЯ, ЧИСЛО

За допомогою цієї команди можна з РІЗНИЦІ відняти ЧИСЛО. Результат поміщається в РІЗНИЦЯ.

Ця команда може змінювати прапори (в залежності від результату). Прапори змінюються таким же чином, як при виконанні команди ADD. Читайте про це тут.

ЧИСЛОМ може бути один з наступних:

Область пам'яті (MEM)

Регістр загального призначення (REG)

Безпосереднє значення (наприклад, число) (IMM)

Різниці може бути один з наступних:

Область пам'яті (MEM)

Регістр загального призначення (REG)

Ця команда, також як і команда складання, не працює з сегментними регістрами. Комбінації РІЗНИЦЯ-ЧИСЛО можуть бути наступними:

REG, MEM

MEM, REG

REG, REG

MEM, IMM

REG, IMM

Наприклад, подальшу поведінку вашої програми може залежати від того, який результат отримано після виконання команди SUB. В такому випадку треба перевірити стан прапора SF, і в залежності від його стану перейти на ту чи іншу ділянку програми за допомогою міток і спеціальних інструкцій. Але ці інструкції ми поки не вивчали, тому в даній статті я про них говорити не буду. Чекайте наступних уроків - підписуйтесь на новини сайту, щоб нічого не пропустити.

Інструкція DIV в Асемблері виконує ділення без знака. Використання цієї інструкції схоже на роботу команди MUL, хоча, звичайно, має деякі особливості, тому що поділ - це не множення)))

Отже, синтаксис команди DIV такий:

DIV ЧИСЛО

ЧИСЛОМ може бути один з наступних:

Область пам'яті (MEM)

Регістр загального призначення (REG)

Ця команда не працює з сегментними регістрами, а також не працює безпосередньо з числами. Тобто ось так

DIV 200; неправильно, так робити не можна.

А тепер алгоритм роботи команди DIV:

Якщо ЧИСЛО - це БАЙТ, то AL = AX / ЧИСЛО

Якщо ЧИСЛО - це СЛОВО, то AX = (DX AX) / ЧИСЛО

Якщо ви вже вивчили інструкцію MUL, то нічого особливо нового для вас тут немає. Ну а якщо не вивчали, то трохи нагадаю.

Зверніть увагу, що інструкція DIV працює або з регістром AX, або з парою регістрів DX AX. Тобто перед виконанням цієї команди нам треба записати в регістр AX або пару регістрів DX AX значення, яке потрібно розділити. Зробити це можна, наприклад, за допомогою вже відомої нам команди MOV.

Потім треба в область пам'яті або в регістр загального призначення записати дільник - тобто число, на яке будемо ділити.

Далі ми виконуємо поділ, і отримуємо результат або в регістр AL (якщо ЧИСЛО - це байт), або в регістр AX (якщо ЧИСЛО - це слово).

Залишок від ділення

Як ви розумієте, інструкція DIV виконує цілочисельне ділення. При цьому залишок від ділення, якщо такий є, буде записаний:

У регістр AH, якщо ЧИСЛО - це байт

У регістр DX, якщо ЧИСЛО - це слово

Ніякі прапори при цьому не змінюються. А якщо і змінюються, то про це нічого не сказано в документації, отже, перевіряти прапори немає необхідності.

Просто якщо є сумніви, що розподіл виконано без залишку, треба перевірити вміст регістрів AL або DX в залежності від того, який розмір має ЧИСЛО.

Приклад розподілу в Асемблері

Отже, например, нам треба 250 розділіті на 150. Тоді ми робимо так:

MOV AX, 250; Ділене в РЕГІСТР AX

MOV BL, 150; Дільнік в РЕГІСТР BL

DIV BL; Тепер AL = 250/150 = 1, AH = 100

Зверніть увагу, что нам доводитися два рази використовувати команду MOV, так як команда DIV Не працює безпосередно з числами, а только з регістрамі Загальне призначення або з якихось пам'яттю.

После Виконання цього коду в регістрі AL буде результат цілочісельного ділення числа 250 на число 150, тобто число 1, а в регістрі AH буде Залишок від ділення - число 100 (64 в шістнадцятковій системі).

Тепер Спробуємо число 50000000 розділити на 60000.

MOV DX, 762; Ділене - в пару регістрів DX AX

MOV AX, 61568; (DX AX) = 50000000

MOV BX, 60000; Дільнік в РЕГІСТР BX

DIV BX; Тепер AX = 50000000/60000 = 833 (341h)

; DX = 20000 (4E20h)

Для запису подільника в пару регістрів DX і AX використовуються дві команди MOV. У нашому прикладі в регістр DX буде записано число 762 (02FA - в шістнадцятковій системі), а в регістр AX - число 61568 (F080 - в шістнадцятковій системі). А якщо розглядати їх як одне число (подвійне слово), де в старшому слові 762, а в молодшому - 61568, то це і буде 50000000 (2FAF080 - в шістнадцятковій системі).

Потім в регістр BX ми записуємо число 60000 і виконуємо команду ділення. В результаті в регістрі AX буде число 833 (або 341 в шістнадцятковій системі), в регістрі DX - залишок від ділення, який в нашому випадку буде дорівнює 20000 (або 4E20 в шістнадцятковій системі).

В кінці як завжди розповім, чому ця команда асемблера називається DIV. Це скорочення від англійського слова DIVIDE, яке можна перевести як "розділити".

3.2 Алгоритм роботи тренажера

Алгоритм роботи тренажера практичної частини наступний:

Крок 1. При виборі теми, користувачу пропонується розпочати роботу з завданням.

Крок 2. Згідно завданню користувач повинен обрати комірку з правильною відповіддю, якщо введена відповідь правильна то комірка зафарбується в зелений колір, та користувач переходить до наступного питання. Якщо відповідь не правильна, то комірка зафарбовується в червоний, та користувач переноситься до наступного питання. Користувач має можливість перейти до теоретичного матеріалу в будь-який момент.

Алгоритм роботи тренажера теоретичної частини наступний:

Крок 1. На початку роботи з програмою-тренажером користувач має лише одну кнопку, для переходу в головне меню, та його інформують про те на яку тему йому надають інформацію.

Крок 2. В головному меню користувач ознайомлюється з короткою інформацією про тему яка буде тут представлена, та буде дві кнопки: перша для переходу до вибору практичного заняття (див. алгоритм роботи тренажера практична частина), друга кнопка переносе користувача до вибору теоретичної теми для опрацювання.

Крок 3. Користувач повинен обрати тему натиснув відповідну кнопку.

Крок 4. Обрав відповідну тему користувачу буде надана відповідна до теми інформація. Тут буде дві кнопки: перша для повернення на минулу сторінку, друга для переходу до наступної сторінки.

Крок 5. На останній сторінці теми, у користувача буде дві кнопки: перша повертає до минулої сторінки, друга дає можливість відразу перейти до меню вибору теми.

Крок 6. На протязі всієї роботи з програмою-тренажером користувач може натиснути на кнопку виходу, яка закінчує роботу програми.

Під час роботи з тренажером використовуються питання з декількома

варіантами відповідей, після вибору правильного варіанта відповіді буде можливість перейти до наступного питання, але в разі не правильної відповіді є можливість перейти до теоретичного матеріалу.

Алгоритм роботи з практичним матеріалом буде розглянуто на прикладі типової задачі.

Приклад:

```
var a, b : integer;
sum: integer;
begin
  writeln(`Ввудіть a і b: ');
  readln(a);
  readln(b);
  asm
     mov ax, a
     add ax, b
     _________, ax
  end;
  writeln(`Сума:', sum);
end.
```

Варіанти відповідей:

1) mov sum

2) sum mov

3) всі варіанти правильні

Правильний варіант відповіді – «mov sum»

