Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____ ” _____ 2025 р.

**ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА**

**за першим (бакалаврським) рівнем вищої освіти**

на тему

# “Програмне забезпечення системи емулятора перетворення логічної адреси в лінійну та лінійної у фізичну для навчальних цілей”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____С.А.Борохович
«____» _____ 2025 р.
Керівник проекту
кандидат технічних наук, доцент
_____ О.В. Коваленко
«____» _____ 2025 р.
Рецензент _____
_____



м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“Інформаційні технології”*_____

Спеціальність *123 “* Комп’ютерна інженерія *”*

Освітньо-професійна (освітньо-наукова) програма _*“* Комп’ютерна інженерія *”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

# ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

*Борохович Святослав Андрійович*
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи емулятора перетворення логічної адреси в лінійну та лінійної у фізичну для навчальних цілей*

2. Керівник роботи *Коваленко Олександр Володимирович д.т.н., доцент*
( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 46-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи емулятора перетворення логічної адреси в лінійну та лінійної у фізичну для навчальних цілей*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

*1. Призначення та область використання.*

*2.Перегляд аналогічних існуючих систем.*

*3. Опис і обґрунтування проектних рішень.*

*4. Етапи програмування системи.*

*5. Впровадження системи в промислову експлуатацію.*

*6. Висновки*

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

| | |
|---|---|
| *Структурна схема системи кібербезпеки* | *1 аркуш* |
| *Функціональна схема системи кібербезпеки* | *1 аркуш* |
| *Діаграма процесів* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |

7. Дата видачі завдання « 17 » січня 2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.03.2025 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.03.2025 р. | |
| 3. | Розробка моделі компонента | 20.03.2025 р. | |
| 4. | Розробка структур даних | 25.03.2025 р. | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.03.2025 р. | |
| 6. | Програмування алгоритмів | 10.04.2025 р. | |
| 7. | Оформлення ПЗ | 17.04.2025 р. | |
| 8. | Попередній захист роботи | 23.05.2025 р. | |
| | | | |
| | | | |
| | | | |
| | | | |

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Коваленко О.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Борохович С.А.
(прізвище та ініціали)

# АНОТАЦІЯ

**Борохович С.А. Програмне забезпечення системи емулятора перетворення логічної адреси в лінійну та лінійної у фізичну для навчальних цілей. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.**

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення системи емулятора, призначене для візуалізації процесів перетворення логічних адрес у лінійні та лінійних у фізичні. Метою розробки є створення програмного засобу, який моделює роботу системи керування пам'яттю з використанням сегментації, сторінкової організації пам'яті, таблиці сторінок та буфера трансляцій (TLB) у навчальних цілях.

Під час реалізації проєкту було проаналізовано принципи адресації у комп'ютерних системах, розроблено архітектуру програмного забезпечення та реалізовано основні компоненти: логічне адресування, обчислення лінійної адреси, доступ до сторінок фізичної пам'яті, обробку збоїв сторінок (page fault), оновлення таблиці сторінок і TLB. Програма дозволяє користувачеві отримати вичерпну інформацію про процес трансляції адрес, переглянути фізичні значення та статистику звернень. Розробка виконана на мові C# у середовищі Visual Studio. Програмне забезпечення сумісне з операційними системами Windows 10/11.

**Ключові слова**: керування пам'яттю, логічна адресація, лінійна адреса, фізична пам'ять, сегментація, сторінкова організація, TLB, емулятор.

# ABSTRACT

**Borokhovych S.A. Software of the emulator system for converting logical addresses to linear ones and linear to physical addresses for educational purposes.**

**123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025**

This bachelor's qualification project presents the development of software for an emulator system designed to simulate the process of converting logical addresses into linear and linear into physical addresses. The aim of the development is to provide a software tool that models memory management mechanisms such as segmentation, paging, page tables, and a Translation Lookaside Buffer (TLB) for educational use.

During the development process, an analysis of modern memory management techniques and related hardware/software implementations was conducted. The structure and functioning of each software component are thoroughly described. A user-friendly interface was developed to visualize address translation and memory access operations. The software outputs detailed information including physical address calculations, memory values, and statistical metrics such as page faults and TLB hits.

The program is compatible with IBM PC architecture and Windows 10/11 operating systems. The software was developed using the C# programming language in the Visual Studio environment.

**Keywords**: memory management, logical address, linear address, physical memory, segmentation, paging, TLB, emulator.

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ

TLB – Translation lookaside buffer

TI - Table Indicator

MVT – Model View Template

SPA – Single-Page Applications

UML – Unified Modeling Language

UI – User Interface

САПР – Система автоматизованого проектування і розрахунку

PDF – Portable Document Format

HTTP – HyperText Transfer Protocol

ПЗ – Програмне Забезпечення

# ВСТУП

Сучасні комп’ютерні системи характеризуються складною структурою управління пам’яттю, що включає сегментацію, сторінкову організацію, багаторівневі таблиці сторінок, буфер швидкого доступу до трансляцій (TLB) тощо. Ці процеси є невід’ємною частиною архітектури процесорів і операційних систем, оскільки забезпечують ефективне і безпечне використання фізичної пам’яті. У зв’язку з цим вивчення принципів трансляції логічних адрес у фізичні є обов’язковим елементом підготовки фахівців у галузі комп’ютерної інженерії.

На практиці вивчення таких процесів є складним для студентів через абстрактність викладу та відсутність можливості взаємодії з реальними механізмами трансляції. У більшості навчальних курсів викладання обмежується теоретичним поясненням або базовими схемами. Це знижує ефективність засвоєння матеріалу та ускладнює формування системного мислення майбутніх інженерів.

У зв’язку з цим актуальною є розробка програмного забезпечення, яке дозволяє моделювати повний цикл перетворення адрес від логічної до фізичної із візуалізацією усіх проміжних етапів. Такий емулятор може стати ефективним інструментом для практичного засвоєння знань, проведення лабораторних і демонстраційних занять, а також самостійної роботи студентів.

**Актуальність теми**

Актуальність теми зумовлена потребою в ефективних навчальних засобах, що дозволяють студентам глибше зрозуміти механізми управління пам’яттю в сучасних обчислювальних системах. Аналіз існуючих рішень показує, що більшість з них або надто спрощені, або не мають належної інтерактивності й візуалізації. Розробка спеціалізованого програмного забезпечення, орієнтованого на навчальні цілі, дозволяє суттєво підвищити якість підготовки студентів у галузі комп’ютерної інженерії.

**Мета і завдання роботи**

Метою дипломної роботи є розробка програмного забезпечення, яке імітує процес трансляції логічної адреси в лінійну та лінійної у фізичну для навчального використання.

Для досягнення поставленої мети у роботі визначено наступні завдання:

a) Провести аналіз архітектури управління пам’яттю в сучасних комп’ютерних системах.

b) Визначити вимоги до функціональності навчального емулятора.

c) Розробити алгоритм перетворення адрес, який враховує етапи сегментації, сторінкової трансляції та використання TLB.

d) Реалізувати програмне забезпечення з візуальним інтерфейсом користувача.

e) Протестувати програму на типових сценаріях використання.

f) Надати методичні рекомендації з використання програми у навчальному процесі.

**Практичне значення одержаних результатів**

Розроблене програмне забезпечення є повноцінним інструментом для навчання основам роботи систем управління пам’яттю. Воно дозволяє моделювати процеси трансляції адрес із наочним відображенням усіх етапів: від обробки логічної адреси до формування фізичної. Програма забезпечує автоматичну генерацію адрес, виявлення збоїв сторінок, наповнення таблиць сторінок і TLB, а також веде статистику ефективності трансляцій. ПЗ може використовуватись у закладах вищої освіти як допоміжний засіб для вивчення архітектури ЕОМ, операційних систем, організації комп’ютерної пам’яті.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ СИСТЕМИ

## 1.1 Призначення системи

Розроблена система-емулятор призначена для демонстраційного моделювання механізмів управління пам’яттю в комп’ютерних системах. Її мета – забезпечити студентів наочним інструментом для дослідження процесу перетворення логічних (віртуальних) адрес у лінійні та лінійних адрес у фізичні в середовищі, наближеному до реальних операційних систем. Емулятор реалізує основні концепції сегментації й сторінкового розбиття пам’яті, а також особливості кешування таблиці сторінок через буфер асоціативної трансляції адрес (TLB).

Сегментація пам’яті в системі забезпечує поділ логічного адресного простору на умовно виділені області – сегменти. Логічна адреса при цьому складається з ідентифікатора сегмента та зсуву всередині нього, що перетворюються на лінійну (або «сегментну») адресу шляхом додавання бази сегмента до зміщення. Таке моделювання дає змогу простежити, як формуються лінійні адреси в результаті застосування технології сегментації пам’яті. Після сегментації здійснюється відображення лінійних адрес у фізичні за допомогою сторінкової організації пам’яті. Сторінкова пам’ять – це підхід до організації віртуальної пам’яті, при якому вона розбивається на регіони (сторінки) константного розміру. Кожна лінійна адреса в емульованій системі зіставляється з фізичною адресою через таблицю сторінок, що імітує відображення сторінок у фрейми фізичної пам’яті.

Окрім цього, система-емулятор відтворює функціонування буфера асоціативної трансляції адрес (TLB) – спеціалізованого кешу ЦПУ, що прискорює перетворення віртуальної (лінійної) адреси у фізичну. У моделях сучасних процесорів запис у TLB містить відомості про нещодавно використані пари віртуальна–фізична адреса, що знижує затримки доступу до пам’яті. Такий

емульований механізм ідеально підходить для засвоєння студентами принципів роботи ізольованих адресних просторів, захисту пам’яті та оптимізації доступу до даних.

Загалом система-емулятор виступає навчальним засобом, що дозволяє наочно вивчати та контролювати процеси формування і трансляції адрес у багаторівневій пам’яті. Роль цього програмного забезпечення у навчальному процесі полягає в тому, щоб підвищити ефективність викладання курсу «Операційні системи» та суміжних дисциплін, зробити матеріал більш наочним і доступним. Студенти можуть експериментально досліджувати роботу сегментної і сторінкової адресації, аналізувати поведінку TLB і зрозуміти, як логічна адреса перетворюється на фізичну в реальних комп’ютерних архітектурах.

### 1.2 Область застосування

Розроблене програмне забезпечення має широкі можливості використання у сфері освіти, зокрема у підготовці фахівців у галузі комп’ютерної інженерії, програмної інженерії та інформаційних технологій. Його основна функція – забезпечення наочного й інтерактивного вивчення принципів управління пам’яттю в сучасних комп’ютерних системах, зокрема трансляції логічних адрес у лінійні та лінійних у фізичні з використанням механізмів сегментації, сторінкової організації пам’яті та буфера TLB.

**Основні напрями використання програмного забезпечення:**

У закладах вищої освіти програма може бути інтегрована до навчального процесу технічних спеціальностей при вивченні курсів. В межах цих курсів емулятор використовується як допоміжний засіб для проведення лекцій, лабораторних і практичних занять. Він дозволяє викладачам демонструвати процес трансляції адрес у реальному часі, показуючи структуру та зміну даних у таблиці сторінок, TLB та фізичній пам’яті.

Для самостійного навчання студентів Емулятор дозволяє студентам виконувати самостійні завдання і дослідження, моделюючи різні сценарії управління пам’яттю. Зокрема, студент може самостійно задавати параметри логічних адрес, сегментів, розмірів сторінок та кількості кадрів, а також аналізувати результативність доступу до пам’яті з урахуванням наявності або відсутності TLB. Це сприяє кращому розумінню принципів побудови адресного простору і підвищує рівень засвоєння теоретичних знань.

У навчальних лабораторіях та IT-центрах Програмне забезпечення може бути впроваджене у спеціалізовані лабораторії при факультетах комп’ютерних наук для комплексного вивчення роботи апаратного і програмного рівнів обробки адрес. Лабораторії можуть проводити курси, тренінги та семінари з використанням цієї системи, надаючи слухачам інструмент для глибшого розуміння функціонування віртуальної пам’яті, багаторівневих таблиць сторінок, кешування адрес та обробки сторінкових переривань.

Для використання в системах дистанційного навчання (e-learning) Завдяки модульній структурі, програма може бути адаптована для інтеграції у віртуальні навчальні середовища (LMS – Learning Management System). Вона може стати частиною онлайн-курсів або платформ дистанційного навчання, забезпечуючи студентів можливістю практичного досвіду з теми адресації у зручному інтерактивному форматі.

У технічній підготовці фахівців підвищення кваліфікації: Система може використовуватись у програмах підвищення кваліфікації для IT-фахівців, які прагнуть поглибити свої знання з комп’ютерної архітектури, операційних систем або системного програмування. Це особливо актуально для викладачів коледжів, технікумів, а також фахівців з експлуатації комп’ютерних систем.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень

### Опис наявних емуляторів і тренажерів пам’яті

У процесі вивчення архітектури комп’ютера та організації пам’яті важливу роль відіграють програмні інструменти, що дозволяють емулювати роботу процесора, управління пам’яттю, віртуалізацію та трансляцію адрес. Серед доступних навчальних та дослідницьких емуляторів можна виділити наступні системи:

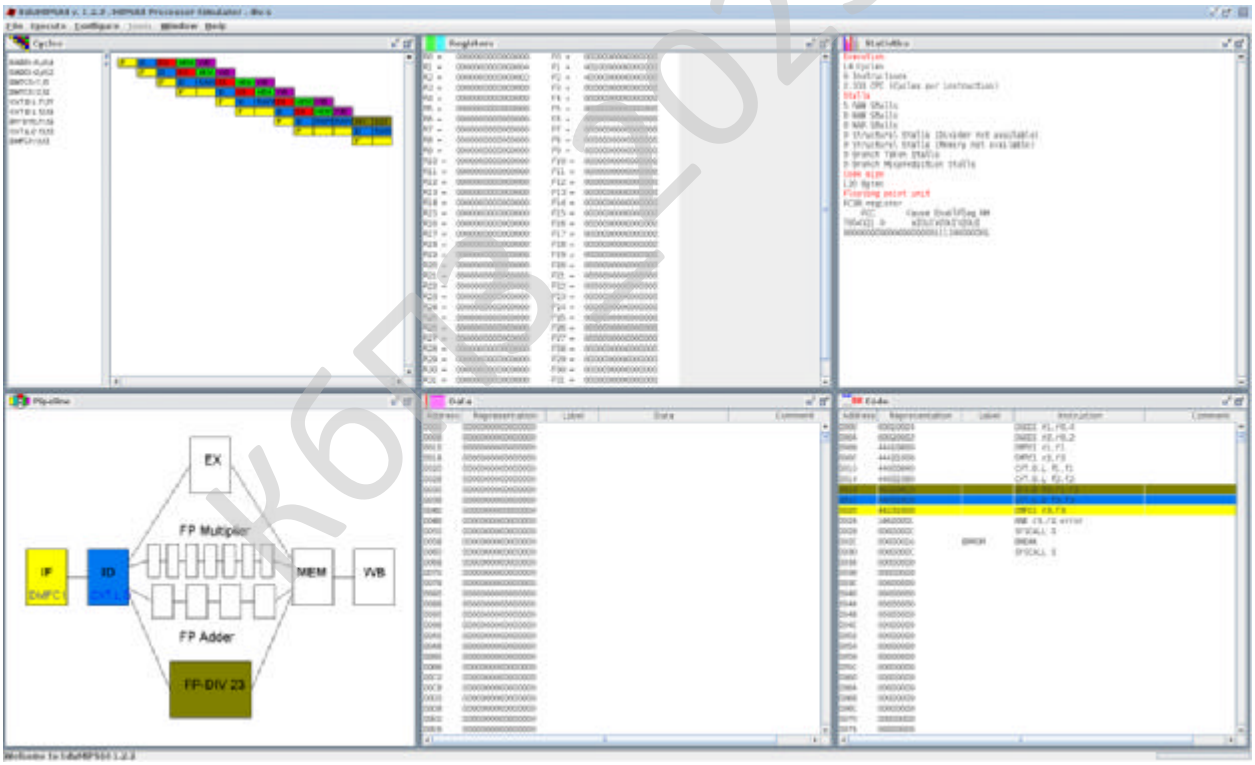


Рисунок 2.1— Інтерфейс додатку EduMIPS64

Під час дослідження існуючих інструментів для вивчення архітектурних особливостей мікропроцесорів особливу увагу було звернено на графічний емулятор EduMIPS64(рисунок 2.1).

Цей симулятор реалізовано з прицілом на навчальні потреби: користувач отримує можливість подавати програми на MIPS64-асемблері та спостерігати, як змінюються вміст регістрів і ділянки пам’яті під час покрокового виконання інструкцій. Інтерфейс EduMIPS64 дозволяє працювати з символьними мітками, досліджувати порожні й заповнені області пам’яті, а також переглядати всі ключові регістри процесора в режимі реального часу.

Незважаючи на ці переваги, у рамках нашої дипломної роботи було виявлено, що EduMIPS64 не моделює жодних механізмів сегментації чи багаторівневого сторінкового відображення, характерних для архітектури x86. У симуляторі відсутні як Translation Lookaside Buffer, так і логіка обробки page-fault, тобто він не відображає сценарії, коли звернення виходить за межі фізичної пам’яті або коли необхідно підвантажити сторінку з «диска». Відтак, хоча EduMIPS64 вдало демонструє базові принципи роботи асемблера та конвеєра MIPS, його функціональність виявилася недостатньою для цілей глибокого вивчення сучасних механізмів віртуальної пам’яті. Це й зумовило потребу розробити власний емулятор із підтримкою сегментації, багаторівневого сторінкування та TLB, що дасть змогу студентам наочно побачити всі етапи трансляції адрес від логічної до фізичної.

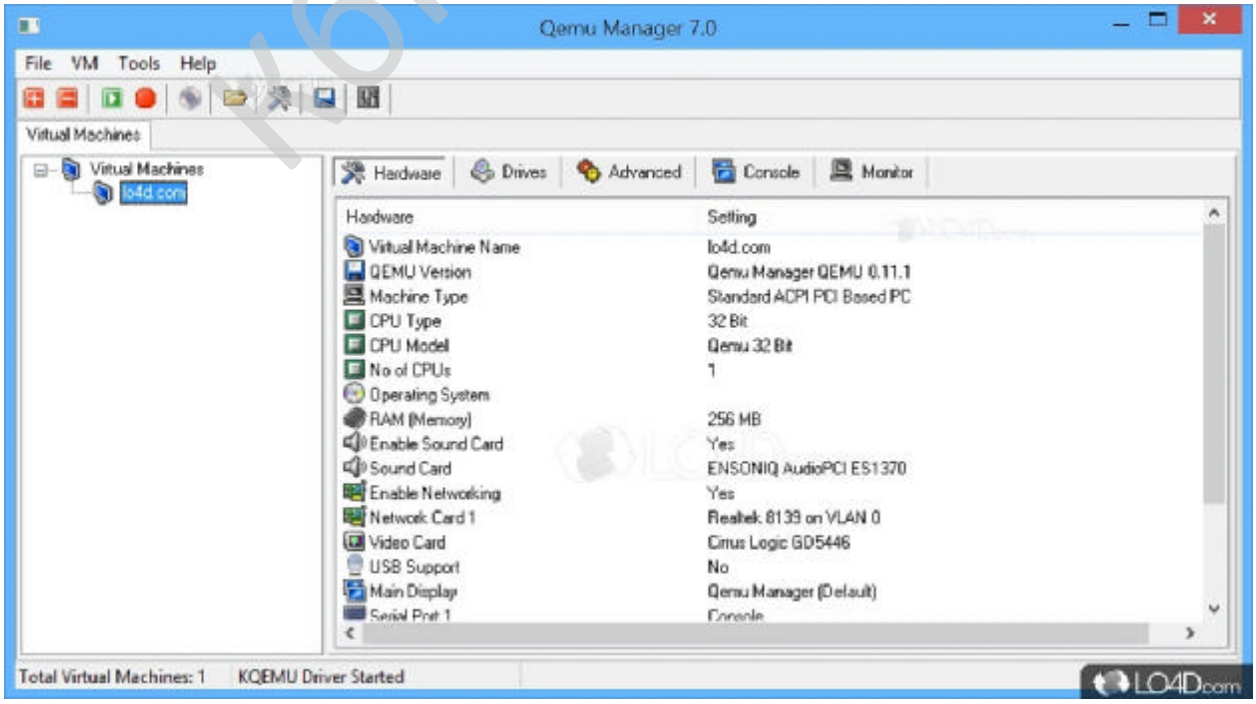

Рисунок 2.2 — Інтерфейс додатку QEMU

При підготовці огляду наявних засобів для емуляції апаратних платформ було досліджено можливості QEMU, потужного програмного рішення, яке дозволяє відтворювати різноманітні архітектури CPU (x86, ARM, RISC-V та інші) разом із відповідними підсистемами пам’яті та периферії. У QEMU реалізовано повну емуляцію апаратної платформи: віртуальний процесор інтерпретує машинні інструкції вибраної архітектури, модель пам’яті підтримує багаторівневе сторінкове відображення з апаратним MMU та кешем TLB, а периферійні пристрої — таймери, контролери вводу-виводу, мережеві адаптери — імітуються за допомогою внутрішніх моделей.

