Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2023 р

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Розробка програмної моделі навчального цифрового процесора»

Захищено на засіданні
Атестаційної комісії № 40
протокол № __ від __.06.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ Скоб Ю. О.
(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи КІ– 41
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 – «Комп'ютерна інженерія»
КОМЕРИСТИЙ Владислав Сергійович
(підпис)

Керівник: кандидат технічних наук, доцент
РЕВА Сергій Миколайович
(підпис)

Рецензент: доцент кафедри штучного інтелекту та програмного забезпечення, кандидат фізико-математичних наук
СПОРОВ Олександр Євгенович
(підпис)

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і 4 додатків. Загальний обсяг роботи складає 78 сторінок, із яких 52 сторінки основної частини з 22 рисунками, 3 таблицями, 14 найменуваннями списку використаних джерел та 4 додатками.

Метою кваліфікаційної роботи є створення програмної моделі цифрового процесора DPM-08 для вдосконалення навчального процесу та покращення базових знань студентів з мікропроцесорної техніки.

Об'єкт дослідження – реальна модель цифрового процесору DPM-08.

Предмет дослідження – програмна модель того ж процесору, а саме емулятор, який повністю імітує роботу реальної моделі.

Проблема, яка вирішується в кваліфікаційній роботі полягає в тому, щоб на основі аналізу апаратних та програмних можливостей реального процесора створити програмну модель, яку можна використовувати під час навчального процесу у дистанційній формі та для самостійного відлагодження програм студентами в домашніх умовах.

Область застосування – покращення навчального процесу. Таким чином, дана програма допоможе студентам попрацювати з моделлю цифрового процесора під час дистанційного навчання, не маючи фізичного приладу.

Ключові слова: емулятор, процесор, модель, Qt, C++, команда, регістр, пам’ять, програма, байт, біт, генератор, кнопка.

ABSTRACT

The explanatory note to the bachelor's qualification work consists of an introduction, three sections, conclusions, a list of used sources and 4 appendices. The total volume of the work is 78 pages, of which 52 pages are the main part with 22 figures, 3 tables, 14 titles of the list of used sources and 4 appendices.

The purpose of the qualification work is to create a software model of the DPM-08 digital processor to improve the educational process and improve students' basic knowledge of microprocessor technology.

The research object is a real model of the DPM-08 digital processor.

The subject of research is a software model of the same processor, namely an emulator, which fully simulates the operation of a real model.

The problem that is solved in the qualification work is to create a software model based on the analysis of the hardware and software capabilities of a real processor, which can be used during the educational process in remote form and for independent debugging of programs by students at home.

The field of application is the improvement of the educational process. Thus, this program will help students to work with a model of a digital processor during distance learning without having a physical device.

Keywords: emulator, processor, model, Qt, C++, command, register, memory, program, byte, bit, generator, button.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ – електронна обчислювальна машина

DPM-08 – Digital process model 08 (8-розрядний процесор)

ПЗ – програмне забезпечення

МК – мікроконтролер

АЛП – арифметико-логічний пристрій

ОЗП – оперативний запам’ятовуючий пристрій

ПК – персональний комп’ютер

GUI (Graphics User Interface) – графічний користувацький інтерфейс

ВСТУП

Сучасний світ неможливо уявити без електронних обчислювальних машин (ЕОМ). Літаки, кораблі, автомобілі, різноманітна техніка, така як пральна машина, пилосос, посудомийка та інші – усе це не було б можливим без створення та розвитку обчислювальних машин. Основою ЕОМ є головний процесор. Історія розвитку цього компоненту бере свій початок ще в 70-х роках 20 століття. Сучасні процесори є складними компонентами, які містять велику кількість внутрішніх елементів і здатні виконувати велику кількість обчислювальних дій за секунду. Але для розуміння базових принципів роботи мікропроцесорів було б непогано використовувати більш прості моделі. Прикладом є процесор DPM-08, створений викладачами разом зі студентами факультету комп'ютерних наук харківського університету ім. В.Н.Каразіна. Але окрім реальної моделі є необхідність ще створити програмний застосунок, який буде імітувати роботу реальної моделі процесора, іншими словами – емулятор. На сьогоднішній день існують різні емулятори для роботи з процесами, такі як Keil uVision, Turbo Debugger, Proteus тощо. Але всі вони не дуже добре підходять для навчального процесу і для розуміння того, як взагалі працює процесор.

Актуальність роботи. Останні роки є дуже складними для нашої країни. Спочатку пандемія, потім напад російських загарбників. У зв'язку з цим студенти вимушені навчатися в дистанційному форматі, не маючи можливості взаємодіяти з певними приладами. Наприклад, із вказаною вище моделлю процесора. Тому створення емулятору є актуальним як ніколи. Саме завдяки такому програмному застосунку студенти будуть мати можливість попрацювати з процесором, розібратися в принципах його роботи, познайомитися з програмуванням машинних кодів та мови асемблера, розглянути етапи низькорівневого

програмування проаналізувати кожен етап виконання команди, відстежити зміни даних, завантажити та запустити деякі найпростіші програми.

Метою дослідження є покращення навчального процесу та отримання досвіду програмування на мовах низького рівня під час вивчення мікропроцесорної техніки. Для цього необхідно створити програмну модель із зручним графічним інтерфейсом, яка допоможе в проведенні дистанційних занять відповідних навчальних курсів, таких як «Мікропроцесори», «Основи комп'ютерної техніки». Програма буде імітувати роботу справжньої моделі процесора. Окрім того, що вона буде показувати результати роботи процесора, ще можна буде наочно спостерігати за тим, як цей результат взагалі було отримано. Емулятор буде працювати під системами Windows XP, Windows 7, Windows 8 та Windows 10, що надасть можливість кожному студенту запустити застосунок на своєму комп'ютері, навіть не маючи великих потужностей ЕОМ.

Об'єктом дослідження є реальна модель процесору DPM-08. Для написання програми-емулятора необхідно мати чітке уявлення про те, які архітектурні та програмні можливості має реальна модель процесора. Таким чином, її роботу можна ефективно промоделювати на екрані комп'ютера.

Методи дослідження: аналіз, синтез, експериментальні дослідження.

Предмет дослідження – програмна модель цифрового процесора DPM-08, яка покладена в основу створення програми-емулятора.

Завдання дослідження:

1. Провести аналіз існуючих рішень, тобто дослідити наявні емулятори та визначити їх недоліки

2. Розробити власну програмну модель у вигляді емулятора, яка буде в повному обсязі реалізувати апаратні та програмні можливості реальної моделі цифрового процесора.

3. Провести тестування власного рішення та виправити помилки. Після завершення роботи над емулятором, створити інструкцію по роботі з емулятором.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ПРОГРАМ ЕМУЛЯТОРІВ

На сьогоднішній день існує чимало різних емуляторів для роботи з процесорами, мікроконтролерами. Деякі дозволяють писати програми і одразу дивитись на результат їх роботи, деякі дозволяють взагалі побудувати ледь не цілий комп'ютер. Інші дозволяють під часи виконання програми спостерігати за станом пам'яті та результатом виконання окремих команд. Я пропоную до розгляду три програми-емулятори: Keil uVision, Proteus та Turbo Debugger.

1.1.Програма Keil uVision

Keil uVision – це віконна платформа розробки ПЗ, яка об'єднує в собі всі інструменти, що необхідні для розробки вбудованих програм. Сюди також включається компілятор C/C++, асемблер макросів, компонувальник та генератор HEX файлів.

Якщо казати про функції даної програми, то вона допомагає прискорити процес розробки вбудованих програм. Для цього в даному емуляторі продумано наступні можливості:

- Редактор вихідного коду, який є повноцінним і повнофункціональним;
- База даних пристроїв, що допомагає налаштовувати інструменти розробки;
- Можливість створювати нові проекти або відкривати створені раніше;
- Є така вбудована функція як Make Utility, що дозволяє складати, компілювати та підключати ваші вбудовані програми;
- Вбудований налагоджувач на рівні асемблера з високошвидкісним процесором та симулятором периферійних пристроїв;
- Утиліта Flash-програмування для завантаження програм у Flash ROM [1].

Звичайно, це не всі функції, але основні. На рисунку 1.1 можна побачити, як виглядає інтерфейс цієї програми:

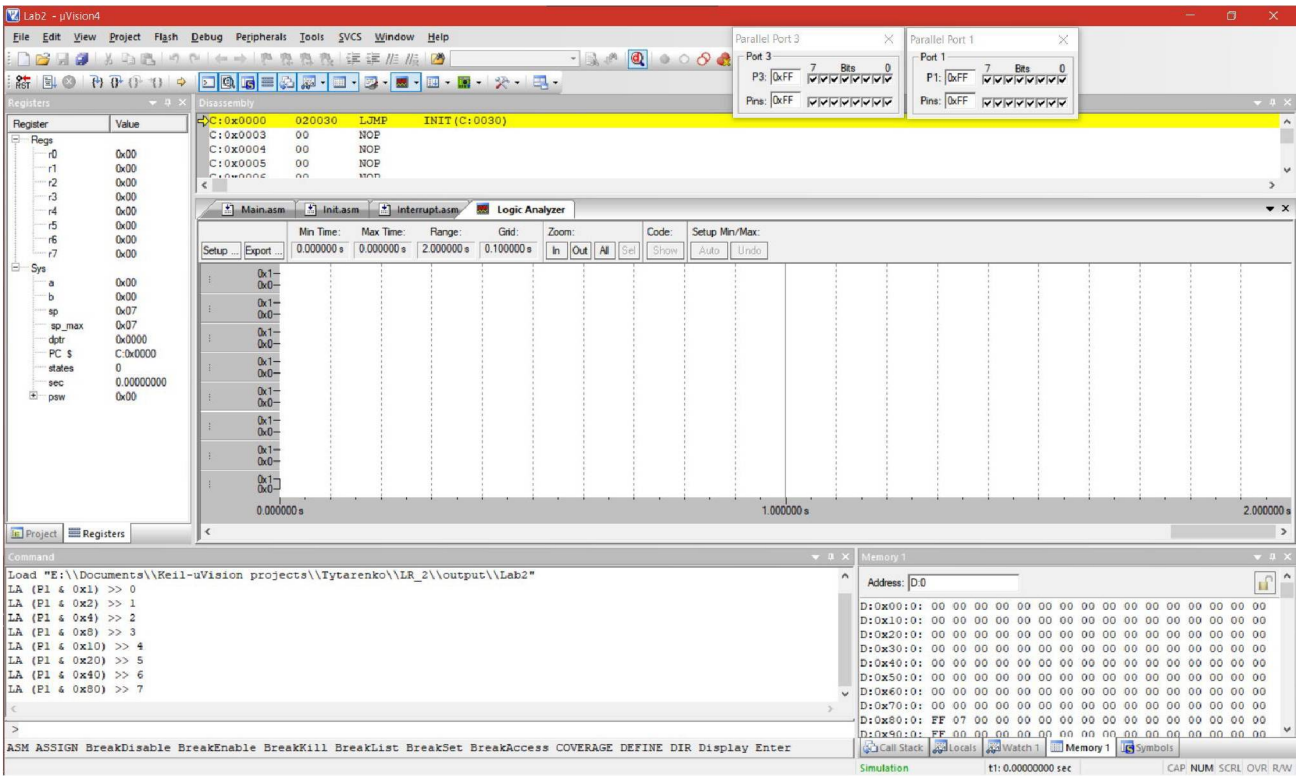

Рисунок 1.1 – Вікно відлагоджувальника Keil uVision

Можна відмітити, що можливість написання коду прямо в програмі — це є значною перевагою даного емулятору, якщо користувачеві потрібно написати програму під якийсь конкретний процесор. Вбудований налагоджувач дозволить виконати програму покроково, при цьому можна відстежувати зміни в регістрах (на рисунку 1.1 відображено ліворуч), в блоці пам’яті, в портах, дивитись стан змінних, що використовуються в програмі. Також можна прослідкувати за роботою таймерів, наприклад. Більше того, Keil uVision містить вбудований аналізатор, який надає можливість спостерігати за рівнями сигналів на портах. Віконце, що показує стан пам’яті, показано на рисунку 1.2.

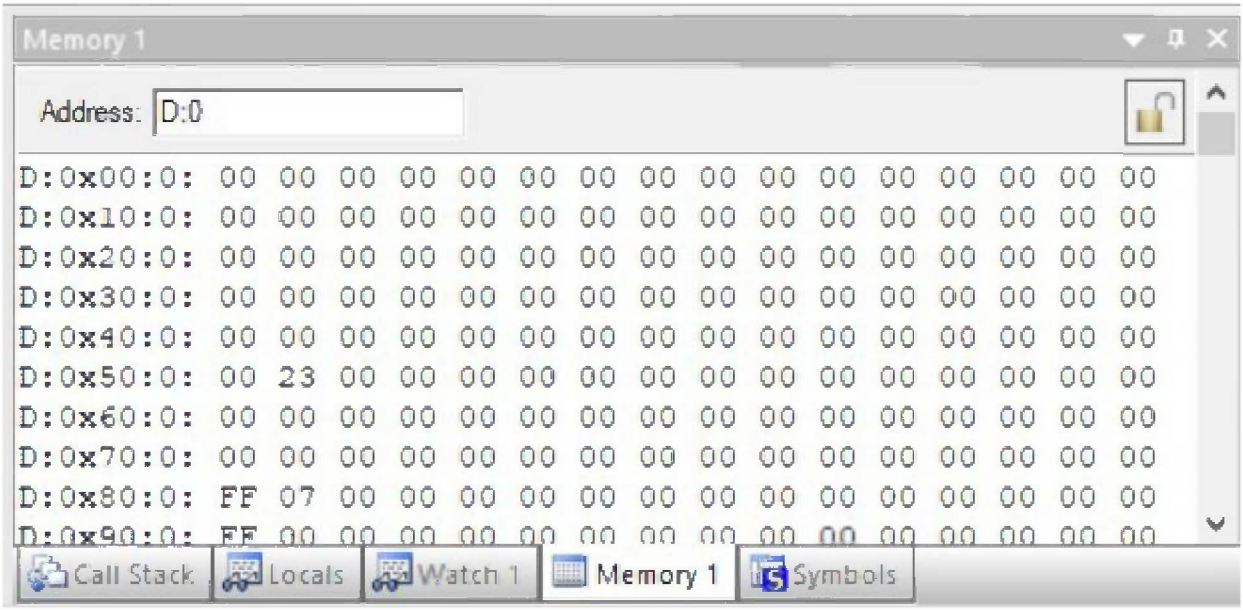

Рисунок 1.2 – Вікно пам’яті в Keil uVision

Keil uVision надає можливість працювати із дуже великою кількістю різноманітних процесорів та мікроконтролерів: Atmel, STM, Chipcon, Daewoo, і це лише мала частина. Під час створення проекту можна обрати той МК, який потрібен, і працювати з ним: писати код програми, компілювати його, налагоджувати, виконувати покроково, відстежувати стан регістрів, тощо.

Недоліком такої програми-емулятора є те, що вона не має необхідної ступені наочності. Користувач може бачити стан регістрів і зрозуміти, як відбувається процес однієї команди на рівні машинного циклу. Але відсутня можливість спостерігати за тим, звідкіля на кожному кроці зчитується значення, як взаємодіє процесор з пам’яттю, як аналізуються команди. Основним недоліком є те, що ця програма, як і ті, що будуть розглянуті далі, призначені для відлагодження програм, але ніяк не для вивчення роботи процесору.