3.3 Блок-схема програми-тренажера

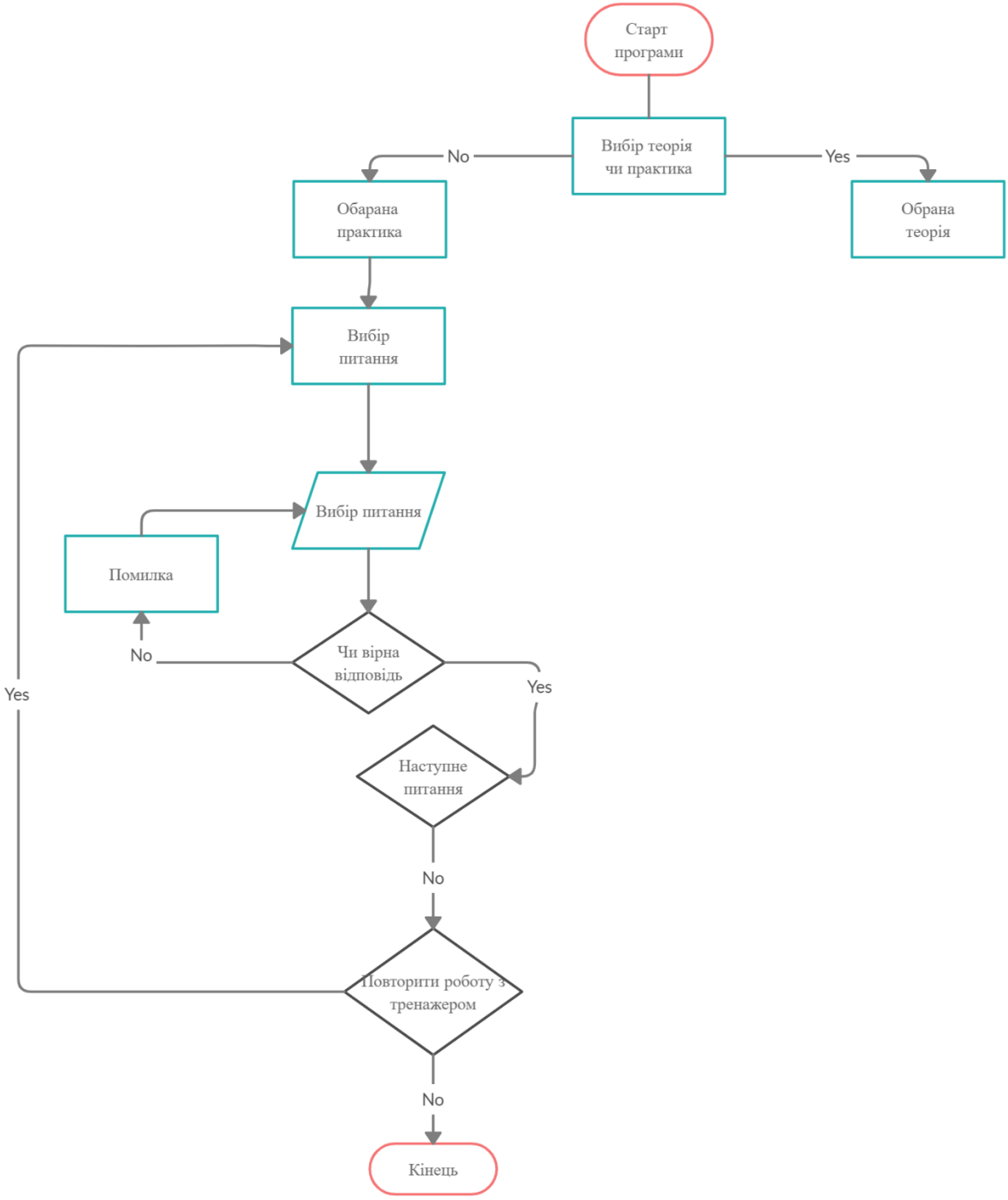

Рисунок 3.1 – Блок-схема програми-тренажера

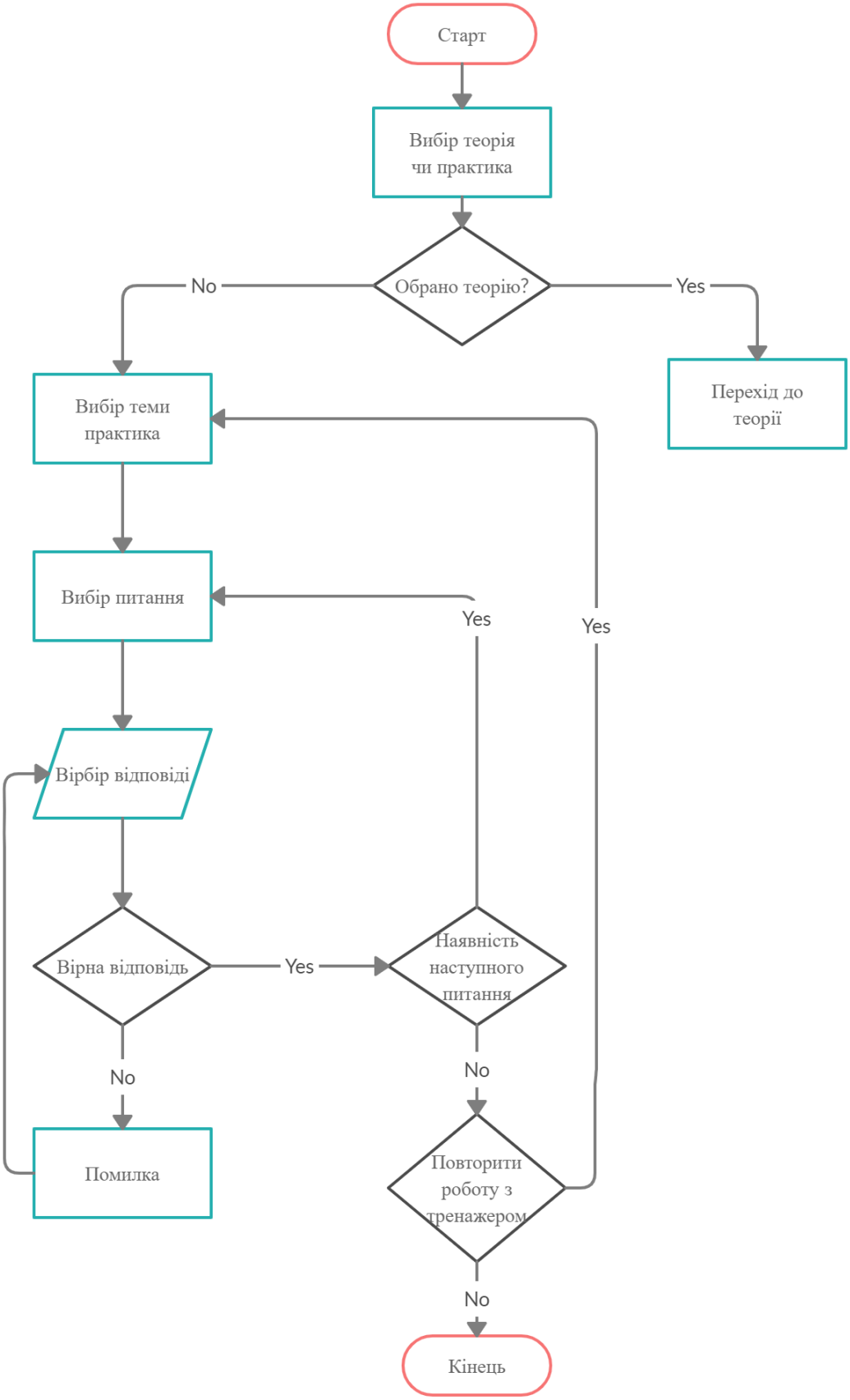

Рисунок 3.2 – Блок-схема роботи з практичним матеріалом

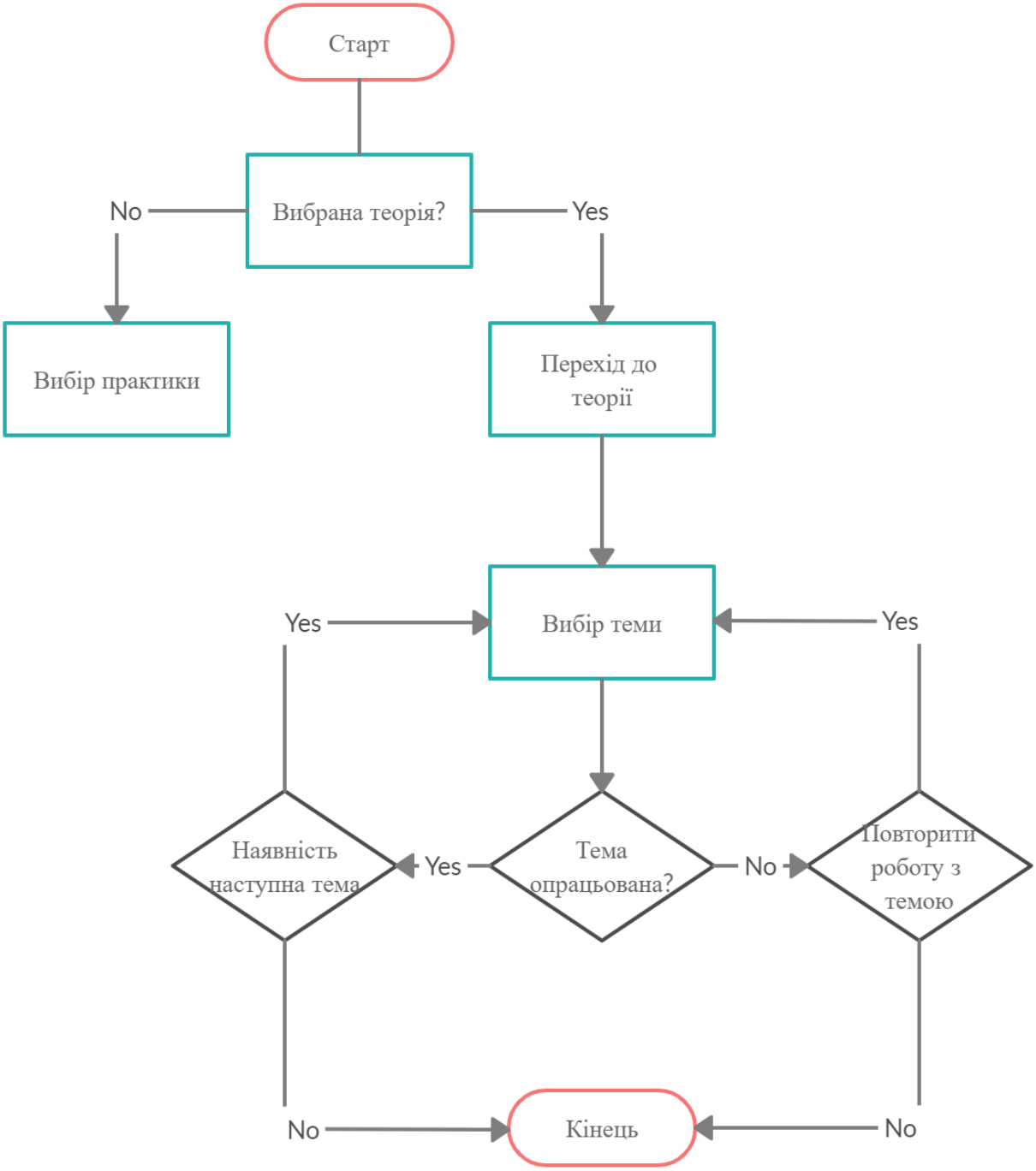

Рисунок 3.3 – Блок-схема роботи з теоретичним матеріалом

4 Практична частина

4.1 Опис програмного забезпечення

Дана програма створена за допомогою IDE MS Visual Studio у середовищі Unity. Такий підхід до розробки тренажеру має свої обмеження, але також багато плюсів. Наприклад до плюсів роботи з Unity можна віднести те, що код для переходу між кнопками достатньо написати лише раз, а потім викликати його за допомогою інтерфейсу програми. Також до переваг такого вибору програмних засобів можна віднести можливість портувати тренажер на будь-яку платформу за вибором. Для даного тренажеру портування було виконане на універсальну платформу Windows, що дає змогу запустити тренажер на будь-якому пристрої з такою операційною системою.

Для переходу між слайдами тренажеру реалізовано за допомогою кнопок «Далі» та «Назад». Кнопка «Далі» базується на закритті поточного слайду та відкритті наступного. У разі закінчення роботи з завданнями теоретичного типу, в кінці кнопки «Далі» замінена на кнопку «В меню».

Код для роботи кнопки «Далі»:

```
Slide.SetActive(false);
Slide.SetActive(true);
```

Кнопка «Назад» теж працює по цьому принципу.

Код для роботи кнопки «Назад»:

```
Slide2.SetActiive(false);
Slide2.SetActiive(true);
```

Робота з логічними змінними в Unity оформлення у вигляді вбудованого скрипту для кнопок (див. Рисунок 4.1)

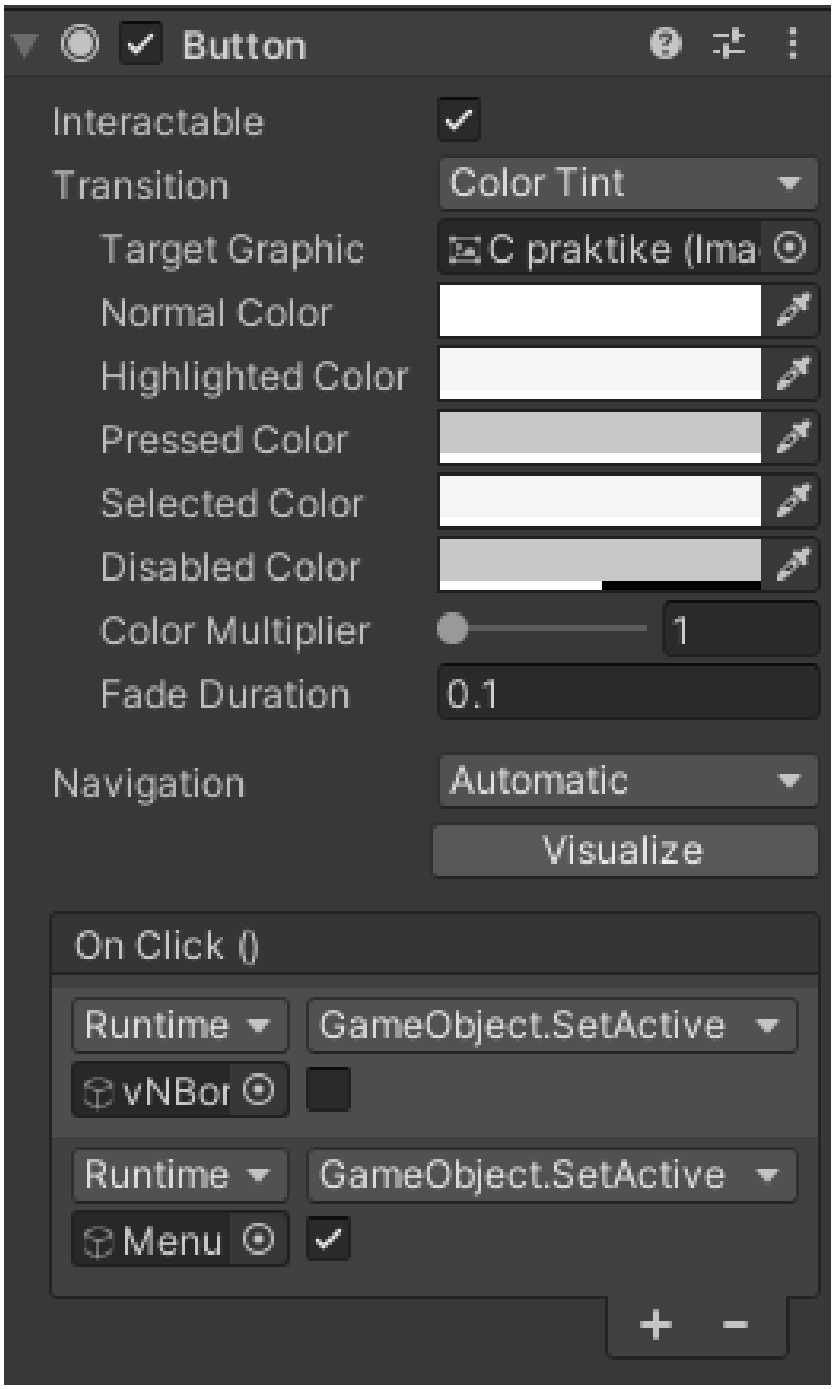

Рисунок 4.1 – Скрипт для виконання роботи кнопок

У стартовому меню тренажера (див Рисунок 4.2) знаходиться кнопка «Старт» яка працює абсолютно так як і кнопка «Далі», а також тут знаходиться привітання до користувача.