Крім чистої емуляції, QEMU може працювати в режимі системної віртуалізації із використанням прискорення KVM, що значно підвищує продуктивність гостьової ОС. У такій конфігурації гостьова система взаємодіє з реальним апаратним MMU й TLB, тоді як QEMU покриває лише частину інтерпретації інструкцій та вводу-виводу.

Незважаючи на всю потужність і гнучкість QEMU, у контексті освітнього процесу його застосування має суттєві обмеження. По-перше, немає жодного графічного або текстового інтерфейсу, призначеного для покрокового відображення процесу трансляції віртуальної адреси в фізичну. По-друге, для налаштування та контролю MMU, TLB та page-fault handler необхідні глибокі знання внутрішньої архітектури операційної системи та системного програмування. У результаті студенти швидше знайомляться з конфігурацією віртуальних машин і налаштуванням прийомів віртуалізації, але не бачать наочної візуалізації ключових моментів роботи віртуальної пам’яті.

Отже, незважаючи на те, що QEMU залишається незамінним інструментом для дослідників і розробників віртуалізованих середовищ, його функціональність виявилася недостатньою для навчальних цілей, де важливі інтуїтивне розуміння та наочна демонстрація роботи сегментації, багаторівневого сторінкування і кешування через TLB. Ці обмеження обґрунтовують необхідність створення власного освітнього емулятора з інтерактивною візуалізацією всіх етапів трансляції адрес.

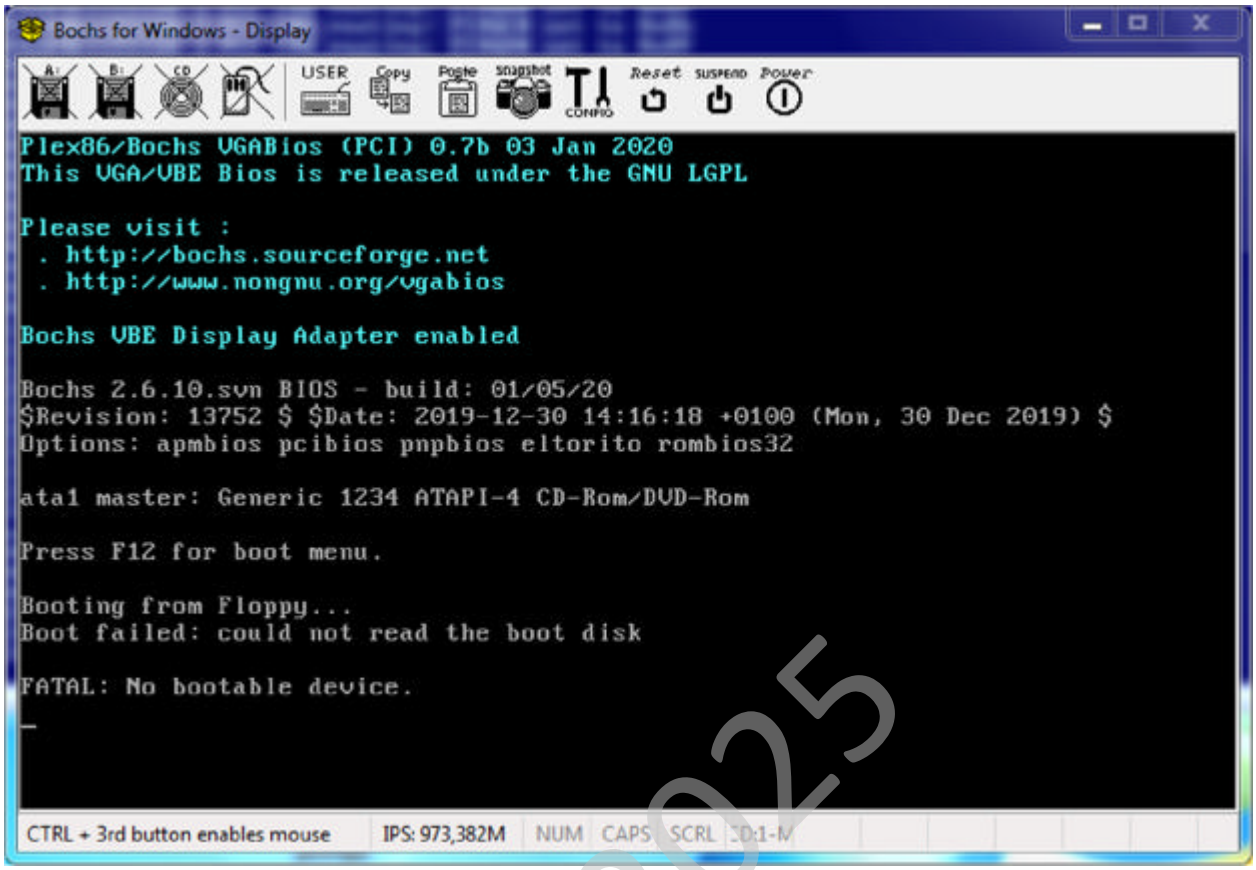


Рисунок 2.3 — Інтерфейс додатку BOCHS

У процесі аналізу наявних емуляторів апаратного забезпечення для освітніх цілей особливу увагу було приділено Bochs — Basic Operating System Hardware Emulator. Цей інструмент спроектовано з урахуванням покрокового вивчення механізмів роботи простих операційних систем, тому він моделює ключові апаратні компоненти: центральний процесор, фізичну пам’ять і базові пристрої введення-виведення. У Bochs передбачений власний графічний інтерфейс, який дозволяє відслідковувати виконання інструкцій і зміну вмісту регістрів у реальному часі, а також спрощену модель диспетчера пам’яті для керування виділенням та звільненням блоків пам’яті.

Що стосується підтримки віртуальної пам’яті, Bochs реалізує базові принципи сегментації: можна встановити дескриптори сегментів із відповідними базовими адресами та межами, після чого спостерігати, як відбувається перевірка доступу при спробі звернення за межі сегмента. У частині сторінкової організації

Bochs дозволяє задати таблицю сторінок і виконати первинне мапування лінійних адрес на фізичні фрейми, однак ця підтримка обмежується єдиним рівнем сторінкування без реалізації асоціативного кешу TLB.

Крім того, у Bochs відсутній механізм обробки page-fault у класичному розумінні: якщо програма звертається до неініціалізованої сторінки, емулятор або аварійно завершує роботу, або імітує лише базове повідомлення про помилку, не надаючи можливості підвантажити сторінку з “диску” та продовжити обчислення. Також у Bochs не передбачено демонстрації політик витіснення сторінок — немає можливості побачити, за яким алгоритмом (LRU, FIFO чи іншим) здійснюється заміна фреймів.

Таким чином, Bochs хоча й корисний для ознайомлення з базовими підходами до сегментації та сторінкової організації, не забезпечує повноцінного покрокового моделювання механізмів TLB і page-fault handler, необхідних для глибокого розуміння сучасної архітектури віртуальної пам’яті. У зв’язку з цим у рамках даної дипломної роботи постає завдання розробити власний емулятор із інтерактивною візуалізацією всіх етапів трансляції адрес, підтримкою багаторівневого сторінкування, асоціативного кешу перекладів і коректною обробкою page-fault.

Розглянуті емулятори мають різний рівень деталізації та призначення. EduMIPS64 орієнтований на початкове знайомство з машинними командами, QEMU — це високотехнологічний засіб для професіоналів, а BOCHS — базовий симулятор для системного навчання. Жодна з цих систем не реалізує повноцінну візуалізацію покрокової трансляції адрес із сегментацією, сторінковістю, TLB та обробкою page-fault у навчальному контексті. Це підтверджує доцільність розробки спеціалізованого програмного забезпечення, яке би дозволяло студентам наочно зрозуміти процеси трансляції логічної адреси у фізичну.

**Аналіз переваг і недоліків**

Для оцінки доцільності використання існуючих рішень було проведено стислий порівняльний аналіз найпоширеніших навчальних і дослідницьких емуляторів пам’яті за такими критеріями:

a) Зручність використання, наскільки швидко користувач може освоїти інтерфейс та логіку роботи.

b) Гнучкість налаштувань — можливість модифікувати конфігурацію емуляції.

c) Наявність української або російської мови — локалізація інтерфейсу для зручності студентів.

d) Підтримка TLB — чи реалізована трансляція адрес із використанням буфера асоціативної пам’яті (TLB).

e) Візуалізація процесу трансляції — наочне відображення етапів трансляції логічної адреси у фізичну.

Таблиця 2.1 - Порівняльна таблиця

| Критерій | EduMIPS64 | QEMU | BOSHE |
|---|---|---|---|
| Зручність використання | Висока | Низька (CLI, складність) | Середня |
| Гнучкість налаштувань | Низька | Висока | Середня |
| Наявність української/російської мови | Ні | Ні | Частково (англ.+локаліз.) |
| Підтримка TLB | Ні | Так | Ні |
| Візуалізація трансляції | Ні | Ні | Часткова |

Проведений аналіз показує, що жодна з існуючих систем не забезпечує одночасно візуалізацію процесу трансляції, підтримку TLB, простоту використання та навчальну орієнтованість**.** Таким чином, існує потреба в розробці спеціалізованого програмного забезпечення**,** яке би забезпечувало

покрокову візуалізацію трансляції логічної адреси у фізичну, враховуючи всі основні етапи — сегментацію, сторінкову організацію, TLB і обробку page-fault.

Порівняльний аналіз трьох провідних навчальних та дослідницьких емуляторів пам’яті — EduMIPS64, QEMU та Bochs — показав, що жоден із них не задовольняє всіх вимог для повноцінного викладання сучасних механізмів трансляції адрес. По-перше, інтерфейси наявних рішень застаріли або надмірно спрощені й не дають змоги наочно демонструвати роботу сегментації, багаторівневого сторінкування та кешування в TLB. EduMIPS64, хоч і підтримує асемблерний режим із відображенням регістрів і конвеєра, не містить жодного візуального компонента для аналізу керування пам’яттю; Bochs, незважаючи на потужні можливості емуляції x86-систем, не забезпечує інтерактивних інструментів для покрокового відстеження трансляції адрес; а QEMU взагалі орієнтований на емулювання апаратних платформ із мінімальними текстовими повідомленнями, але без будь-яких графічних пояснень внутрішніх структур пам’яті.

По-друге, жоден із цих інструментів не реалізує глибоке моделювання всіх етапів віртуальної пам’яті в інтерактивному режимі. Актуальні платформи не демонструють, як на практиці формується лінійна адреса на основі сегментного регістру, як відбувається багаторівневе відображення через каталог і таблицю сторінок, як працює апаратний кеш перекладу (TLB) з відстеженням попадань та промахів, і як операційна система обробляє page fault із завантаженням відсутніх сторінок.

По-третє, навіть підтримка окремих алгоритмів виявилася неповною. TLB-моделювання присутнє лише в QEMU, але без жодних засобів для візуального аналізу чи отримання статистики на рівні користувача. Обробники page fault у Bochs та EduMIPS64 обмежуються попередньо заданими статичними зверненнями і не дають студентам змоги вручну ініціювати та відстежувати підвантаження сторінок із «диска». Щодо політик евікції кешу — наприклад, LRU чи FIFO — то сучасні підходи або взагалі відсутні в інтерфейсі, або ж приховані всередині ядра емулятора, недоступні для експериментів.

Нарешті, існуючі рішення недостатньо адаптовані саме під освітні потреби. Жоден із розглянутих емуляторів не має локалізованого інтерфейсу українською мовою або україномовної документації, що значно ускладнює їхнє використання в наших навчальних закладах. Також відсутні готові сценарії лабораторних занять, які б демонстрували різні аспекти віртуального адресного простору, залишаючи викладачам дедалі менше можливостей залучати студентів до активного вивчення ключових концепцій сегментації, сторінкування та кешування.

Таким чином, вищезазначені обмеження існуючих інструментів відкривають вікно для розробки нового емулятора, що поєднуватиме інтерактивну візуалізацію кожного кроку трансляції адрес із можливістю гнучкого налаштування алгоритмів кешування й обробки виключень, а також матиме зручний україномовний інтерфейс і готові навчальні сценарії.

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

У якості основи для розробки навчального емулятора трансляції адрес було обрано мову C# у поєднанні з платформою .NET. Перш за все, C# пропонує потужну об’єктно-орієнтовану модель, що дозволяє чітко розділяти функціональність на незалежні класи та модулі. Завдяки інкапсуляції, наслідуванню й поліморфізму можна окремо реалізувати компоненти сегментації, керування TLB, обробки page-fault та графічного інтерфейсу, що значно спрощує підтримку коду, його тестування та подальший розвиток.

Вибір платформи .NET (версій Core, 5, 6) забезпечує додаткові переваги: завдяки крос-платформенності створений додаток може стабільно працювати не тільки під Windows, а й під Linux або macOS. Це робить систему доступною у більшості лабораторних середовищ і дає змогу студентам використовувати улюблене оточення без додаткових налаштувань.

Для побудови зручного та наочного інтерфейсу обрано Windows Presentation Foundation (WPF). Ця технологія підтримує гнучке масштабування вікон, динамічні візуалізації через data-binding та готові контроли, анімації підсвічування й переміщення блоків. Саме такі можливості необхідні для показу покрокових перетворень логічної та лінійної адрес, а також для демонстрації роботи TLB і page-fault handler у режимі реального часу.

Практичне застосування мов програмування у вітчизняній освіті також зіграло свою роль: C# є однією з найпоширеніших мов, якою вже володіють студенти технічних спеціальностей. Використання знайомого синтаксису та звичних бібліотек дозволяє зосередитися на вивченні алгоритмів трансляції адрес замість додаткового опанування нових мов програмування.

Крім того, екосистема .NET містить багатий набір готових компонентів для роботи з файлами (System.IO), колекціями (System.Collections.Generic), обробки виключень і багатопоточності. Це спрощує реалізацію бекстору для моделювання підвантаження сторінок із “диска”, симуляцію фізичної пам’яті та ведення статистики в реальному часі.

Нарешті, C# забезпечує автоматичне керування пам’яттю (GC), сувору типізацію та потужні механізми виняткової обробки. Для навчального програмного забезпечення, де надійність і стабільність мають першочергове значення, такі засоби захисту від типових помилок (null-посилання, переповнення буферів тощо) є суттєвою перевагою.

Таким чином, застосування C# та .NET із WPF дозволяє поєднати високий рівень продуктивності, легкість розробки й підтримки, а також якісний, інтуїтивно зрозумілий інтерфейс, повністю відповідаючи вимогам освітнього середовища.

### 2.3 Розгорнута постановка завдання

Метою даної випускної кваліфікаційної роботи є створення програмного забезпечення, яке моделює процес перетворення логічної адреси у лінійну, а

лінійної у фізичну, з візуалізацією усіх етапів трансляції з урахуванням таких механізмів, як сегментація, сторінкова організація пам’яті та робота буфера трансляції сторінок (TLB). Розроблена система має використовуватися у навчальному процесі для пояснення складних аспектів адресації в операційних системах.

– З огляду на поставлену мету було сформульовано наступні основні задачі розробки:—

– Реалізувати логіку перетворення логічної адреси у лінійну відповідно до принципів сегментації, які використовуються у сучасних комп’ютерних системах. Програмне забезпечення повинне враховувати сегментні регістри, бази та межі сегментів, доступні режими адресації.

– Реалізувати механізм сторінкового перетворення лінійної адреси у фізичну, що включає побудову і використання таблиць сторінок, двохрівневої трансляції адрес, а також перевірки на сторінкові помилки (page fault).

– Впровадити модуль емуляції буфера TLB з підтримкою кешування адрес та обліком промахів (misses) і влучень (hits), що дозволяє вивчати ефективність роботи буфера в реальному часі.

– Забезпечити можливість крокового виконання трансляції з коментарями та поясненнями кожного етапу для полегшення розуміння користувачем.

– Реалізувати окремий навчальний режим, у якому студенти зможуть виконувати завдання з адресації з автоматичною перевіркою відповідей та підказками.

– Забезпечити збереження та завантаження конфігурацій для повторного використання на лабораторних заняттях або в самостійній роботі.

– Розробити документацію для викладачів і студентів, яка пояснює використання програмного забезпечення, а також методичні рекомендації щодо інтеграції інструменту в освітній процес.

Таким чином, програмне забезпечення повинно стати не лише інструментом візуалізації, але і інтерактивною навчальною платформою для поглибленого вивчення механізмів управління пам’яттю в комп’ютерних системах.

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

У цьому підрозділі розглядаються базові поняття та загальна логіка роботи емульованої системи трансляції адрес. Емулятор моделює основні механізми керування пам’яттю сучасних процесорів, зокрема:

a) Логічна адреса — адреса, яку формує програма, складається з двох частин: сегментного селектора (який вказує на конкретний сегмент пам’яті) та внутрішнього зсуву всередині цього сегмента.

b) Лінійна адреса — результат простого додавання базової адреси сегмента до зсуву. На цьому етапі відбувається первинна трансляція без урахування сторінкової організації.

c) Фізична адреса — справжня адреса в оперативній пам’яті, отримана після багаторівневої сторінкової трансляції лінійної адреси через структури Page Directory і Page Table.

d) Сегментація — спосіб поділу адресного простору на незалежні ділянки-сегменти, кожен із яких має власну базу та межу (limit). Використовується для ізоляції та захисту пам’яті.

e) Сторінкова організація (Paging) — поділ лінійного адресного простору на фіксовані блоки — сторінки (pages) та їх відображення в фізичні блоки — фрейми (frames) за допомогою таблиць.

f) TLB (Translation Lookaside Buffer) — невеликий кеш швидких перекладів «номер сторінки → фізичний фрейм», що дозволяє прискорювати багаторазові звернення до тих самих ділянок пам’яті.

Емулятор поєднує ці механізми в єдиний послідовний процес:

a) Сегментація переводить програмну (логічну) адресу у лінійну, перевіряючи межі сегмента.

b) Сторінкова трансляція розбиває лінійну адресу на індекси у Page Directory та Page Table, знаходить відповідний фізичний фрейм і обчислює остаточну фізичну адресу.

c) TLB використовується як кеш-прискорювач: перед зверненням до таблиць система перевіряє, чи не було нещодавно виконано аналогічний переклад, щоб у разі успіху миттєво отримати результат.