1.2.Програма Proteus

Proteus – програмне забезпечення, яке поєднує моделювання схем, проектування друкованої плати та віртуальне прототипування. Він може імітувати мікроконтролери від різних виробників, таких як Microchip, Atmel, Texas Instruments тощо. Має в своєму розпорядженні такі функції, як інтерактивне моделювання, змішане моделювання, захоплення схем і компонування друкованих плат [2]. На рисунку 1.3 показано, як виглядає ця програма з відкритим проектом:

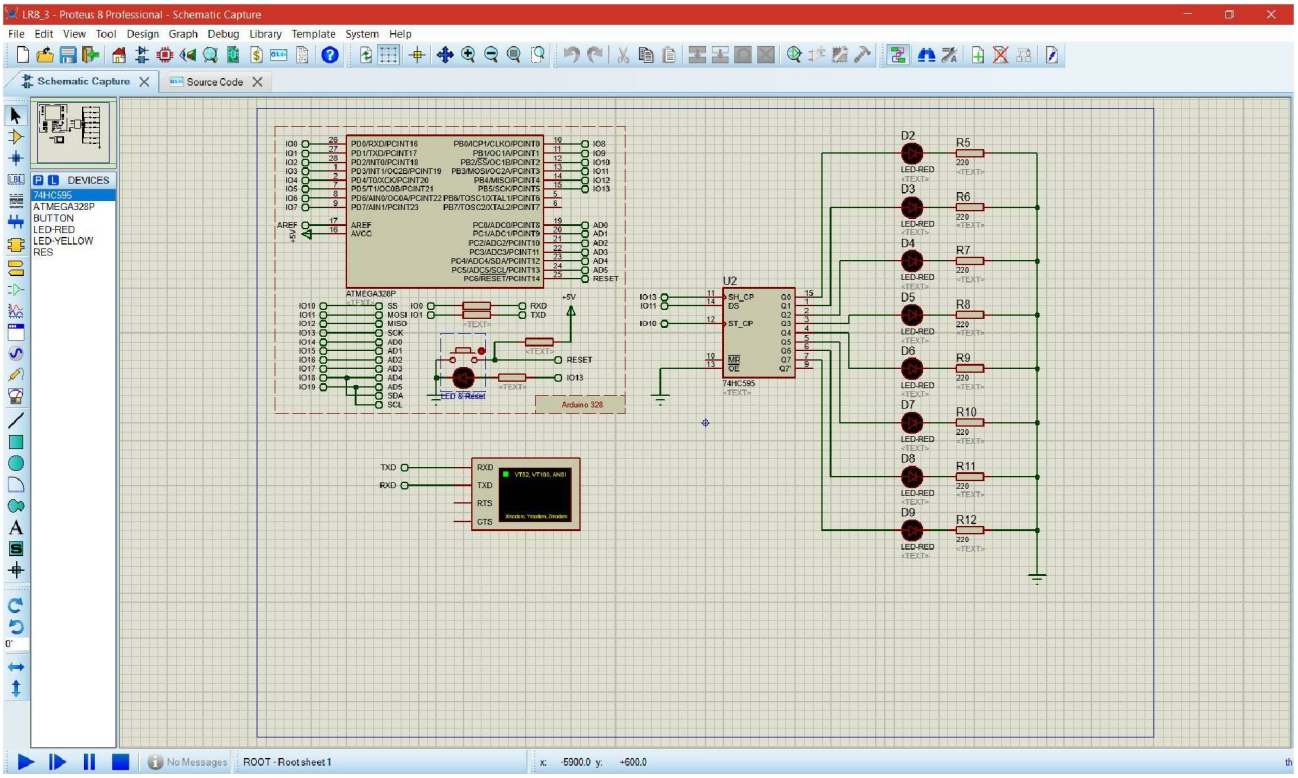

Рисунок 1.3 – Робоче вікно програми Proteus

Незважаючи на те, що даний програмний продукт більше використовується для побудови різноманітних складних схем, його можна використовувати і для імітації роботи різних процесорів. Емулятор містить в собі дуже велику кількість різноманітних МК та мікропроцесорів. Як і в Keil, в Proteus наявний редактор коду, компілятор. Код можна писати на C/C++, Assembly, BASIC тощо. Для

перевірки роботи коду можна використовувати вбудовані або сторонні налагоджувачі.

Цей емулятор має досить вагомий недолік. Пов'язаний він з тим, що користувач може розміщувати процесори, будувати схеми з ними і спостерігати за станом вхідних та вихідних портів. Проте відсутня можливість подивитись на те, що відбувається в середині процесора. Більше того, в порівнянні з Keil в Proteus робота з пам'яттю є менш зручною. Наприклад, на рисунку 1.4 показано, як це виглядає під час паузи симуляції:

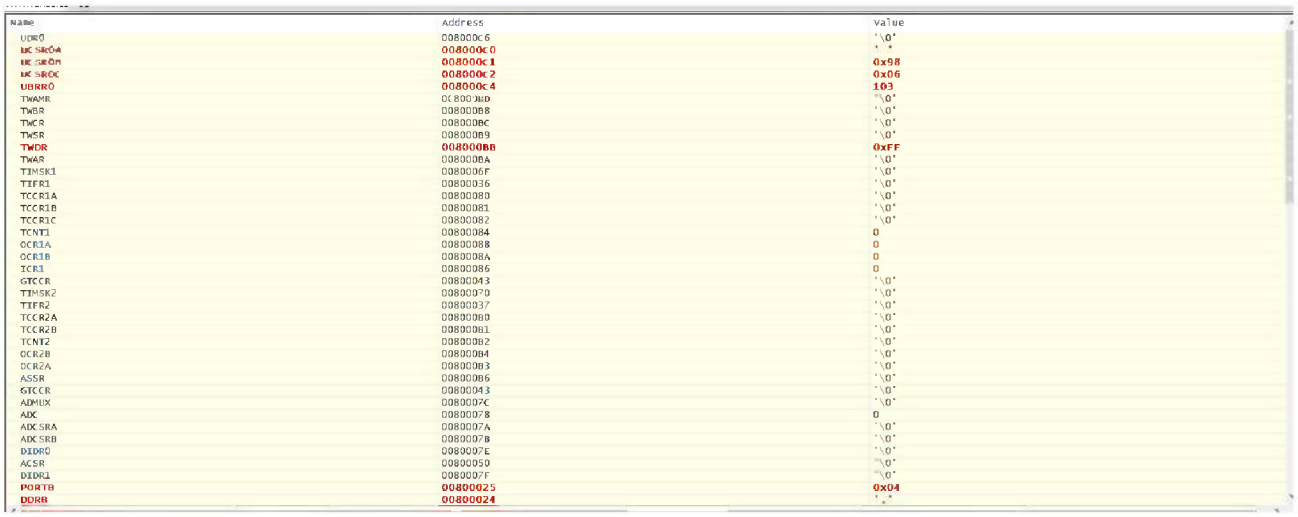

Рисунок 1.4 – Модуль пам'яті в Proteus

Якщо уважно роздивитись цей рисунок, можна побачити наявність назв регістрів, відповідні адреси та значення в цих регістрах. Більше того, доки симуляція запущена в реальному часі, ці регістри не можна передивитись таким чином. Звичайно, є способи використання різноманітних моніторів, аналізаторів тощо. Але такі методи не відображають гарно структуровану пам'ять.

Даний програмний продукт має багато можливостей і переваг. Це достатньо потужна програма, яка дозволяє будувати складні електронні схеми, проектувати різні модулі, використовувати готові компоненти або створювати їх власноруч.

Але, як було зазначено вище, для вивчення принципів роботи процесорів Proteus не володіє достатніми демонстраційними можливостями.

1.3.Програма Turbo Debugger

Turbo Debugger – це продукт компанії Borland, який був створений для роботи з програмами DOS. Він може налагоджувати 16-бітні та 32-бітні програми, які були написані на асемблері, C/C++, Pascal та інших мовах. Він також може налагоджувати програми, що виконуються на цільовому обладнанні через паралельний порт.

Основні можливості цього налагоджувача наступні:

- Точки зупинки, що дозволяють призупинити виконання програми у певних місцях коду для аналізу стану програми;
- Спостереження за змінними, що дозволяло відстежувати зміни змінних під час виконання програми;
- Крокування, що дозволило виконувати по одній інструкції за раз і детально аналізувати, що було отримано;
- Дамп пам’яті, який містив дані для перегляду;
- Дизасемблювання – перетворення машинного коду на зрозумілий для програміста вигляд [3].

На рисунку 1.5 показано, який вигляд має даний програмний продукт.

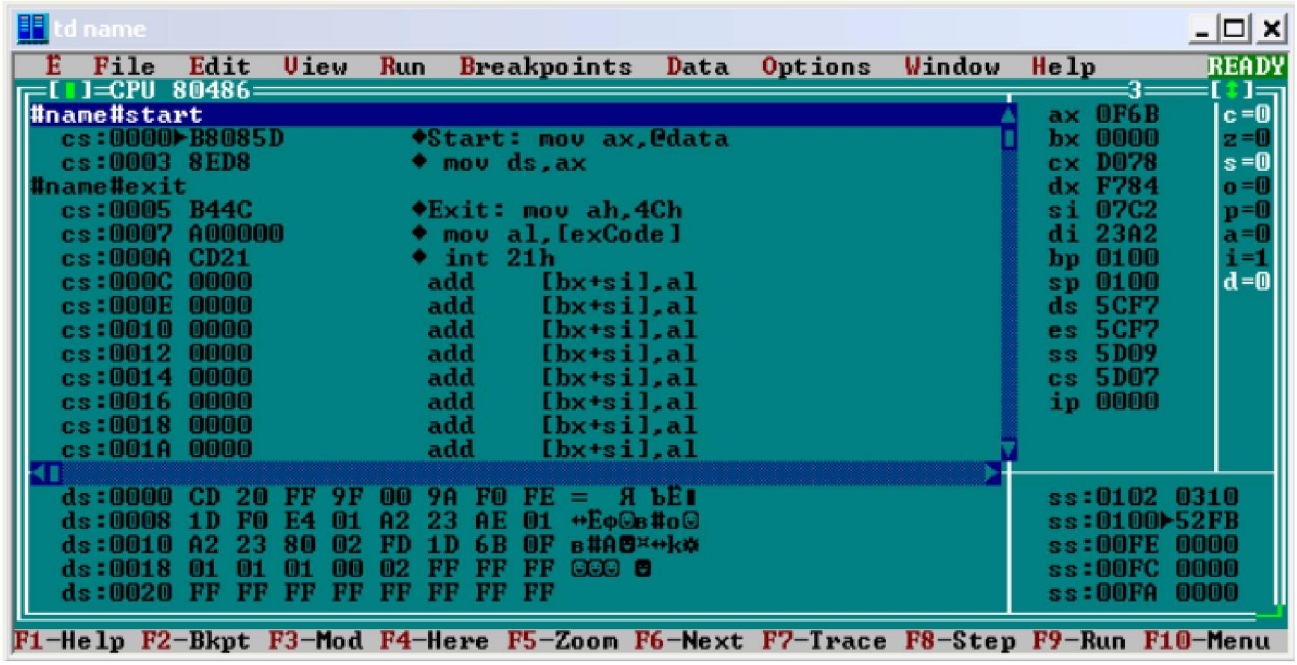

Рисунок 1.5 – Робоче вікно налагоджувача Turbo Debugger

Незважаючи на те, що це є налагоджувач, а не емулятор, за своїм функціоналом він дуже схожий на те програмне рішення, яке буде представлене надалі. Робота з пам’яттю, можливість покрокового виконання, відображення вмісту регістрів та змінних. Все це буде також наявним в розробленому новому рішенні. Але є відмінність – нова програма буде більш зручною та наочною, аніж Turbo Debugger. Насамперед, це буде можливість відстежувати, звідки дані беруться, куди вони записуються, які дії при цьому виконуються. Таким чином, буде простіше зрозуміти, яким чином виконується та чи інша команда.

Висновки за розділом 1

Підводячи підсумки аналізу існуючих рішень, можна зробити висновок, що всі переглянуті емулятори по-своєму унікальні, мають певні переваги і недоліки. Але якщо потрібно покращити та зробити більш ефективним навчальний процес у складні часи дистанційного навчання і зрозуміти основні

принципи роботи процесорів, використовуючи модель DPM-08, то перераховані вище моделі не можуть бути використані, оскільки вони не передбачають налаштування на систему команд цієї моделі. Окрім цього, вони призначені для роботи професіоналів для відлагодження програм, і з цієї точки зору не є настільки наочними, щоб на їх основі можна було вивчати принцип дії процесора. В усіх прикладах відсутня можливість демонстрації проходження всіх внутрішніх сигналів, послідовності виконання машинного циклу, взаємодії процесора з пам'яттю за допомогою зовнішніх сигналів сполучення між ними. Через те, що весь цей функціонал є необхідним для знайомства з процесором, виникає необхідність створити власний емулятор, який буде реалізувати можливості, що відсутні в інших програмах. Тобто вони призначені для роботи з тим чи іншим процесором, що промислово випускаються, але всі ці процесори складні за своєю архітектурою, тому не підходять для вивчення принципів основ роботи мікропроцесорної техніки.

РОЗДІЛ 2. ПРОГРАМНА МОДЕЛЬ ПРОЦЕСОРА DPM-08

Проаналізувавши деякі існуючі аналоги програм-емуляторів і з'ясувавши, що всі вони не виконують потрібні функції для використання в навчальному процесі, було прийнято рішення про необхідність створення власної програмної моделі для процесора DPM-08, в якій буде передбачено всі необхідні нюанси та деталі, що допоможуть в подальшому студентам ознайомитися з принципами роботи процесорів, навчитися програмувати, використовуючи можливості мов низького рівня програмування та завантажувати їх для перевірки роботи та розгляду процесу виконання створених програм.

1.1. Вибір програмного забезпечення для реалізації програмної моделі

Для створення програмної моделі процесора DPM-08 було проаналізовано можливі шляхи реалізації. Вибір стояв між створенням окремого додатку або ж у вигляді односторінкового веб-сайту. Через відсутність знань і досвіду у веб програмуванні, цей варіант був відкинутий, адже для цього знадобився би певний час для вивчення концепцій створення веб-сторінок, вибору набору інструментів тощо. Враховуючи умови дистанційного заняття на сьогоднішній день, така програмна модель є конче необхідною для студентів уже в теперішній час. Для створення окремого застосунку з графічним інтерфейсом треба було визначитися з мовою програмування та можливим додатковим інструментарієм. Вибір зупинився на мові програмування C++. Але через відсутність вбудованих бібліотек для роботи з графічним інтерфейсом, також було проаналізовано можливість використання сторонніх бібліотек для вирішення цього питання. Вибір серед цих бібліотек достатньо великий. Проте було обрано бібліотеку Qt, і разом з нею було використано фреймворк Qt Creator. Фреймворк — це певне програмне середовище, яке дозволяє прискорити та полегшити роботу

по створенню ПЗ. При роботі з фреймворком не потрібно робити додаткову роботу по налаштуванню баз даних (якщо вони потрібні), автентифікації. Можна писати код, який лише реалізує логіку для певного програмного продукту [4].

Мова програмування C++ – мова високого рівня, що може підтримувати декілька різних парадигм. Серед таких можна виокремити наступні: процедурна парадигма; функціональна парадигма; об'єктноорієнтована парадигма. Дана мова програмування вважається розширеною версією мови низького рівня C. Додавання нових парадигм, в особливості об'єктноорієнтованої, надало ще більше можливостей для створення програм.

Серед переваг мови програмування C++ перед іншими мовами можна виокремити наступні:

- Багатопарадигмова мова – про це вже було сказано вище, що C++ підтримує декілька різних парадигм.
- Управління пам'яттю – C++ дозволяє працювати з пам'яттю статично, динамічно, автоматично та поточно. Зі статичною пам'яттю все більш менш зрозуміло: об'єкт або змінна були створені і їх час життя обмежується роботою програми. Якщо ж казати про динамічну пам'ять, то об'єкти та змінні створюються лиш тоді, коли викликані відповідні функції, і тривалість їх життя залежить саме від викликів цих функцій.
- Нульова абстракція – це такий принцип конструкції C++, який каже наступне: «Ви не сплачуєте за те, що не використовуєте. Те, що ви використовуєте, настільки ж ефективне, як і те, що ви могли б написати вручну»[4]. В якості прикладу такої абстракції можна привести C-масиви. Їх можна замінити на масив з STL, який використовується в C++, і при цьому не зазнати втрат в часі або використанні пам'яті.

- Перевантаження функцій – корисна перевага, яка надає можливість створювати функції з однаковою назвою, але з різною кількістю або різними типами параметрів. Таким чином, під час компіляції буде викликано саме ту функцію, яка буде знайдена для вказаних параметрів [5-6].

Це лише деякі з переваг C++ над іншими мовами.

Набір графічних бібліотек Qt – швидкий, зручний, гнучкий та кросплатформенний інструментарій. Було прийнято рішення використовувати саме Qt через те, що він містить багато різних бібліотек та модулів, крім GUI. Наприклад, обробка векторних зображень, можливість малювати різноманітні фігури на віджетах, тощо. Також в Qt є додаткові функції, такі як сигнали та слоти. Це реалізовано наступним чином: кожен компонент містить свої певні сигнали, які посилаються у випадку, якщо були виконані певні дії з цим компонентом. Далі створюється новий або перевизначається старий слот, в якому описуються дії, які повинні бути виконані, коли цей слот отримає сигнал від даного компоненту. Далі сигнали і слоти поєднуються між собою командою connect, і таким чином налаштовується робота компонента [7]. На рисунках 2.1 та 2.2 наведено приклад такої реалізації:

```
connect(ui->memoryTable, &QTableWidget::itemClicked, this, &Window::handleItemClicked);
```

Рисунок 2.1 – використання команди connect

```
void Window::handleItemClicked(QTableWidgetItem *item)
{
    int row = item->row();
    int column = item->column();
    if(!secFunc->getIsBigMemory()){
        if(row > 0)
            row = 0;
        secFunc->setPc(column);
    } else {
        if(row == 0){
            secFunc->setPc(column);
        } else secFunc->setPc(row * 8 + column);
    }
    ui->memoryTable->setCurrentCell(row, column);
}
```

Рисунок 2.2 – код реалізації слоту handleItemClicked

У цьому прикладі при натисканні на елемент таблиці посилається відповідний сигнал ***itemClicked***, який поєднаний із слотом ***handleItemClicked***. Таким чином, при натисканні на айтем в таблиці буде викликатися цей слот і виконуватися дії, що там описані.

До переваг Qt відносяться:

- Багатоплатформеність: дозволяє створювати додатки, що будуть працювати на різних системах. Емулятор буде працювати на системах Windows XP, Windows 7, Windows 8 та Windows 10, тому дана можливість буде актуальною. В подальшому, можливо, програма буде підтримуватися і на системах Linux та MacOS.
- Широкий функціонал: як було сказано вище, Qt містить дуже великий набір бібліотек та модулів. Завдяки цьому можна створювати складні та повноцінні додатки, використовуючи однаковий інструментарій.
- Легкість використання: Qt має свій фреймворк Qt Creator. Це середовище розробки надає зручний редактор коду, де підсвічується синтаксис, автодоповнюється код та виконується динамічна перевірка. Тобто, якщо

в коді було допущено помилку, Qt Creator одразу сповістить про це, підкресливши помилку червоною лінією [8].