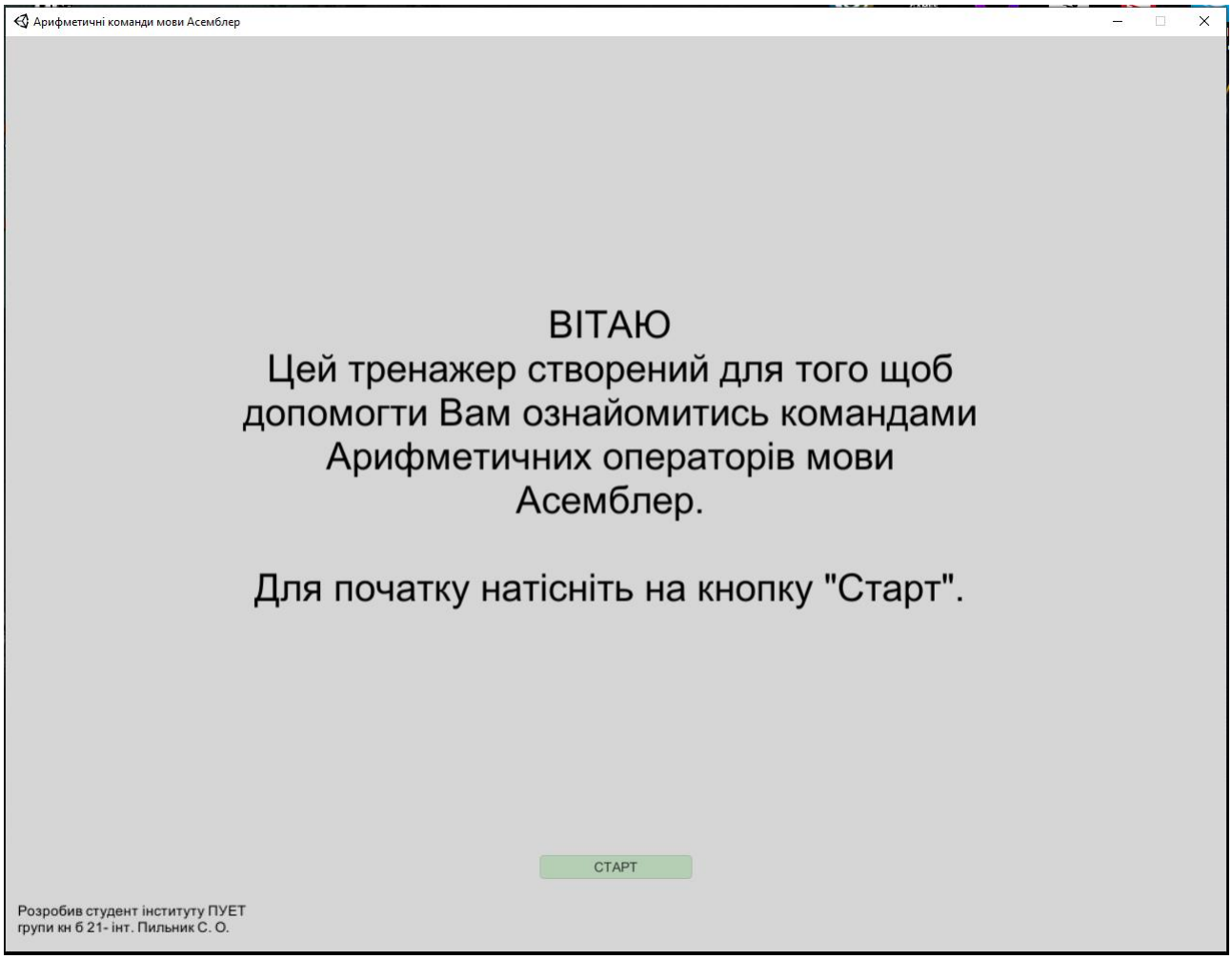

Рисунок 4.2 – Стартове меню тренажера

Головне меню програми (див. Рисунок 4.3) дозволяє обирати між теоретичним матеріалом та практичними завданнями, та показує коротку інформацію про тренажер.

Кнопка «Практичні завдання» перенаправляє на слайд з темами для практичної роботи (див. Рисунок 4.4).

Код для роботи кнопки «Практичні завдання»:

Slide(MainMenu).SetActive(false);

Slide.(PrMenu)SetActive(true);

Кнопка «Теорія» перенаправляє на слайд з теоретичною інформацією (див. Рисунок 4.5).

Код для роботи кнопки «Теорія»:

Slide.(MainMenu)SetActive(false);

Slide.(ThMenu)SetActive(true);