Користувач взаємодіє з інтерфейсом крок за кроком, вводячи селектор і зсув, спостерігаючи підсвічені блоки в таблицях і отримуючи змістовні повідомлення про кожний етап трансляції. Такий поетапний підхід робить складні внутрішні процеси управління пам'яттю наочними й зрозумілими для студентів.

**Трансляція логічної адреси у лінійну**

У захищеному режимі x86 програмний код посилається на пам'ять за допомогою логічних адрес, що складаються з двох 16-бітних компонентів: селектора сегмента та внутрішнього зсуву (offset). Селектор сегмента, рисунок 3.1, одночасно містить інформацію про те, який дескриптор вибрано (індекс у таблиці GDT або LDT), і кілька бітів привілеїв процесу, що звертається до пам'яті.

У нашому емуляторі всі дескриптори з індексами від 0 до N зберігаються в класі SegmentDescriptorView, а користувач вибирає їх із випадаючого списку, у якому селектор формується як бітовий зсув номера дескриптора вліво на три позиції. Це забезпечує точну відповідність форматам апаратних селекторів та дозволяє наочно показати, як саме формуються адресні поля на рівні процесора.

Наступним етапом є обчислення внутрішнього зсуву — тієї частини логічної адреси, яка позначає відстань від початку сегмента до конкретного байта. У реальних процесорах ця величина часто називають ефективною адресою (EA) і вона формується з урахуванням кількох складових: значення базового регістра, значення індексного регістра, масштабного множника (Scale) і константного зміщення (Displacement). У навчальному емуляторі ця складна формула попередньо обчислюється поза межами графічного інтерфейсу, а користу-

вачеві пропонується ввести вже готове 16-бітне значення зсуву, що значно спрощує взаємодію й одночасно дає змогу зосередитися на самому механізмі сегментування.

Після вибору селектора та введення offset система перевіряє, чи не виходить зсув за межі сегмента. Для цього порівнюється Offset із полем Limit обраного дескриптора. У разі перевищення кордонів сегмента емулятор миттєво генерує повідомлення про помилку захисту пам’яті й підсвічує відповідний рядок у таблиці дескрипторів червоним кольором, доки користувач не виправить значення. Якщо Offset знаходиться в межах допустимого діапазону, програма обчислює лінійну адресу шляхом простого додавання базової лінійної адреси сегмента (SegmentBase) та значення offset. Цей результат у вигляді єдиного 32-бітного числа відображається в окремому блоці інтерфейсу, де він одразу підсвічується зеленим як успішно обчислений.

У графічному інтерфейсі кожний із цих кроків супроводжується анімацією та текстовими підказками: вибір селектора призводить до виділення відповідного рядка в таблиці дескрипторів, перевірка offset ілюструється миготінням комірок у блоці offset, а обчислення лінійної адреси відбувається за рахунок плавного переміщення значень полів Base і Offset у блок результату. Такий підхід дозволяє не лише формально відпрацювати алгоритм сегментації, але й глибше зрозуміти внутрішню логіку роботи апаратних компонентів — від формування селекторів і обчислення ефективної адреси до контролю доступу та остаточного складання лінійної адреси, яка стане вхідним параметром для наступного етапу сторінкового перетворення пам’яті.

**Сторінкова трансляція та кешування в TLB**

Після трансляції логічної адреси у 32-бітну лінійну, відбувається її розбиття на поля, необхідні для багаторівневої сторінкової трансляції. Старші 10 біт цієї адреси визначають індекс рядка у таблиці директорій сторінок (Page Directory), яка складається з 1024 записів. Кожен запис, так званий PDE (Page Directory Entry), містить інформацію про наявність сторінки в пам’яті (біт

Present/Absent), а також базову фізичну адресу відповідної таблиці сторінок (Page Table).

Наступні 10 біт адреси використовуються як індекс для вибору запису у відповідній таблиці сторінок, що також є масивом з 1024 елементів. Кожен запис у цій таблиці — PTE (Page Table Entry) — містить біт наявності фрейму (Present/Absent) і фізичну адресу початку конкретної сторінки.

Молодші 12 біт лінійної адреси є зсувом (Offset) всередині сторінки розміром 4 КБ, що дозволяє доступ до конкретного байта в межах обраного фрейму.

Перед здійсненням повної трансляції спочатку виконується перевірка Translation Lookaside Buffer (TLB) — кешу, який зберігає останні результати трансляції сторінкових адрес. Для цього з лінійної адреси вилучається номер сторінки (PageNumber), який складається з усіх бітів, окрім Offset. Якщо у TLB є відповідний запис (TLB-hit), фізична адреса початку фрейму (FrameBase) береться безпосередньо звідти, і етапи звернення до Page Directory та Page Table пропускаються. У разі TLB-miss починається повна процедура сторінкової трансляції.

На першому етапі обирається відповідний запис у Page Directory за значенням DirectoryIndex. Якщо біт Present у PDE не встановлений, система генерує виняток — Page Fault. Інакше з цього запису отримується фізична адреса початку потрібної таблиці сторінок.

Далі виконується звернення до таблиці сторінок за індексом TableIndex. Аналогічно, якщо сторінка не присутня в пам’яті, виникає Page Fault. Якщо ж доступ дозволено, отримується адреса фрейму з відповідного PTE.

Після успішної трансляції у кеш TLB заноситься новий запис, що містить відповідність між PageNumber і FrameBase. Якщо кеш заповнений, один зі старих записів видаляється згідно з політикою заміщення: наприклад, за методом LRU (Least Recently Used) або FIFO (First-In First-Out).

Фізична адреса обчислюється додаванням зсуву до базової адреси фрейму( Physical_Address = FrameBase + Offset ).

TLB є невеликим кешем, зазвичай на 16–64 записів, що забезпечує дуже швидкий доступ до сторінкових трансляцій. Записи у TLB зберігають трійки: PageNumber, FrameBase та додаткові метадані про використання. Пошук здійснюється асоціативно, шляхом паралельного порівняння всіх записів, що дозволяє швидко знайти потрібний фрейм. Політика заміщення визначає, який запис буде видалено у разі переповнення кешу.

Перевагою використання TLB є значне зменшення кількості повних звернень до таблиць директорій і сторінок. При високому відсотку TLB-hit (від 70 до 99%) доступ до пам’яті стає значно швидшим.

Емулятор у навчальному середовищі демонструє всі ці етапи поетапно, підсвічуючи відповідні записи у візуальному інтерфейсі. Користувач отримує повідомлення про події, як-от TLB-hit, TLB-miss або Page Fault, що дозволяє чітко зрозуміти логіку роботи сучасних механізмів трансляції адрес у реальних процесорах.

**Фізична адреса**

Після завершення всіх кроків трансляції — сегментації, багаторівневої сторінкової адресації та кешування в TLB — утворюється фізична адреса**.**

Фізична адреса — це остаточна двійкова адреса, за якою центральний процесор звертається до комірок оперативної пам’яті.

У ній об’єднані:

a) FrameBase (адреса початку фізичного фрейму), отримана на етапі сторінкової трансляції (або з TLB у разі попадання).

b) Offset — залишковий зсув у межах сторінки (нижні 12 біт лінійної адреси).

Ця адреса потрапляє у зовнішню шину пам’яті та використовується апаратними контролерами DRAM для читання чи запису даних.

Саме фізична адреса є «останньою точкою» перетворень для процесора: далі він уже не оперує ні селекторами, ні номерами сторінок — лише цією абсолютною адресою.

Всі внутрішні механізми сегментації і сторінкування служать виключно для захисту, ізоляції та віртуалізації пам’яті; але без остаточної перекладки в фізичний простір сам процесор не може здійснити фактичний доступ до модулів оперативної пам’яті чи кешів нижчих рівнів.

Таким чином, фізична адреса — це кінцевий результат усієї послідовності перетворень, що реалізовані в нашому емуляторі, і саме вона є тією величиною, з якою працюють мікроархітектурні блоки пам’яті для зчитування та запису даних.

### 3.2 Розробка структурної схеми

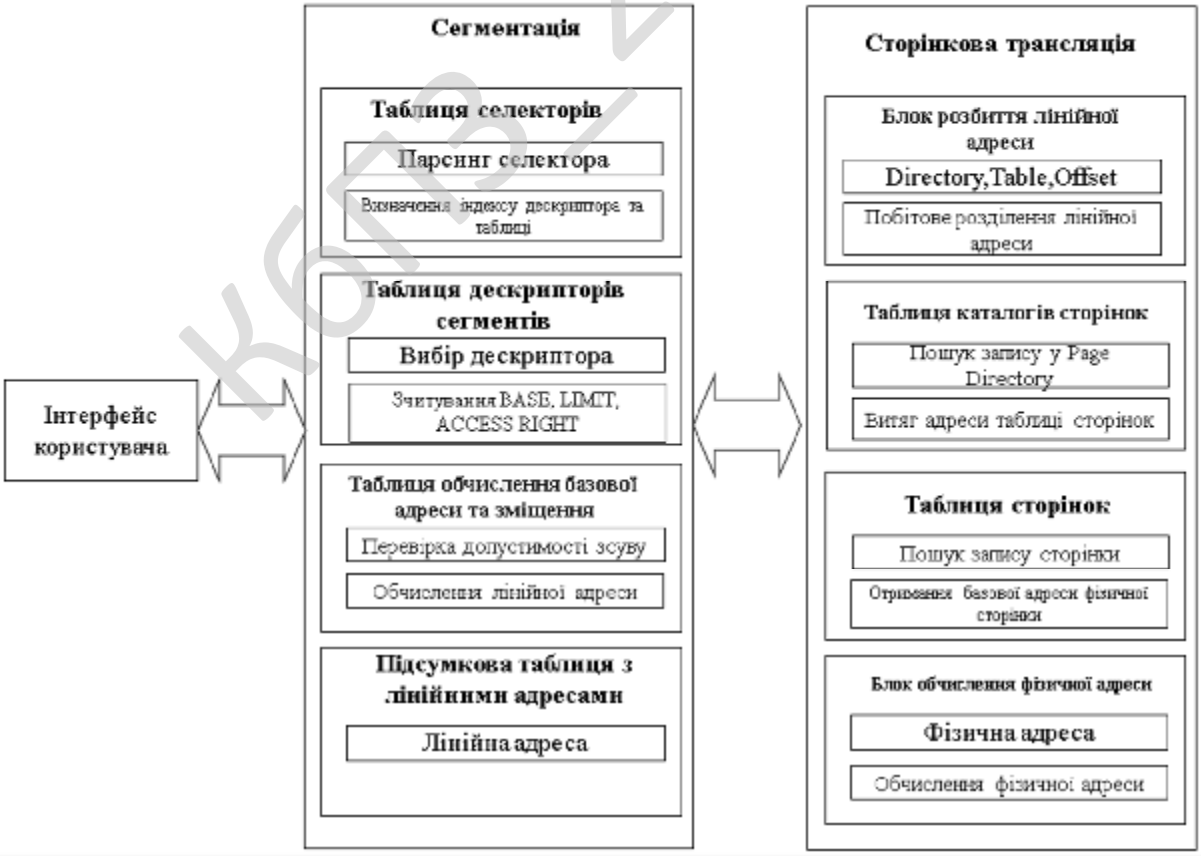


Рисунок 3.1 — Структурна схема системи

Для реалізації програмного забезпечення системи трансляції логічних адрес у фізичні було розроблено структурну схему, що представлена на рисунку 3.1.

Вона описує логічну побудову системи, розділену на окремі функціональні модулі, які взаємодіють між собою відповідно до етапів перетворення адрес. Схема є центральним елементом архітектури програмного забезпечення, оскільки визначає послідовність обробки введеної користувачем логічної адреси.

Структура системи поділена на три основні модулі.

**Інтерфейс користувача**

Цей модуль відповідає за взаємодію користувача з програмним забезпеченням. Він реалізований у вигляді настільного WPF-додатку, який має графічний інтерфейс та дозволяє здійснювати повноцінне керування процесом трансляції адрес. На початковому етапі користувач вводить логічну адресу у форматі "селектор:зсув", після чого система переходить до її обробки. Програма надає можливість переглядати значення селектора, обраного дескриптора та всі розраховані проміжні адреси. Усі результати, включно з розбиттям селектора, полями дескриптора, обчисленою лінійною та фізичною адресами, відображаються у зручному для сприйняття вигляді: через таблиці, підписані поля та повідомлення. Інтерфейс забезпечує покроковий перехід між етапами трансляції, дозволяючи користувачеві краще зрозуміти внутрішню логіку роботи системи. Таким чином, модуль інтерфейсу користувача виконує не лише функцію вводу та виводу, але й виконує роль навчального інструменту з візуалізації процесу трансляції адрес.

**Сегментація**

Модуль сегментації відповідає за перетворення логічної адреси у лінійну, що є першим етапом трансляції адрес у системах на архітектурі x86. Логічна адреса складається з двох основних компонентів: селектора та зсуву. На початку обробки виконується розбір селектора, який ділиться на три основні частини. Зокрема, з нього виділяється індекс, що дозволяє знайти номер запису в таблиці

дескрипторів; біт TI, який вказує, чи потрібно використовувати глобальну таблицю дескрипторів (GDT) чи локальну (LDT); а також поле RPL, що визначає рівень привілеїв запиту.

Після розбору селектора програма визначає, до якої саме таблиці слід звернутися — GDT чи LDT, — та витягує з неї дескриптор за вказаним індексом. З отриманого дескриптора вилучаються всі необхідні для подальших обчислень поля: базова адреса сегмента (BASE), межа сегмента (LIMIT), а також інформація про права доступу, яка дозволяє дізнатися, чи є сегмент кодовим або даними, а також з яким рівнем доступу можна працювати з цим сегментом.

Наступним кроком є перевірка правильності вказаного зсуву. Значення зсуву порівнюється з межами сегмента, і у випадку, якщо воно перевищує допустимий ліміт, система генерує помилку доступу за межі пам’яті. Якщо ж зсув знаходиться в межах допустимого, до базової адреси додається його значення, і таким чином формується лінійна адреса.

Після успішного завершення всіх обчислень лінійна адреса передається на подальшу обробку в модуль сторінкової трансляції, де вже відбуватиметься трансляція у фізичну адресу. Таким чином, модуль сегментації забезпечує коректну та безпечну трансляцію логічної адреси у лінійну відповідно до архітектурних принципів роботи пам’яті в системах x86.

**Сторінкова трансляція**

Модуль сторінкової трансляції виконує завершальний етап процесу трансляції адрес, а саме — перетворення лінійної адреси у фізичну. Він моделює роботу блоку керування пам’яттю (MMU) процесора, реалізуючи базовий принцип сторінкової організації пам’яті, що широко використовується в сучасних операційних системах для ефективного керування доступом до фізичної пам’яті.

На початку трансляції 32-бітна лінійна адреса розбивається на три логічні частини. Перша частина — це індекс у таблиці каталогів сторінок (Page Directory), який займає 10 старших бітів. Друга частина — індекс у таблиці сторінок (Page Table), що також має розмір 10 біт. Нарешті, третя частина — це зсув усередині сторінки (Offset), який займає 12 молодших бітів. Для виділення

кожної з цих частин з адреси використовуються бітові маски та побітові зсуви, що дозволяє точно визначити, до яких записів у таблицях слід звертатися.

Після розбиття адреси модуль виконує звернення до таблиці каталогів сторінок. За обчисленим індексом із цієї таблиці витягується 32-бітовий запис, який містить адресу на базову таблицю сторінок. З цього запису видобувається базова адреса таблиці сторінок, що потрібна для наступного етапу трансляції.

Отримавши доступ до таблиці сторінок, програма за другим індексом — індексом таблиці сторінок — витягує запис, який вказує на базову адресу фізичної сторінки пам’яті. У цьому записі зберігається фізична адреса фрейму, до якого належить потрібна сторінка. Важливо, що на цьому етапі ще не визначено остаточну фізичну адресу — лише визначено її базову частину.

На завершальному етапі обчислюється остаточна фізична адреса. Це здійснюється шляхом додавання зсуву, виділеного на самому початку, до базової адреси фізичної сторінки, отриманої з таблиці сторінок. У результаті цих обчислень формується повна фізична адреса, яка й буде використана для безпосереднього доступу до відповідної комірки фізичної пам’яті.

Таким чином, модуль сторінкової трансляції забезпечує завершення процесу трансляції логічної адреси у фізичну, демонструючи повний цикл адресного перетворення в умовах багаторівневої організації пам’яті.

Таким чином, наведена схема реалізує повноцінну двоетапну трансляцію логічної адреси у фізичну, охоплюючи всі етапи, передбачені в архітектурі x86, включаючи захист пам’яті, багаторівневу адресацію та механізм сторінкової організації.

### 3.3 Розробка функціональної схеми


Рисунок 3.2 – Функціональна схема системи

Функціональна схема є важливим етапом моделювання системи, що дозволяє відобразити логіку її роботи та розподіл функціональних обов'язків між ключовими компонентами.

Вона дозволяє детально описати, які дії виконує кожен блок програмного забезпечення, як дані передаються між цими блоками, а також як здійснюється обробка і трансляція логічної адреси у фізичну.

**Призначення функціональної схеми**

Функціональна схема системи виконує такі основні завдання:

− Демонструє, як саме реалізується поетапна трансляція логічної адреси у фізичну;

− Дозволяє побачити потоки даних між компонентами;

− Забезпечує візуалізацію обробки помилок;

− Описує взаємодію з інтерфейсом користувача.

Функціональна схема охоплює всі етапи обробки адреси — від введення даних користувачем до виведення результату трансляції. Вона поділена на логічні блоки з наступною функціональністю:

− обробка логічної адреси;

− обчислення лінійної адреси;

− реалізація сторінкової організації пам’яті;

− взаємодія з буфером трансляцій (TLB);

− моделювання помилок;

− формування кінцевого результату.

**Опис функціональних блоків системи**

Блок обробки введених даних відповідає за початкову взаємодію з користувачем. На цьому етапі здійснюється введення селектора сегмента та зсуву, визначається режим трансляції — із використанням TLB, без нього або з моделюванням помилок. Також ініціалізуються таблиці пам’яті та встановлюються базові параметри емуляції.

У структурі моделі використовуються ключові елементи пам’яті: таблиця дескрипторів сегментів, директорія сторінок (Page Directory), таблиця сторінок (Page Table), буфер асоціативної пам’яті (TLB) та простір фізичної пам’яті. Ці структури зберігають важливу інформацію для забезпечення правильної трансляції адрес.

Операційна частина системи виконує перевірку коректності селектора, здійснює пошук відповідного дескриптора сегмента та аналізує зсув на предмет його відповідності встановленому діапазону. Далі обчислюється базова адреса сегмента, формується лінійна адреса, перевіряється наявність запису в TLB. У разі його відсутності здійснюється звернення до записів у директорії та таблиці сторінок, після чого формується фізична адреса і при потребі оновлюється TLB.

Системна частина відповідає за контроль прав доступу до пам’яті, а також за обробку помилок трансляції. Це може бути як класичний Page Fault, так і ситуація порушення меж сегмента або спеціально змодельовані виняткові

обставини. Крім того, ця частина контролює правильність усіх вхідних параметрів, що надходять у систему.

Інтерфейс користувача дозволяє вручну ввести логічну адресу або згенерувати її автоматично. Після цього можна ініціалізувати процес трансляції та переглядати його поетапно. На кожному кроці користувач бачить проміжні й фінальні результати, а також має змогу переглянути вміст TLB, Page Table та Page Directory. Крім того, доступні функції очищення або оновлення даних.