Це лише частина з того, чим Qt кращий за інші інструменти та графічні бібліотеки. Можливо в подальшому програма буде перероблена з використанням якихось інших бібліотек, якщо вони виявляться більш швидкодіючими і менш затратними в плані використання пам'яті.

Отже, для розробки програмної моделі було обрано саме мову програмування C++ з використанням Qt, тому що з їх допомогою можна вирішити всі необхідні функціональні задачі, що виникають під час розробки програми.

1.2. Опис принципів роботи моделі процесора DPM-08

Перш ніж описувати принципи роботи моделі, варто визначитися, які компоненти взагалі має даний процесор і як вони поєднані. Для цього на рисунку 2.3 приведене зображення реальної моделі:

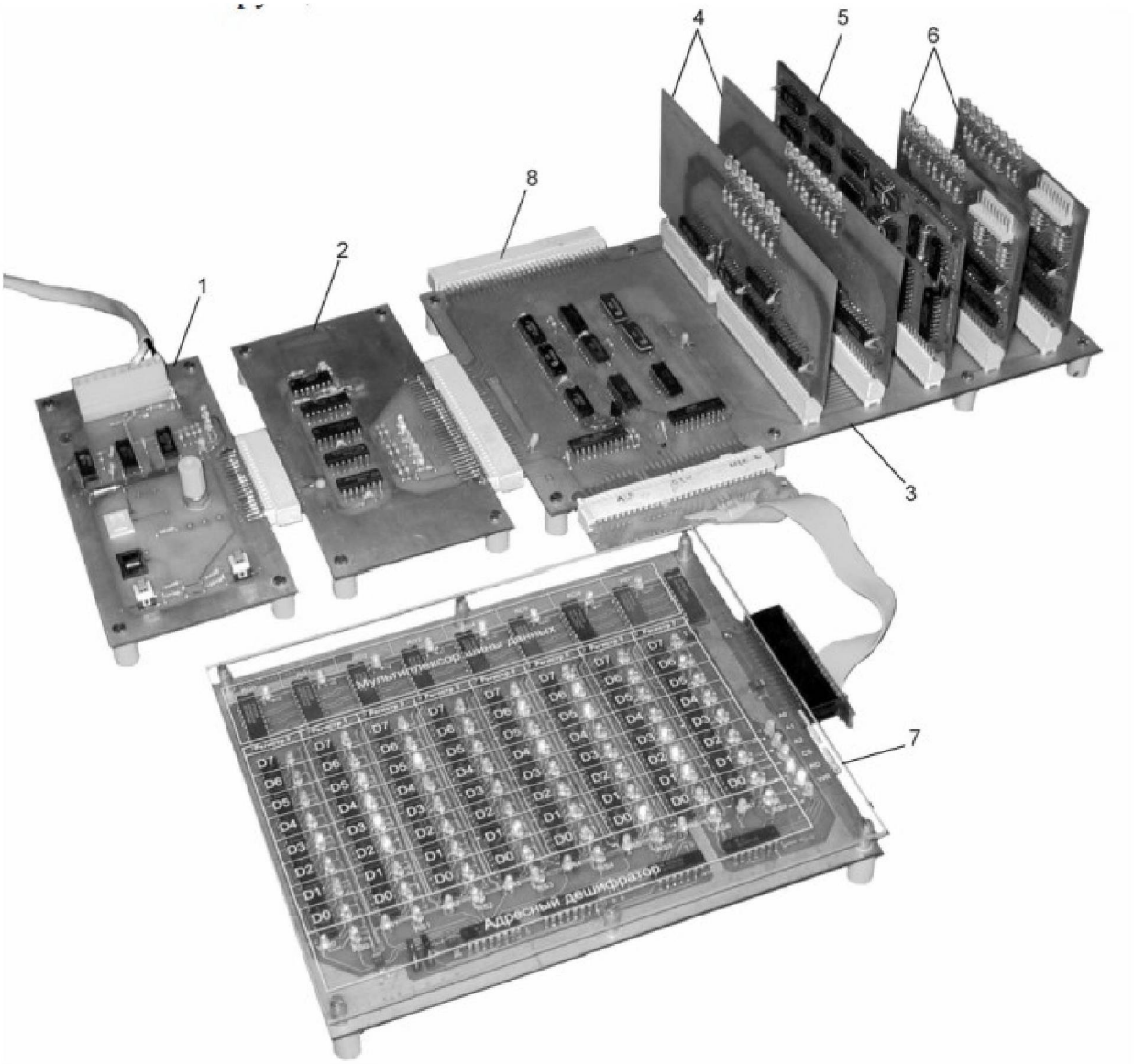

Рисунок 2.3 – Зовнішній вигляд моделі цифрового процесору [9]

Далі наведено невеликий опис компонентів, які входять до моделі, з урахуванням рисунку 2.3 та їх нумерації:

1. **Генератор сигналів управління**: генерує сигнали, які будуть поступати до генератору сигналів машинних циклів. Складається з двох частин: генератор тактових імпульсів та генератор сигналу скидання. Генератор тактових імпульсів може генерувати як 1 такт при натисканні кнопки, так і більше.

Окрім цього, при наявності автогенератора, може генерувати сигнали доти, доки кнопку не буде натиснуто знову. Якщо ж згенеровано сигнал скидання, то всі регістри очищуються, генератор сигналів машинного циклу не отримує ніякого сигналу, відповідно нічого і не генерує.

2. **Генератор сигналів машинного циклу**: Приймає сигнали, що генерує генератор сигналів управління, та перетворює ці сигнали на сигнали машинного циклу. В залежності від тактової частоти, при натисканні на кнопку може виконуватися 1 такт (покрокове виконання), 8 тактів (покомандне виконання) або отримувати велику кількість тактів щосекунди і працювати в безперервному режимі.
3. **Основний блок моделі процесора**: Містить в собі порти для підключення регістрів A,B,C,D, а також АЛП. Окрім цього, є роз'єми для підключення різних інтерфейсів. Також перед портами встановлено регістр команд і дешифратор. Поєднані всі ці елементи 8-розрядними шинами. До входів регістрів та АЛП підключено 3 шини: шина даних, шина керування та шина команд. Шина команд також підключена до інтерфейсу ПК та до дешифратору. Шина керування, в свою чергу, підключена ще й до дешифратора, інтерфейсу ПК та лічильника команд. Шина даних під'єднана до лічильника команд, регістра команд та роз'єму ОЗП, до якого під'єднано блок пам'яті.
4. **Регістри A та B**: Звичайні регістри, які використовуються для тимчасового зберігання даних. Мають світлодіоди, щоб відстежувати, які саме дані на поточний момент там знаходяться. Якщо якісь біти встановлені в одиницю, то відповідні їм світлодіоди будуть горіти. Окрім цього, регістр A має більше функцій, ніж інші регістри. Сюди можна віднести побітове

зміщення даних, наприклад. Також ці два регістри використовуються для виконання різних арифметичних та логічних операцій.

5. **Арифметико-логічний пристрій (АЛП)**: пристрій, що зберігає в собі декілька мікросхем різноманітних арифметичних та логічних операцій. Таким чином, до його складу входять наступні операції: очищення регістра, логічне додавання значень з регістрів A та B (OR), логічне перемноження значень з тих же регістрів (AND), виключне АБО (XOR), інверсія регістра A, зміщення даних в регістрі A побітово вліво/вправо, арифметичне додавання значень з регістрів A та B. Варто сказати про те, що АЛП працює в асинхронному режимі. Це означає, що як тільки змінюються стани регістрів A та/або B, АЛП змінює стан всіх внутрішніх регістрів незалежно від того, чи закінчився машинний цикл.
6. **Регістри C та D**: Такі ж регістри, як A та B, за виключенням того, що ці регістри лише можуть зберігати дані і не приймають участь в арифметичних та логічних операціях. До регістру D може бути під'єднана цифрова клавіатура, щоб надати можливість користувачеві вводити якісь значення в цей регістр.
7. **Модуль пам'яті програм**: Власне пам'ять, яка може бути під'єднана до роз'єму ОПЗ. Зберігає команди, які представлені у вигляді чисел в шістнадцятковому вигляді. В моделі пам'яті, представленій на рисунку 2.3, всього 8 байт і кожен байт розділений на біти, таким чином користувач може встановити відповідні біти в одиницю. Також можна підключити дещо інший варіант блоку пам'яті на 256 байт і завантажувати туди якусь програму з комп'ютеру.

8. **Роз'єм підключення інтерфейсного модулю для з'єднання з ПК**: спеціальний інтерфейс, який дозволяє контролювати та формувати всі основні сигнали моделі за допомогою персонального комп'ютера [9].

Таким чином, було розглянуто будову процесора і коротко описано його компоненти. Спираючись на це при створенні програмної моделі процесора буде реалізовано окремо модуль пам'яті та окремо модуль самого процесора, який буде містити в собі всі потрібні регістри, АЛП, та генератор сигналів машинного циклу, а також шину даних, шину керування та шину адреси. Також варто показати, як ці два модулі поєднані між собою.

Для створення програмної моделі потрібно проаналізувати роботу моделі процесора на програмному рівні та вплив окремих бітів командного слова на його функціонування. Командне слово в цій моделі процесора становить 8 біт, тобто 1 байт. В таблиці 2.1 наведено призначення кожного біта одного командного слова:

Таблиця 2.1

Призначення біт командного слова

Біт	Призначення
CM7	Тип команди
CM6	
CM5	Адреса джерела даних
CM4	
CM3	
CM2	
CM1	Адреса регістра-отримувача даних
CM0	

Два старші біти командного слова визначають тип команди, яка була зчитана з пам'яті. Всього існує 4 типи команд:

- 00 – команди нульового типу. Це однобайтні команди, які в якості адреси джерела можуть використовувати як регістри A,B,C,D, так і арифметичні або логічні операції з АЛП, в залежності від того, що записано в біти СМ5 – СМ2. Тоді два наймолодші біти відповідають за адресу регістра отримувача даних. Стосовно їх комбінацій пізніше.
- 01 – команди першого типу (СМ7 – 0, СМ6 – 1). Команди першого типу є двобайтними командами, які використовуються для запису значення, яке вказане в другому байті, до одного з регістрів-отримувачів (A,B,C,D). При цьому біти СМ5 – СМ2 не використовуються.
- 10 – команди другого типу (СМ7 – 1, СМ6 – 0). До команд другого типу відноситься лише одна команда безумовного переходу. Вона є двобайтною: перший байт – це саме інструкція, другий байт – адреса, на яку потрібно перейти. Так як це саме безумовний перехід, він відбудеться в будь-якому випадку, незалежно від того, які стани регістрів або інші деталі. Біти СМ5 – СМ0 в цьому випадку не використовуються.
- 11 – команди третього типу. Це також є двобайтні команди. На відміну від минулої комбінації, до третього типу відносяться команди умовного переходу. Таким чином, коли в програмі досягнуто команди третього типу, перевіряється певна умова в залежності від бітів СМ1 – СМ0. Якщо вона виконується, то програма переходить на адресу, яка була вказана в другому байті. В іншому випадку, програма ігнорує цей перехід і переміщується на наступну команду в пам'яті.

Далі буде розглянуто біти СМ5 – СМ2, які формують адресу джерела інформації. Тут варто зазначити, що біт СМ5 вказує на те, чи будуть дані

обиратися з одного з регістрів оперативного призначення (A – D), або ж з АЛП. Таким чином, далі наведено комбінації, коли біт CM5 скинуто в 0:

- 000 – джерелом даних буде регістр A;
- 001 – джерелом даних буде регістр B;
- 010 – джерелом даних буде регістр C;
- 011 – джерелом даних буде регістр D;

У випадку, коли CM5 встановлено в 1, перелік комбінацій буде наведено в таблиці 2.2.

Таблиця 2.2.

Перелік команд нульового типу, коли біт CM5 = 1

Комбінація команди	Функціональне призначення
000	Команда очищення регістра – запис в регістр числа 0
001	Команда запису інвертованого значення регістра A
010	Команда запису результату логічної суми вмісту регістрів A та B
011	Команда запису результату логічного множення вмісту регістрів A та B
100	Команда запису результату виключного АБО вмісту регістрів A та B
101	Команда запису зміщеного побітово вліво значення регістра A
110	Команда запису зміщеного побітово вправо значення регістра A
111	Команда запису результату арифметичного додавання вмісту регістрів A та B

Варто зазначити, що команди, вказані в таблиці 2.2 використовують в якості джерела саме АЛП. Якщо казати стосовно того, куди саме записуються результати операцій або вміст якогось з регістрів, то треба звернути увагу на біти CM1 та CM0. Далі буде розглянуто саме ці 2 біти.

Адреса регістра-отримувача даних визначається двома наймолодшими бітами CM1 та CM0. Але ця їх функціональність працює лише для команд нульового та першого типу. В командах другого типу, як було зазначено вище, ці біти не використовуються, а в командах третього типу вони відповідають не за адресу регістра-отримувача, а за умови для виконання переходу. Таким чином, в таблиці 2.3 зазначено комбінації бітів та відповідні їм функції для різних типів команд:

Таблиця 2.3.

Призначення біт CM1 та CM0 в різних типах команд

CM1 та CM0	Команди нульового та першого типу	Команди третього типу
00	Регістр-отримувач A	Якщо в регістрі A міститься значення 0
01	Регістр-отримувач B	Якщо в регістрі A міститься значення, відмінне від 0
10	Регістр-отримувач C	Якщо під час виконання команди арифметичного додавання вмісту регістрів A та B було встановлено біт переносу
11	Регістр-отримувач D	Якщо біт переносу дорівнює нулю

Таким чином, переглянувши третій стовпець таблиці 2.3 можна побачити всі можливі умови для виконання умовного переходу. Якщо умова виконується – робиться перехід на адресу, інакше – перехід на наступну команду.

Структура командного слова була розглянута з тою метою, щоб було зрозуміло, як процесор розуміє, що саме потрібно йому робити. В програмній моделі це буде реалізовано в регістрі інструкцій, щоб наочно за допомогою сигналів показати, звідки дані беруться і куди записуються.

Вище в цьому підрозділі я застосовував таке поняття як машинний цикл, тому є необхідність розглянути його більш докладно. Для цього на рисунку 2.4 наведено часову діаграму тривалості машинного циклу:

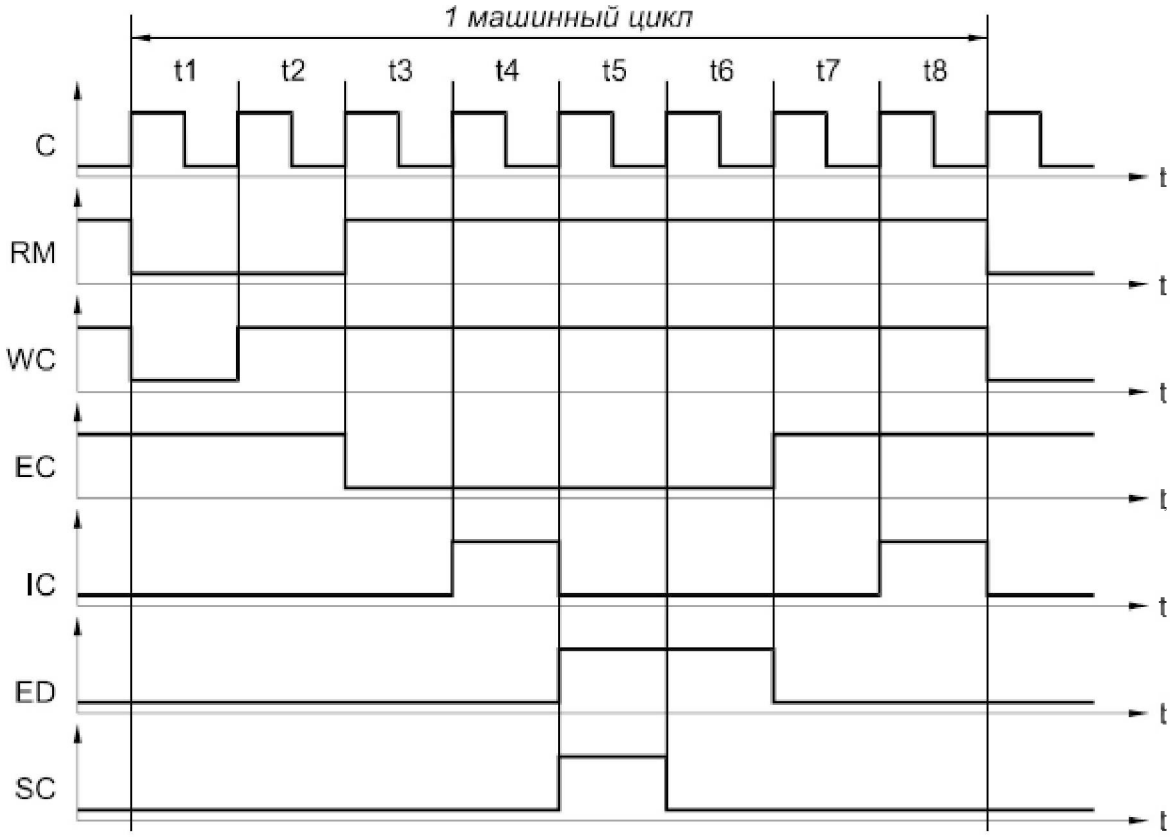

Рисунок 2.4 – Послідовність сигналів машинного циклу

Як видно з рисунку 2.4, машинний цикл складається з 8 періодів тактової частоти. Починається цей процес з встановлення низького логічного рівня сигналу RM (Read Memory) та сигналу WC (Write Command). Перший сигнал дозволяє зчитати дані з пам’яті та виставити їх на шину даних. Триває він два періоди, щоб дані повністю встигли зчитатися і не було спотворень. Далі, після затримки в один період на сигналі WC формується перехід з низького логічного рівня до високого, що призводить до запису числа з шини даних до регістру команд. По завершенню