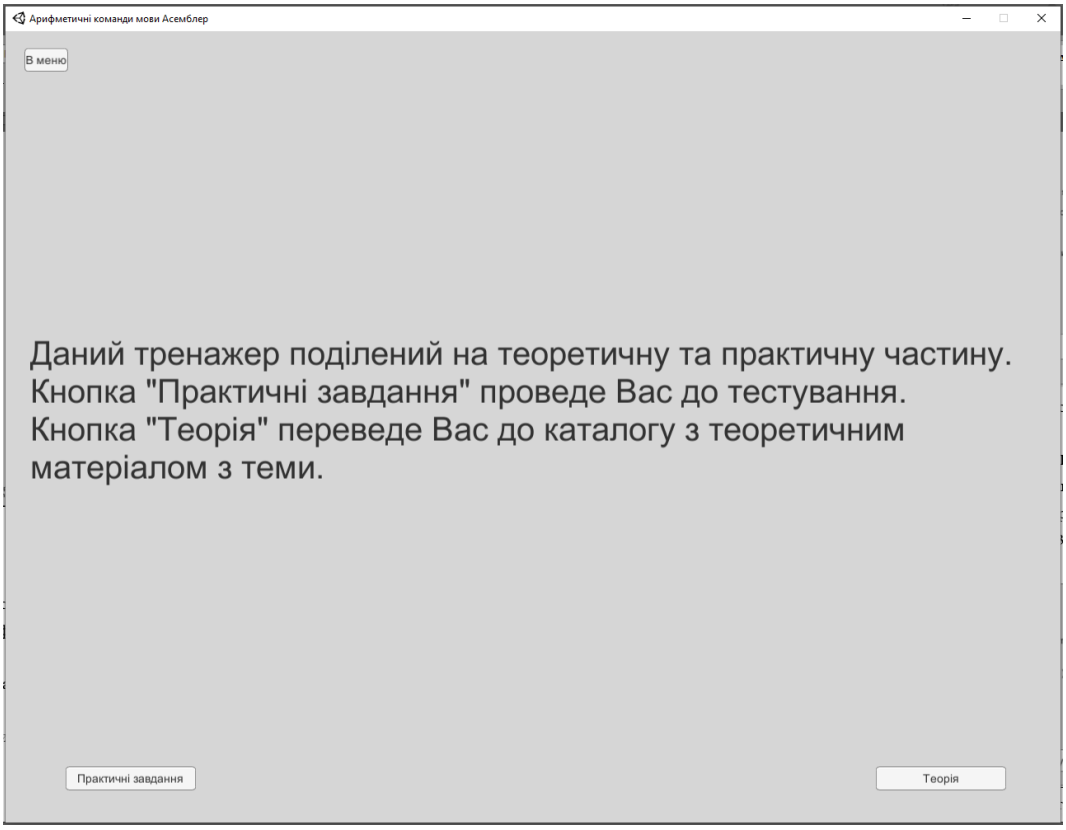

Рисунок 4.3 – Головне меню тренажера

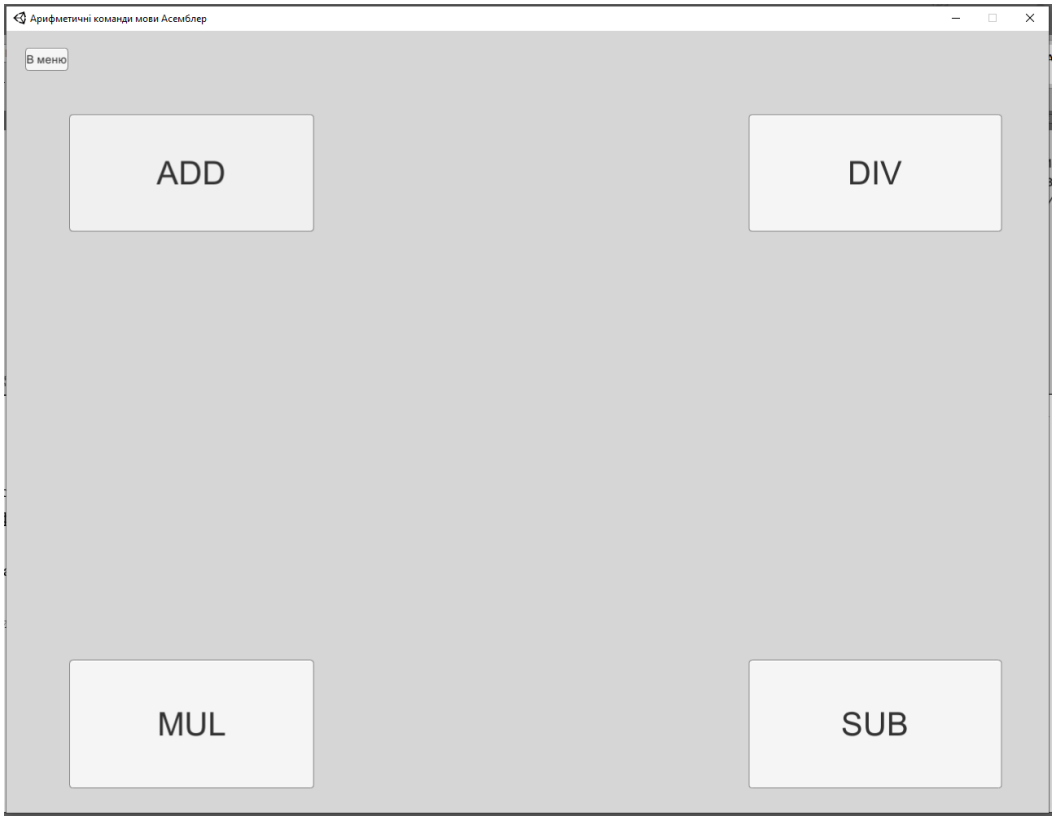

Рисунок 4.4 – Меню для вибору практичної теми

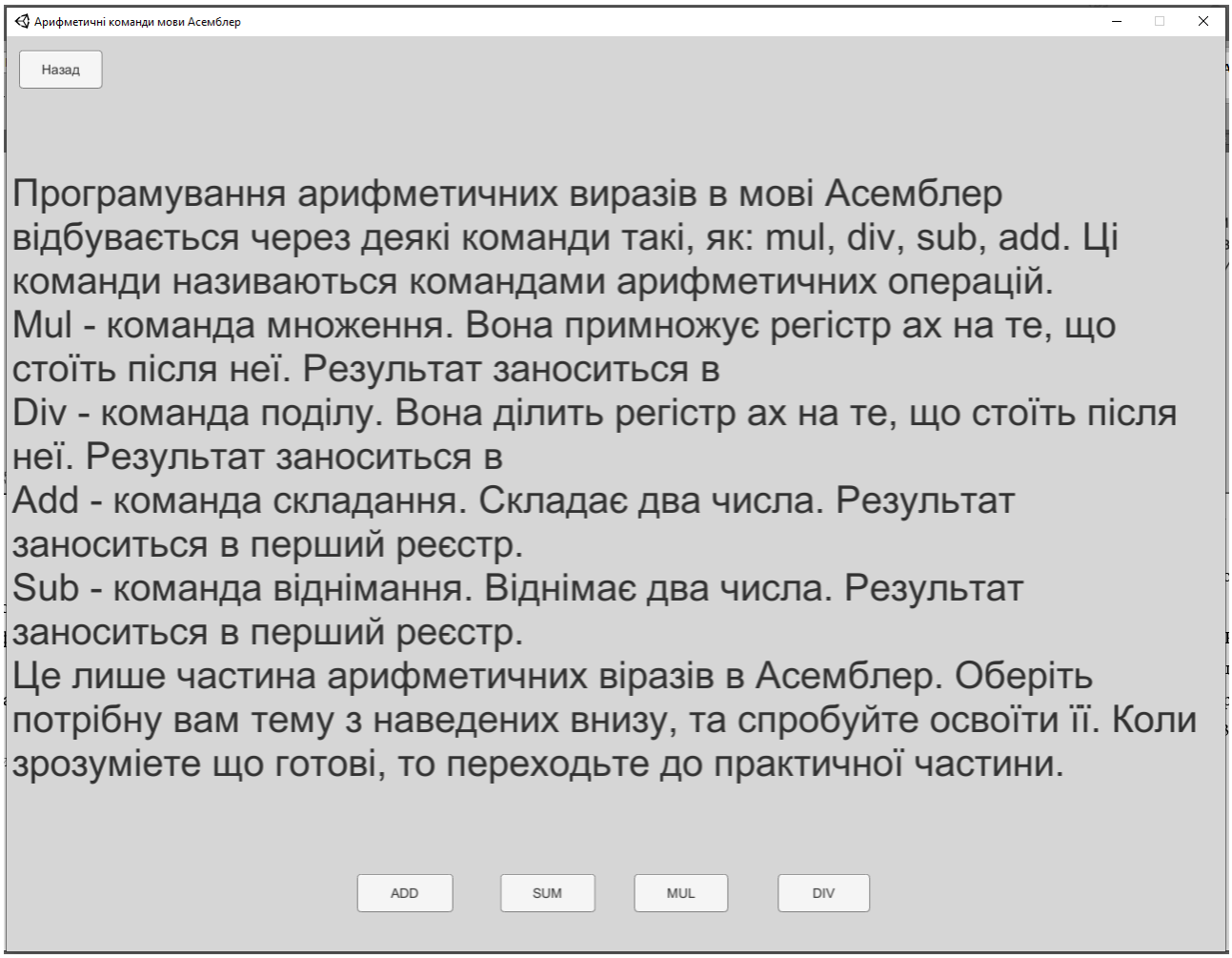

Рисунок 4.5 – Меню для вибору теми для теоретичного опрацювання

При виборі теми відкривається її перший слайд (див. Рисунок 4.6), на ньому та наступних слайдах буде видана теоретична інформація відповідно до теми. На останньому слайді теми встановлена кнопка повернення до меню вибору теми для опрацювання наступної теми (див. Рисунок 4.7).

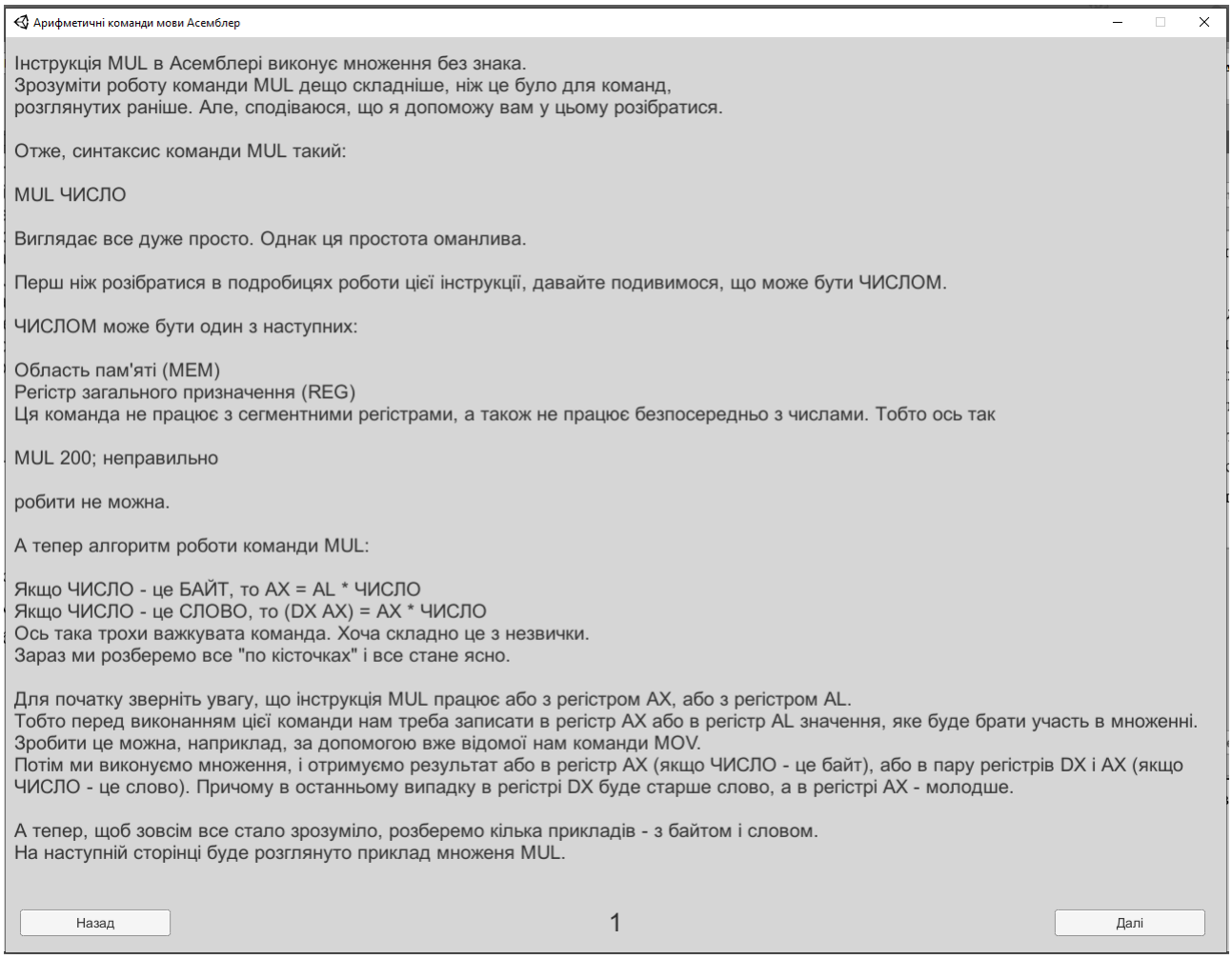

Рисунок 4.6 – Перший слайд з теоретичним матеріалом

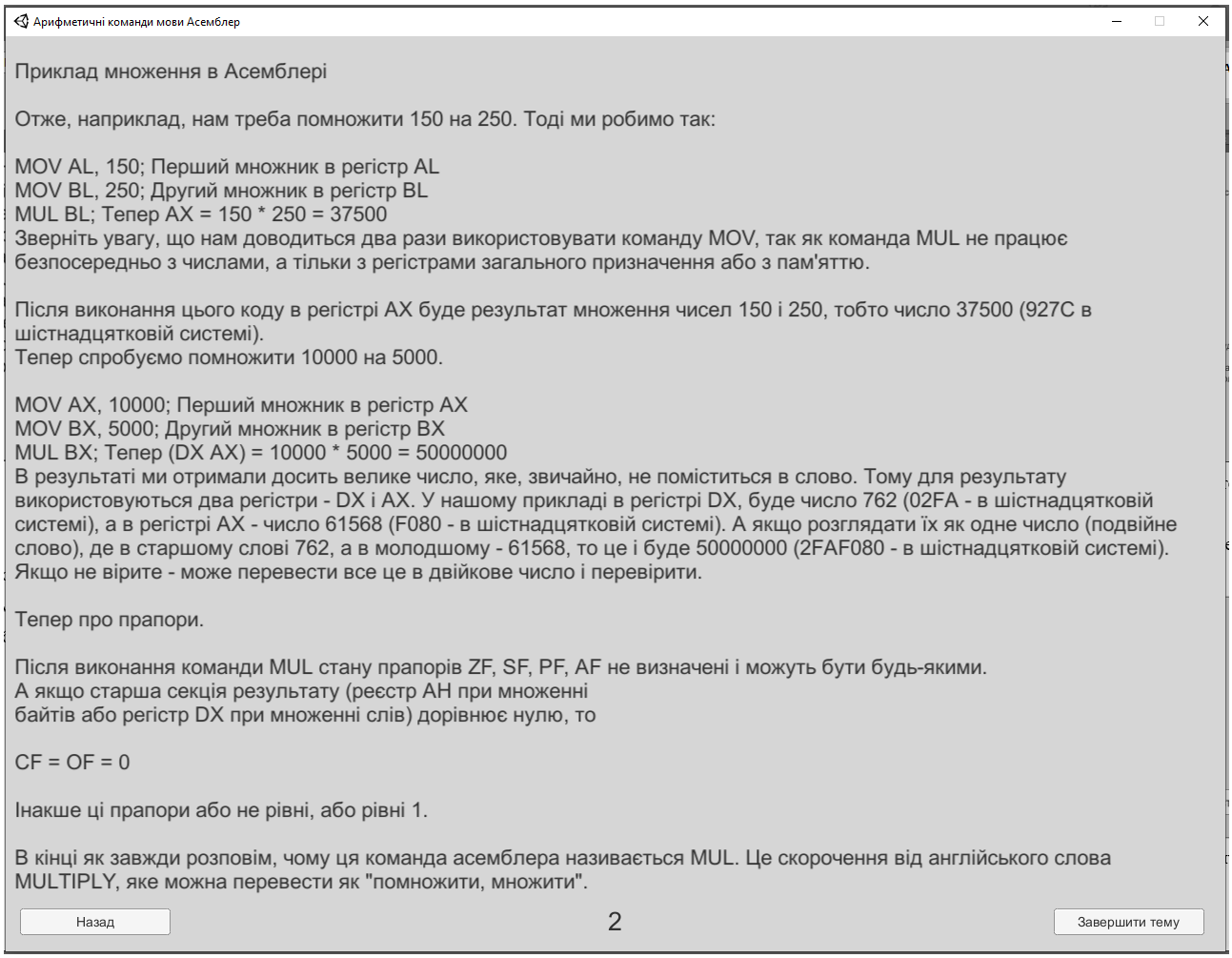

Рисунок 4.7 – Другий слайд з теоретичним матеріалом

Розглянемо практичне завдання. Під час роботи з практичним завданням користувач отримує три варіанта відповіді, одна з яких являється обов’язково правильною, якщо користувач вибере її то він отримає повідомлення «Правильно!» (див. Рисунок 4.9), якщо користувач обирає неправильну відповідь, то він отримає повідомлення «Не правильно!» (див. Рисунок 4.10).