Логіка виконання системи побудована на послідовному проходженні всіх етапів трансляції. Після введення логічної адреси здійснюється перевірка її коректності. З таблиці дескрипторів визначається базова адреса сегмента, яка слугує основою для формування лінійної адреси. Система перевіряє, чи містить TLB відповідний запис. Якщо такий запис присутній, фізична адреса отримується безпосередньо. У протилежному випадку система звертається до Page Directory та Page Table для завершення трансляції. У результаті формується фізична адреса або генерується помилка на кшталт Page Fault. Отриманий результат відображається на екрані користувача.

Усі етапи трансляції візуалізуються за допомогою графічного інтерфейсу, таблиць та повідомлень.

На рисунку 3.2 зображено функціональну схему системи у вигляді діаграми потоків даних. Вона включає наступні компоненти:

a) Джерело даних: користувач (логічна адреса, параметри).

b) Процеси: обробка селектора, формування адрес, перевірки, звернення до таблиць.

c) Сховища: дескриптори, TLB, Page Directory, Page Table.

d) Результат: виведення фізичної адреси та повідомлень.

Розробка функціональної схеми дозволила сформувати чітку логічну структуру системи. Завдяки розмежуванню компонентів за функціональним призначенням вдалося забезпечити модульність, прозорість та гнучкість програмного забезпечення. Такий підхід спрощує налагодження, подальший розвиток та використання емулятора у навчальному процесі.

### 3.4 Розробка діаграми процесів


Рисунок 3.3 — Діаграма процесів

У цьому підрозділі подано опис логіки виконання основних процесів емуляції трансляції логічної адреси у фізичну за допомогою сегментації, сторінкової організації пам’яті та буфера TLB. Для наочного подання використано діаграму потоків управління, що демонструє послідовність дій та взаємозв’язки між ключовими етапами трансляції адрес.

**Основні процеси системи**

На діаграмі (рисунок 3.3) відображено три основні обчислювальні процеси:

а) Сегментація - початковий етап, під час якого здійснюється перетворення логічної адреси (яка складається із селектора та зсуву) у лінійну адресу, враховуючи базову адресу сегмента та його обмеження.

b) Сторінкова трансляція - процес розбиття лінійної адреси на індекси Page Directory та Page Table з подальшим отриманням фізичної адреси за допомогою відповідних записів.

c) Буфер трансляцій (TLB) - механізм кешування результатів попередніх трансляцій для зменшення затримок при повторному зверненні до тих самих адрес.

**Детальний опис процесів**

1. Процес трансляції адреси розпочинається з моменту введення логічної адреси користувачем або програмою. Ця логічна адреса містить селектор та зсув, які далі обробляються в рамках сегментації пам’яті.

2. На етапі сегментації з селектора визначається індекс сегмента, за яким здійснюється звернення до таблиці дескрипторів. Після отримання дескриптора перевіряється, чи не перевищує зсув дозволене значення обмеження сегмента (Limit). У разі успішної перевірки формується лінійна адреса — як сума базової адреси сегмента та зсуву.

3. Наступним кроком є перевірка буфера трансляцій (TLB), який зберігає останні виконані трансляції. Якщо відповідного запису у TLB не виявлено, система звертається до таблиць сторінок: спочатку до директорії сторінок (Page Directory), а потім — до таблиці сторінок (Page Table). Отримані записи PDE та PTE використовуються для формування остаточної фізичної адреси шляхом підстановки фізичної бази сторінки та зсуву.

4. У разі успішного завершення трансляції результат зберігається у TLB, що дозволяє прискорити подальші звернення до пам’яті. Завершення процесу відбувається після виведення отриманої фізичної адреси користувачеві або використання її для подальших операцій системи.

**Аналіз діаграми**

У діаграмі (рисунок 3.3) логічно та послідовно представлено основні етапи обробки: від введення даних до генерації кінцевого результату. Також передбачено розгалуження залежно від наявності записів у TLB або результатів перевірки меж зсуву, що відображає реалізацію умовної логіки в системі.

Діаграма процесів демонструє ефективну структуру обробки трансляції адрес. Розділення процесів на окремі логічні блоки дозволяє легко адаптувати або модифікувати компоненти системи. Такий підхід забезпечує зрозумілість логіки, наочність виконання та зручність у підтримці та розширенні програмного забезпечення.

# 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ

## 4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Уся система емулює процес трансляції адрес, побудована навколо кількох логічно відокремлених, але тісно взаємопов'язаних компонентів. Перший модуль відповідає за інтерактивний інтерфейс користувача, приймаючи на вхід сегментний селектор і зсув або файл із переліком адрес, відображаючи таблиці дескрипторів і списки введених значень. Йому підпорядкований модуль сегментації, який розшифровує селектор, звертається до внутрішньої колекції дескрипторів, перевіряє межі та обчислює плоску лінійну адресу. Далі лінійна адреса передається в модуль TLB, що спершу шукає відображення в кеші швидкого доступу й у разі промаху делегує завдання модулю сторінкового транслятора. Останній звертається до двох рівнів таблиць — Page Directory і Page Table — підсвічуючи відповідні записи, проводить перевірку присутності сторінок та за необхідності оновлює кеш TLB згідно з політикою найменш часто використовуваних записів (LRU). Нарешті, після отримання номера фізичного кадру модулі обчислюють остаточну фізичну адресу та передають її назад інтерфейсу для відображення користувачеві, разом із деталями про TLB-хіти або сторінкові звернення. Уся ця взаємодія відбувається в єдиному цикл керування, забезпечуючи цілісність та зрозумілість процесу трансляції адрес на кожному з етапів.

**Сегментація**

У момент, коли користувач вводить логічну адресу у вигляді пари "селектор:зсув", система перш за все розпізнає значення селектора та зсуву й приводить їх до внутрішнього формату. Селектор містить у собі номер дескриптора сегмента, бітову ознаку таблиці індексів (TI) та рівень привілеїв (RPL). Перетворивши 16-бітовий селектор у три складові, емулятор звертається

до списку дескрипторів сегментів і за індексом знаходить запис із інформацією про базову адресу сегмента (Base) та максимально допустимий зсув (Limit). Одночасно перевіряється, чи не перевищує введений зсув розмір сегмента: якщо Offset перевищує Limit, видається помилка доступу до пам'яті. Якщо ж перевірка успішна, інтерпретатор просто додає значення зсуву до базової адреси дескриптора, у результаті отримуючи однорідну лінійну адресу. Таким чином, на виході цього етапу система отримує безпосередню адресу в лінійній (віртуальній) пам'яті, готову до подальшого кроку сторінкової

Реалізовано послідовний механізм перетворення кожного вхідного запису виду «Selector:Offset» у плоску лінійну адресу. Насамперед система зчитує до 256 рядків з текстового файлу формату 0xSSSS:0xOOOO і перетворює їх у числові значення типу ushort. Цей крок забезпечується фрагментом коду:

```
        var parts = line.Trim().Split(':');
        if (!ushort.TryParse(parts[0].Replace("0x", ""), NumberStyles.HexNumber,
null, out var sel)) continue;
        if (!ushort.TryParse(parts[1].Replace("0x", ""), NumberStyles.HexNumber,
null, out var off)) continue;
```

Далі з отриманого «сирого» селектора sel виділяються три поля: індекс дескриптора сегмента (idx), індикатор таблиці (TI) та рівень привілеїв (RPL):

```
        int idx = sel >> 3;
        int ti  = (sel >> 2) & 1;
        int rpl = sel & 0b11;
```

Для кожного idx у конструкторі вікна LogicToLinear було попередньо створено 256 дескрипторів з базовими адресами, кратними 1 МБ, і фіксованим лімітом 0xFFFF:

```
        for (int i = 0; i < 256; i++)
            Descriptors.Add(new SegmentDescriptorView(i, (uint)i * 0x0010_0000,
0xFFFF));
```

Зі списку дескрипторів знаходиться той, індекс якого співпадає з idx. Витягнуті з дескриптора поля BaseAddress і Limit перевіряються так, щоб Offset не перевищував допустимий ліміт:

```
        var desc = view.ToDescriptor();
        if (off > desc.Limit) continue;
```

Після успішної верифікації лімітів формується лінійна адреса як сума базової адреси сегмента та зсуву:

```
uint linear = desc.BaseAddress + off;
```

Кожний етап обчислень — від «сирого» селектора до остаточної лінійної адреси — відображається у відповідних колекціях, прив’язаних до трьох DataGrid: перший показує поля селектора (SelectorRaw, Index, TI, RPL), другий — OffsetRaw і базову адресу (BaseAddress), третій — уже обчислену LinearAddress.

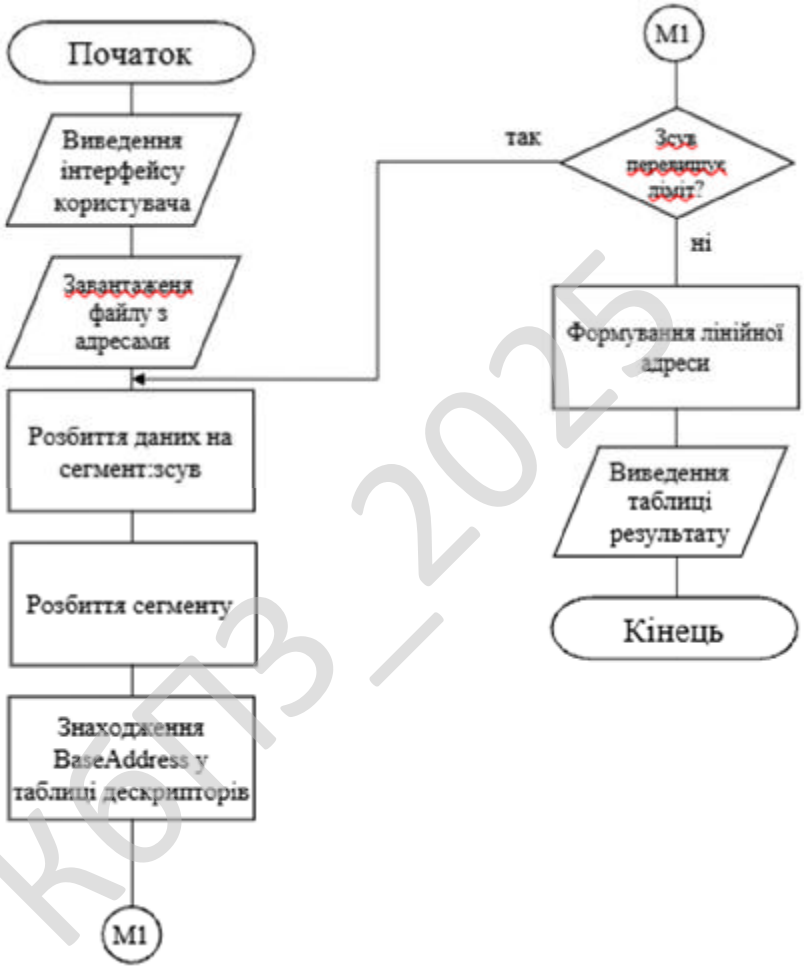


Рисунок 4.1 – Блок схема роботи сегментації

**Сторінкова трансляція**

Після отримання лінійної адреси система визначає три складові: індекс запису в Page Directory, індекс запису в Page Table та зміщення всередині сторінки. Першою здійснюється спроба швидкого пошуку відповідного відображення «номер сторінки → фізичний кадр» у буфері трансляції (TLB) з політикою видалення найменш недавно використаних записів. Якщо запис знайдено (TLB-hit), значення кадру миттєво повертається для подальшого

обчислення фізичної адреси. У разі промаху (TLB-miss) модуль трансляції звертається до Page Directory, вичитує запис PDE, звідки отримує базову адресу таблиці сторінок, а потім, використовуючи індекс сторінкової таблиці, знаходить відповідний запис PTE та перевіряє біти присутності. Після успішного зчитування адреси кадру він додається до TLB із оновленням часової мітки, що гарантує ефективну роботу кешу у наступних зверненнях. Нарешті для завершального кроку змінна кадру сумується зі зміщенням всередині сторінки, утворюючи кінцеву фізичну адресу, яка передається далі для моделювання звернення до оперативної пам'яті.

У реалізації емулятора сторінкова трансляція лінійної адреси розбивається на три основні етапи: виділення індексів у Page Directory та Page Table, пошук у кеші TLB з політикою LRU, та, у разі промаху в кеші, звернення до відповідних записів сторінкових таблиць. В кінці виконується обчислення фізичної адреси як суми базової адреси кадру та внутрішнього зсуву (рисунок 4.2).

**Розбиття лінійної адреси**

На початку з кожної лінійної адреси Linear виділяються три поля:

```
        uint dirIndex = (Linear >> 22) & 0x3FF;   // старші 10 біт → індекс Page
Directory
        uint tblIndex = (Linear >> 12) & 0x3FF;   // середні 10 біт → індекс Page
Table
        uint offset   = Linear         & 0xFFF;   // молодші 12 біт → байтовий
зсув усередині сторінки
```

Цей механізм повторено у методі ShowAddressInfo та в обробнику CalculatePhysicalAddress_Click, що демонструє, як кожну адресу з колекції _linearAddresses ми розбираємо й виводимо у таблицю DirTableOffsetGrid.

**Пошук у TLB з політикою LRU**

Після отримання номера сторінки (pageNumber = Linear >> 12), алгоритм намагається знайти відповідний запис у списку _tlb. Сам TLB реалізований як List<TLBEntry>, кожен елемент якого містить поля:

```
        public class TLBEntry
        {
            public uint   PageNumber;    // номер сторінки
            public uint   FrameAddress;  // базова адреса кадру
            public DateTime LastUsed;    // мітка часу останнього звернення
        }
```

У коді пошук виглядає так:

```
var tlbEntry = _tlb.FirstOrDefault(e => e.PageNumber == pageNumber);
if (tlbEntry != null)
{
    // TLB Hit
    tlbEntry.LastUsed = DateTime.Now;
    frameAddress = tlbEntry.FrameAddress;
}
```

Якщо запис знайдено, одразу оновлюється його час використання (LastUsed) і виконується прямий перехід до обчислення фізичної адреси.

**Обробка TLB-miss та звернення до сторінкових таблиць**

У разі, коли запис у TLB відсутній, виконується підсвічування відповідних рядків у PageDirectoryGrid і PageTableGrid, а потім отримуємо адресу кадру зі сторінкової таблиці:

```
_pageDirectory[(int)dirIndex].IsHighlighted = true;
_pageTable    [(int)tblIndex].IsHighlighted = true;
frameAddress = _pageTable[(int)tblIndex].FrameAddress;
```

Після цього виконуємо політику заміщення LRU: якщо кеш TLB уже заповнений (кількість записів ≥ TLB_CAPACITY), видаляємо найстаріший запис:

```
if (_tlb.Count >= TLB_CAPACITY)
{
    var oldest = _tlb.OrderBy(e => e.LastUsed).First();
    _tlb.Remove(oldest);
}
```

Потім додаємо новий запис із сучасною міткою часу:

```
_tlb.Add(new TLBEntry {
    PageNumber   = pageNumber,
    FrameAddress = frameAddress,
    LastUsed     = DateTime.Now
});
```

**Обчислення фізичної адреси**

Після отримання frameAddress фізична адреса формується додаванням внутрішнього зсуву:

```
uint physicalAddress = frameAddress + offset;
```

Результат записується у список results типу PhysicalAddressResult, який потім виводиться в PhysicalAddressesGrid:

```
results.Add(new PhysicalAddressResult {
    Linear   = entry.Linear,
    Physical = physicalAddress,
    Status   = (tlbEntry != null) ? "TLB Hit" : "TLB Miss"
});
PhysicalAddressesGrid.ItemsSource = results;
```

Таким чином, реалізація суттєво відповідає апаратному прототипу сторінкової трансляції з кешем TLB і дає наочне уявлення про кожну стадію перетворення адреси.

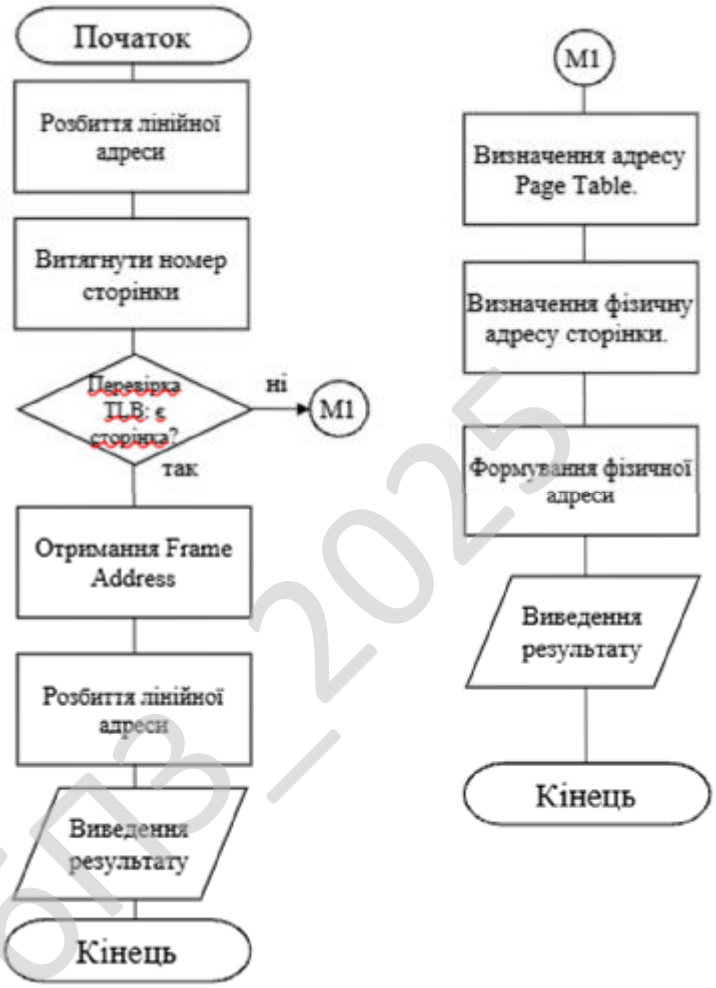




Рисунок 4.2– Блок схема роботи сторінкової трасляції

### 4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення реалізоване у вигляді виконуваного файлу з розширенням .exe для операційної системи Windows, що передбачає необхідність захисту його структури та цілісності. Хоча система має навчальне призначення, все ж важливо забезпечити її базову безпеку для уникнення несанкціонованих змін, збоїв у роботі та неправомірного використання.

Передусім захист починається з компіляції застосунку у формат, який ускладнює модифікацію — вихідний код перетворюється на машинний, і змінити його без реверс-інжинірингу практично неможливо. Однак навіть у такому вигляді програму можуть намагатися змінити або декомпілювати. Тому доцільно застосовувати додаткові заходи, такі як обфускація коду, яка ускладнює розуміння структури програми сторонніми особами. Крім того, забезпечується перевірка цілісності програми під час запуску, що дозволяє виявити сторонні втручання у виконуваний файл.

Захист функціональності всередині програми реалізується через контроль доступу до різних етапів емуляції. Інтерфейс побудований таким чином, що користувач не може пропустити етапи або змінити порядок виконання логіки, що унеможливлює некоректне використання програми. Завдяки цьому підтримується стабільність роботи програмного забезпечення навіть у випадках помилкових дій користувача.

Щоб уникнути запуску програми у модифікованому середовищі, наприклад, під керуванням відладчика або у віртуальному оточенні, передбачені механізми перевірки оточення, в якому виконується застосунок. Це дозволяє виявити потенційні спроби злому або несанкціонованого аналізу. Програма також оснащена системою обробки виняткових ситуацій, що забезпечує стійкість до помилок та аварійне збереження даних при критичних збоях.