другого періоду сигнал RM набуває високого логічного рівня і звільняє шину даних. В цей же час на третьому періоді машинного циклу формується низький логічний рівень сигналу EC (Enable Command). Даний сигнал дозволяє виконувати перезапис даних із джерела до регістра-отримувача. На 4 етапі циклу можна помітити формування високого рівня сигналу IC (Increment Counter). Даний сигнал спрацьовує лише в тому випадку, коли команда двобайтна, адже дані потрібно зчитувати з наступної лунки пам'яті. В іншому випадку цей сигнал замасковано протягом всього часу, доки на EC встановлено низький логічний рівень. На 5 етапі машинного циклу формується високий логічний рівень сигналів ED (Enable Data) та SC (Strobe Command). Перший сигнал дозволяє виставити дані на шину даних, а другий дозволяє роботу дешифратора сигналів запису в регістри-отримувачі. В кінці 5 етапу можна спостерігати перехід сигналу SC в низький логічний рівень. Саме в цей час виконується запис даних до потрібного регістра-отримувача. По завершенню 6 етапу сигнал ED спадає до нульового рівня, тим самим звільняє шину даних. Окрім цього, сигнал EC переходить до високого логічного рівня, тим самим блокуючи можливість перезапису даних. Сьомий етап є проміжним, там нічого не відбувається. Завершується машинний цикл після 8 періоду. В цей час формується позитивний імпульс сигналу IC, який вже не маскується і призводить до інкременту лічильника команд. Таким чином, по завершенню машинного циклу на шину адреси встановлюється адреса, за якою розміщується перший байт наступної команди [9].

Машинний цикл був описаний тому, що це основа роботи процесора. Звідси слідує, що в програмній моделі увесь цей процес повинен бути також реалізований. Тому буде розроблено генератор, який буде містити всі сигнали, що описані в машинному циклі. Більше того, користувач матиме можливість спостерігати, які сигнали на кожному кроці активні та на що вони впливають.

Підсумовуючи все, що було описано в підрозділі 2.2, можна сказати, що є уявлення стосовно апаратних та програмних можливостей реальної моделі. Усі ці можливості необхідно буде реалізувати в програмній моделі. Більше того, емулятор повинен бути наочним, зрозумілим і простим у використанні.

1.3.Розробка програмної моделі

Після аналізу апаратних та програмних можливостей реальної моделі, розробка починається зі створення дизайну програми. На рисунку 2.5 показано зовнішній вигляд програмної моделі:

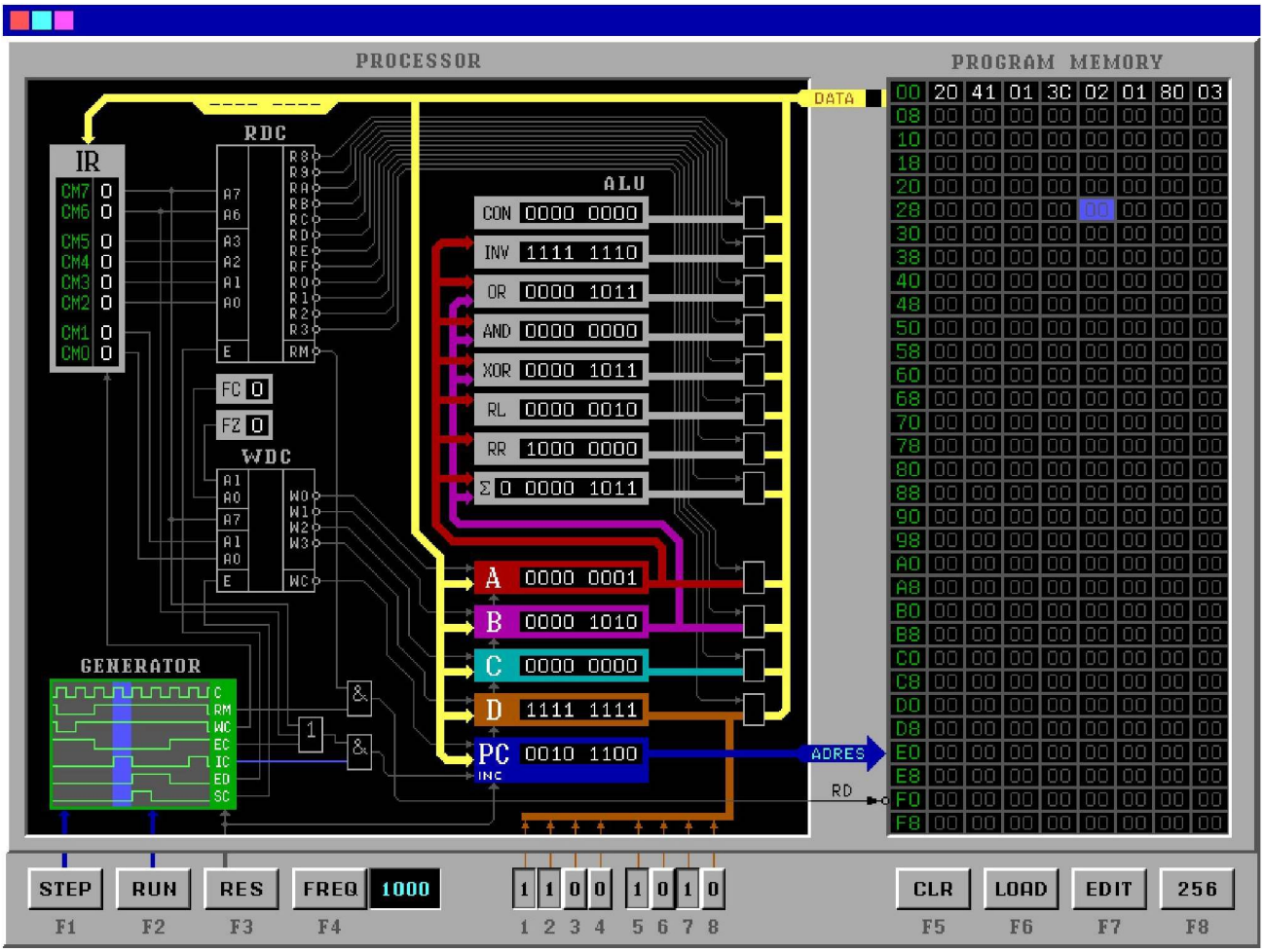

Рисунок 2.5 – Розроблений дизайн програмної моделі

Програмна модель умовно складається з двох частин: вікно дисплея, де відображені всі регістри, генератор та провідники, що з'єднують між собою компоненти, а також шина даних та шина адреси; вікно модуля пам'яті, де будуть зберігатися завантажені програми. Варто звернути увагу на те, що кнопки розміщені в нижній частині програми таким чином, що вони вказують, до якого модуля відносяться. Окрім цього можна побачити, що модулі зв'язані між собою шиною даних та шиною адреси. Також зображено під'єднаний сигнал RD (Read Memory), щоб було наочно видно, що дані зчитуються з модуля пам'яті. Після розробки дизайну, маючи перед очима картинку, розпочинається частина програмування.

При створенні нового проекту в Qt Creator автоматично генерується найпростіший додаток, який представляє собою порожнє вікно. Центральним елементом цього вікна є об'єкт класу QWidget, який формує основний віджет з розміром, пропорційним розміру вікна додатку. Перейшовши до файлу з розширенням ***.ui*** можна потрапити до редактору графічного інтерфейсу, де надається можливість редагувати головне вікно, додавати туди різні компоненти, налаштовувати більшість атрибутів кожного компонента тощо. Представлення цього файлу зображену на рисунку 2.6:

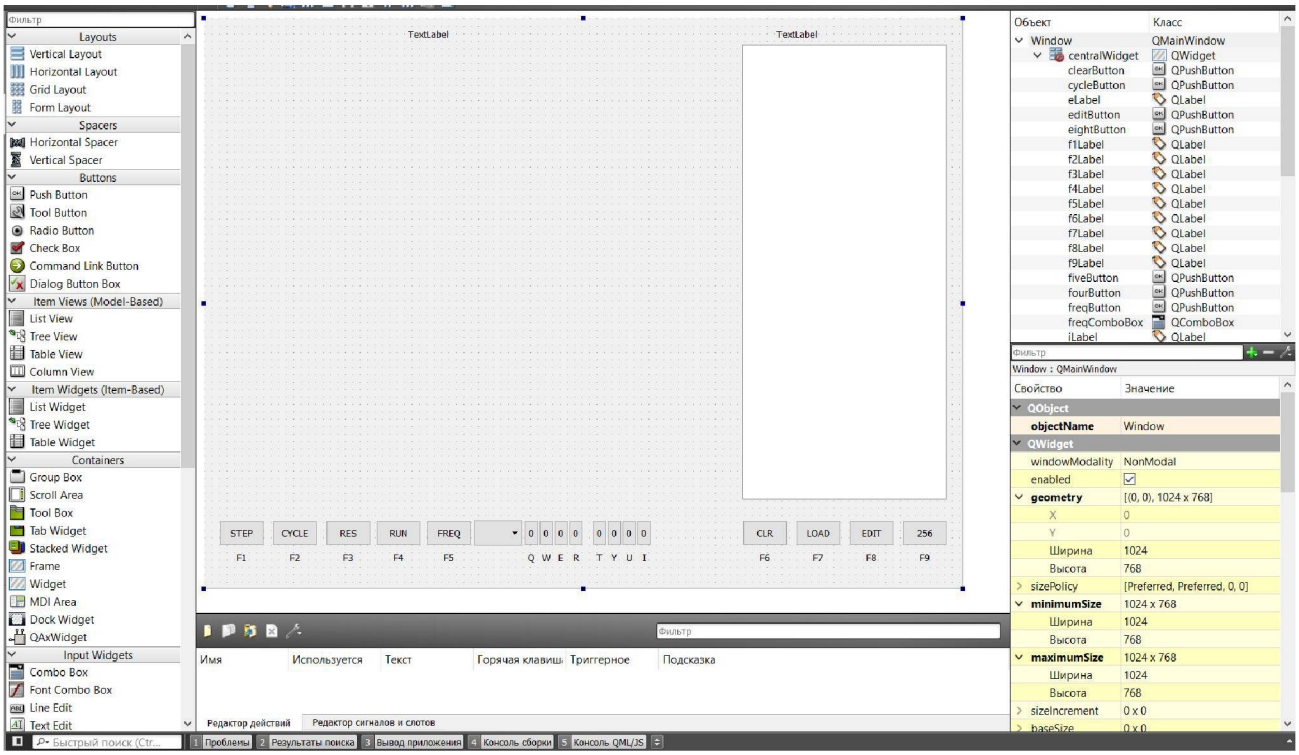

Рисунок 2.6 – Редактор GUI

Окрім цього файлу, Qt Creator завжди генерує файли з розширенням ***.h*** та ***.cpp***. Перший файл – це хеддер. В ньому містяться всі виклики бібліотек, оголошені класи, змінні, прототипи функцій, сигналів та слотів. Другий файл містить в собі вже сам код реалізації функцій, роботи з класами, об'єктами, тощо. Окрім цього, в наявності також файл ***main.cpp***, в якому описано власне функцію ***main***, без якої програма взагалі не буде працювати. Вміст цієї функції показано на рисунку 2.7:

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Window w;
    w.show();

    return a.exec();
}
```

Рисунок 2.7 – код функції main

Якщо коротко про те, що тут описано, то спочатку створюється об'єкт типу ***QApplication***. ***QApplication*** – це клас, який керує основними функціями додатку Qt, такими як ініціалізація, обробка подій, стиль інтерфейсу тощо. Далі створюється об'єкт типу Window. Це є той клас, який представляє вікно додатку. Саме в ньому описується весь функціонал для того, щоб додаток працював. Метод ***w.show()*** дозволяє відобразити вікно на екрані. ***a.exec()*** – це метод класу ***QApplication***, що запускає основний цикл подій додатку. Може приймати події як від джерел вводу, так і з інших джерел. По завершенню програми повертає код цієї події.

Загалом проект містить три допоміжні класи і один основний – саме клас Window, який згадувався вище. В основному класі описано поведінку кожного елементу графічного інтерфейсу і кожного об'єкта. Також назначено певні стилі для кожного елемента. Один з допоміжних класів, що має назву ***StyleHelper***, створений саме для того, щоб задавати потрібні стилі для елементів. Потреба в ньому виникла через те, що дизайн кожного компоненту задається за допомогою атрибуту StyleSheet. Окрім цього, використовується механізм QSS (Qt Style Sheets), який є дуже схожим на стилі HTML та CSS. Такий функціонал дозволяє задавати стилі для всього класу, для певних об'єктів класу, для різних станів

компоненту (як то наведення курсору на кнопку або натискання) тощо [10]. Таким чином, щоб не займати багато місця і не ускладнювати основний код, було прийнято рішення винести в окремий клас опис всіх необхідних стилів. Тобто в цьому класі реалізовані функції, які повертають рядок типу ***QString***, в якому записані всі необхідні атрибути для стилю. Приклад коду наведений на рисунку 2.8:

```
QString StyleHelper::getCentralWidgetStyle()
{
    return "QWidget{"
           "    border-top: 4px solid #FFFFFF;"
           "    border-left: 4px solid #FFFFFF;"
           "    border-bottom: 4px solid #555555;"
           "    border-right: 4px solid #555555;"
           "}";
}
```

Рисунок 2.8 – вміст функції getCentralWidgetStyle

Наступним допоміжним класом є такий незвичайний клас як MyDelegate. Він наслідує клас QStyledItemDelegate і потрібен для того, щоб задавати певний стиль для комірок таблиці. Так як в програмі буде реалізовано можливість редагувати пам’ять і змінювати дані, для цього необхідно використовувати делегати. Загалом це певні класи, що наслідуються від QAbstractItemDelegate та надають можливість відображати та редагувати елементи даних з моделі у вигляді ***QListView***, ***QTableView*** або ***QTreeView*** [11]. Таким чином, в цьому класі перевизначено методи ***createEditor, setEditorData,*** та ***setModelData***. Перший метод викликається одразу, як тільки таблиця переводить свої комірки в режим редагування. В середині методу лише описана зміна стилю комірки, а також поєднано сигнал ***editingFinished*** класу ***QLineEdit*** із створеним власноруч слотом ***commitAndCloseEditor***. Якщо простіше, то під час завершення редагування

комірки надсилається сигнал, і в цей час викликається користувацький слот. Другий метод працює з даними, що були введені в комірку. Так як в пам'яті можуть зберігатися лише числа в шістнадцятковому форматі не більше значення FF, то є необхідність обмежити користувача, щоб він не вводив некоректні дані. Це можливо, якщо використати маску, яка буде обмежувати введення даних, щоб було не більше 2 символів і лише ті, які використовуються в шістнадцятковій системі числення (1..9, A..F). Літери можна вводити як великі, так і маленькі. В будь-якому випадку вони будуть приведені до одного розміру. Останній метод змінює дані в моделі, щоб вони збереглися в комірці після завершення роботи з нею.

Третім допоміжним класом є ***SecondaryFunctions.*** Цей клас було створено для того, щоб винести з основного класу частину коду та запобігти нагромадження файлу. Він містить багато різних функцій, які потрібні для того, щоб замінити ними повторювані блоки коду. Сюди ж відноситься велика кількість геттерів та сеттерів, які необхідні для того, щоб отримати значення закритої змінної або ж змінити її стан. Для зручності було прийнято рішення винести певну кількість змінних в цей клас з основного, щоб можна було працювати з ними напряму. Так як більшість змінних використовується лише один-два рази в основному класі, більш доцільно використати геттери та сеттери в допоміжному класі, аніж створювати їх в основному та більше викликати цих функцій в допоміжному класі. Тут буде зберігатися список всіх кольорів, які використовуються для дизайну програми та для відображення різноманітних сигналів. Також в цьому класі будуть визначатися всі змінні, що відповідають за текст в регістрах, а також різноманітні флаги, такі як ***fc, fz, cycleClicked*** та інші, які необхідні для повноцінної та правильної роботи емулятора.

При розробці графічного інтерфейсу було прийнято рішення частину графічних об’єктів малювати не шляхом малювання по пікселях, а створити векторні зображення та за допомогою класу ***QSvgRenderer*** рендерити їх. Це було зроблено з тією метою, щоб дещо зменшити витрати пам’яті, адже при векторній графіці зберігається лише математична інформація про зображення [12]. Окрім цього, спочатку було заплановано зробити можливість масштабувати вікно, але з часом було прийнято рішення обмежити вікно заданим розміром, а саме 1024x768. Також під час розробки зовнішній вигляд програми був дещо змінений на відміну від того, що було показано на рисунку 2.5. Тому, кінцевий продукт наведено на рисунку 2.9:

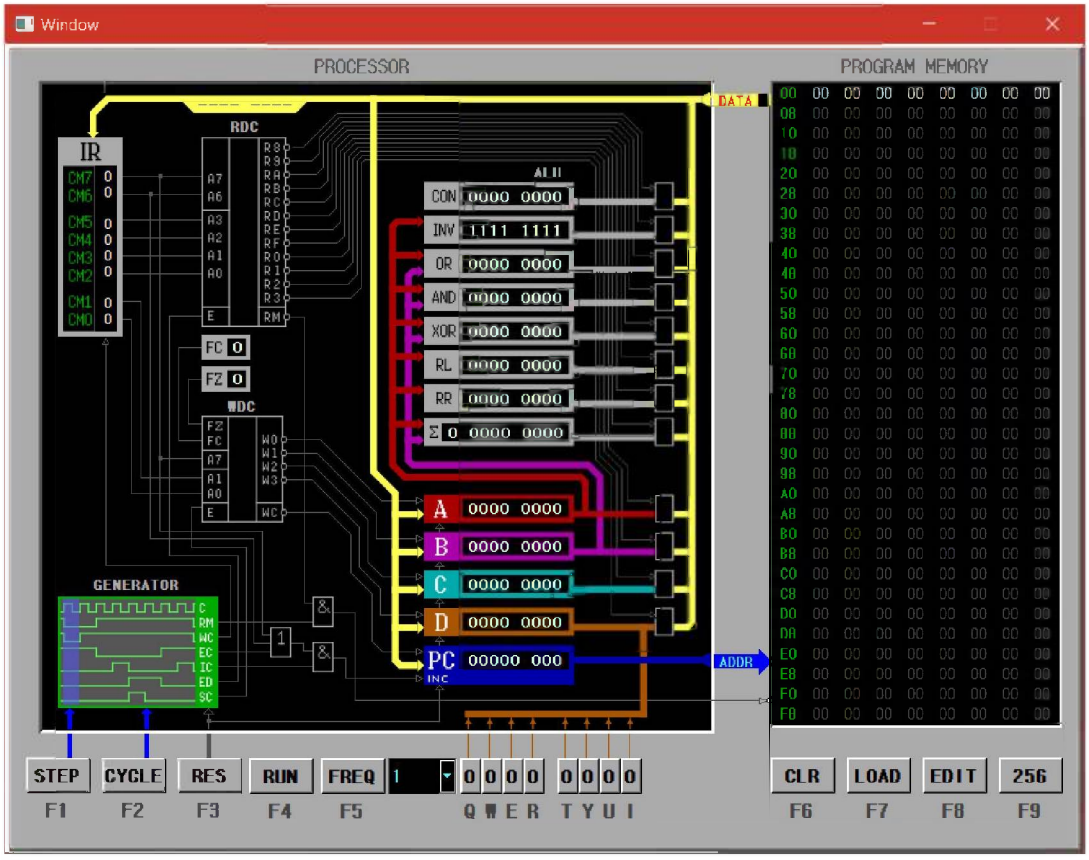

Рисунок 2.9 – Зовнішній вигляд готової програмної моделі

Під час програмування графічного інтерфейсу одним з найскладніших частин було відтворити всі необхідні сигнали (лінії сірого кольору). Необхідно було розрахувати координати кожної точки, за якими потім будуть малюватися лінії та ламані. Загалом все малювання на віджеті відбувається за допомогою перевизначеного методу ***paintEvent***. Це основа для малювання, без якої нічого не буде відображатися. Використовується об’єкт класу ***QPainter***, який є чимось типу олівця. Без цього також неможливо щось малювати. ***QPainter*** у Qt надає оптимізовані функції для малювання графічних елементів у більшості програм. Він вміє малювати різноманітні об'єкти - від простих ліній до складних форм, таких як сектори та хорди. Також він підтримує вирівняний текст та піктограми. Зазвичай ***QPainter*** працює в "природній" системі координат, але може виконувати перегляд та трансформацію світу. Важливо зазначити, що ***QPainter*** може співпрацювати з будь-яким об'єктом, який успадковує клас ***QPaintDevice*** [13].

Після того, як дизайн програмної моделі вже запрограмовано, можна переходити до реалізації програмних можливостей. Для цього необхідно для кожної кнопки, що розташовані внизу програми, прописати слоти для сигналу натискання на них. Далі буде коротко описано кожну кнопку та що було реалізовано:

- Кнопка Step: кнопка покрокового виконання машинного циклу. В слоті спочатку перевіряється флаг, який відповідає за те, чи натиснута кнопка, чи ні. Потрібно це, перш за все, для того, щоб змінювати дизайн кнопки при натисканні. Окрім цього, на деяких етапах машинного циклу при натисканні на кнопку виконуються одні дії, а після відпускання інші. Коли кнопка натиснута, виконується блок коду, в якому зберігається лічильник, який визначає, на якому етапі машинного циклу зараз знаходиться програма. В залежності від цього існує 8 різних послідовностей дій. Опис машинного

циклу був наведений в розділі 2.2, тому саме спираючись на нього було реалізовано ті ж самі дії в програмній моделі. Зміна кольорів потрібних сигналів, аналіз команди, зміна стану регістрів, зміна обраної комірки пам'яті – все це описано в цьому слоті, який викликається при кожному натисканні кнопки Step.

- Кнопка Cycle: кнопка покомандного виконання програми. Тут вже немає розподілу на 8 етапів, проте частина дій схожа на ту, що була описана для кнопки Step. Відмінність полягає в тому, що тут немає необхідності показувати зміну сигналів. Лише кінцеві стани регістрів після виконання команди, а також значення в АЛП і в лічильнику команд, адже при команді переходу стан лічильника команд буде змінено. Якщо команда є двобайтною, то виділяються обидва байти, щоб було зрозуміло, що це саме так.
- Кнопка Run: кнопка запуску емулятора в реальному часі. Після того, як ця кнопка була натиснута, починається послідовне виконання всіх команд в пам'яті. При цьому ми спостерігаємо лише за зміною стану регістрів A,B,C,D. На цьому етапі нам не потрібно знати, що там в АЛП в якийсь проміжний час, адже важливо лише відстежити результат. Ця кнопка може бути переведена в неактивний стан не тільки другим натисканням на неї, а також натисканням на кнопки Reset або 256, які буде описано далі. Стосовно реалізації цієї кнопки, було використано внутрішній QTimer. Так як далі буде описано кнопку, яка допомагає змінювати частоту, з якою працює процесор, а як наслідок, змінювати швидкість виконання, то в залежності від частоти інтервал таймера буде різним. Коли таймер досягає стану таймауту, викликається слот ***runWorkOfProcessor***, який є користувацьким. В ньому

описано майже ті ж дії, що і в кнопці Cycle, за винятком того, що перемальовуються лише регістри.

- Кнопка Reset: кнопка скидання емулятора в початковий стан. Коли дана кнопка натиснута, шина даних закрита, дані в ней не зберігаються. Всі регістри скидаються в нульові значення, АЛП також змінює значення в залежності від регістрів A та B. В блоці пам'яті виділяється перша комірка, з якої буде зчитана команда. Стан генератора сигналів машинних циклів невідомий, доки натиснута кнопка. Всі сигнали вимкнені і регістри неактивні. Після того, як кнопку було відпущено, на генераторі сигналів машинних циклів з'являється маркер, що вказує на те, на якому етапі машинного циклу знаходиться програма. Також виділено сигнали RM, WC та EC (стосовно призначення сигналів дивитися пункт 2.2), шина даних відкрита для запису даних з модуля пам'яті. Після цього можна заново виконувати завантажену або записану в пам'яті програму.
- Кнопка Freq: кнопка зміни частоти роботи процесора. Поряд з кнопкою є віконце, в якому відображається частота в Герцах. При натисканні на кнопку, з віконця відкривається випадаючий список, в якому є вибір частот. В подальшому цей компонент буде перероблений таким чином, щоб окрім вибору можна було записувати значення частоти самостійно. Змінювати частоту можна прямо під час виконання програми в реальному часі. Після зміни буде збільшуватися або зменшуватися швидкість виконання.
- Кнопка Clr: кнопка очищення пам'яті. Загалом назва демонструє, що ця кнопка робить. При натисканні на неї всі комірки пам'яті заповнюються нулями, інакше кажучи пам'ять очищується.

- Кнопка Load: кнопка завантаження програм до пам’яті. При натисканні на цю кнопку користувачеві відкривається провідник для вибору файлу, який необхідно завантажити. Емулятор здатний завантажувати файли з розширенням .BIN та .HEX. Також можна завантажувати .OBJ, але в такому випадку відповідальність за коректність прочитаних даних лежить на плечах користувача.
- Кнопка Edit: кнопка редагування пам’яті. Якщо казати простіше, то дана кнопка переводить комірки таблиці в стан редагування, тим самим надає можливість користувачеві вводити дані. Цей процес був реалізований за допомогою користувацького делегата, про який було описано вище. Окрім цього, в циклі перебирається кожна комірка та переводиться в стан редагування за допомогою вбудованого методу ***openPersistentEditor***. Приклад реалізації наведено на рисунку 2.10:

```
for(int row = 0; row < ui->memoryTable->rowCount(); row++){
    for(int col = 0; col < ui->memoryTable->columnCount(); col++){
        item = ui->memoryTable->item(row, col);
        item->setFlags(item->flags() | Qt::ItemIsEditable);
        if(item){
            ui->memoryTable->openPersistentEditor(item);
        }
    }
    if(!secFunc->getIsBigMemory())
        break;
}
```

Рисунок 2.10 – використання методу openPersistentEditor

- Кнопка «256»: кнопка зміну розміру пам’яті. В реальній моделі для того, щоб змінити об’єм пам’яті, потрібно відключити один модуль і підключити інший. В програмній моделі це реалізовано шляхом натискання відповідної кнопки. Тобто, коли ця кнопка натиснута, в пам’яті стають доступними для роботи 256 байт. Коли кнопка відпущена, знову користувач отримує всього 8 байт в своєму розпорядженні.

Якщо уважно подивитись на рисунок 2.9, то можна помітити такі компоненти, як FC та FZ. Перший компонент – це флаг переповнення (Flag Carry). Він встановлюється в одиницю в тому випадку, коли в програмі виконується команда арифметичного додавання і при цьому результат перевищує 255 байт. В цьому випадку з'являється біт переповнення. В реальній моделі цього не відстежити, але в програмній моделі така можливість надана з тою метою, щоб студентам було простіше визначати, чому умовний перехід стався або не стався. Другий компонент – це флаг наявності нуля в регістрі A (Flag Zero). Доки в регістр A не записано жодного значення, він буде встановлений в одиницю. Як тільки туди записати хоча б одиницю, флаг одразу буде скинуто в нуль. Цей компонент за аналогією із FC відсутній в реальній моделі наочно.

Не варто залишати без уваги невеличкі 8 кнопок, що під'єднані до регістра D. Це є та сама клавіатура, яка наявна в реальній моделі. Кнопки ці мають лише два стани: 0 та 1. Коли кнопка натиснута, вона містить 1, коли відтиснута – 0. Коли з регістра D на шину даних надходить його вміст, шляхом побітового множення він зіставляється із даними на клавіатурі. Таким чином, якщо вся клавіатура буде встановлена в 0, то будь-яке значення з регістра D при зіставленні з клавіатурою буде давати 0.

Як видно з рисунка 2.9, під кожною кнопкою підписані гарячі клавіші для того, щоб керувати програмою лише за допомогою клавіатури. Для цього було перевизначено метод ***keyPressEvent***, який спрацьовує тоді, коли натискаються кнопки на фізичній клавіатурі. Таким чином, спочатку перевіряється, яка кнопка була натиснута. Далі, якщо для цієї кнопки є певні дії, а саме посилання сигналу однієї з кнопок програмної моделі, то вони виконуються. Інакше натискання цієї кнопки ігнорується. На рисунку 2.11 наведено приклад коду:

```
switch(event->key()){
case Qt::Key_F1:
    ui->stepButton->clicked();
    break;
case Qt::Key_F2:
    ui->cycleButton->clicked();
    break;
case Qt::Key_F3:
    ui->resButton->clicked();
    break;
case Qt::Key_F4:
    ui->runButton->clicked();
    break;
```

Рисунок 2.11 – частина коду реалізації методу keyPressEvent

Підводячи підсумки, можна сказати, що програмна модель була розроблена і доволі успішно. Описувати тут всю реалізацію потребує багато часу та об'єму і, скоріш за все, не має сенсу. Загалом було описано всі можливості програмної моделі та як вони реалізовані.

Висновки за розділом 2

Отже, в цьому розділі було детально розглянуто апаратні та програмні можливості реальної моделі процесора DPM-08. Було з'ясовано, яку будову має ця модель, як вона працює, було розглянуто структуру командного слова, а також машинний цикл. Спираючись на цей аналіз, було з'ясовано, що потрібно реалізувати в програмній моделі, щоб вона повністю імітувала роботу реальної моделі. Для розробки було обрано мову програмування C++ та набір інструментів Qt, що надає велику кількість бібліотек для створення додатку з графічним інтерфейсом. В програмній моделі було відтворено всі компоненти, які наявні в реальній моделі. Поєднуються ці компоненти за допомогою шин даних, адреси та шини управління. Програмна модель вийшла достатньо зручною і зрозумілою.

Допомогти у роботі з програмою має опис користувача, який наводиться в підрозділі 3.2. Даний емулятор, звичайно, має шляхи для вдосконалення та доопрацювання. Проте на сьогоднішній день навіть цього вже буде достатньо для того, щоб використовувати даний продукт для покращення навчального процесу. Наочна робота процесору, відображення всіх сигналів, позначення джерела та отримувача різними кольорами та інші можливості роблять цю програму ідентичною до реальної моделі, і навіть в деяких випадках зручнішою у використанні.

РОЗДІЛ 3. ВІДЛАГОДЖЕННЯ РОБОТИ ПРОГРАМИ ТА РОЗРОБКА ІНСТРУКЦІЇ ЩОДО КОРИСТУВАННЯ НЕЮ

2.1. Відлагодження програми

Важливим етапом при розробці програми є, звичайно, її відлагодження та тестування. Адже майже неможливо розробити програмний продукт, який буде одразу ідеально працювати. Тому є необхідність перевірити працездатність та стійкість до помилок, щоб програма не зламувалась від того, що користувач щось не те натиснув або не те ввів. Окрім цього, так як цей емулятор повинен працювати під різні системи Windows, потрібно перевірити, щоб він був по-перше, транспортованим, тобто міг запускатися без попередньої компіляції на іншому комп'ютері, а по-друге, щоб програма зберегла свій дизайн та програмні можливості.

Після завершення розробки програми під час першого запуску було помічено деякі недоліки під час роботи. На рисунку 3.1 представлено режим редагування пам'яті:

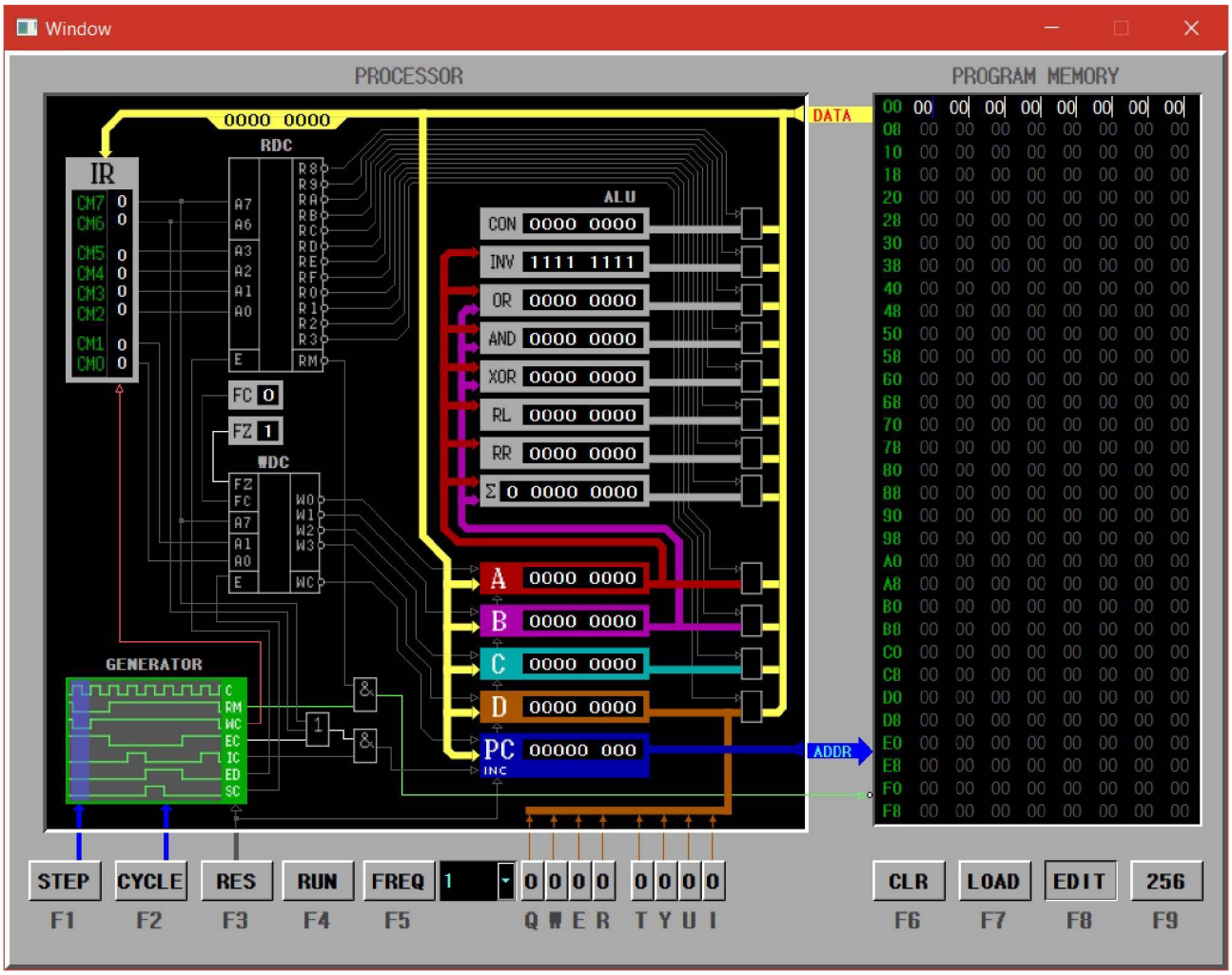

Рисунок 3.1 – Режим редагування вмісту пам’яті процесора

На рисунку помітно, що комірки, які були переведені в режим редагування, дещо відрізняються від інших. Це пов’язано не з кольором, а саме з тим, як зміщується текст: він знаходиться лівіше від інших комірок. Окрім цього, праворуч від тексту кожної комірки видно білу лінію, яку теж неможливо не помітити. До цього ще був дещо різний шрифт тексту, але це вже було виправлено на цьому етапі. Окрім цього, на рисунку 3.2 наведено формат комірки, коли було виконано перехід, а потім повернення на попередню адресу:

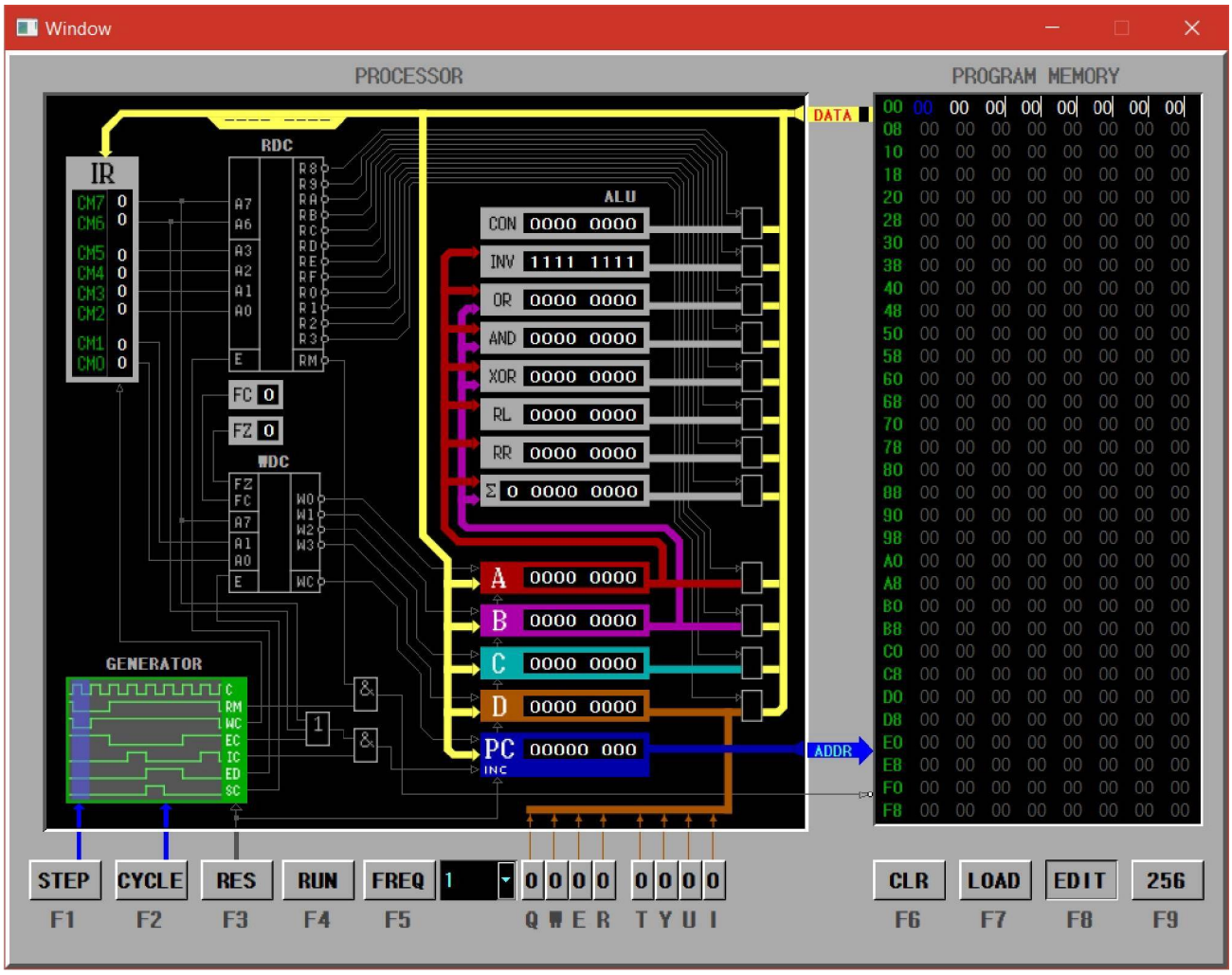

Рисунок 3.2 – Перехід на комірку назад

За допомогою рисунків складно це відобразити, але суть полягає в тому, що при поверненні на комірку, з якою користувач вже попрацював, текст комірки не виділяється. Тобто щоб записати нові дані, потрібно видалити старі, і лише після цього виконати введення команди. Це не зовсім зручно. Таким чином, доведеться виконувати додаткову роботу. Також була проблема, яка пов’язана з тим, що при переключені на будь-яке інше вікно, а потім поверненні до вікна програми, виділення тексту також зникало. Загалом ці проблеми були вирішені. Для вирівнювання тексту було використано клас ***QMargins***, який дозволяє

налаштовувати відступи. Таким чином, в кожній комірці береться рядок, вирівнюється по лівому краю і зміщується на 5 позицій праворуч. Стосовно виділення тексту під час перемикання між комірками та зміни активного вікна. Для цього було використано подію ***eventFilter***. Всередині цієї події перевіряється фокусування на комірці. Якщо фокус є, тоді створюється об'єкт типу ***QLineEdit***, і якщо об'єкт існує, то викликається функція ***selectAll,*** яка виділяє весь текст комірки. Таким чином, на рисунку 3.3 наведено режим редагування після всіх змін:

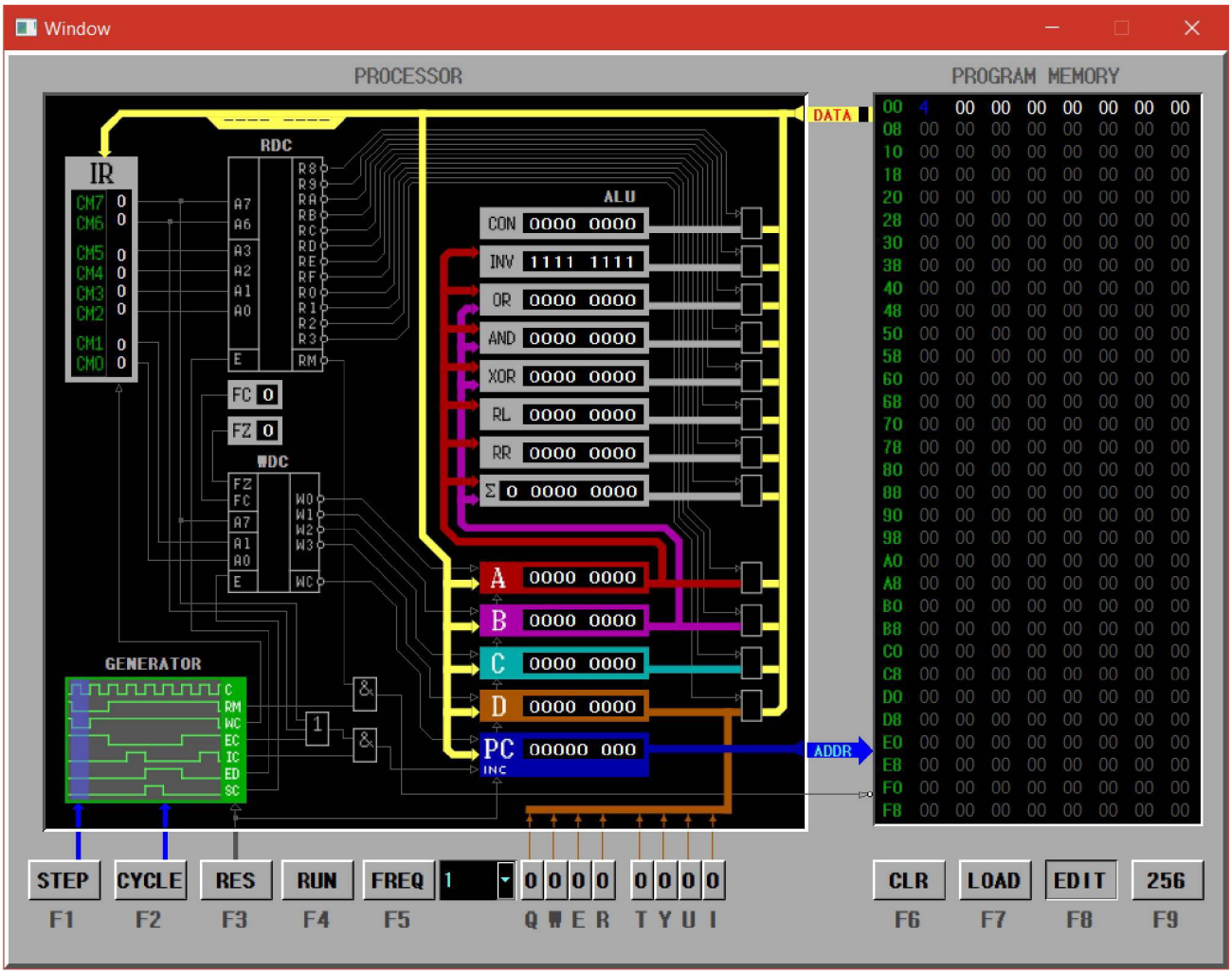

Рис.3.3. Режим редагування після внесення правок

Для наочності показано, що це робота в режимі редагування і можна вводити дані. Тепер комірки не відрізняються від звичайних. Більше того, можна вільно перемикатися між комірками і одразу записувати нові дані. При роботі з блоком пам'яті також було помічено недолік. Стосувався він того, що якщо користувач вводить малі літери, то вони так і залишаються в нижньому регістрі. Для вирішення цієї проблеми було використано функцію ***toUpper***, яка форматує всі літери до верхнього регістру.

Тепер стосовно тестів програми на різних системах. Для цього було використано віртуальну машину, в яку завантажено та встановлено системи Windows XP та Windows 7. Так як програма розроблялася на системі Windows 10, то очевидно, що вона була налагоджена під неї. На рисунках 3.4 та 3.5 наведено відображення програми на цих двох системах:

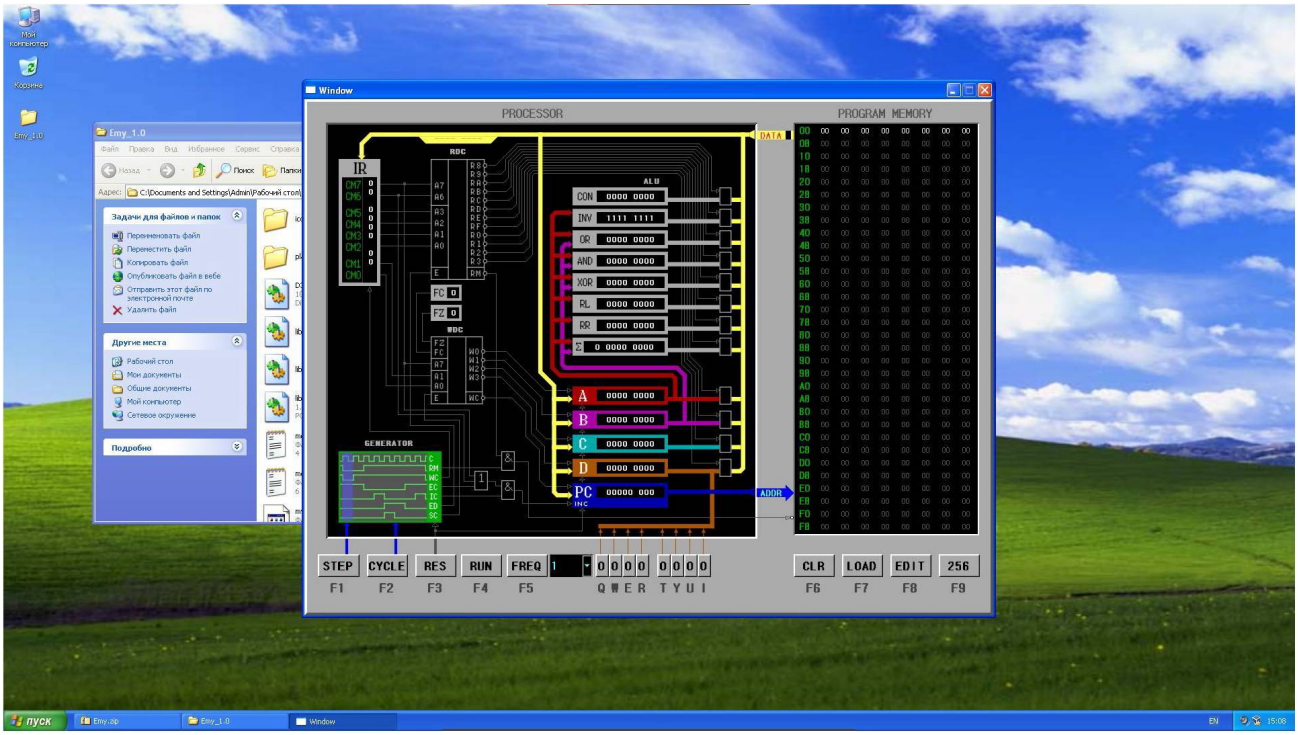

Рисунок 3.4 – Запуск емулятору на Windows XP

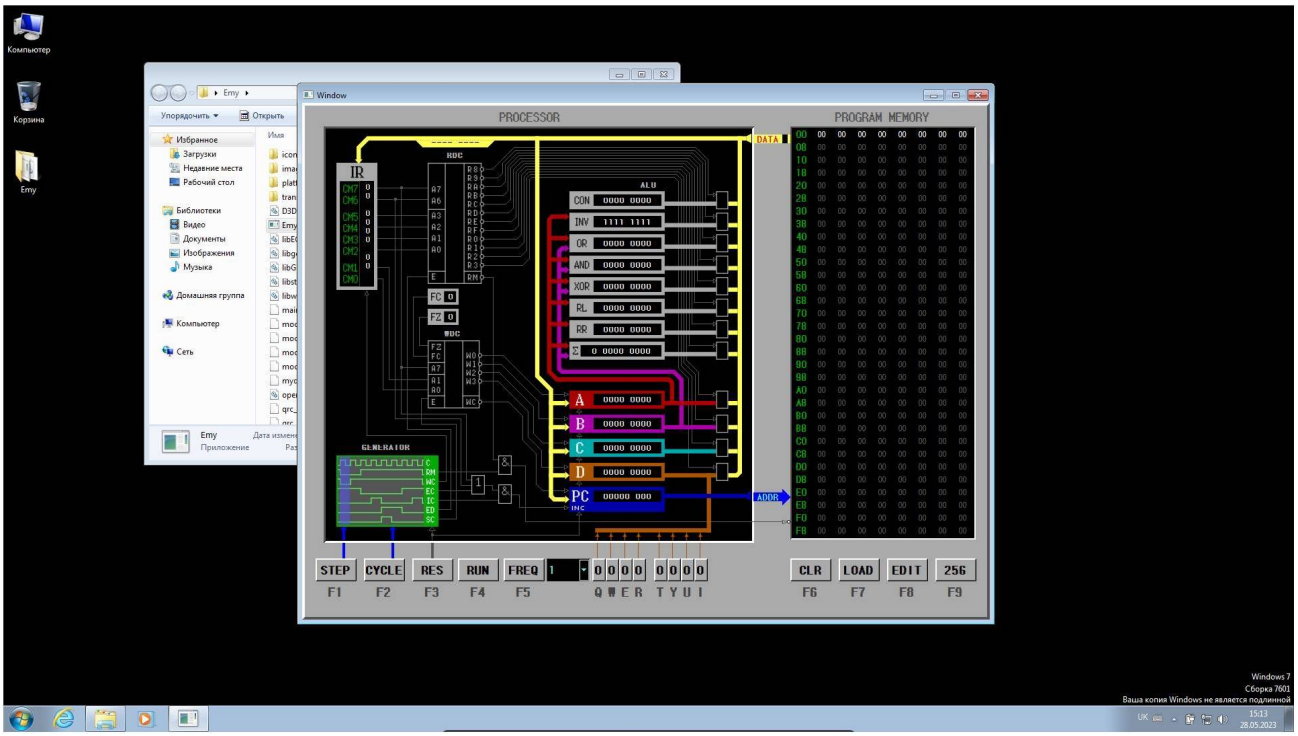

Рисунок 3.5 – Запуск емулятору на Windows 7

Як видно з рисунків, під час запуску на обох системах розмір шрифту відображається меншим за той, який повинен бути. Той же результат був отриманий, коли програма була запущена на цих же системах іншого комп'ютера. Для вирішення цієї проблеми було використано клас ***QSysInfo.*** Даний клас містить інформацію про систему, на якій запущено додаток [14]. Таким чином, цей клас надає можливість за допомогою функції ***windowsVersion*** визначити, на якій саме системі працює програма. Тому виконується перевірка системи, і якщо це є Windows XP або Windows 7, то розмір шрифту встановлюється дещо більшим, щоб результат був таким самим, як і на Windows 10. Таким чином, на рисунках 3.6 та 3.7 показано, як виглядає програма на двох системах після внесення змін в код.

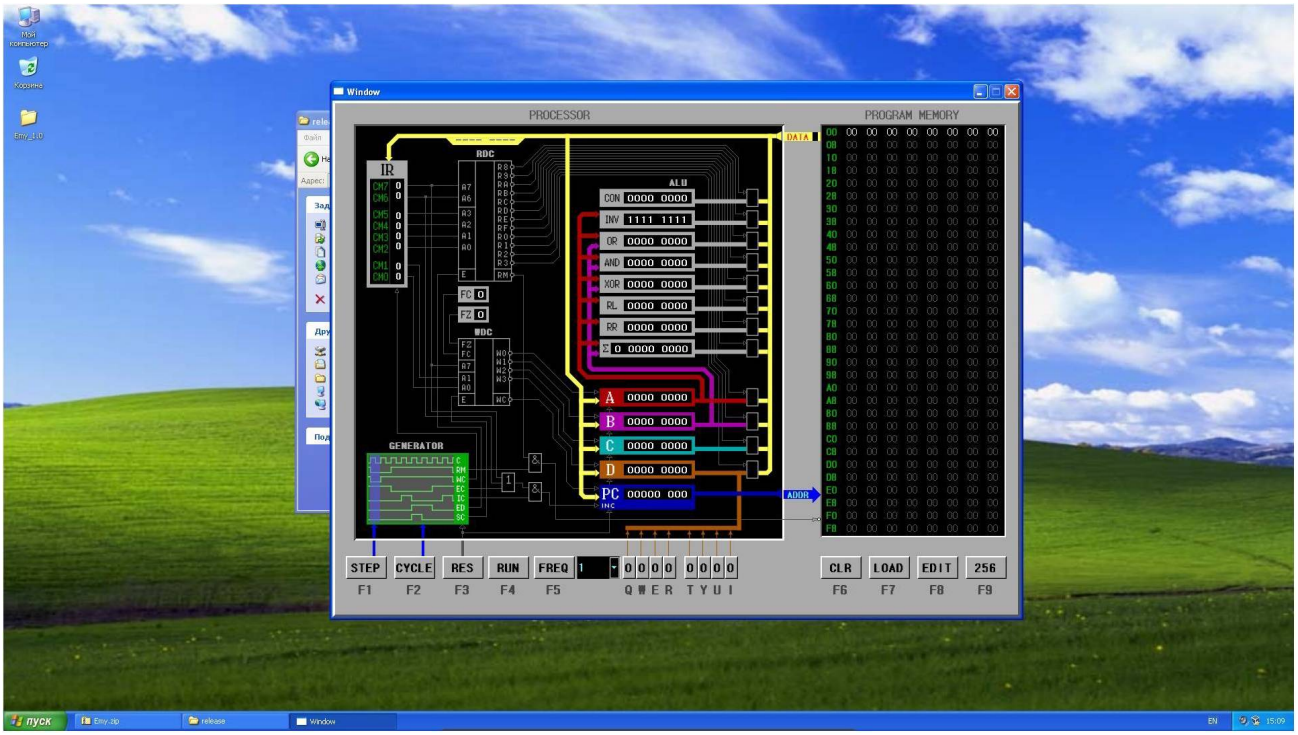

Рисунок 3.6 – Емулятор на Windows XP після змін

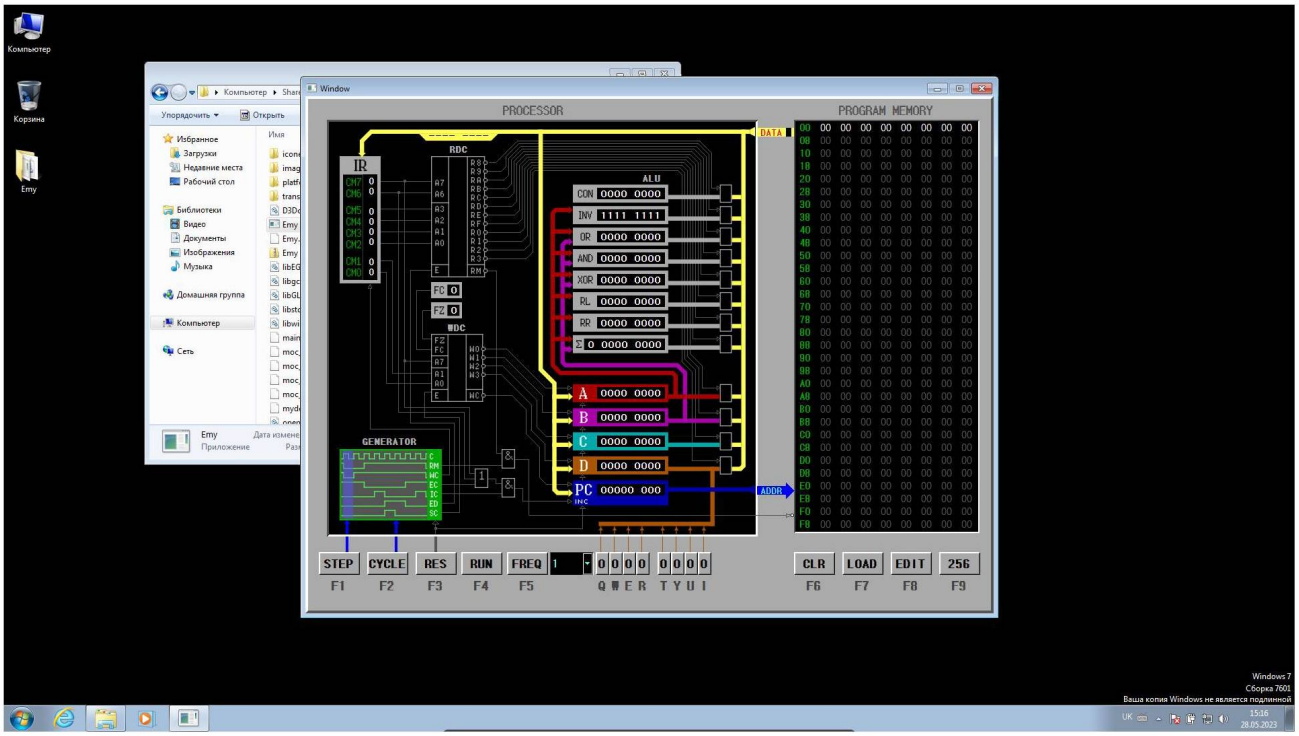

Рисунок 3.7 – Емулятор на Windows 7 після змін

Якихось інших проблем в програмі під час відлагодження не було помічено. Але у разі виявлення таких під час експлуатації, вони будуть обов'язково виправлені.

2.2. Інструкція щодо експлуатації програмного продукту

Після того, як програма створена та налагоджена, вона готова до експлуатації. Але незважаючи на її наочність та легкість у використанні, є потреба створити для користувачів інструкцію щодо того, як правильно нею користуватися.

1. Запустити програму на виконання будь-яким доступним в операційній системі способом;
2. Після запуску перед користувачем відкривається початковий стан процесора. Модуль пам'яті очищений, сигнали всі неактивні. Користувач може, звичайно, виконувати команди покроково, покомандно або в режимі реального часу, але з порожньою пам'яттю це не буде мати сенсу. Тому перш за все, потрібно визначитися з тим, який об'єм пам'яті потрібен. Якщо потрібно мати 256 байт, клікнути мишею на кнопку «256» або натиснути клавішу F9 на клавіатурі. В іншому випадку перейти на наступний пункт.
3. Після того, як було обрано потрібний об'єм пам'яті, є два варіанти: перший – завантажити готову програму з файлу з розширенням .BIN або .HEX; другий – заповнити пам'ять власноруч необхідними командами. В першому випадку необхідно клікнути мишею на кнопку LOAD або натиснути на клавіатурі клавішу F7, та у відкритому діалоговому вікні знайти потрібний файл і відкрити його. Вміст файлу буде проаналізовано та всі команди будуть записані у відповідні комірки. В другому випадку необхідно клікнути на кнопку EDIT або натиснути F8 на клавіатурі, після чого всі комірки пам'яті перейдуть в стан редагування. Після цього можна вводити

потрібні команди, перемикаючись між комірками натисканням кнопки Tab. Для того, щоб повернутися на попередню комірку, треба натиснути комбінацію клавіш Shift+Tab. Після завершення редагування натиснути Enter і потім знову клікнути EDIT або F8.

4. Після цього необхідно клікнути на кнопку RES або натиснути F3, щоб на всяк випадок видалити всі попередні дані та підготувати процесор до роботи.