Під час виконання завдання користувач може перейти до наступного завдання натиснув кнопку «Далі».

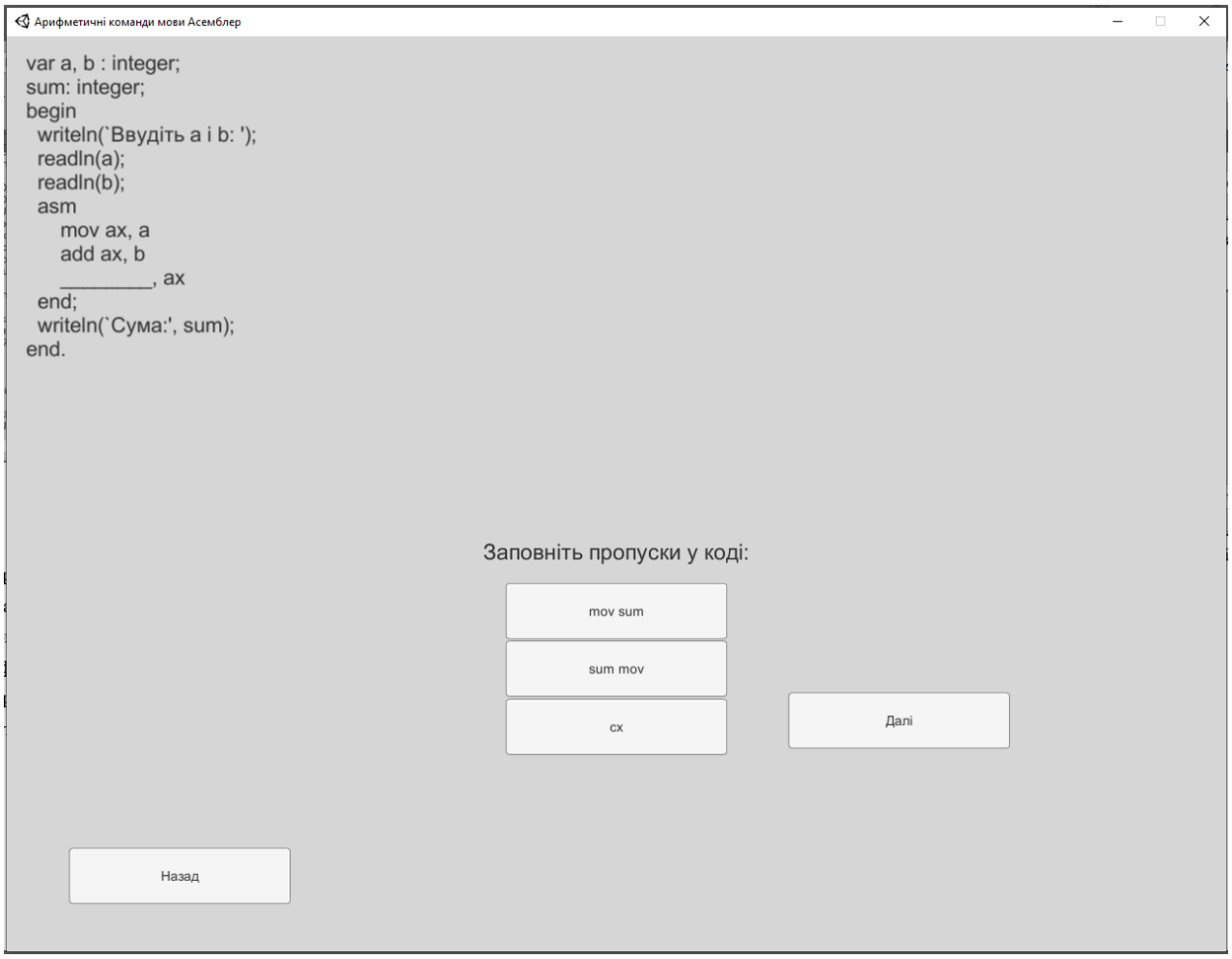

Рисунок 4.8 – Подача програмою завдання практичного типу

Після вибору правильного варіанта відповіді користувач отримує повідомлення «Правильно!»

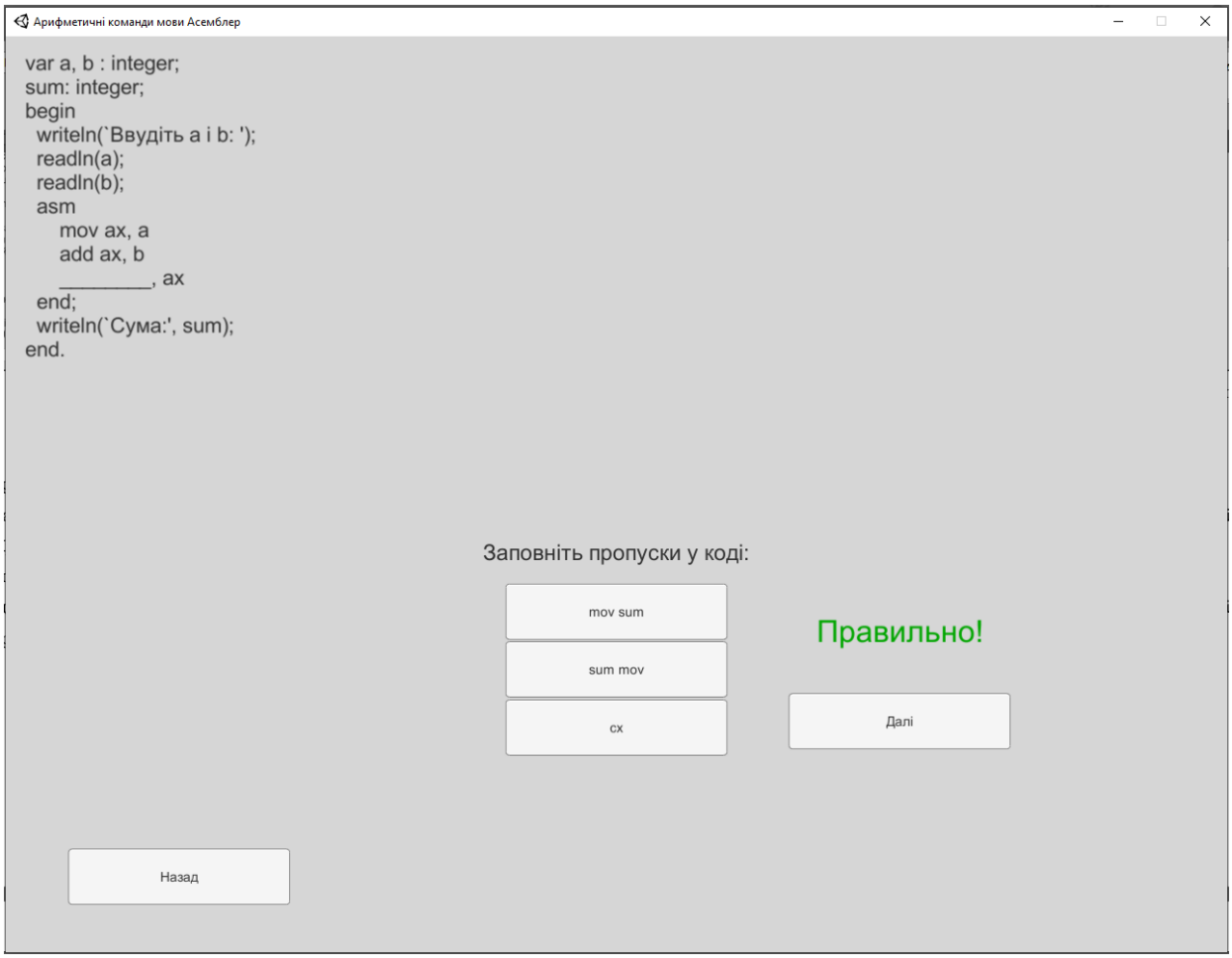

Рисунок 4.9 – Вікно після вибору правильної відповіді

Після вибору неправильного варіанта відповіді користувач отримує повідомлення «Не правильно!»

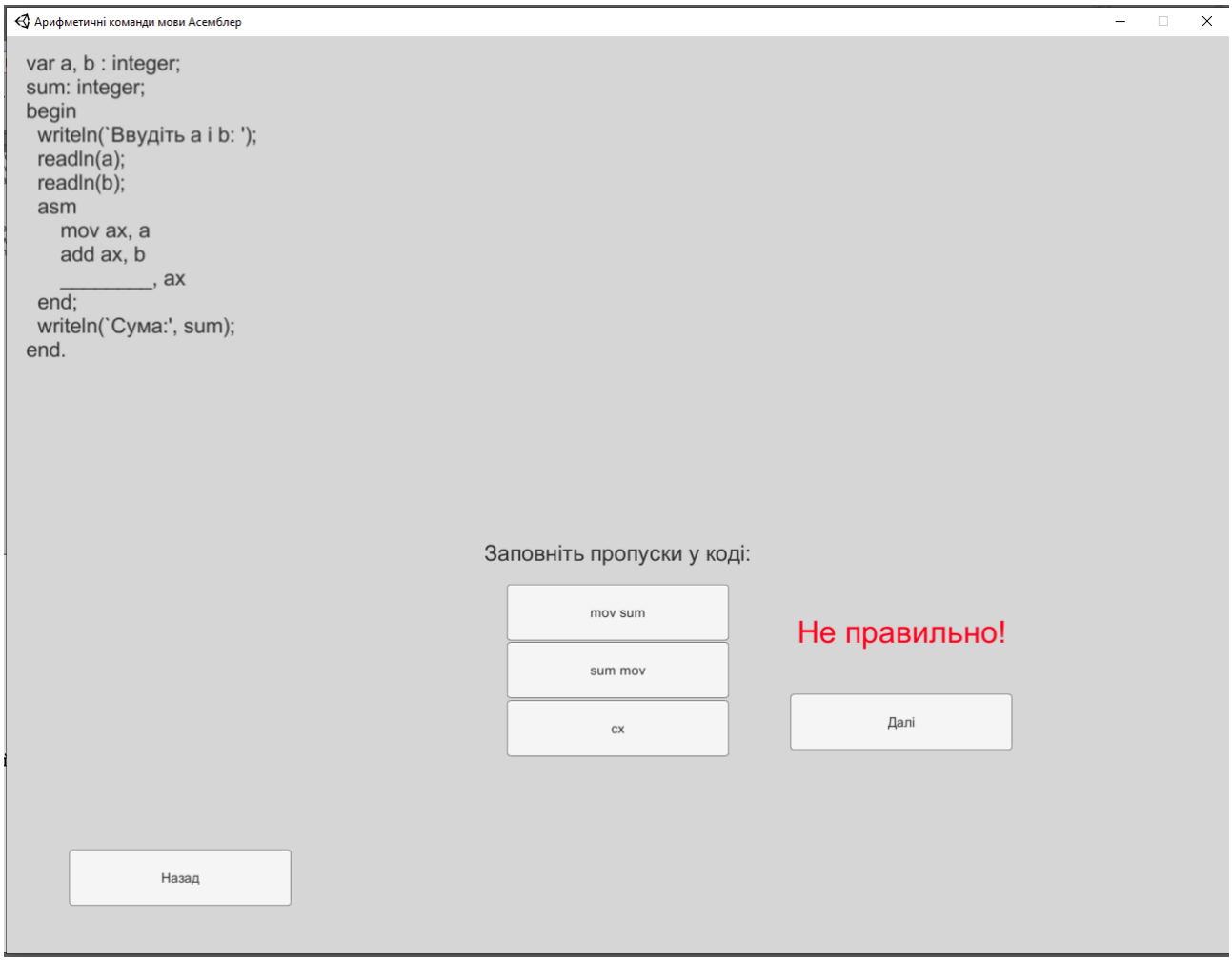

Рисунок 4.10 – Вікно після вибору неправильної відповіді

На будь-якому етапі роботи з програмою користувач має можливість вийти з програми, або в головне меню розділу.