У перспективі розробка може бути доповнена механізмами ліцензування, що дозволить регулювати доступ до неї відповідно до ролі користувача або обмежувати термін її використання. Для зберігання проміжних даних і конфігурацій використовується локальна пам’ять, до якої доступ реалізується з дотриманням правил безпеки операційної системи, що унеможливлює несанкціоноване редагування або видалення важливих налаштувань.

Таким чином, розроблене програмне забезпечення має не лише навчальну, а й захищену структуру, що дозволяє використовувати його в освітніх середовищах із дотриманням принципів безпечного програмування.

# 5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

**Вимоги до апаратного та програмного середовища**

Для забезпечення коректної та стабільної роботи емулятора в навчальному чи лабораторному середовищі рекомендується дотримуватися наступних мінімальних та рекомендованих характеристик:

Таблиця 5.1 – Рекомендовані характеристики

| Компонент | Мінімальні вимоги | Рекомендовано для інтенсивного навантаження |
|---|---|---|
| Процесор (CPU) | Двоядерний x86_64, 2 ГГц | Чотириядерний або вище, 3 ГГц і вище |
| Оперативна пам’ять | 2 ГБ | 8 ГБ або більше |
| Дисковий простір | 200 МБ для інсталяції ПЗ та 100 МБ для файлу образу | 500 МБ з урахуванням логів і тимчасових файлів |
| ОС та середовище | Windows 10/11 (20H2+) із встановленим .NET 6 Runtime | Windows 10/11 або Linux (з .NET 6 Core) |
| Права користувача | Звичайні права для читання/запису в %AppData% та програмні каталоги | Адміністративні права не обов’язкові |



**Скріншоти реалізованої системи**

Для підтвердження функціональності та візуалізації роботи програмного забезпечення, розробленого в рамках даної дипломної роботи, нижче наведено скріншоти основних вікон користувацького інтерфейсу. Програмне забезпечення створене з використанням технології WPF (Windows Presentation Foundation), що забезпечує зручну, інтуїтивно зрозумілу та естетично привабливу взаємодію з користувачем.

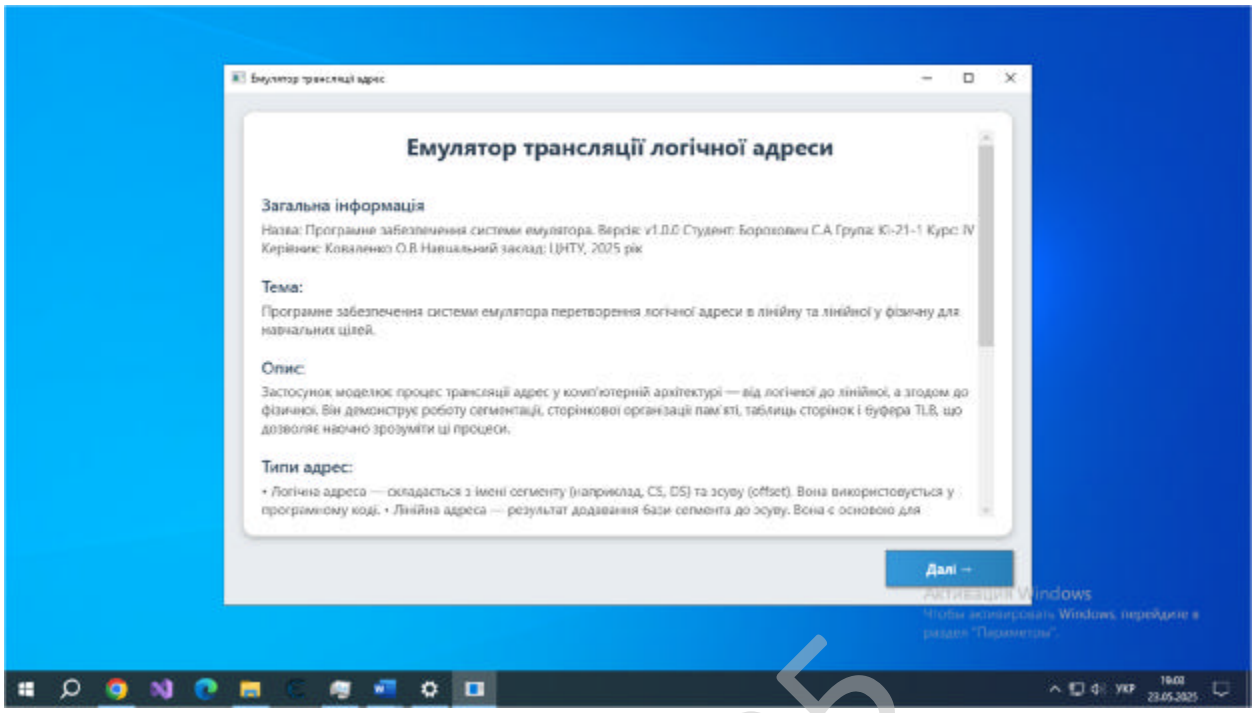


Рисунок 5.1 – Головне вікно системи

На рисунку 5.1 представлено головне вікно програми, яке містить короткий опис принципів адресації: логічної, лінійної та фізичної. Користувач має можливість ознайомитися з теоретичними відомостями, перш ніж перейти до практичної частини моделювання. Нижня частина вікна містить кнопку переходу до наступного етапу – трансляції логічної адреси в лінійну.

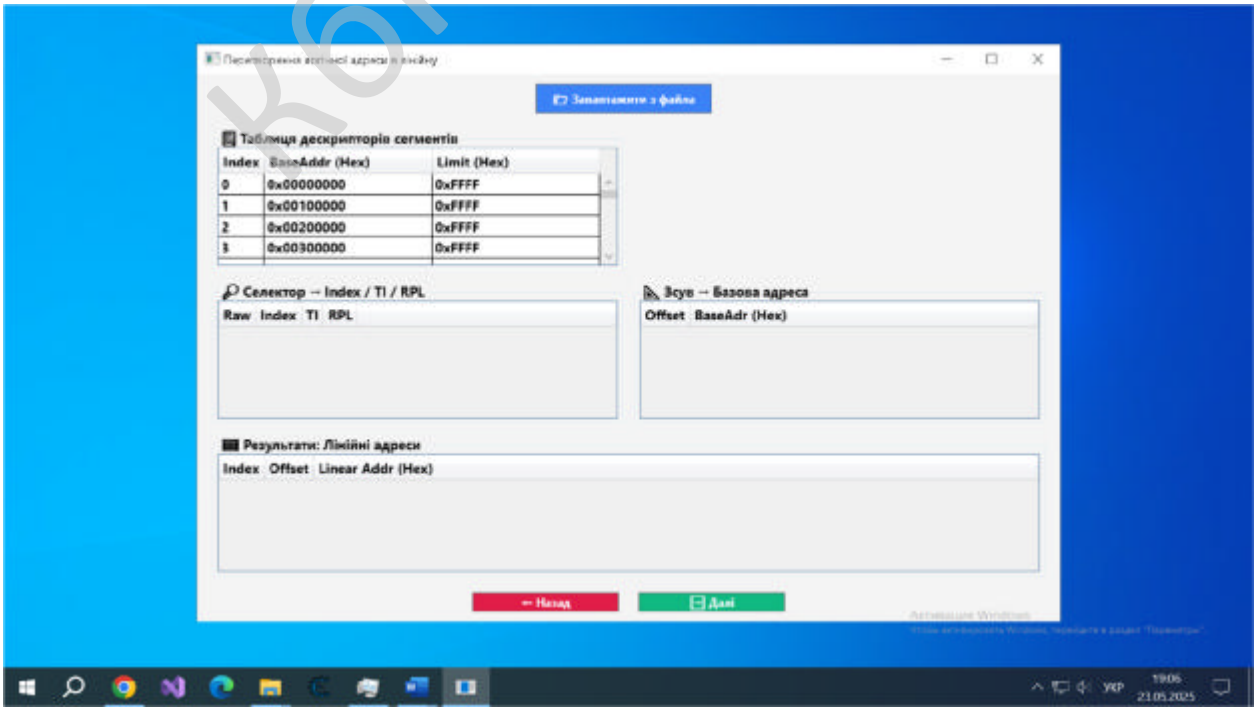


Рисунок 5.2 – Вікно трансляції логічної адреси в лінійну

На рисунку 5.2 зображено основне вікно модуля трансляції логічної адреси в лінійну, виконане у рамках WPF-інтерфейсу. У верхній частині розташована панель управління, що складається з кнопки завантаження вхідного файлу та поля відображення шляху до обраного файлу.

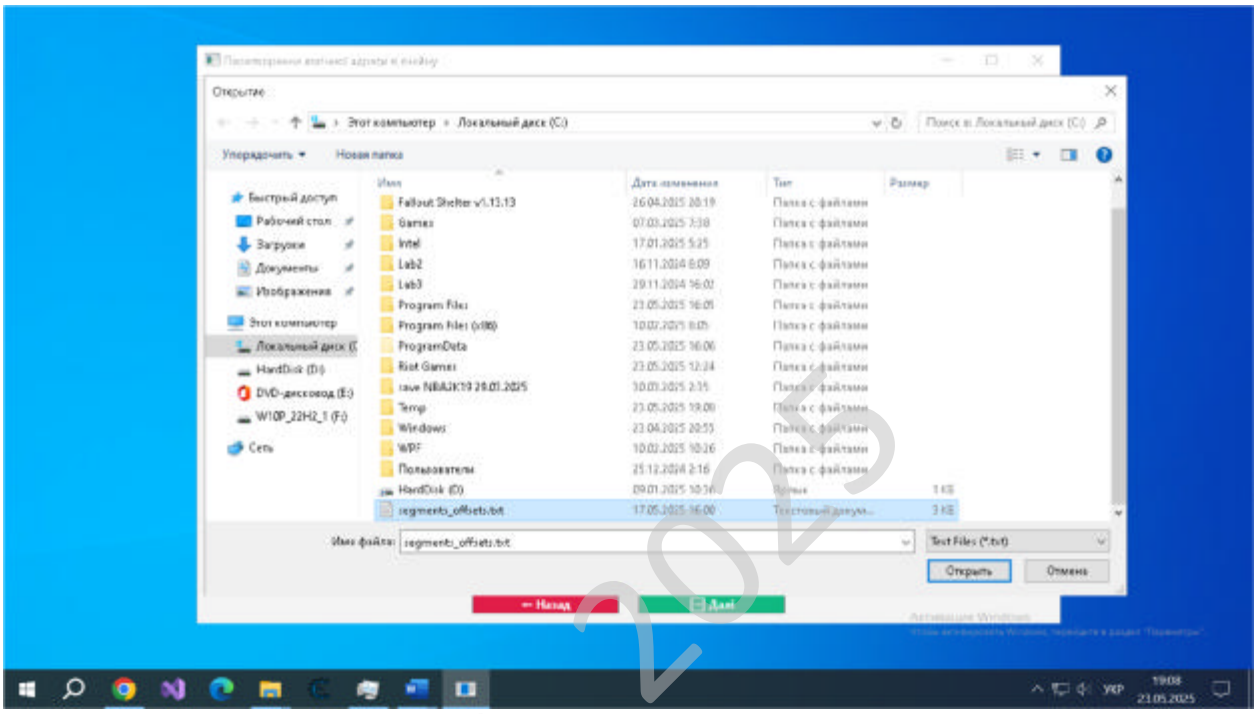

Рисунок 5.3 – Стандартний діалог вибору файлу

При натисканні на кнопку «Відкрити», рисунок 5.3, викликається стандартний діалог вибору файлу — очікується документ у форматі TXT, у якому міститься масив записів про дескриптори сегментів, селектори та зсуви.

Нижче панелі управління розташовані чотири групові блоки, об’єднані в єдиний робочий простір. Перший блок містить DataGrid із таблицею дескрипторів сегментів: кожен рядок демонструє індекс дескриптора, його лінійну основу (Base) та межу (Limit). Завантажені дані підсвічуються різними кольорами в залежності від коректності введених значень — зелений для дійсних дескрипторів, червоний для записів, у яких Base або Limit виходять за допустимі діапазони.

Другий груповий блок відведений під таблицю селекторів. Тут колонки розбиті на індекс дескриптора (Index), біт вибору таблиці дескрипторів (TI) та рівень привілеїв (RPL). У цій таблиці реалізовано вбудовану перевірку формату

селекторів: значення TI і RPL автоматично перетворюються з бітового поля, а некоректні комбінації позначаються жовтим застереженням.

Третій блок демонструє проміжні обчислення лінійної адреси: для кожної пари «дескриптор + селектор» відображається обраний Base, введений Offset та результат арифметичного складання. DataGrid автоматично сортує записи за зменшенням значення лінійної адреси, що полегшує аналіз найістотніших трансляцій. У разі спроби обчислити адресу з Offset, що перевищує межу сегмента, у відповідному рядку з'являється сповіщення про помилку захисту пам'яті.

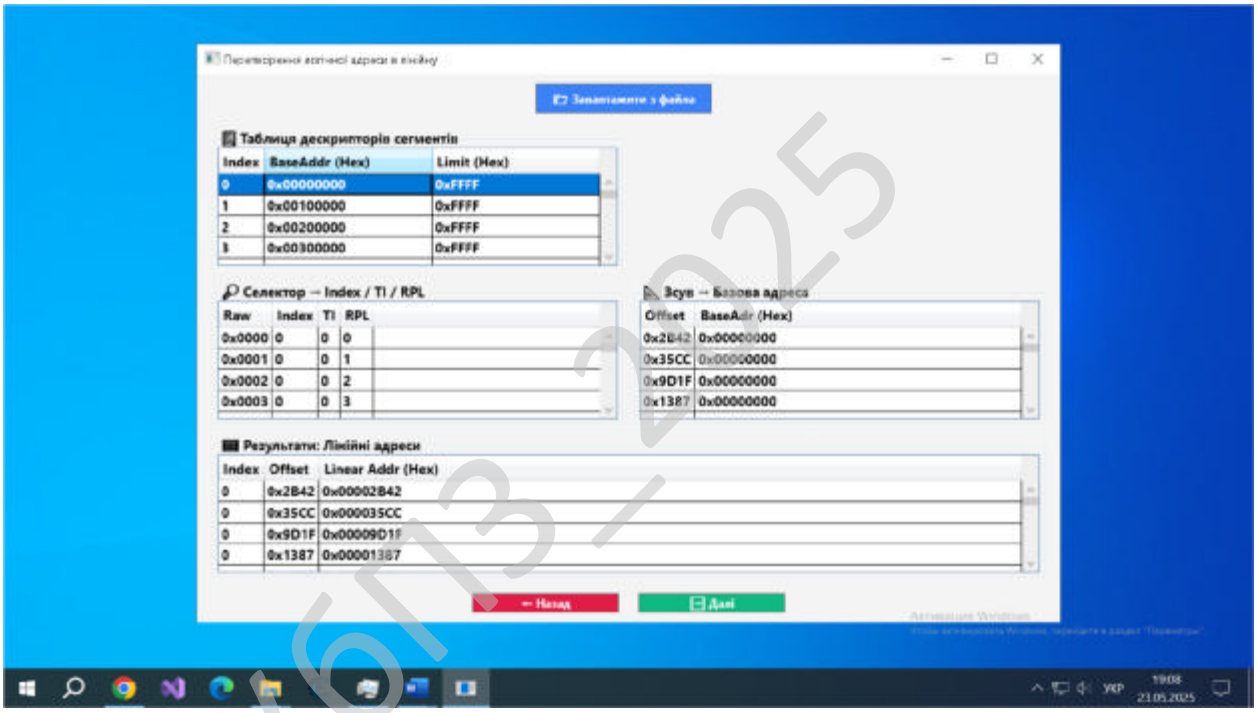


Рисунок 5.4 – Відображення готових лінійних адрес

У четвертому блокові міститься підсумкова таблиця, рисунок 5. 4, яка відображає готові лінійні адреси, готові до подальшої сторінкової трансляції. Кожний елемент цієї таблиці містить колонки «Логічна адреса», «Лінійна адреса» та «Статус» (успішно або із зазначенням типу помилки). Під таблицею розташована кнопка «Обчислити лінійні адреси», яка виконує одночасну перевірку всіх даних і оновлює кольорову індикацію стану рядків.

Управління інтерфейсом реалізовано через прив'язки команд WPF та використання шаблонів стилів для DataGrid: це гарантує єдиний вигляд усіх таблиць і дозволяє невеликою кількістю рядків коду підтримувати консистентність

оформлення. Завдяки групуванню у блоки та динамічній перевірці введених даних, користувач отримує наочний та інтуїтивно зрозумілий засіб для роботи з механізмом перетворення логічної адреси в лінійну.

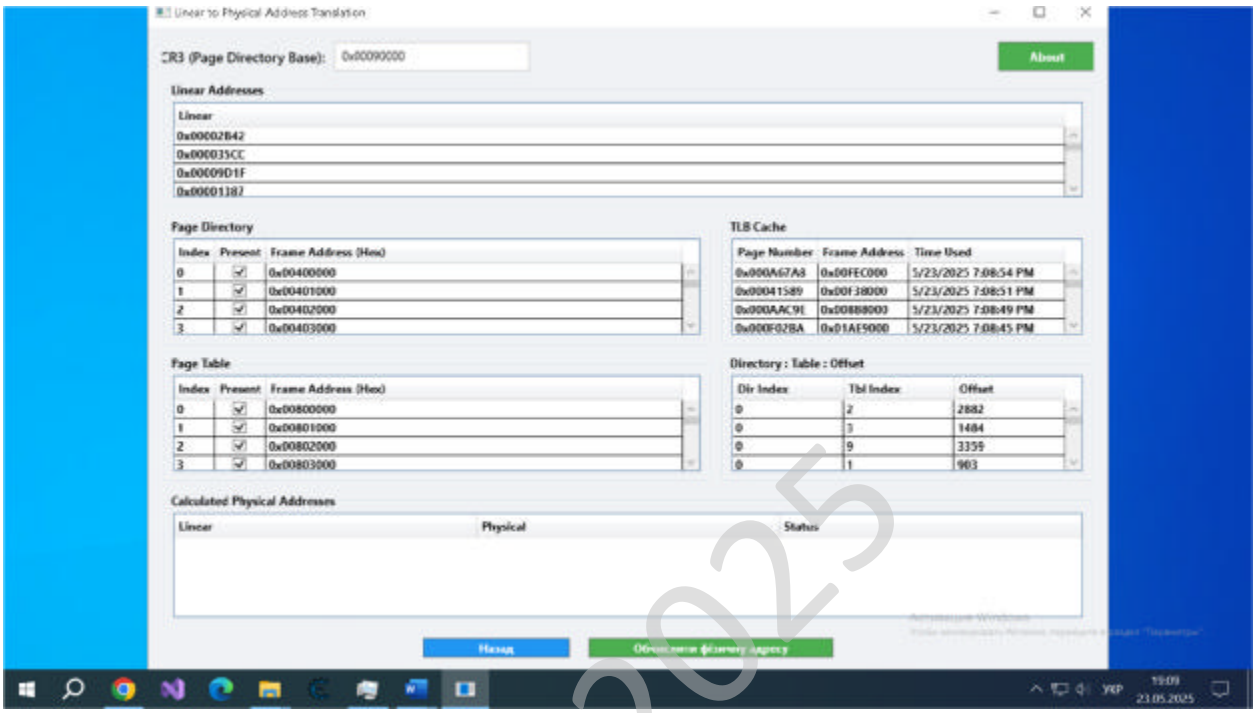


Рисунок 5.5 – Вікно трансляції лінійної адреси у фізичну

У центральній частині вікна трансляції лінійної адреси у фізичну, яке зображено на рисунку 5.5, розміщено три основні області роботи з пам’яттю — каталоги й таблиці сторінок, буфер трансляцій (TLB) та область імітації Page Fault. Ліва область інтерфейсу відведена під DataGrid каталогу сторінок (Page Directory): у ній ідентифікатор кожного запису (Directory Index) автоматично обчислюється з верхніх 10 бітів лінійної адреси, після чого відображається біт наявності (Present) і базова адреса таблиці сторінок (Frame Address). Нижче, у другому DataGrid, реалізовано відображення записів таблиці сторінок (Page Table), куди потрапляє середні 10 біт лінійної адреси; кожен запис показує біт доступу, статус завантаження у фізичну пам’ять та адресу фізичного фрейму. Відразу поруч розташована область TLB, де у вигляді асоціативного кешу представлено останні зіставлення «номер сторінки → фізичний фрейм» із виводом часу останнього звернення.