5. Якщо ви хочете покроково виконувати програму, тоді потрібно клікнути мишею на кнопку STEP або натиснути на клавіатурі F1. Таким чином буде виконано 1 етап машинного циклу. Для кожного наступного етапу потрібно знову натиснути цю кнопку.
6. Якщо ви хочете виконувати програму покомандно, тоді потрібно клікнути мишею на кнопку CYCLE або натиснути на клавіатурі значення F2. Виділення комірки в модулі пам'яті зміниться на фіолетовий колір. Після натискання цієї кнопки буде виконано одну команду – всі 8 етапів машинного циклу, але за один раз. Ви також можете натиснути цю кнопку в той час, коли машинний цикл не був завершений на покроковому виконанні. В цьому випадку завершиться виконання поточної команди і буде виконано перехід на наступну команду.
7. Якщо ви хочете запустити виконання в реальному часі, треба клікнути мишею на кнопку RUN або натиснути F4. Таким чином, програмна модель почне самостійно без зупинки виконувати команди одну за одною. В цей час ви можете змінити частоту роботи процесора. Для цього потрібно натиснути на кнопку FREQ, після чого обрати нове значення. Після вибору програма без зупинки перейде на більшу чи меншу швидкість виконання.

8. Якщо під час виконання програми в реальному часі натиснути кнопку RES, то виконання автоматично зупиниться і модель процесора перейде в початковий стан.
9. Якщо вам необхідно створити якісь маніпуляції з пам'яттю, необхідно спочатку зупинити виконання програми в реальному часі (якщо воно запущено), і лише після цього робити все, що потрібно.
10. Для швидкого очищення пам'яті ви можете клікнути на кнопку CLR або натиснути клавішу F6.
11. Якщо ви змінюєте об'єм пам'яті в той час, коли там вже записані якісь дані, то вони стираються. В реальній моделі для цього необхідно замінити модуль пам'яті, тому очевидно, що вона стає порожньою. Те ж саме реалізовано і в програмній моделі.
12. Для взаємодії з клавіатурою, що під'єднана до регістра D, потрібно або клікнути мишею на відповідні кнопки, або скористатися клавіатурою. Кожна кнопка підписана відповідною клавішею на клавіатурі.
13. Для завершення роботи з емулятором можна клікнути хрестик в кутку вікна або натиснути комбінацію клавіш Ctrl+X.

Висновки за розділом 3

В даному розділі виконувалося тестування та відлагодження готової програмної моделі. Більша частина програми працює коректно. Були виявлені деякі недоліки, які були описані в цьому розділі. Всі вони виправлені та на даний момент в програмі не було помічено серйозних помилок. Програмна модель в повному обсязі імітує роботу реальної моделі процесора. Також було написано інструкцію для користувача, адже незважаючи на те, що програма є простою у використанні, можуть виникнути деякі ускладнення під час першого запуску. І в цьому випадку опис користувача стане у нагоді.

ВИСНОВКИ

Підводячи підсумки виконання даної роботи можна зробити висновок, що всі поставлені задачі були успішно виконані.

1. Зроблено огляд існуючих емуляторів. В процесі аналізу було з'ясовано, що серед наявних програмних рішень відсутні програми, які в повному обсязі задовольняють потреби для покращення навчального процесу. Виходячи з цього було прийнято рішення про створення власної програмної моделі у вигляді програми емулятора.
2. Обрано мову програмування: проаналізувавши можливі варіанти, було обрано мову програмування C++ із набором інструментів Qt. З цією мовою програмування є певний досвід роботи, окрім цього її можливостей вистачає для створення програмної моделі. Інструментарій Qt надає велику кількість бібліотек та модулів для створення додатку з графічним інтерфейсом.
3. Проаналізовано реальну модель процесору: було детально розібрано архітектуру процесора DPM-08, принципи та режими роботи. Проаналізовано структуру командного слова. Наявність всіх необхідних знань щодо можливостей процесора дозволило створити зручну програмну модель, яка повністю імітує роботу реальної моделі процесора.
4. Проведено відлагодження та написано опис користувача: робота програмної моделі була відлагоджена. Всі знайдені недоліки було виправлено. Для користувача було створено інструкцію щодо використання програми-емулятора.

Завдяки успішному виконанню всіх поставлених завдань була створена програмна модель процесора у вигляді емулятора, яка дозволить в подальшому

покращити якість проведення лекцій та лабораторних робіт, а також самостійної роботи студентів при вивченні мікропроцесорної техніки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. µVision User's Guide [Електронний ресурс]. – режим доступу: URL: https://developer.arm.com/documentation/101407/0538/About-uVision (дата звернення – 05.11.2022).
2. Proteus 8 Professional - a useful tool for electronic engineers (techulator.com). [Електронний ресурс]. – режим доступу: URL: https://www.techulator.com/resources/15432-Proteus-8-Professional-a-useful-tool-for-electronic-engineers.aspx (дата звернення – 07.11.2022).
3. Borland Turbo Debugger – Wikipedia [Електронний ресурс]. – режим доступу: URL: https://en.wikipedia.org/wiki/Borland_Turbo_Debugger (дата звернення – 08.11.2022).
4. Що таке фреймворк (framework)? Пояснюємо простими словами [Електронний ресурс]. – режим доступу: URL: https://highload.today/uk/frejmvorki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/ (дата звернення – 03.01.2023).
5. 10 переваг мови C++, яка і досі залишається актуальною | DOU [Електронний ресурс]. – режим доступу: URL: https://dou.ua/forums/topic/42888/ (дата звернення – 05.01.2023).
6. Огляд і основи мови програмування C++ [Електронний ресурс]. – режим доступу: URL: http://www.znannya.org/?view=Cpp_basics (дата звернення – 05.01.2023).
7. Signals & Slots | Qt Core 6.5.1 [Електронний ресурс]. – режим доступу: URL: https://doc.qt.io/qt-6/signalsandslots.html (дата звернення – 10.02.2023).

8. C++ and Qt for embedded systems - Integra Sources [Електронний ресурс]. – режим доступу: URL: https://www.integrasources.com/blog/qt-c-embedded-development-pros-cons-alternatives/ (дата звернення – 11.02.2023).
9. Журавель Ю. А. Модель цифрового процесора / Ю. А. Журавель, С. Н. Рева // Вісник Харківського національного університету імені В.Н. Каразіна. Серія : Математичне моделювання. Інформаційні технології. Автоматизовані системи управління. 2011. № 987, Вип. 18. С. 5-18 [Електронний ресурс]. – режим доступу: URL: http://nbuv.gov.ua/UJRN/VKhIMAM_2011_987_18_3 (дата звернення – 15.02.2023).
10. Qt Style Sheets Reference | Qt Widgets 6.5.1 [Електронний ресурс]. – режим доступу: URL: https://doc.qt.io/qt-6/stylesheet-reference.html (дата звернення – 24.02.2023).
11. QStyledItemDelegate Class | Qt Widgets 5.15.14 [Електронний ресурс]. – режим доступу: URL: https://doc.qt.io/qt-5/qstyleditemdelegate.html (дата звернення – 03.03.2023).
12. Raster vs. vector: What are the differences? | Adobe [Електронний ресурс]. – режим доступу: URL: https://www.adobe.com/creativecloud/file-types/image/comparison/raster-vs-vector.html (дата звернення – 05.02.2023).
13. QPainter Class | Qt GUI 5.15.14 [Електронний ресурс]. – режим доступу: URL: https://doc.qt.io/qt-5/qpainter.html#details. (дата звернення – 15.02.2023).
14. QSysInfo Class | Qt Core 5.15.14 [Електронний ресурс]. – режим доступу: URL: https://doc.qt.io/qt-5/qsysinfo.html#details. (дата звернення – 27.03.2023).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр
Галузь знань: 12 – Інформаційні технології.
Спеціальність 123 – Комп'ютерна інженерія.

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної
та прикладної системотехніки
д.т.н., проф. Шматков С. І.

«17»листопада 2022 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

Комеристого Владислава Сергійовича
прізвище, ім'я та по батькові студента (студентки)

1. Тема роботи: **«Розробка програмної моделі навчального цифрового процесора»**

керівник роботи: Рева Сергій Миколайович, кандидат технічних наук.
прізвище, ім'я, по батькові, науковий ступінь, вчене звання

затверджені наказом по університету від « *23* » *травня* 20 23 року № 4104-5/895 .

2. Строк подання студентом роботи:. 26 травня 2023 .

3. Перелік питань, які потрібно розробити

Провести аналіз предметної області.

Проаналізувати архітектуру процесора DPM-8 на рівні системи команд.

Розробити та віддагодити програму емулятора.

Провести тестування розробленої програми

Написати інструкцію по роботі з програмою для користувача.

Підготувати пояснювальну записку, доповідь та презентацію

4. План роботи.

№ з/п	Назва етапів роботи	Термін виконання етапів роботи
1	Проведення аналізу предметної області, пошук та вивчення літератури	Листопад 2022
2	Проведення аналізу архітектури процесора DPM-8 на рівні системи команд	Грудень 2022
3	Проведення аналізу варіантів створення проекту, вибір інструментарію	Грудень 2022 – Січень 2023