4.2 Інструкція користувача

Для того щоб почати роботу з тренажером необхідно запустити виконуваний файл Арифметичні команди мови Асемблер.exe. Після цього перед користувачем відкривається вікно зі стартовим меню тренажера.

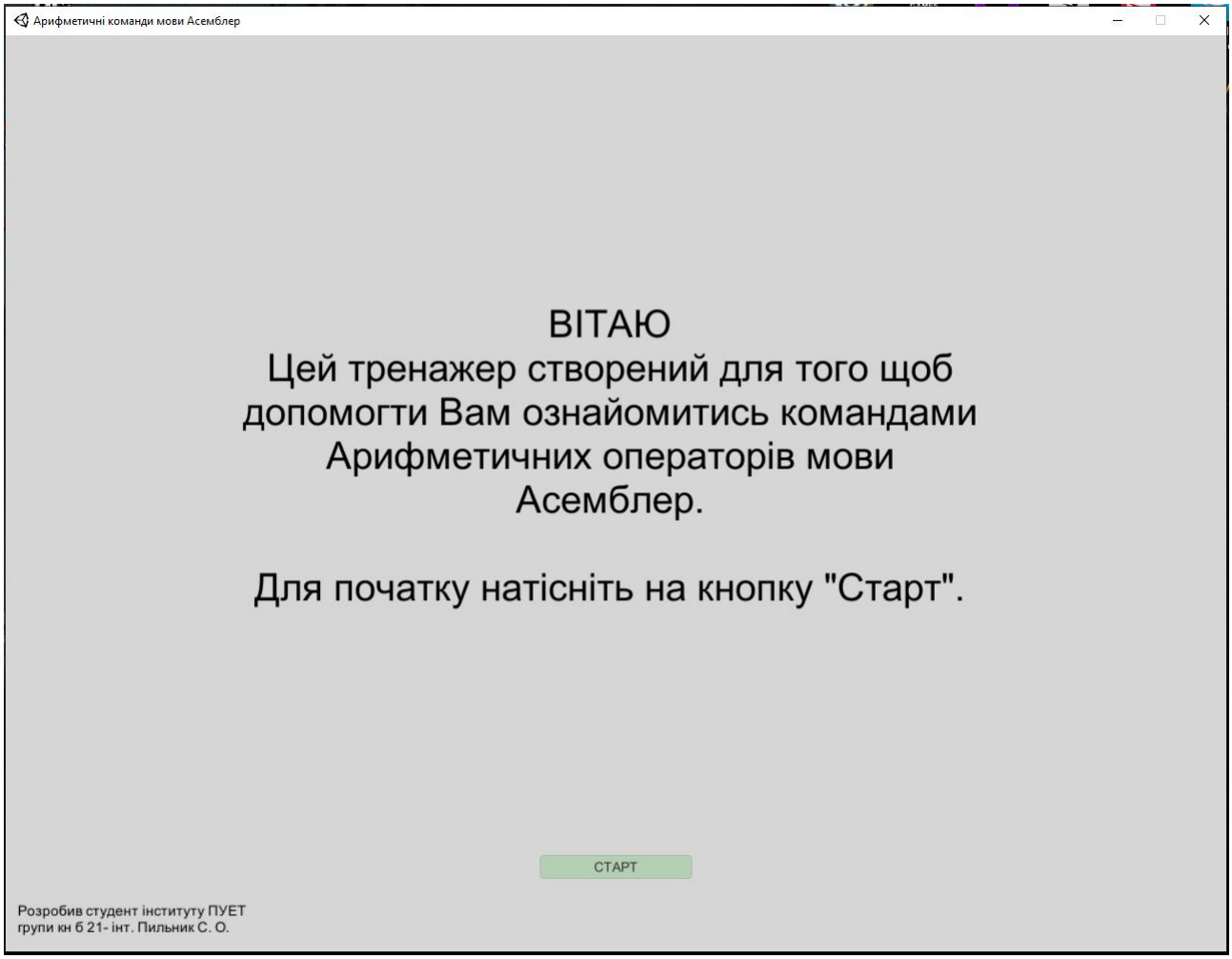

Рисунок 4.11 – Стартове меню тренажера

Після натиснення на кнопку «Старт» користувач переходить до головного меню тренажеру.

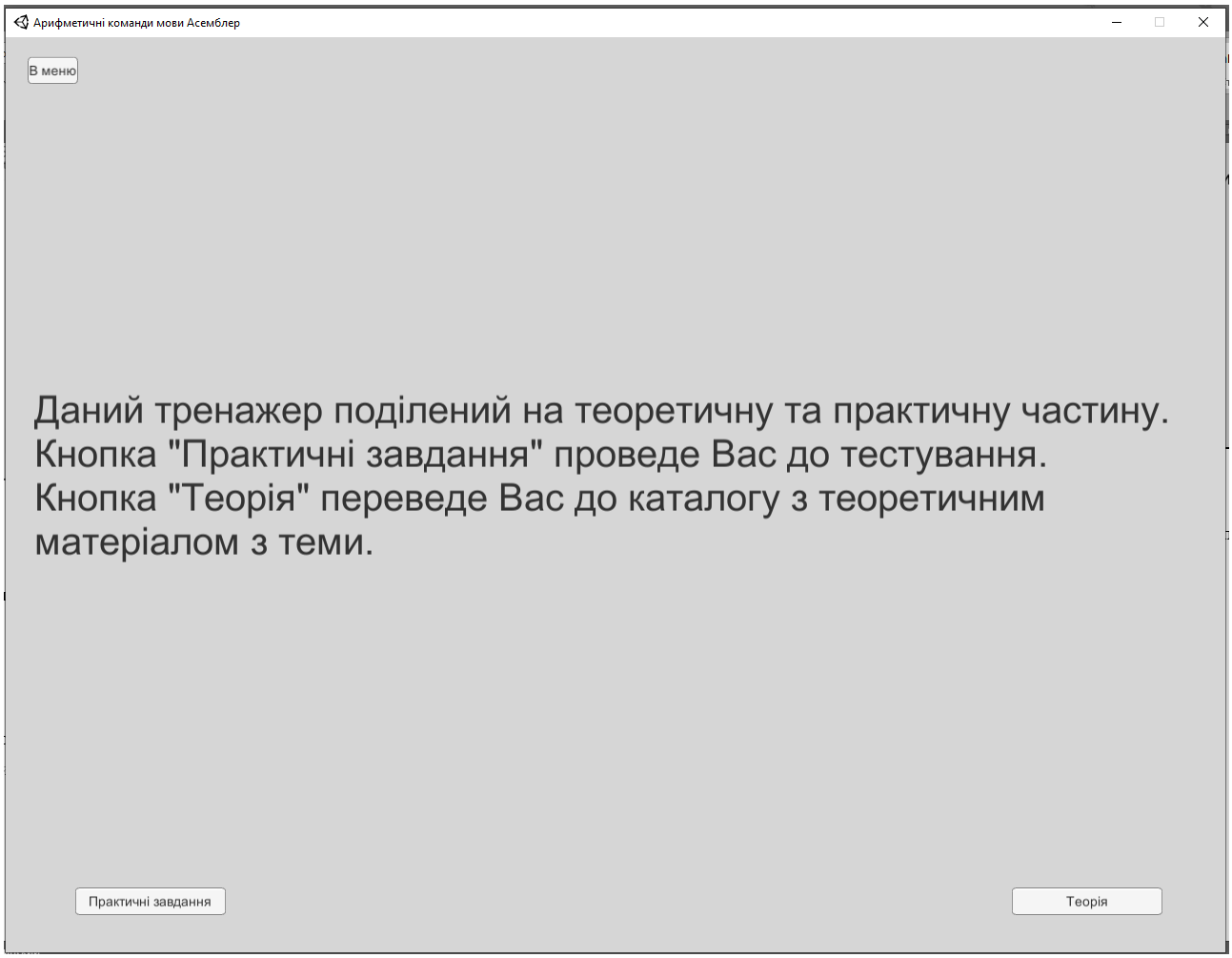

Рисунок 4.12 – Головне меню тренажеру

Після натиснення на кнопку «Теорія» користувач переходить до меню для вибору теорії.

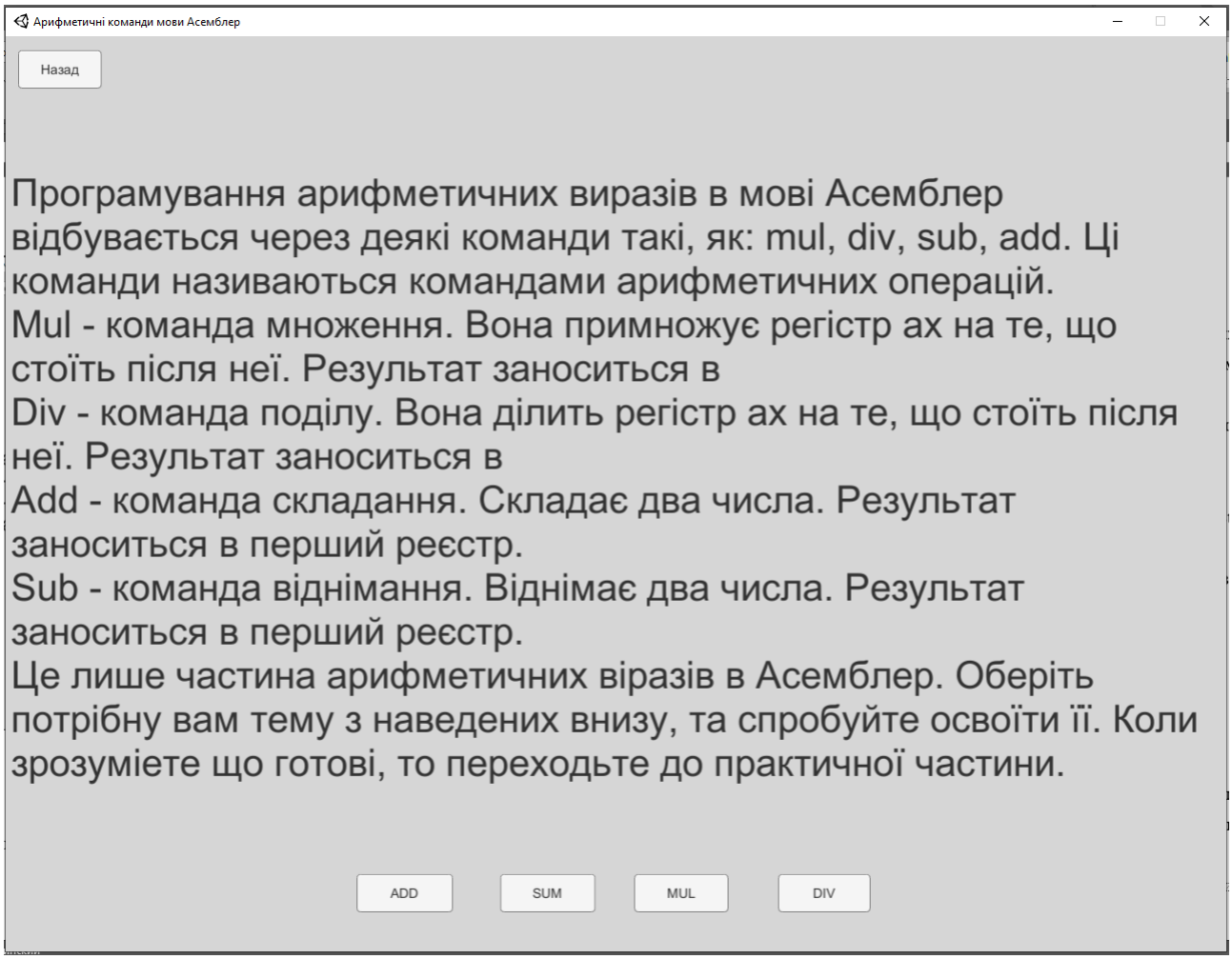

Рисунок 4.13 – Меню для вибору теорії

В тренажері реалізовано чотири основні теми для теоретичного ознайомлення. Після натиснення на будь-яку з них користувач переходить до теоретичних матеріалів з обраної теми.