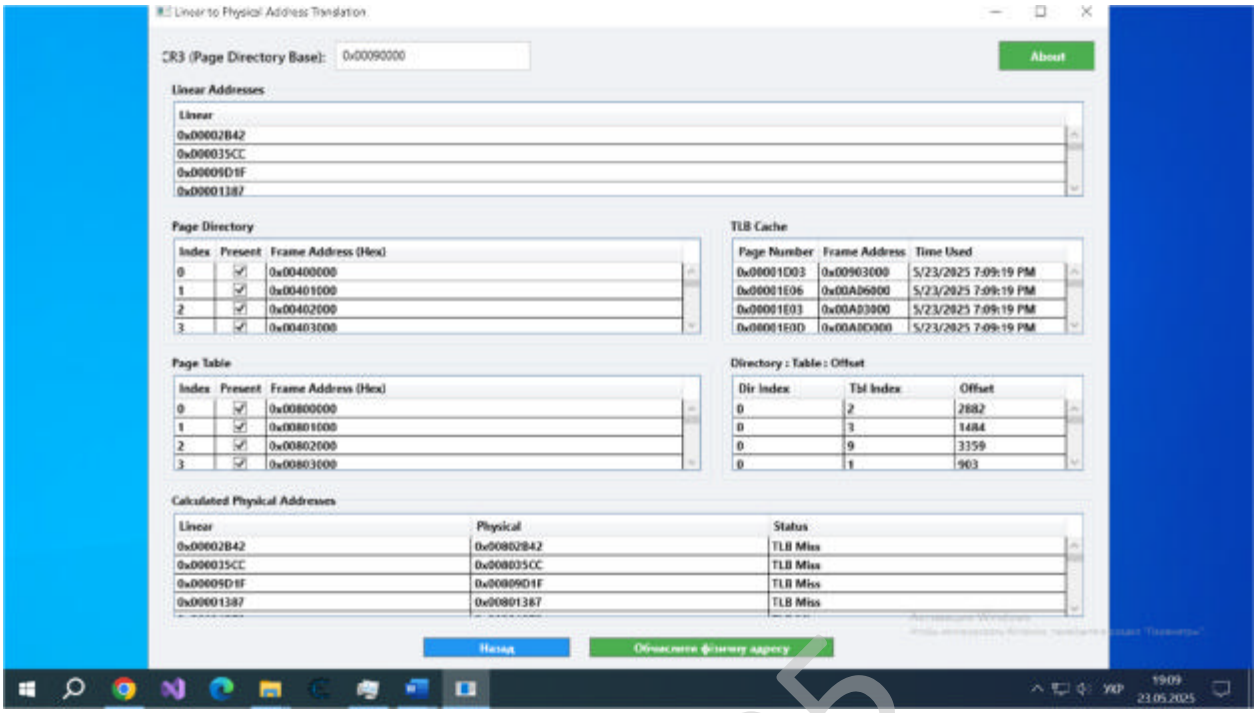


Рисунок 5.6 – Відображення готових фізичних адрес

При натисканні «Обчислити фізичну адресу» програма спочатку перевіряє TLB на наявність відповідного запису; у випадку успіху нова фізична адреса формується без звернення до таблиць сторінок, у протилежному випадку послідовно звертається до каталогу й самої таблиці сторінок, рисунок 5.6 ,а отримане зіставлення негайно записується в TLB. Якщо ж обидва звернення виявляють біт Present у стані «0», внизу інтерфейсу виникає діалогове вікно імітації Page Fault із можливістю «завантажити» відсутню сторінку з зовнішнього сховища; після підтвердження користувач побачить поповнення таблиці сторінок і, за необхідності, оновлення запису в TLB. Завдяки інтерактивній підсвітці рядків із різними статусами (TLB hit/miss, Page Fault) та детальним повідомленням про кожен крок, користувач може наочно прослідкувати, як операційна система обробляє звернення до пам’яті, починаючи з розщеплення лінійної адреси на індекси та завершуючи формуванням остаточної фізичної адреси..

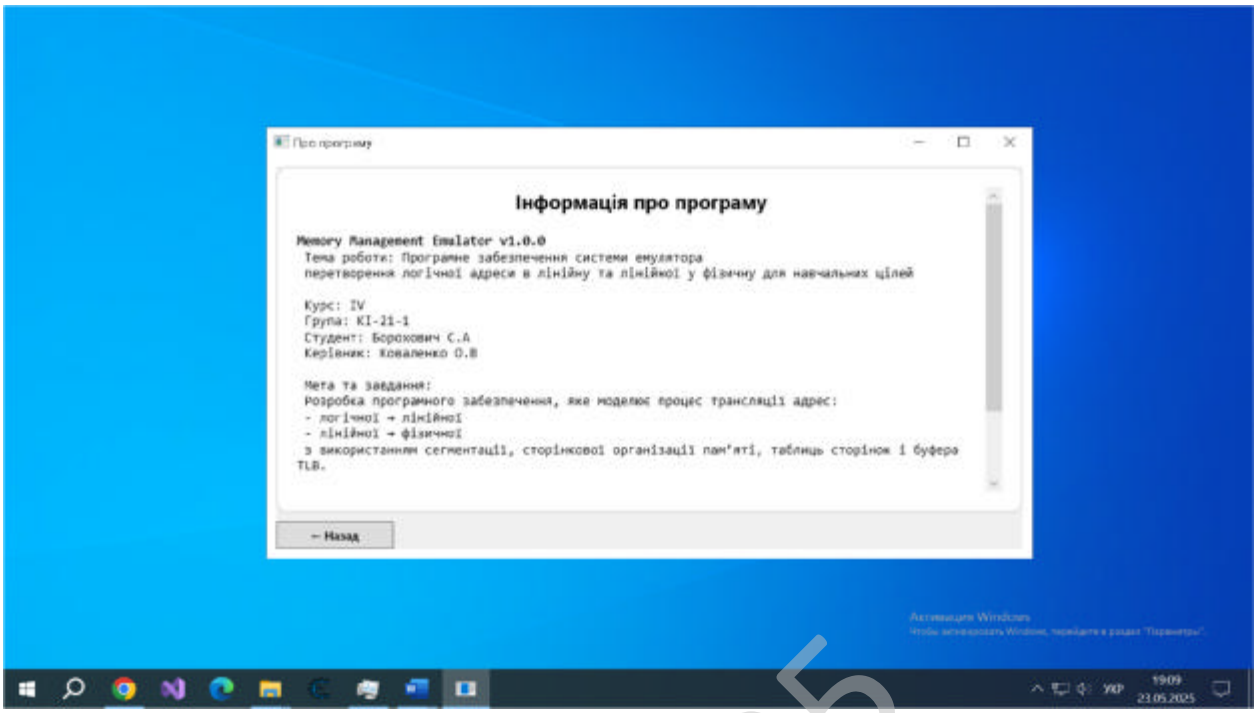


Рисунок 5.7 – Вікно “Про програму”

На рисунку 5.7 показано вікно "Про програму", яке містить загальну інформацію про розроблене програмне забезпечення. Це вікно виконує ознайомлювальну функцію для користувача, дозволяючи отримати базові відомості про призначення програми, її можливості та функціональність.

Вікно містить такі елементи:

a) Назву системи, що відображається у верхній частині;

b) Короткий опис функціоналу: моделювання трансляції логічної адреси у лінійну, а лінійної — у фізичну;

c) Перелік основних компонентів: підтримка таблиць дескрипторів, селекторів, TLB, Page Directory та Page Table;

d) Інформацію про розробника — ПІБ, навчальний заклад, факультет, кафедра;

e) Версію програмного забезпечення**;**

f) Кнопку переходу до модулів трансляції, яка дозволяє одразу розпочати симуляцію.

Інтерфейс оформлено у сучасному стилі з використанням м’якої колірної гами та іконок. Завдяки цьому вікно виглядає зручно для користувача.

# 6 ОСНОВНІ ВИСНОВКИ

У процесі розробки та тестування програмного забезпечення системи-емулятора ми змогли створити доволі гнучкий і наочний інструмент для демонстрації принципів адресної трансляції в сучасних комп'ютерних системах. Створений емултор успішно моделює три ключові етапи: від перетворення логічної адреси у лінійну шляхом додавання бази сегмента до офсету, через поділ лінійної адреси на індекси в Page Directory і Page Table та обчислення фізичної адреси з урахуванням внутрішнього зсуву сторінки, до кешування цих операцій у Translation Lookaside Buffer з політикою Least Recently Used.

Проведені експерименти з набором адрес підтвердили коректність реалізованих алгоритмів: для різноманітних комбінацій значень селектора та офсету емултор завжди обчислює очікувані лінійні й фізичні адреси, а модель TLB показує осмислену поведінку при влучаннях і промахах, що імітує роботу реального буфера швидкого доступу. Окрім того, обробка ситуацій Page Fault працює безвідмовно: при зверненні до відсутньої сторінки емултор коректно завантажує дані з backing store, оновлює таблиці сторінок і TLB, а також відображає цей крок користувачу.

Використання C# та WPF для реалізації інтерфейсу дозволило отримати приємний та інтуїтивно зрозумілий графічний інструмент, який легко адаптувати під різні навчальні сценарії та експериментальні завдання. Завдяки модульній архітектурі коду можна без значних зусиль додати нові режими адресації, розширити обсяг статистики або інтегрувати емултор із зовнішніми системами аналізу. Бібліотеки .NET надали надійну основу для роботи з великими наборами даних і гарантували стабільність під час демонстраційних занять.

Запропоноване рішення вже знайшло практичне застосування в рамках лабораторних занять з операційних систем та архітектури ЕОМ, де викладачі відзначили його здатність «оживити» абстрактні теоретичні концепції. Студенти, у свою чергу, отримали ефективний інструмент для самостійного вивчення

структури таблиць сторінок і поведінки TLB у різних сценаріях звернень до пам’яті.

Перспективним напрямом подальшого розвитку є розширення емултора підтримкою багаторівневого сторінкового відображення, деталізованим моделюванням сегментних прав доступу та впровадженням веб–версії, що дозволить інтегрувати його в дистанційні освітні платформи. Таким чином, розроблений емултор не тільки підтвердив свою ефективність як навчальний засіб, але й заклав основу для подальших досліджень і практичних впроваджень у галузі комп’ютерної інженерії.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Таненбаум А. С., Остін Т. С. Архітектура комп’ютера. — К. : Видавництво «Діалектика», 2019. — 784 с.

2. Сільвестрова Т. О., Жукова Н. В. Архітектура комп’ютерів та організація ЕОМ : навч. посіб. — Харків : ХНУРЕ, 2020. — 156 с.

3. Столярчук О. М., Куцуба Б. С. Системне програмування : навч. посіб. — Львів : Львівська політехніка, 2018. — 210 с.

4. Patterson D. A., Hennessy J. L. Комп’ютерна організація та проектування : інтерфейс апаратного та програмного забезпечення. — Київ : Наука і техніка, 2021. — 680 с.

5. Панкратов К. Є. Операційні системи : навч. посіб. — Дніпро : ДНУ, 2017. — 275 с.

6. Дорошенко С. В. Основи побудови операційних систем : навч. посіб. — Харків : ХНУРЕ, 2019. — 162 с.

7. Боярчук І. О., Кондратенко Ю. М. Комп’ютерна інженерія : основи архітектури мікропроцесорів. — Київ : КНЕУ, 2020. — 304 с.

8. MIPS64 Architecture for Programmers Volume II: The MIPS64 Instruction Set — Imagination Technologies, 2016. — 432 p.

9. Intel 64 and IA-32 Architectures Software Developer’s Manual. Volume 3: System Programming Guide. — Intel Corporation, 2021. — 1815 p.

10. EduMIPS64 Simulator [Електронний ресурс]. — Режим доступу: https://www.edumips.org/

11. QEMU Documentation [Електронний ресурс]. — Режим доступу: https://wiki.qemu.org/Manual

12. Bochs x86 Emulator [Електронний ресурс]. — Режим доступу: https://bochs.sourceforge.io/

13. Кудрявцев О. М., Третяк І. О. Архітектура комп’ютерів : підручник. — Київ : Каравела, 2021. — 328 с.

14. Шестопалов Є. А. Операційні системи. Віртуальна пам'ять : навч. посіб. — Запоріжжя : ЗНТУ, 2019. — 143 с.

15. Бутенко В. С., Білоус А. А. Архітектура ЕОМ : навч. посіб. — Черкаси : ЧДТУ, 2018. — 170 с.

16. Stallings W. Operating Systems: Internals and Design Principles. — 9th ed. — Pearson, 2018. — 784 p.

17. Сидоренко А. Ю. Архітектура комп’ютерних систем : методичні вказівки. — Херсон : ХНТУ, 2020. — 96 с.

18. Лекції з віртуалізації пам’яті на платформі x86 [Електронний ресурс]. — Режим доступу: https://wiki.osdev.org/Paging

19. Сегментація та сторінкова організація пам’яті [Електронний ресурс]. — Режим доступу: https://en.wikibooks.org/wiki/X86_Assembly/Protected_Mode

20. Дьяків О. В. Системне програмування : підручник. — Київ : КНЕУ, 2021. — 248 с.

21. Моделі пам’яті в операційних системах : конспект лекцій / За ред. І. М. Чередніченка. — Одеса : ОНАХТ, 2017. — 94 с.

22. Гайдук В. Є. Операційні системи. Теорія та практика : навч. посіб. — Львів : Видавництво Львівської політехніки, 2020. — 284 с.

23. Силін О. Є., Іванов Ю. О. Комп’ютерна архітектура та організація : навч. посіб. — Київ : КНУБА, 2019. — 256 с.

24. Морозов В. П. Організація та архітектура ЕОМ : навч. посіб. — Харків : ХНАМГ, 2018. — 212 с.

25. Глушков В. М. Основи побудови ЕОМ. — Київ : Наукова думка, 2017. — 300 с.

26. Silberschatz A., Galvin P. B., Gagne G. Operating System Concepts. — 10th ed. — Wiley, 2018. — 976 p.

27. Карпов Ю. М. Архітектура комп’ютера. — СПб. : Питер, 2021. — 416 с.

28. Угрюмов С. В. Комп’ютерна логіка і архітектура. — Одеса : ОНПУ, 2020. — 198 с.

29. Брауде Д. Основи операційних систем. — Харків : ХНУРЕ, 2019. — 174 с.

30. Віртуалізація пам’яті: теорія та практика [Електронний ресурс]. — Режим доступу: https://developer.ibm.com/articles/pa-linuxvm

31. Архітектура IA-32 [Електронний ресурс]. — Режим доступу: https://en.wikibooks.org/wiki/X86_Assembly

32. Пам’ять комп’ютера: керування, організація, типи [Електронний ресурс]. — Режим доступу: https://www.cs.cmu.edu/afs/cs/academic/class/15213-f03/doc/memory.pdf

33. Білозор О. В. Принципи побудови комп’ютерів : навч. посіб. — Львів : Львівська політехніка, 2021. — 242 с.

34. Архітектура і організація комп’ютерних систем / За ред. Л. В. Чумаченка. — Київ : КНУ, 2018. — 290 с.

35. Денисюк І. І. Організація пам’яті в обчислювальних системах. — Івано-Франківськ : ІФНТУНГ, 2020. — 154 с.

36. Архітектура систем обробки інформації. — Харків : ХНАУ, 2017. — 188 с.

37. Конспект лекцій з архітектури комп’ютерів / Національний технічний університет України «КПІ». — Київ, 2019. — 110 с.

38. Віртуальна пам’ять: основи організації [Електронний ресурс]. — Режим доступу: https://www.tldp.org/LDP/tlk/mm/memory.html

39. Внутрішня будова процесорів Intel [Електронний ресурс]. — Режим доступу: https://www.intel.com/content/www/us/en/processors/processor-architecture.html

40. Організація пам’яті у Linux [Електронний ресурс]. — Режим доступу: https://www.kernel.org/doc/html/latest/vm

41. Структура таблиць сторінок у x86 [Електронний ресурс]. — Режим доступу: https://wiki.osdev.org/Page_Table

42. Чорненький І. М. Основи програмування системного рівня. — Харків : ХНУРЕ, 2018. — 203 с.

43. Юрченко В. А. Архітектура мікропроцесорних систем. — Київ : НАУ, 2020. — 185 с.

44. Архітектура програмно-апаратних засобів / За ред. П. І. Сікорського. — Київ : КНЕУ, 2019. — 264 с.

45. Маліновський О. А. Віртуалізація та емулювання систем : навч. посіб. — Вінниця : ВНТУ, 2021. — 134 с.

46. Архітектура та організація комп'ютерних систем. — Чернівці : ЧНУ, 2018. — 222 с.

47. Оперативна пам'ять: типи та характеристики [Електронний ресурс]. — Режим доступу: https://www.crucial.com/memory-types

48. Memory Management in Modern OS [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/memory-management-in-operating-system

49. Дубровін О. І. Архітектура обчислювальних систем : навч. посіб. — Херсон : ХДУ, 2020. — 164 с.

50. Системи управління пам'яттю / Навч. матеріали. — Київ : Університет «КРОК», 2022. — 112 с.

51. Розподіл і віртуалізація пам'яті [Електронний ресурс]. — Режим доступу: https://www.tutorialspoint.com/operating_system/os_memory_management.htm

52. Сучасні архітектури процесорів Intel і AMD [Електронний ресурс]. — Режим доступу: https://www.hardwaretimes.com/category/architecture

ДОДАТОК А
(обов’язковий)

**Технічне завдання**

**ЗМІСТ**

## 1 Найменування та область застосування

Це технічне завдання розроблено для створення програмного забезпечення, яке реалізує емуляцію процесу трансляції логічної адреси у фізичну з використанням механізмів сегментації, сторінкової організації пам’яті та кешування в буфері TLB. Розроблюване програмне забезпечення є частиною навчального комплексу і призначене для використання у вищих навчальних закладах при вивченні архітектури комп’ютерів та принципів організації пам’яті в сучасних мікропроцесорних системах.

## 2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 46-02 від 17.01.2025 р.).

## 3 Мета та призначення розробки

Мета та призначення розробки полягає у створенні програмного забезпечення для емуляції процесу трансляції логічної адреси у фізичну з використанням сегментації, сторінкової організації пам’яті та кешування в буфері TLB, що забезпечить наочне відображення алгоритмів керування пам’яттю в сучасних комп’ютерних системах. Розроблений програмний продукт має слугувати навчальним засобом для студентів, що вивчають принципи функціонування підсистеми управління пам’яттю процесора, і сприяти кращому засвоєнню теоретичного матеріалу за допомогою інтерактивної візуалізації процесів сегментації та сторінкової трансляції.

## 4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробки, які ведуться у Центральноукраїнському національному технічному університеті кафедрою кібербезпеки та програмного забезпечення і стосовні до теми бібліографічні джерела.