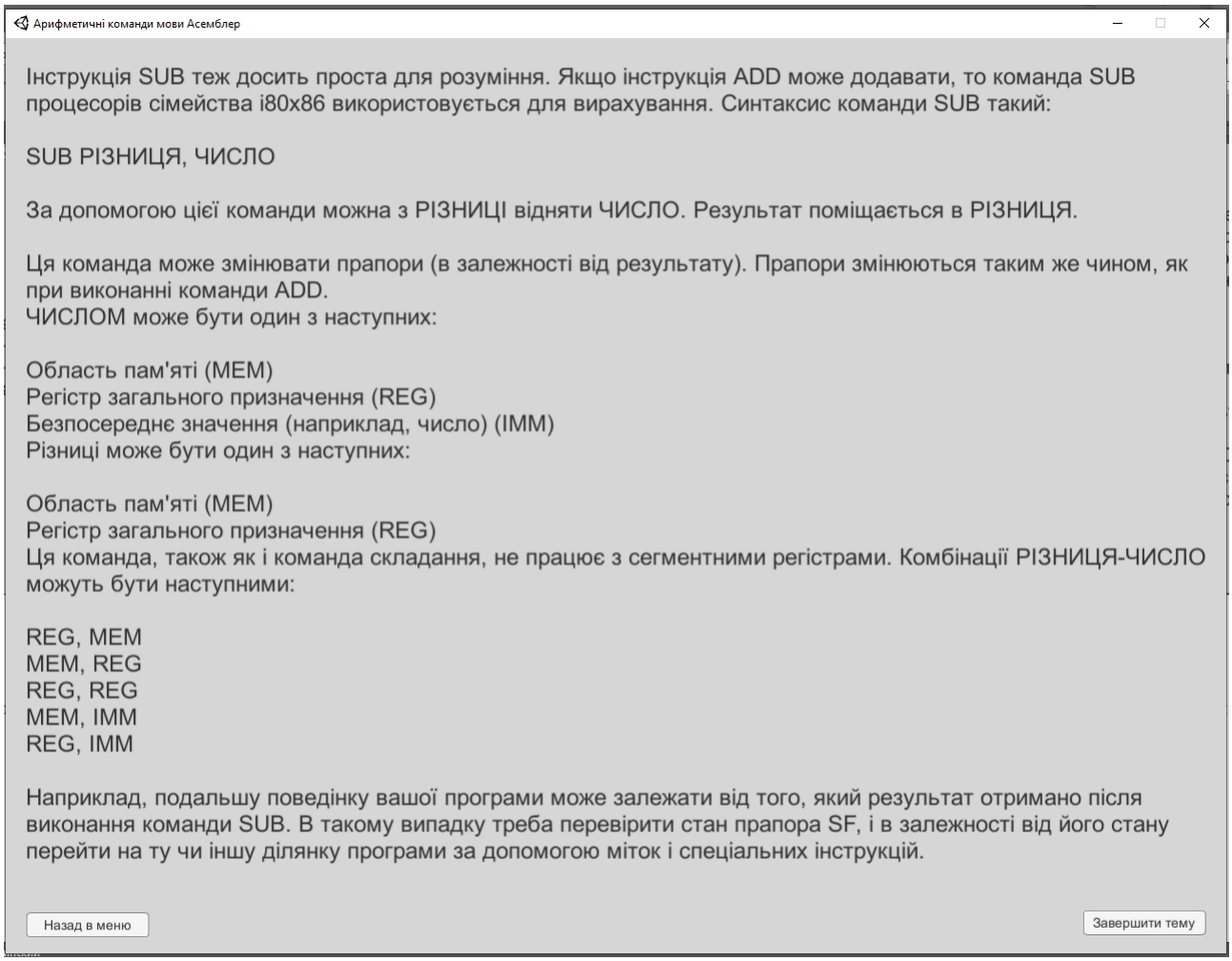

Рисунок 4.14- Теоретична інформація в тренажері

На будь-якому кроці роботи з теоретичним матеріалом можна повернутися в меню через відповідну кнопку, після завершення ознайомлення з теорією користувач повертається в меню вибору теоретичної теми через кнопку «Завершити тему»

Після натиснення кнопки «Практичні завдання» в головному меню тренажера користувач переходить до меню вибору практичної теми.

Як і теоретичних тем, реалізовано чотири різні практичні теми.

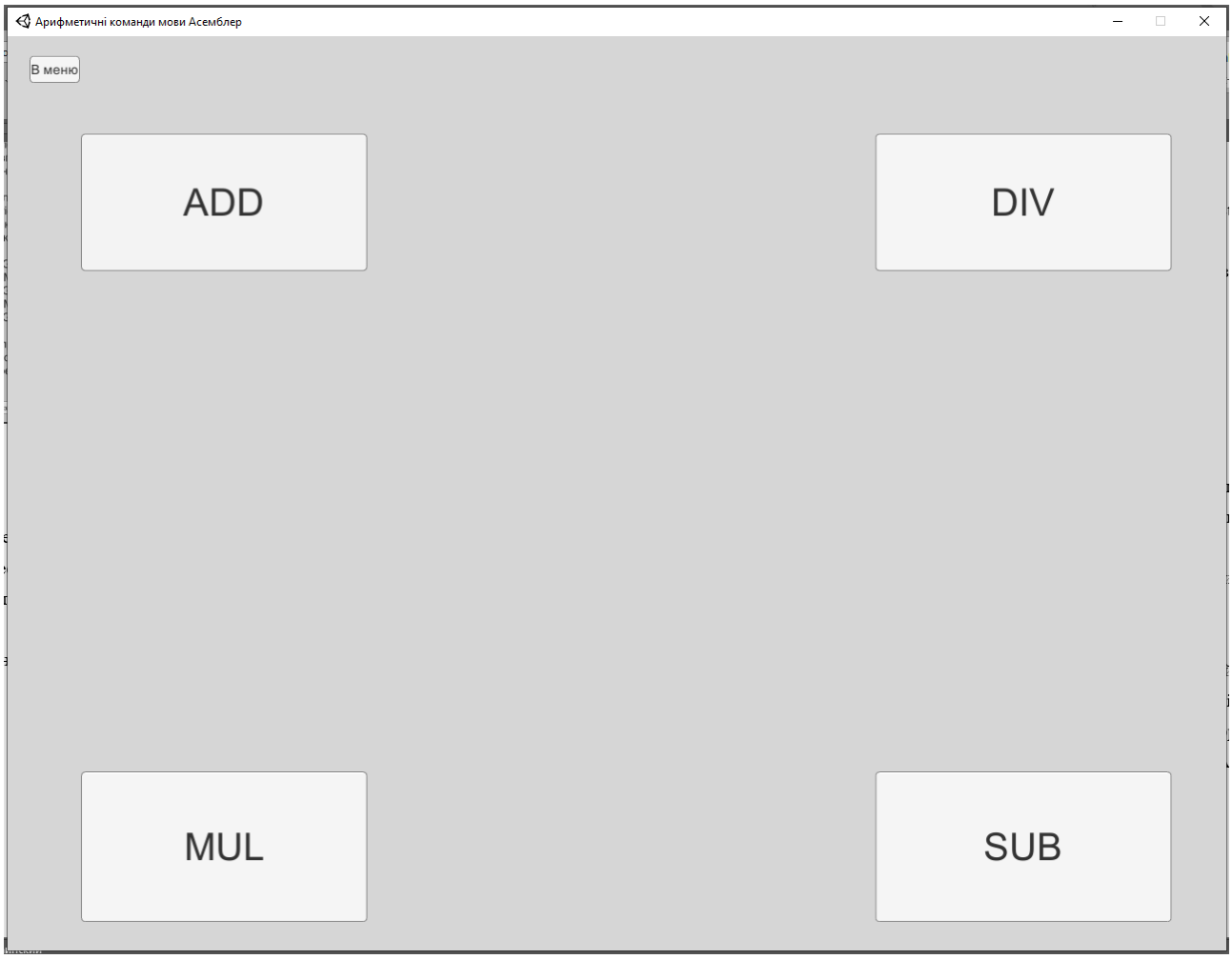

Рисунок 4.15 – Меню для вибору практичних тем

Після вибору та натиснення на будь-яку з практичних тем користувач переходить до практичних завдань з обраної теми.

Для прикладу було обрано тему ADD.

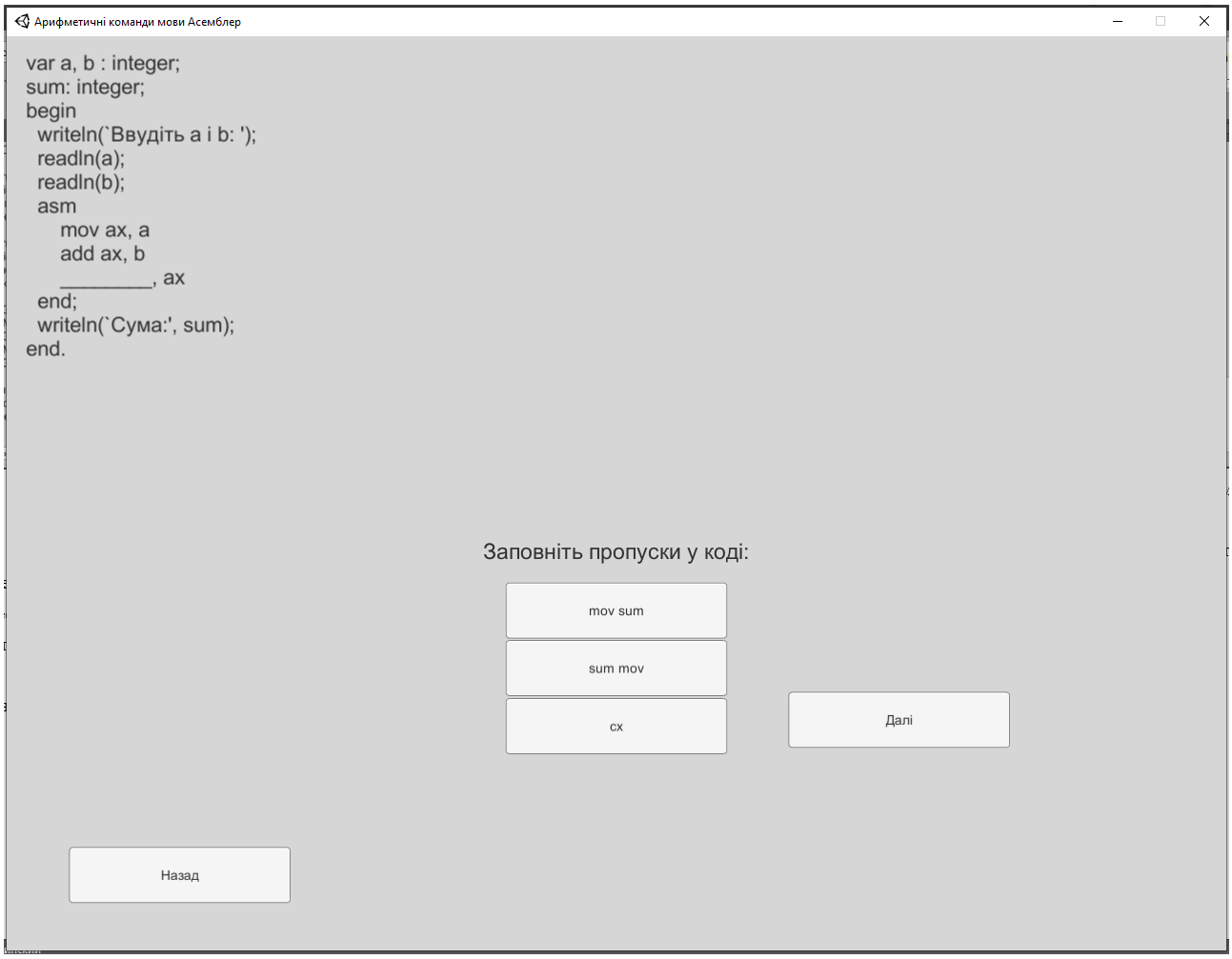

Рисунок 4.16 – Практичне завдання в тренажері

Після вибору варіанту відповіді користувач отримує відповідне повідомлення та може продовжити роботу з практичними завданнями.

Рисунок 4.17 – Перевірка введеної відповіді

Після закінчення роботи з практичними матеріалами з теми кнопка «Далі» змінюється на кнопку «Завершити тестування»

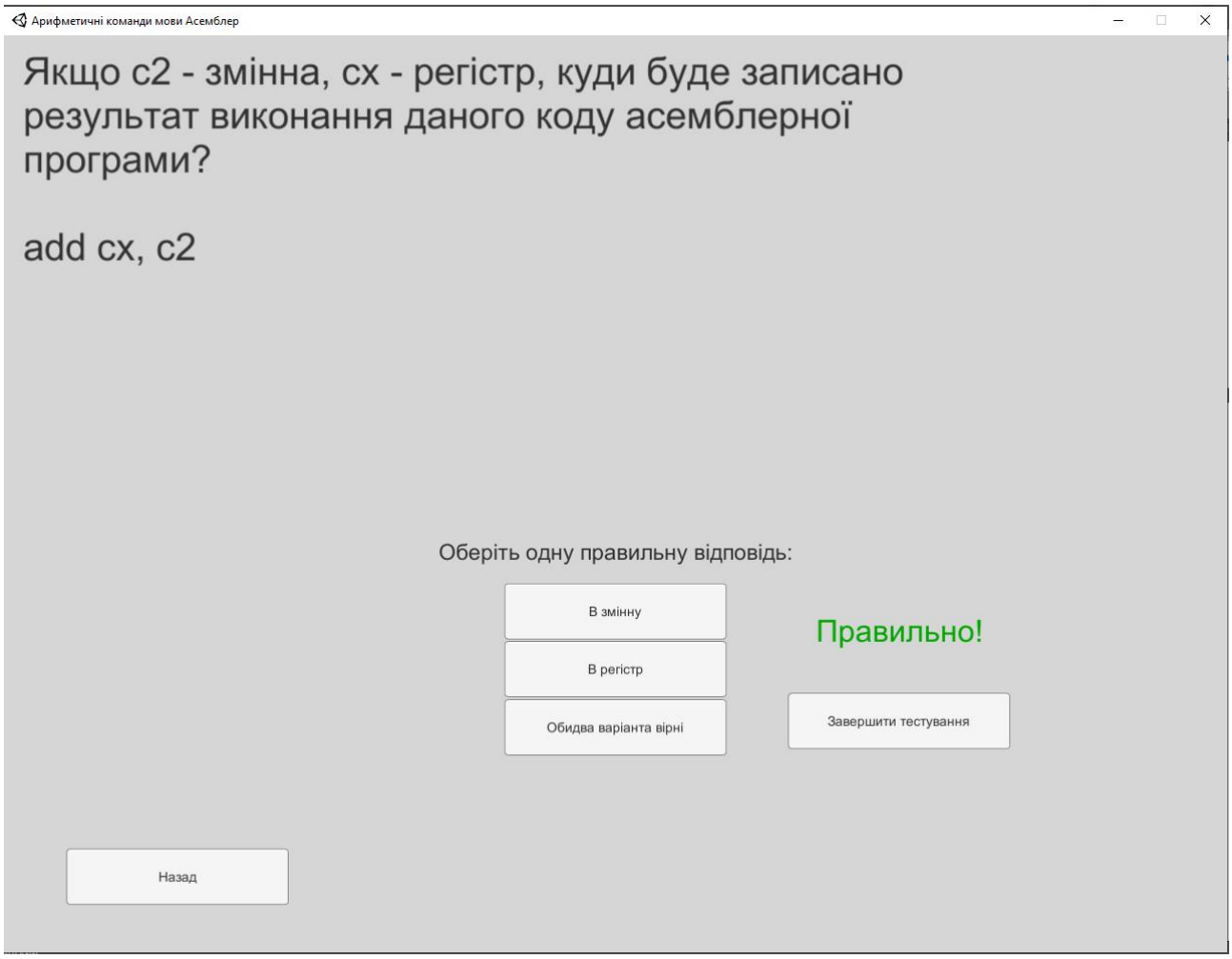

Рисунок 4.18 – Завершення роботи з практичною темою

Робота з усіма іншими навчальними матеріалами тренажеру виконана за розглянутим зразком.

ВИСНОВКИ

Дана програма створена на основі сучасних комп’ютерних технологій, навчає студентів темі «Арифметичні команди мови асемблер», має теоретичну та практичну частину та зручний перехід між ними.

Під час виконання бакалаврської роботи було розроблено програму-тренажер з теми «Арифметичні команди мови асемблер.

Результатами бакалаврської роботи є:

1. Обрано програмне середовище – MS Visual Studio та мову програмування C#;
2. Обрано платформу для розробки програмного забезпечення – Unity 2020;
3. Розроблено алгоритм роботи тренажера з теми «Арифметичні команди мови асемблер» для дистанційного курсу «Архітектура обчислювальних систем»;
4. Складено блок-схему до алгоритму роботи програми з врахуванням усіх можливих варіантів роботи;
5. Програмно реалізовано програму-тренажер на платформі Unity 2020 у середовищі IDE MS Visual Studio за допомогою мови програмування C#;
6. Виконано перевірку валідності програми. Помилок в роботі тренажера не знайдено.

Плюсами розробленого програмного забезпечення є;

1. Мінімалістичний та інтуїтивно зрозумілий інтерфейс програми;
2. Докладно та зрозуміло поданий теоретичний матеріал;
3. Роботу з тренажером поділено на блоки;
4. Навігація між блоками реалізовано через відповідні меню для вибору практичних та теоретичних тем.

Мету та завдання бакалаврської роботи виконано, створено тренажер для навчання студентів «Арифметичні команди мови Асемблер» для

дистанційного курсу «Архітектура обчислювальних систем».

Розроблений тренажер може бути впроваджено в систему дистанційного навчання MOODLE ПУЕТ в дистанційний курс «Архітектура обчислювальних систем», про що є акт про впровадження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мороз Артур Вадимович, з теми Пояснювальна записка до бакалаврської роботи на тему Оптимізація виробництва столів: програмна реалізація тренажера (моделювання) дистанційного курсу «Проекне навчання з курсу «Методи оптимізації та дослідження операцій / Мороз Артур Вадимович [Електронний ресурс].– Режим доступу до ресурсу: http://dspace.puet.edu.ua/handle/123456789/9017

2. Григор'єв Владислав Віталійович, Пояснювальна записка до дипломної роботи на тему Розробка програмного забезпечення тренажеру з теми «Побудова математичних моделей лінійних задача планування виробництва» дистанційного навчального курсу «Сучасні методи оптимізації та їх програмування / Григор'єв Владислав Віталійович [Електронний ресурс].– Режим доступу до ресурсу: http://dspace.puet.edu.ua/handle/123456789/8992

3. Голубенко Роман Володимирович, Пояснювальна записка до бакалаврської роботи на тему Розробка та програмна реалізація тренажера з теми «Метод Дальтона-Ллевеліна» дистанційного навчального курсу «Методи оптимізації та дослідження операцій / Голубенко Роман Володимирович [Електронний ресурс].– Режим доступу до ресурсу: http://dspace.puet.edu.ua/handle/123456789/9002

4. Вікіпедія, Unity [Електронний ресурс].– Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Unity_(%D1%80%D1%83%D1%88%D1%96%D0%B9_%D0%B3%D1%80%D0%B8)

5. Інструкції при роботі з Unity [Електронний ресурс].– Режим доступу до ресурсу: https://unity.com/

6. Арифметичні та логічні операції мови асемблер мікропроцесора Intel8086 [Електронний ресурс].– Режим доступу до ресурсу: https://knowledge.allbest.ru/programming/2c0a65625a3bd68b4c53a88421216d36_0.html

7. Арифметичні команди мови Асемблер. [Електронний ресурс].–

Режим доступу до ресурсу: https://life-prog.ru/ukr/view_zam.php?id=36&cat=3&page=1

8. Ємець О.О. Методичні рекомендації до виконання бакалаврської роботи для студентів спеціальності 122 «Комп'ютерні науки та інформаційні технології» освітня програма «Комп'ютерні науки» галузь знань – 12 «Інформаційні технології»/ О.О. Ємець,–Полтава; ПУЕТ, 2017, - 71 с.

9. Ю. Магда Ассемблер для процессоров Intel Pentium /Ю. Магда

10. Р. Марек Assembler на примерах. Базовый курс / Р. Марек

ДОДАТОК А

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

public class Inputs : MonoBehaviour
{
    public GameObject Input;
    public GameObject AfterInput;
    public InputField inputedTxt;
    public Text ShowInpTxt;



    public void Next()
    {
        Input.SetActive(false);
        AfterInput.SetActive(true);

        ShowInpTxt.text = inputedTxt.text;
    }
}

public class QuitButton : MonoBehaviour
{
    public void QuitGame()
    {
        Debug.Log ("QUIT!!");
        Application.Quit();
```

```
    }

void Start()
    {
        Button btn = nxtButton.GetComponent<Button>();
        btn.onClick.AddListener(TaskOnClick);
    }

void TaskOnClick()
    {
        Theme1.SetActive(false);
        Theme2.SetActive(true);

        Debug.Log("You touched this button.");
    }

void Start()
    {
        Button btn = menuButton.GetComponent<Button>();
        btn.onClick.AddListener(TaskOnClick);
    }

void TaskOnClick()
    {
        Theme6.SetActive(false);
        MainMenu.SetActive(true);

        Debug.Log("You touched this button.");
    }
public void Next()
```

```
    {
        Input.SetActive(false);
        AfterInput.SetActive(true);

        ShowInpTxt.text = inputedTxt.text;
    }
void TaskOnClick Theory()
{
Slide.(MainMenu)SetActive(false);
Slide.(ThMenu)SetActive(true);
}
void TaskOnClick Practic ()
{
Slide.(MainMenu)SetActive(false);
Slide.(PrMenu)SetActive(true);
}
}
```