## 5 Технічні вимоги

### 5.1 Вміст проекту

**Складовими розробки є:**

– аналіз існуючих систем емуляції трансляції логічних адрес у фізичні з використанням сегментації, сторінкової організації пам’яті та буфера TLB;

– вибір і обґрунтування методики побудови програмного забезпечення на основі сучасних технологій програмування, зокрема з використанням C# та WPF;

– розробка структур даних, що відображають сегментні дескриптори, таблиці сторінок, TLB і фізичну пам’ять, а також механізмів їхньої взаємодії;

– створення інтерфейсу користувача для візуалізації процесу трансляції та відображення кроків трансформації адреси;

– реалізація програми, що імітує всі етапи трансляції адреси, з повідомленнями про виникнення TLB-hit, TLB-miss та Page Fault, а також розрахунок ефективності за допомогою часових характеристик у рамках додаткового завдання.

### 5.2 Показники призначення

Система повинна забезпечувати: – візуалізацію процесу трансляції логічної адреси у фізичну з використанням сегментації, сторінкової організації пам’яті та буфера TLB; – простий та інтуїтивно зрозумілий інтерфейс взаємодії з користувачем для демонстрації кожного етапу трансляції адрес; – коректність і цілісність даних, що моделюють таблиці сторінок, TLB та фізичну пам’ять.

### 5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення повинно забезпечувати можливість розширення логіки трансляції, додавання нових режимів моделювання, зміни конфігурації пам’яті, TLB, а також налаштування параметрів емуляції без необхідності суттєвої модифікації коду.

### 5.4 Вимоги до архітектури

Розроблюваний компонент повинен базуватись на об’єктно-орієнтованій архітектурі, реалізованій мовою програмування C# з використанням фреймворку WPF, що забезпечує поділ на логіку моделі, інтерфейс користувача та механізми взаємодії (MVVM-підхід).

### 5.5 Вимоги до надійності

Компонент повинен використати існуючі угоди по стандартним викликам процедур, функцій, засобів і форм, визначених технічною документацією на середовище розробки.

### 5.6 Умови експлуатації

Автоматизовані робочі місця користувачів системи повинні задовольняти наступним умовам експлуатації:

– температура повітря: 18-220 С;

– відносна вологість повітря при 200 С до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу і параметрів технічних засобів

Компонент повинен бути реалізований на ЕОМ типу IBM PC в операційному середовищі Windows 10 і орієнтований на сумісні з цією платформою зовнішні пристрої, мережне обладнання і прикладне програмне забезпечення.

### 5.8 Вимоги до інформаційної та програмної сумісності

Розроблюване програмне забезпечення повинно бути сумісним із сучасними операційними системами родини Windows, зокрема Windows 10 або новішими. Забезпечення сумісності гарантується за рахунок використання об'єктно-орієнтованого підходу в розробці та платформи .NET, а також застосування технології WPF (Windows Presentation Foundation).

#### 5.8.1 Обладнання

Персональний комп’ютер із процесором не нижче Intel® Core i3, оперативною пам’яттю не менше 4 ГБ, дисплеєм з роздільною здатністю щонайменше 1366×768 пікселів та підтримкою GPU, сумісного з Windows

### 5.8.2 Мова програмування

C# у середовищі розробки Microsoft Visual Studio з використанням .NET Framework або .NET Core.

### 5.8.3 Вхідні дані

Параметри логічної адреси, значення сегментних дескрипторів, вміст таблиць сторінок, стан буфера TLB.

**Вихідні дані**

Візуалізовані етапи трансляції адрес, повідомлення про TLB-hit, TLB-miss, Page Fault, а також результати трансляції у вигляді фізичної адреси.

## 6 Вимоги до програмної документації

Програмна продукція повинна бути представлена в вигляді опису структури даних, схем і описів алгоритмів, інструкції користувача, а також текстів вхідних модулів програмного забезпечення в відповідності з ЄСПД

## 7 Перелік документів, які необхідно розробити

– Структурна схема системи повна – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схеми алгоритму роботи програми – 2 аркуша.
–

## 8 Етапи розробки

На рівні проекту розробляються (терміни виконання етапів див. в "Завданні на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти"):

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок-схем алгоритмів роботи програмного забезпечення компоненту.

8.4 Побудова схем взаємодії структур даних.

8.5 Створення прототипу компоненту. Створення програмного продукту.

8.6 Відлагодження компоненту, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.



## 9 Порядок контролю і приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 24.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 2.06.2025 р.

ДОДАТОК Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

О.В. Коваленко

Програмне забезпечення системи емулятора перетворення логічної адреси в лінійну та лінійної у фізичну для навчальних цілей

Лістинг програми

Носій: DVD-диск

Код документу 12

Загальна кількість аркушів: 22

Літера: РП

Кропивницький - 2025 року

## MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace diplom
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {


        public MainWindow()
        {
            InitializeComponent();
        }

        private void OpenLogicToLinear_Click(object sender, RoutedEventArgs e)
        {
            var window = new LogicToLinear();
            window.Show(); // Відкрити нову форму

            this.Close(); // Закрити MainWindow, якщо потрібно
        }

    }
}
```



## MainWindow.xaml

```
<Window x:Class="diplom.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:diplom"
        mc:Ignorable="d"
        Title="Емулятор трансляції адрес" Height="600" Width="900"
        Background="#e8ecf0" WindowStartupLocation="CenterScreen">

    <Grid Margin="20">
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
```

```
        <!-- Панель з тінню -->
        <Border Grid.Row="0"
                Background="White"
                CornerRadius="12"
                Padding="20"
                BorderBrush="#D0D0D0"
                BorderThickness="1"
                Margin="0,0,0,15">
            <Border.Effect>
                <DropShadowEffect BlurRadius="10" ShadowDepth="3"
Direction="270" Color="#999999" Opacity="0.3"/>
            </Border.Effect>
            <ScrollViewer VerticalScrollBarVisibility="Auto">
                <StackPanel>
                    <!-- Заголовок -->
                    <TextBlock Text="Емулятор трансляції логічної адреси"
                               FontSize="26"
                               FontWeight="Bold"
                               Foreground="#2c3e50"
                               TextAlignment="Center"
                               Margin="0 0 0 25"/>

                    <!-- Розділ: Загальна інформація -->
                    <TextBlock Text="Загальна інформація" FontSize="18"
FontWeight="SemiBold" Margin="0 10 0 5" Foreground="#34495e"/>
                    <TextBlock FontSize="15" TextWrapping="Wrap"
Foreground="#555">
                        Назва: Програмне забезпечення системи емулятора.
                        Версія: v1.0.0
                        Студент: Борохович С.А
                        Група: КІ-21-1
                        Курс: IV
                        Керівник: Коваленко О.В
                        Навчальний заклад: ЦНТУ, 2025 рік
                    </TextBlock>

                    <!-- Тема -->
                    <TextBlock Text="Тема:" FontSize="18" FontWeight="SemiBold"
Margin="0 20 0 5" Foreground="#34495e"/>
                    <TextBlock FontSize="15" TextWrapping="Wrap"
Foreground="#555">
                        Програмне забезпечення системи емулятора перетворення
логічної адреси в лінійну
                        та лінійної у фізичну для навчальних цілей.
                    </TextBlock>

                    <!-- Опис -->
                    <TextBlock Text="Опис:" FontSize="18" FontWeight="SemiBold"
Margin="0 20 0 5" Foreground="#34495e"/>
                    <TextBlock FontSize="15" TextWrapping="Wrap"
Foreground="#555">
                        Застосунок моделює процес трансляції адрес у
комп’ютерній архітектурі — від логічної до лінійної,
                        а згодом до фізичної. Він демонструє роботу сегментації,
сторінкової організації пам’яті,
                        таблиць сторінок і буфера TLB, що дозволяє наочно
зрозуміти ці процеси.
                    </TextBlock>

                    <!-- Адреси -->
                    <TextBlock Text="Типи адрес:" FontSize="18"
FontWeight="SemiBold" Margin="0 20 0 5" Foreground="#34495e"/>
```

```
                    <TextBlock FontSize="15" TextWrapping="Wrap"
Foreground="#555">
                        • Логічна адреса — складається з імені сегменту
(наприклад, CS, DS) та зсуву (offset). Вона використовується у програмному коді.
                        • Лінійна адреса — результат додавання бази сегмента до
зсуву. Вона є основою для подальшої трансляції.
                        • Фізична адреса — справжня адреса в оперативній
пам’яті, отримана шляхом трансляції через таблиці сторінок.
                    </TextBlock>

                    <!-- Трансляція -->
                    <TextBlock Text="Процес трансляції адрес:" FontSize="18"
FontWeight="SemiBold" Margin="0 20 0 5" Foreground="#34495e"/>
                    <TextBlock FontSize="15" TextWrapping="Wrap"
Foreground="#555">
                        1. Логічна → Лінійна: сегментний дескриптор забезпечує
базову адресу сегменту. До неї додається зсув.
                        2. Лінійна → Фізична: старші біти адреси визначають
індекси в Page Directory та Page Table,
                        а молодші — зсув усередині сторінки. Завдяки таблицям
сторінок та TLB відбувається трансляція у фізичну адресу.
                    </TextBlock>

                    <!-- Приклад -->
                    <TextBlock Text="Приклад трансляції:" FontSize="18"
FontWeight="SemiBold" Margin="0 20 0 5" Foreground="#34495e"/>
                    <TextBlock FontSize="15" TextWrapping="Wrap"
Foreground="#555">
                        Лінійна адреса: 0x1234ABCD → Page Directory Index: 0x48,
Page Table Index: 0x34A, Offset: 0xBCD.
                        Фізична адреса: 0xA1B2C3D4 (в результаті трансляції
через таблиці сторінок).
                    </TextBlock>

                    <!-- Технології -->
                    <TextBlock Text="Використані технології:" FontSize="18"
FontWeight="SemiBold" Margin="0 20 0 5" Foreground="#34495e"/>
                    <TextBlock FontSize="15" TextWrapping="Wrap"
Foreground="#555">
                        • Сегментація пам’яті
                        • Буфер TLB (Translation Lookaside Buffer)
                        • Сторінкова організація пам’яті (Paging)
                        • Механізм Page Fault
                        • Таблиці Page Directory і Page Table
                    </TextBlock>
                </StackPanel>
            </ScrollViewer>
        </Border>

        <!-- Кнопка переходу -->
        <StackPanel Grid.Row="1" Orientation="Horizontal"
HorizontalAlignment="Right">
            <Button Content="Далі →"
                    Width="140"
                    Height="40"
                    Margin="0"
                    FontWeight="Bold"
                    Foreground="White"
                    FontSize="15"
                    Cursor="Hand"
                    Click="OpenLogicToLinear_Click">
                <Button.Background>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
```

```
                        <GradientStop Color="#3498db" Offset="0.0"/>
                        <GradientStop Color="#2980b9" Offset="1.0"/>
                    </LinearGradientBrush>
                </Button.Background>
                <Button.Effect>
                    <DropShadowEffect BlurRadius="5" ShadowDepth="2"
Opacity="0.4"/>
                </Button.Effect>
            </Button>
        </StackPanel>
    </Grid>
</Window>
```

## LogicToLinear.xaml

```
<Window x:Class="diplom.LogicToLinear"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:diplom"
        mc:Ignorable="d"
        Title="Перетворення логічної адреси в лінійну" Height="600" Width="900"
        Background="#F4F4F4"
        WindowStartupLocation="CenterScreen">

    <Grid Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>

        <!-- Верхня панель з кнопкою завантаження -->
        <DockPanel Margin="0 0 0 10">
            <Button Content="🗀 Завантажити з файла"
Click="OnLoadFromFile_Click" Width="180" Height="30" Margin="0,0,10,0"
                    Background="#3B82F6" Foreground="White" FontWeight="Bold" />
        </DockPanel>

        <!-- Основна сітка з групами -->
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="*"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <!-- Дескриптори -->
            <GroupBox Header="▧ Таблиця дескрипторів сегментів" Margin="5"
                      FontWeight="Bold" FontSize="14" Grid.Row="0"
Grid.Column="0">
                <DataGrid x:Name="AllDescriptorsGrid"
AutoGenerateColumns="False" IsReadOnly="True"
```

```
                          HeadersVisibility="Column" CanUserAddRows="False"
GridLinesVisibility="All">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Index" Binding="{Binding
Index}" Width="Auto"/>
                        <DataGridTextColumn Header="BaseAddr (Hex)"
Binding="{Binding BaseAddress, StringFormat=0x{0:X8}}" Width="*"/>
                        <DataGridTextColumn Header="Limit (Hex)"
Binding="{Binding Limit, StringFormat=0x{0:X4}}" Width="*"/>
                    </DataGrid.Columns>
                </DataGrid>
            </GroupBox>

            <!-- Селектори -->
            <GroupBox Header="🔎 Селектор → Index / TI / RPL" Margin="5"
                      FontWeight="Bold" FontSize="14" Grid.Row="1"
Grid.Column="0">
                <DataGrid x:Name="SelectorsGrid" AutoGenerateColumns="False"
IsReadOnly="True"
                          HeadersVisibility="Column" CanUserAddRows="False"
GridLinesVisibility="All">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Raw" Binding="{Binding
SelectorRaw}" Width="Auto"/>
                        <DataGridTextColumn Header="Index" Binding="{Binding
Index}" Width="Auto"/>
                        <DataGridTextColumn Header="TI" Binding="{Binding TI}"
Width="Auto"/>
                        <DataGridTextColumn Header="RPL" Binding="{Binding RPL}"
Width="Auto"/>
                    </DataGrid.Columns>
                </DataGrid>
            </GroupBox>

            <!-- Зсуви та бази -->
            <GroupBox Header="📐 Зсув → Базова адреса" Margin="5"
                      FontWeight="Bold" FontSize="14" Grid.Row="1"
Grid.Column="1">
                <DataGrid x:Name="BasesGrid" AutoGenerateColumns="False"
IsReadOnly="True"
                          HeadersVisibility="Column" CanUserAddRows="False"
GridLinesVisibility="All">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Offset" Binding="{Binding
OffsetRaw}" Width="Auto"/>
                        <DataGridTextColumn Header="BaseAdr (Hex)"
Binding="{Binding BaseAddress, StringFormat=0x{0:X8}}" Width="*"/>
                    </DataGrid.Columns>
                </DataGrid>
            </GroupBox>

            <!-- Лінійні адреси -->
            <GroupBox Header="  Результати: Лінійні адреси" Grid.Row="2"
Grid.ColumnSpan="2" Margin="5"
                      FontWeight="Bold" FontSize="14">
                <DataGrid x:Name="LinearGrid" AutoGenerateColumns="False"
IsReadOnly="True"
                          HeadersVisibility="Column" CanUserAddRows="False"
GridLinesVisibility="All">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Index" Binding="{Binding
Index}" Width="Auto"/>
```

```
                        <DataGridTextColumn Header="Offset" Binding="{Binding
OffsetRaw}" Width="Auto"/>
                        <DataGridTextColumn Header="Linear Addr (Hex)"
Binding="{Binding LinearAddress, StringFormat=0x{0:X8}}" Width="*"/>
                    </DataGrid.Columns>
                </DataGrid>
            </GroupBox>
        </Grid>

        <!-- Нижні кнопки -->
        <StackPanel Grid.Row="2" Orientation="Horizontal"
HorizontalAlignment="Center" Margin="0 10 0 0">
            <Button Content="← Назад" Width="150" Margin="10,0"
Click="OnBack_Click"
                    Background="#E11D48" Foreground="White" FontWeight="Bold"/>
            <Button Content="→ Далі" Width="150" Margin="10,0"
Click="Button_Click"
                    Background="#10B981" Foreground="White" FontWeight="Bold"/>
        </StackPanel>
    </Grid>
</Window>
```

## LogicToLinear.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using Microsoft.Win32;

namespace diplom
{
    public class InputAddressExtended
    {
        public string SelectorRaw { get; set; }   // оригінальний текст,
наприклад "0x0018"
        public int Index { get; set; }   // selector >> 3
        public int TI { get; set; }   // таблиця (Table Indicator): (selector >>
2) & 1
        public int RPL { get; set; }   // Requestor Privilege Level: selector &
0b11
        public string OffsetRaw { get; set; }   // "0x00FF"
        public uint BaseAddress { get; set; }   // взято з дескриптора
        public uint LinearAddress { get; set; }   // BaseAddress + Offset
    }
    /// <summary>
    /// Логика взаимодействия для LogicToLinear.xaml
    /// </summary>
    public partial class LogicToLinear : Window
    {
```

```
        public List<SegmentDescriptorView> Descriptors = new
List<SegmentDescriptorView>();
        public int DescriptorCount => Descriptors.Count;
        private ObservableCollection<InputAddressExtended> _inputs = new
ObservableCollection<InputAddressExtended>();

        private List<uint> _linearAddresses = new List<uint>();
        public LogicToLinear()
        {
            InitializeComponent();
            LoadDescriptors();
            // підв'язуємо гріди
            SelectorsGrid.ItemsSource = _inputs;
            BasesGrid.ItemsSource = _inputs;
            LinearGrid.ItemsSource = _inputs;
            AllDescriptorsGrid.ItemsSource = Descriptors;

        }
        public void LoadDescriptors()
        {
            // Генеруємо 256 дескрипторів
            const uint SEGMENT_SIZE = 0x0010_0000;   // 1 Мб = 0x0010_0000
            const uint DEFAULT_LIMIT = 0xFFFF;

            for (int idx = 0; idx < 256; idx++)
            {
                uint baseAddr = (uint)idx * SEGMENT_SIZE;
                Descriptors.Add(new SegmentDescriptorView(idx, baseAddr,
DEFAULT_LIMIT));
            }
        }

        private void OnLoadFromFile_Click(object sender, RoutedEventArgs e)
        {
            var dlg = new OpenFileDialog
            {
                Filter = "Text Files (*.txt)|*.txt"
            };
            if (dlg.ShowDialog() != true) return;

            var lines = File.ReadAllLines(dlg.FileName)
                            .Where(l => !string.IsNullOrWhiteSpace(l))
                            .Take(256)
                            .ToArray();

            _inputs.Clear();

            foreach (var line in lines)
            {
                var parts = line.Trim().Split(':');
                if (parts.Length != 2) continue;

                var selText = parts[0].Replace("0x", "").Trim();
                var offText = parts[1].Replace("0x", "").Trim();
                if (!ushort.TryParse(selText, NumberStyles.HexNumber, null, out
var sel)) continue;
                if (!ushort.TryParse(offText, NumberStyles.HexNumber, null, out
var off)) continue;

                // розбір селектора
                int idx = sel >> 3;
                int ti = (sel >> 2) & 1;
                int rpl = sel & 0b11;
```

```
                // знаходимо дескриптор
                var view = Descriptors.FirstOrDefault(d => d.Index == idx);
                if (view == null) continue;
                var desc = view.ToDescriptor();

                if (off > desc.Limit) continue; // перевірка межі

                uint baseAdr = desc.BaseAddress;
                uint linear = baseAdr + off;
                _linearAddresses.Add(linear);

                _inputs.Add(new InputAddressExtended
                {
                    SelectorRaw = "0x" + selText,
                    Index = idx,
                    TI = ti,
                    RPL = rpl,
                    OffsetRaw = "0x" + offText,
                    BaseAddress = baseAdr,
                    LinearAddress = linear
                });
            }
        }
        public void OpenLinearToPhysical()
        {
            var window = new LinearToPhysical(_linearAddresses);
            window.Show(); // Відкрити нову форму

            this.Close();
        }

        private void OnBack_Click(object sender, RoutedEventArgs e)
        {
            var window = new MainWindow();
            window.Show();
            this.Close();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
           OpenLinearToPhysical();
        }
    }
}
```



## LinearToPhysical.xaml

```
<Window x:Class="diplom.LinearToPhysical"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:diplom"
        mc:Ignorable="d"
        Title="Linear to Physical Address Translation" Height="700" Width="1000"
        Background="#f0f2f5"
        WindowStartupLocation="CenterScreen">

    <Grid Margin="15">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
```

```
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="3*"/>
            <ColumnDefinition Width="2*"/>
        </Grid.ColumnDefinitions>

        <!-- CR3 -->
        <StackPanel Orientation="Horizontal" Grid.ColumnSpan="3" Margin="0 0 0
10" HorizontalAlignment="Center">
            <TextBlock Text="CR3 (Page Directory Base):" FontWeight="SemiBold"
FontSize="14" VerticalAlignment="Center" Margin="0 0 10 0"/>
            <TextBox x:Name="CR3TextBox" Width="200" IsReadOnly="True"
Background="#fff" Padding="5" BorderThickness="1" BorderBrush="#ccc"/>
            <Button Content="About" Height="30" Width="100" Click="Button_Click"
Margin="480 0 0 0"
                    Background="#4CAF50" Foreground="White" FontWeight="Bold"/>
        </StackPanel>

        <!-- Linear Addresses -->
        <GroupBox Header="Linear Addresses" Grid.Row="1" Grid.ColumnSpan="2"
Margin="0 0 0 10" FontWeight="Bold">
            <DataGrid x:Name="LinearAddressesGrid" AutoGenerateColumns="False"
IsReadOnly="True" Margin="5"
                      HeadersVisibility="Column" CanUserAddRows="False"
Background="White">
                <DataGrid.Columns>
                    <DataGridTextColumn Header="Linear" Binding="{Binding
Linear, StringFormat=0x{0:X8}}" Width="*"/>
                </DataGrid.Columns>
            </DataGrid>
        </GroupBox>

        <!-- Page Directory -->
        <GroupBox Header="Page Directory" Grid.Row="2" Grid.Column="0" Margin="0
0 10 10" FontWeight="Bold">
            <DataGrid x:Name="PageDirectoryGrid" AutoGenerateColumns="False"
IsReadOnly="True" Margin="5"
                      HeadersVisibility="Column" CanUserAddRows="False"
Background="White">
                <DataGrid.Columns>
                    <DataGridTextColumn Header="Index" Binding="{Binding Index}"
Width="Auto"/>
                    <DataGridCheckBoxColumn Header="Present" Binding="{Binding
Present}" Width="Auto"/>
                    <DataGridTextColumn Header="Frame Address (Hex)"
Binding="{Binding FrameAddressHex}" Width="*"/>
                </DataGrid.Columns>
            </DataGrid>
        </GroupBox>

        <!-- TLB -->
        <GroupBox Header="TLB Cache" Grid.Row="2" Grid.Column="1" Margin="0 0 0
10" FontWeight="Bold">
            <DataGrid x:Name="TLBGrid" AutoGenerateColumns="False"
IsReadOnly="True" Margin="5"
                      HeadersVisibility="Column" CanUserAddRows="False"
Background="White">
                <DataGrid.Columns>
```

```
                    <DataGridTextColumn Header="Page Number" Binding="{Binding
PageNumber, StringFormat=0x{0:X8}}" Width="Auto"/>
                    <DataGridTextColumn Header="Frame Address" Binding="{Binding
FrameAddress, StringFormat=0x{0:X8}}" Width="Auto"/>
                    <DataGridTextColumn Header="Time Used" Binding="{Binding
LastUsed}" Width="*"/>
                </DataGrid.Columns>
            </DataGrid>
        </GroupBox>

        <!-- Page Table -->
        <GroupBox Header="Page Table" Grid.Row="3" Grid.Column="0" Margin="0 0
10 10" FontWeight="Bold">
            <DataGrid x:Name="PageTableGrid" AutoGenerateColumns="False"
IsReadOnly="True" Margin="5"
                      HeadersVisibility="Column" CanUserAddRows="False"
Background="White">
                <DataGrid.Columns>
                    <DataGridTextColumn Header="Index" Binding="{Binding Index}"
Width="Auto"/>
                    <DataGridCheckBoxColumn Header="Present" Binding="{Binding
Present}" Width="Auto"/>
                    <DataGridTextColumn Header="Frame Address (Hex)"
Binding="{Binding FrameAddressHex}" Width="*"/>
                </DataGrid.Columns>
            </DataGrid>
        </GroupBox>

        <!-- Dir:Tbl:Offset -->
        <GroupBox Header="Directory : Table : Offset" Grid.Row="3"
Grid.Column="1" Margin="0 0 0 10" FontWeight="Bold">
            <DataGrid x:Name="DirTableOffsetGrid" AutoGenerateColumns="False"
IsReadOnly="True" Margin="5"
                      HeadersVisibility="Column" CanUserAddRows="False"
Background="White">
                <DataGrid.Columns>
                    <DataGridTextColumn Header="Dir Index" Binding="{Binding
DirIndex}" Width="*"/>
                    <DataGridTextColumn Header="Tbl Index" Binding="{Binding
TblIndex}" Width="*"/>
                    <DataGridTextColumn Header="Offset" Binding="{Binding
Offset}" Width="*"/>
                </DataGrid.Columns>
            </DataGrid>
        </GroupBox>

        <!-- Physical Addresses -->
        <GroupBox Header="Calculated Physical Addresses" Grid.Row="4"
Grid.ColumnSpan="2" FontWeight="Bold">
            <DataGrid x:Name="PhysicalAddressesGrid" AutoGenerateColumns="False"
IsReadOnly="True" Margin="5"
                      HeadersVisibility="Column" CanUserAddRows="False"
Background="White">
                <DataGrid.Columns>
                    <DataGridTextColumn Header="Linear" Binding="{Binding
Linear , StringFormat=0x{0:X8}}" Width="*"/>
                    <DataGridTextColumn Header="Physical" Binding="{Binding
Physical , StringFormat=0x{0:X8}}" Width="*"/>
                    <DataGridTextColumn Header="Status" Binding="{Binding
Status}" Width="*"/>
                </DataGrid.Columns>
            </DataGrid>
        </GroupBox>
```

```
        <!-- Buttons -->
        <StackPanel Grid.Row="5" Grid.ColumnSpan="2" Orientation="Horizontal"
HorizontalAlignment="Center" Margin="0 10 0 0">
            <Button Content="Назад" Width="150" Margin="0 0 20 0"
Click="OnBack_Click"
                    Background="#2196F3" Foreground="White" FontWeight="Bold"/>
            <Button Content="Обчислити фізичну адресу" Width="250"
Click="CalculatePhysicalAddress_Click"
                    Background="#4CAF50" Foreground="White" FontWeight="Bold"/>

        </StackPanel>
    </Grid>
</Window>
```

## LinearToPhysical.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Remoting.Contexts;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Xml;

namespace diplom
{
    /// <summary>
    /// Логика взаимодействия для LinearToPhysical.xaml
    /// </summary>
    public partial class LinearToPhysical : Window
    {
        private const int TLB_CAPACITY = 16; // Максимальна кількість записів у
TLB
        private uint _linearAddress;
        private uint _cr3 = 0x0009_0000; // CR3 — базовый адрес Page Directory
        private List<PageDirectoryEntry> _pageDirectory;
        private List<PageTableEntry> _pageTable;
        private List<TLBEntry> _tlb = new List<TLBEntry>();


        public LinearToPhysical()
        {
            InitializeComponent();
            InitializeTLB();

        }
        public LinearToPhysical(IEnumerable<uint> linearAddresses)
        {
            InitializeComponent();

            LinearAddressesGrid.ItemsSource = linearAddresses
            .Select(addr => new { Linear = addr })
```

```
            .ToList();
            ShowAddressInfo(linearAddresses);
            LoadPageDirectory();
            LoadPageTable();
            InitializeTLB();

        }
        private void ShowAddressInfo(IEnumerable<uint> linearAddresses)
        {
            var entries = linearAddresses.Select(addr =>
            {
                return new LinearAddressEntry
                {
                    Linear = addr,
                    DirIndex = (addr >> 22) & 0x3FF,
                    TblIndex = (addr >> 12) & 0x3FF,
                    Offset = addr & 0xFFF
                };
            }).ToList();

            DirTableOffsetGrid.ItemsSource = entries;

            CR3TextBox.Text = $"0x{_cr3:X8}";
        }
        private void InitializeTLB()
        {
            var random = new Random();
            _tlb = new List<TLBEntry>();

            for (int i = 0; i < 16; i++)
            {
                var pageNumber = random.Next(0x00000, 0xFFFFF);         // 20-
бітовий номер сторінки
                var frameAddress = random.Next(0x00100000, 0x01FFFFFF); //
Адреса фрейму (вирівняна по сторінці, умовно)

                _tlb.Add(new TLBEntry
                {
                    PageNumber = (uint)pageNumber,
                    FrameAddress = (uint)frameAddress & 0xFFFFF000, //
Вирівнюємо на кордоні сторінки (4Кб)
                    LastUsed = DateTime.Now.AddMilliseconds(-random.Next(0,
10000)) // Імітація давнішнього використання
                });
            }

            TLBGrid.ItemsSource = _tlb;
        }

        private void LoadPageDirectory()
        {
            _pageDirectory = new List<PageDirectoryEntry>();
            for (int i = 0; i < 1024; i++)
            {
                _pageDirectory.Add(new PageDirectoryEntry
                {
                    Index = i,
                    Present = true,
                    FrameAddress = 0x00400000 + (uint)(i * 0x1000),
                    IsHighlighted = false
                });
            }
```

```
            PageDirectoryGrid.ItemsSource = _pageDirectory;
        }

        private void LoadPageTable()
        {
            _pageTable = new List<PageTableEntry>();
            for (int i = 0; i < 1024; i++)
            {
                _pageTable.Add(new PageTableEntry
                {
                    Index = i,
                    Present = true,
                    FrameAddress = 0x00800000 + (uint)(i * 0x1000),
                    IsHighlighted = false
                });
            }

            PageTableGrid.ItemsSource = _pageTable;
        }


        private void CalculatePhysicalAddress_Click(object sender,
RoutedEventArgs e)
        {
            if (DirTableOffsetGrid.Items.Count == 0) return;

            // Очистити підсвічування
            foreach (var pdEntry in _pageDirectory) pdEntry.IsHighlighted =
false;
            foreach (var ptEntry in _pageTable) ptEntry.IsHighlighted = false;

            List<PhysicalAddressResult> results = new
List<PhysicalAddressResult>();

            foreach (LinearAddressEntry entry in DirTableOffsetGrid.Items)
            {
                if (entry == null) continue;

                uint dirIndex = entry.DirIndex;
                uint tblIndex = entry.TblIndex;
                uint offset = entry.Offset;
                uint pageNumber = (entry.Linear >> 12) & 0xFFFFF;

                uint frameAddress;

                // Спроба знайти в TLB
                var tlbEntry = _tlb.FirstOrDefault(en => en.PageNumber ==
pageNumber);

                if (tlbEntry != null)
                {
                    frameAddress = tlbEntry.FrameAddress;
                    tlbEntry.LastUsed = DateTime.Now;
                }
                else
                {
                    if (dirIndex >= _pageDirectory.Count || tblIndex >=
_pageTable.Count)
                    {
                        results.Add(new PhysicalAddressResult
                        {
                            Linear = entry.Linear,
                            Physical = 0,
```

```
                    Status = "Помилка: Невірні індекси"
                });
                continue;
            }

            _pageDirectory[(int)dirIndex].IsHighlighted = true;
            _pageTable[(int)tblIndex].IsHighlighted = true;

            frameAddress = _pageTable[(int)tblIndex].FrameAddress;

            if (_tlb.Count >= TLB_CAPACITY)
            {
                var oldest = _tlb.OrderBy(en => en.LastUsed).First();
                _tlb.Remove(oldest);
            }

            _tlb.Add(new TLBEntry
            {
                PageNumber = pageNumber,
                FrameAddress = frameAddress,
                LastUsed = DateTime.Now
            });
        }

        uint physicalAddress = frameAddress + offset;

        results.Add(new PhysicalAddressResult
        {
            Linear = entry.Linear,
            Physical = physicalAddress,
            Status = tlbEntry != null ? "TLB Hit" : "TLB Miss"
        });
    }

    // Оновити відображення
    PageDirectoryGrid.Items.Refresh();
    PageTableGrid.Items.Refresh();
    TLBGrid.Items.Refresh();
    // Вивести результати у нову таблицю або текстове поле
    PhysicalAddressesGrid.ItemsSource = results;
}



private void OnBack_Click(object sender, RoutedEventArgs e)
{
    var window = new LogicToLinear();
    window.Show();
    this.Close();
}

void ToAbout()
{
    var window = new About();
    window.Show();
    this.Close();
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    ToAbout();
}
```

```
    }
}
```

## About.xaml

```
<Window x:Class="diplom.About"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:diplom"
        mc:Ignorable="d"
        Title="Про програму" Height="450" Width="800">

    <Grid Background="#f0f0f0" Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>

        <!-- Інформаційний блок -->
        <Border Grid.Row="0"
                Background="White"
                CornerRadius="10"
                Padding="20"
                BorderBrush="#CCC"
                BorderThickness="1"
                Margin="0,0,0,10">
            <ScrollViewer VerticalScrollBarVisibility="Auto">
                <StackPanel>
                    <!-- Заголовок -->
                    <TextBlock Text="Інформація про програму"
                               FontSize="20"
                               FontWeight="Bold"
                               TextAlignment="Center"
                               Margin="0 0 0 20"/>

                    <!-- Основна інформація -->
                    <TextBlock FontSize="14" FontFamily="Consolas"
TextWrapping="Wrap">
                        <Run Text="Memory Management Emulator v1.0.0&#x0a;"
FontWeight="Bold"/>
                        <Run Text="Тема роботи: Програмне забезпечення системи
емулятора&#x0a;"/>
                        <Run Text="перетворення логічної адреси в лінійну та
лінійної у фізичну для навчальних цілей&#x0a;&#x0a;"/>

                        <Run Text="Курс: IV&#x0a;"/>
                        <Run Text="Група: КІ-21-1&#x0a;"/>
                        <Run Text="Студент: Борохович С.А&#x0a;"/>
                        <Run Text="Керівник: Коваленко О.В&#x0a;&#x0a;"/>

                        <Run Text="Мета та завдання:&#x0a;"/>
                        <Run Text="Розробка програмного забезпечення, яке
моделює процес трансляції адрес:&#x0a;"/>
                        <Run Text="- логічної → лінійної&#x0a;"/>
                        <Run Text="- лінійної → фізичної&#x0a;"/>
                        <Run Text="з використанням сегментації, сторінкової
організації пам’яті, таблиць сторінок і буфера TLB.&#x0a;&#x0a;"/>

                        <Run Text="Використані технології:&#x0a;"/>
```

```
                        <Run Text="• Сегментація пам’яті&#x0a;"/>
                        <Run Text="• Сторінкова організація пам’яті
(Paging)&#x0a;"/>
                        <Run Text="• Буфер TLB&#x0a;&#x0a;"/>

                        <Run Text="© 2025 ЦНТУ" FontStyle="Italic"/>
                    </TextBlock>
                </StackPanel>
            </ScrollViewer>
        </Border>

        <!-- Кнопка повернення -->
        <StackPanel Grid.Row="1" Orientation="Horizontal"
HorizontalAlignment="Left">
            <Button Content="← Назад"
                    Width="120"
                    Padding="5"
                    FontWeight="Bold"
                    Margin="0,0,0,0"
                    Click="OnBack_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

About.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace diplom
{
    /// <summary>
    /// Логика взаимодействия для About.xaml
    /// </summary>
    public partial class About : Window
    {
        public About()
        {
            InitializeComponent();
        }

        private void OnBack_Click(object sender, RoutedEventArgs e)
        {
            var window = new LogicToLinear();
            window.Show();
            this.Close();
        }
    }
}
```

## LinearAddressEntry.cs

```
namespace diplom
{
    internal class LinearAddressEntry
    {
        public uint Linear { get; set; }
        public uint DirIndex { get; set; }
        public uint TblIndex { get; set; }
        public uint Offset { get; set; }

        public uint PageNumber => Linear >> 12;
    }
}
```

## PageDirectoryEntry.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class PageDirectoryEntry
    {
        public int Index { get; set; }
        public bool Present { get; set; }
        public uint FrameAddress { get; set; }
        public string FrameAddressHex => $"0x{FrameAddress:X8}";

        public bool IsHighlighted { get; set; }
    }
}
```



## PageTableEntry.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class PageTableEntry
    {
        public int Index { get; set; }
        public bool Present { get; set; }
        public uint FrameAddress { get; set; }
        public string FrameAddressHex => $"0x{FrameAddress:X8}";
        public bool IsHighlighted { get; set; }

    }
```

```
}
```

## PhysicalAddressResult.cs

```
namespace diplom
{
    internal class PhysicalAddressResult
    {
        public uint Linear { get; set; }
        public uint Physical { get; set; }
        public string Status { get; set; }
    }
}
```

## TLBEntry.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class TLBEntry
    {
        public uint PageNumber { get; set; }     // Верхні 20 біт лінійної
адреси
        public uint FrameAddress { get; set; }

        public DateTime LastUsed { get; set; }// Фізична адреса сторінки
    }
}
```

## SegmentDescriptor.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class SegmentDescriptor
    {
        public uint BaseAddress { get; set; }
        public uint Limit { get; set; }

        public SegmentDescriptor(uint baseAddr, uint limit)
        {
            BaseAddress = baseAddr;
            Limit = limit;
        }

        public override string ToString()
        {
            return $"Base: 0x{BaseAddress:X}, Limit: 0x{Limit:X}";
        }
    }
```

```
}
```

## SegmentDescriptorView.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class SegmentDescriptorView
    {
        public int Index { get; set; }
        public uint BaseAddress { get; set; }
        public uint Limit { get; set; }

        public SegmentDescriptorView(int index, uint baseAddr, uint limit)
        {
            Index = index;
            BaseAddress = baseAddr;
            Limit = limit;
        }

        public SegmentDescriptor ToDescriptor() => new
SegmentDescriptor(BaseAddress, Limit);
    }
}
```

## SegmentSelector.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class SegmentSelector
    {
        public ushort RawValue { get; }

        public int Index => (RawValue >> 3) & 0x1FFF; // 13 біт
        public bool TableIndicator => ((RawValue >> 2) & 0x1) == 1; // 1 біт:
false = GDT, true = LDT
        public int RPL => RawValue & 0x3; // 2 біти

        public SegmentSelector(ushort value)
        {
            RawValue = value;
        }

        public override string ToString()
        {
            return $"Index: {Index}, TI: {(TableIndicator ? "LDT" : "GDT")},
RPL: {RPL}";
```

```
        }
    }
}
```

SegmentSelectorView.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class SegmentSelectorView
    {
        public int Index { get; set; }
        public ushort Value { get; set; }
        public string DisplayName { get; set; }
    }
}
```

TranslationResult.cs

```
namespace diplom
{
    public class TranslationResult
    {
        public string OffsetHex { get; set; }
        public string LinearHex { get; set; }
    }
}
```



AddressTranslator.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace diplom
{
    public class AddressTranslator
    {

        private readonly SegmentDescriptor[] gdt;

        public AddressTranslator(SegmentDescriptor[] gdtEntries)
        {
            gdt = gdtEntries;
        }

        public uint TranslateToLinear(LogicalAddress logical)
        {
            if (logical.Selector.TableIndicator)
                throw new NotSupportedException("LDT is not supported in this
model.");

            int index = logical.Selector.Index;
            if (index < 0 || index >= gdt.Length)
                throw new IndexOutOfRangeException("Invalid segment selector
index.");

            var descriptor = gdt[index];
            if (logical.Offset > descriptor.Limit)
                throw new ArgumentOutOfRangeException("Offset exceeds segment
limit.");

            return descriptor.BaseAddress + logical.Offset;
        }
    }
}
```