4	Розробка алгоритму майбутньої програми	Січень 2023
5	Розробка та віддагодження програми емулятора	Лютий 2023 – Квітень 2023
6	Тестування застосунку та виправлення помилок	Квітень 2023 – Травень 2023
7	Написання інструкції по роботі з програмою для користувача	Травень 2023
8	Оформлення пояснювальної записки	Травень 2023 – Червень 2023
9	Представлення кваліфікаційної роботи керівнику та рецензенту	Червень 2023

5. Дата видачі завдання «14» жовтня 2022 року.

Студент _____ В.С. Комеристий
підпис ініціали, прізвище

Керівник роботи _____ С.М. Рева
підпис ініціали, прізвище

Додаток Б

Технічне завдання
на розробку програмного виробу
«Програмна модель навчального цифрового процесора»

Назва розділу	Назва та зміст підрозділу
1. Вступ	1.1. Назва програмного виробу «Програмна модель навчального цифрового процесора» 1.2. Галузь застосування: використання виробу для навчання студентів принципам роботи процесора
2.Підстава для розробки	2.1. Навчальний план за спеціальністю 123 «Комп'ютерна інженерія» 2.2. Завдання на кваліфікаційну роботу затверджено наказом №4101-5/895 ХНУ імені В.Н.Каразіна від 23.05.2023.
3.Призначення розробки	3.1. Мета розробки програмного виробу: покращення методики викладання основ комп'ютерної схемотехніки на факультеті комп'ютерних наук. 3.2. Призначення програмного виробу: надання можливості знайомства із будовою та розвитком мов програмування, починаючи з найнижчого рівня програмування машинних кодів. 3.3. Вхідні дані для розробки: реальна модель процесора DPM-08, вхідні параметри змінюються від 0 до 255 (FF), блок пам'яті, регістри A,B,C,D та регістри арифметико-логічного пристрою, клавіатура для вводу значень в регістр D.
4.Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: точність емуляції, швидкодія, сумісність з операційними системами, простота використання, розширені можливості, надійність. 4.2. Вимоги до надійності: стабільність роботи, емуляція процесу виконання програм, можливість роботи із трансляторами програм. 4.3. Вимоги до умов експлуатації: якихось серйозних умов немає, потрібно мати комп'ютер або ноутбук із встановленою системою Windows 10/8/7/XP. Програмний виріб не потребує великої потужності та великого об'єму пам'яті. 4.4. Вимоги до складу параметрів технічних засобів: комп'ютер або ноутбук із встановленою системою Windows. Не потрібні якісь надсучасні комп'ютери з потужним процесором та великою оперативною пам'яттю. 4.5. Вимоги до інформаційної та програмної сумісності: програмний виріб має сумісність із системами Windows XP, Windows 7, Windows 8 та Windows 10 4.6. Вимоги до маркування та пакування: відсутні 4.7. Вимоги до транспортування та зберігання: відсутні 4.8. Спеціальні вимоги: відсутні
5. Вимоги до програмної документації.	Програмною документацією до виробу «Програмна модель навчального цифрового процесора» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи)

<table>
<tr><td>документації.</td><td>1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи)
2) Опис програмного виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).
3) Опис користувача по роботі з емулятором (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).</td></tr>
<tr><td>6.Техніко-економічні показники</td><td>Техніко-економічні показники відсутні</td></tr>
<tr><td>7. Стадії та етапи розробки</td><td>
<table>
<tr><th>№ з/п</th><th>Назва етапів роботи</th><th>Термін виконання етапів роботи</th></tr>
<tr><td>1</td><td>Проведення аналізу предметної області, пошук та вивчення літератури</td><td>Листопад 2022</td></tr>
<tr><td>2</td><td>Проведення аналізу архітектури процесора DPM-8 на рівні системи команд</td><td>Грудень 2022</td></tr>
<tr><td>3</td><td>Проведення аналізу варіантів створення проекту, вибір інструментарію</td><td>Грудень 2022 – Січень 2023</td></tr>
<tr><td>4</td><td>Розробка алгоритму майбутньої програми</td><td>Січень 2023</td></tr>
<tr><td>5</td><td>Розробка та відлагодження програми емулятора</td><td>Лютий 2023 – Квітень 2023</td></tr>
<tr><td>6</td><td>Тестування застосунку та виправлення помилок</td><td>Квітень 2023 – Травень 2023</td></tr>
<tr><td>7</td><td>Написання інструкції по роботі з програмою для користувача</td><td>Травень 2023</td></tr>
<tr><td>8</td><td>Оформлення пояснювальної записки</td><td>Травень 2023 – Червень 2023</td></tr>
<tr><td>9</td><td>Створення презентації, підготовка до доповіді</td><td>Червень 2023</td></tr>
</table>
</td></tr>
</table>

8.Порядок контролю та приймання	У цьому розділі мають бути зазначені загальні вимоги до приймання розробленого програмного виробу, наприклад: 1) Перевірку відповідності створеної програми технічним характеристикам реальної моделі. 2) Перевірка стабільності та коректності роботи програмного забезпечення 3) Розроблені проектні документи подати в електронному вигляді та на паперових носіях в одному примірнику.

Виконавець

студент групи КІ- 41

Комеристий В.С.

Замовник

к. т. н., доцент

Рева С.М.

Додаток В

Програма та методика випробувань програмного виробу

«Програмна модель навчального цифрового процесора»

1. Об'єкт випробувань

1.1 Найменування програмного виробу «Програмна модель навчального цифрового процесора»

1.2 Область застосування для навчання студентів принципам роботи процесора

2. Мета випробувань

Переконатися в тому, що програмний виріб працює справно, виконує свої функції та не завершується аварійно.

3. Загальні положення

3.1 Підстави щодо випробувань

Перевірка реальної працездатності та відповідність технічним вимогам.

3.2 Місце та тривалість випробувань

Випробування проводяться на різних операційних системах та різних комп'ютерах.

3.3. «Обсяг випробувань»

Приймальні випробування програмного виробу проводяться в обсязі відповідно до цієї Програми та методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться за участю Замовника, Виконавця та членів комісії, призначених для прийому дипломної роботи.

4. Вимоги до програмного виробу

4.1. Вимоги до функціональних характеристик: точність емуляції, швидкодія, сумісність з операційними системами, простота використання, розширені можливості, надійність.

4.2. Вимоги до надійності: стабільність роботи, емуляція процесу виконання програм, можливість роботи із трансляторами програм.

4.3. Вимоги до умов експлуатації: якихось серйозних умов немає, потрібно мати комп'ютер або ноутбук із встановленою системою Windows 10/8/7/XP. Програмний виріб не потребує великої потужності та великого об'єму пам'яті.

4.4. Вимоги до складу параметрів технічних засобів: комп'ютер або ноутбук із встановленою системою Windows. Не потрібні якісь надсучасні комп'ютери з потужним процесором та великою оперативною пам'яттю.

4.5. Вимоги до інформаційної та програмної сумісності: програмний виріб має сумісність із системами Windows XP, Windows 7, Windows 8 та Windows 10

5. Вимоги до програмної документації

«Склад програмної документації, що подається на випробування включає:

1) Технічне завдання на розробку програмного виробу (представлено в Додатку Б до пояснювальної записки до дипломної роботи).

2) Програма та методика випробувань розробленого програмного виробу (представлена в Додатку В до пояснювальної записки до дипломної роботи).

6. Засоби та порядок випробувань

6.1 Засоби випробувань

Випробування проводяться на різних комп'ютерах із встановленими операційними системами Windows XP, Windows 7, Windows 8 або Windows 10 . Не потрібні якісь надсучасні комп'ютери з потужнім процесором та великою оперативною пам'яттю.

6.2 Порядок проведення випробувань

1-й етап. Перелік перевірок, що проводяться на 1-му етапі, включає наступні операції:

-1-й підетап. Перевірка комплектності програмної документації

Перевірка комплектності технічних та програмних засобів здійснюється за критерієм відповідності їх складу Технічному завданню (за потреби).

-2-й підетап. Перевірка якості представленої на випробування програмної документації на відповідність вимогам стандартів ЄСПД.

Якість програмної документації відповідає вимогам ДЕСТ 19301-79 ЄСПД.

2-й етап. Перелік перевірок, що проводяться на другому етапі випробувань, може включати наступні операції:

Перевірка робиться з використанням тестових програм, які призначені для тестування функціональності програмної моделі під час роботи з різними оперативними системами.

Тест 1: Перевірка можливості змінити об'єм доступної пам'яті процесора в емуляторі та завантажити туди програму:

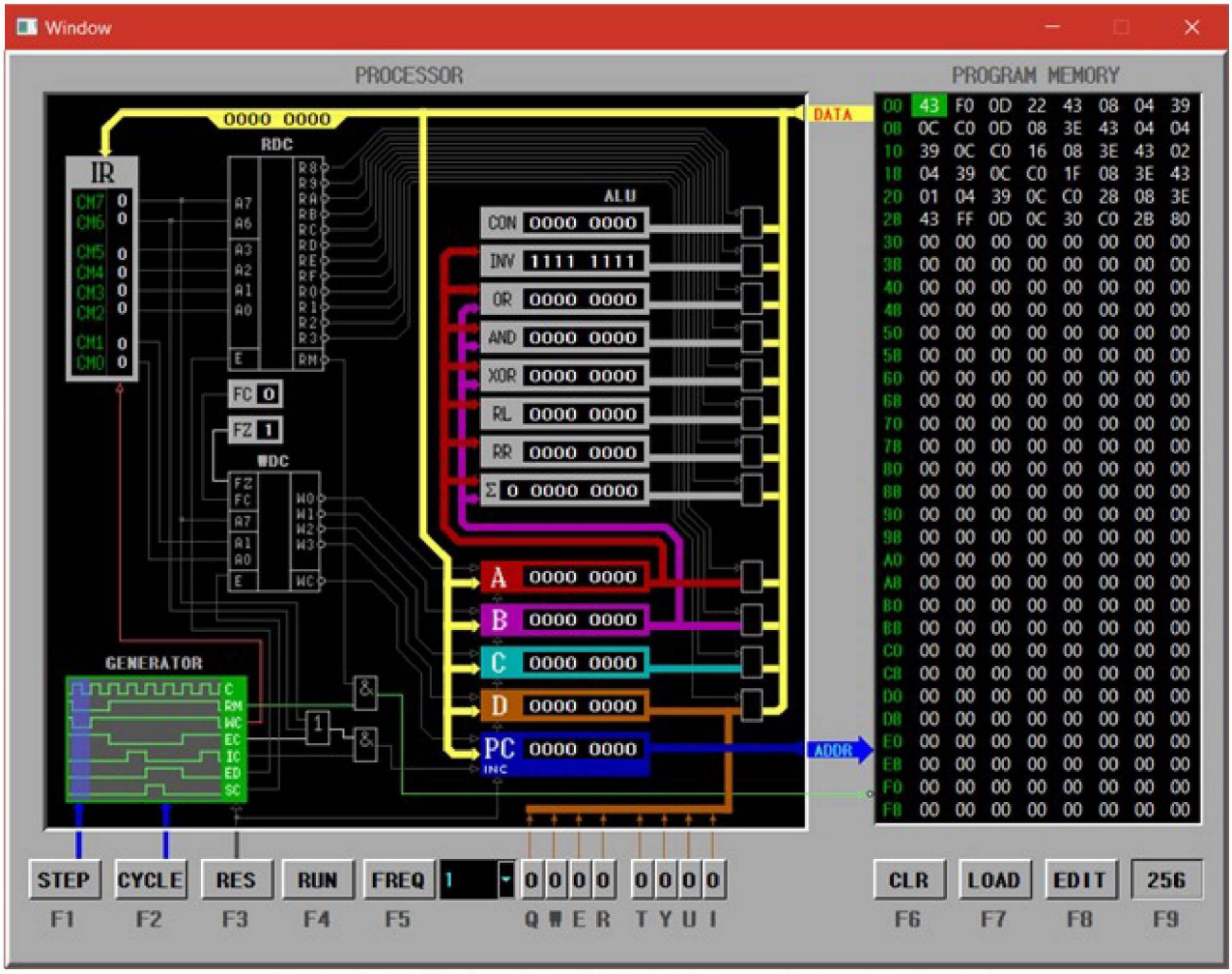

Рисунок В.1 – зміна розміру пам'яті та завантаження програми

Висновок: пам'ять була змінена на 256 байт та завантажено деяку програму. Якщо в лунках пам'яті з'явилися дані, то програма була завантажена. Тест 1 пройшов успішно.

Тест 2: покрокове виконання програми

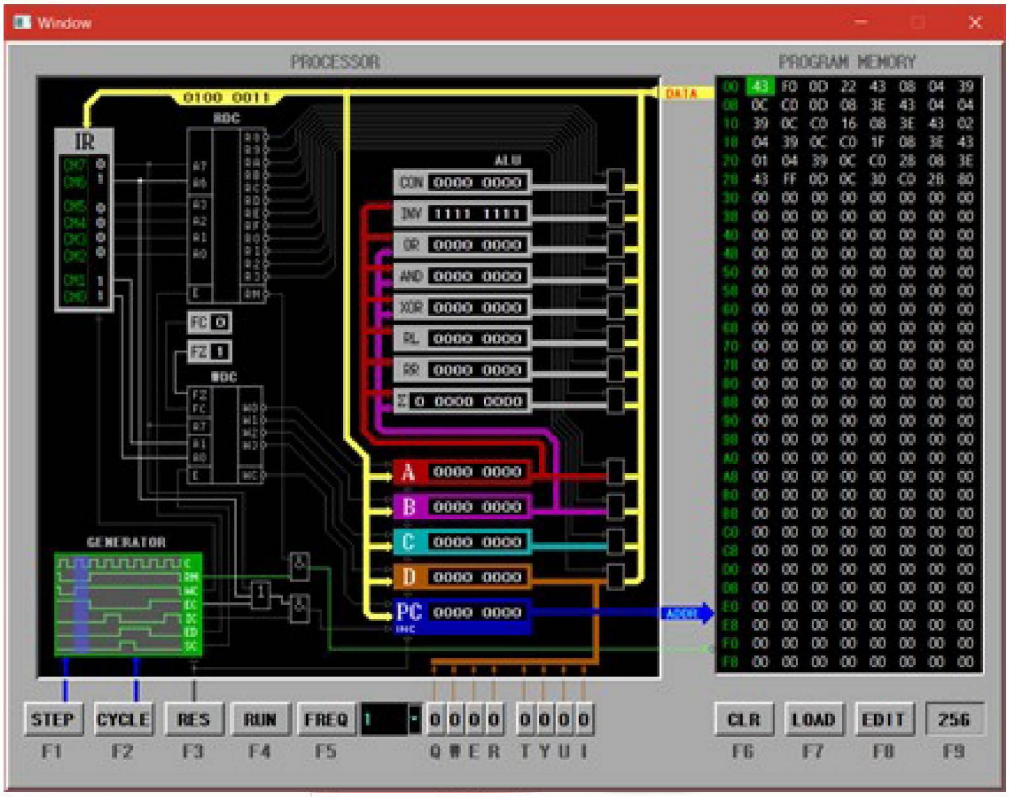

Рисунок В.2 – 2 етап циклу

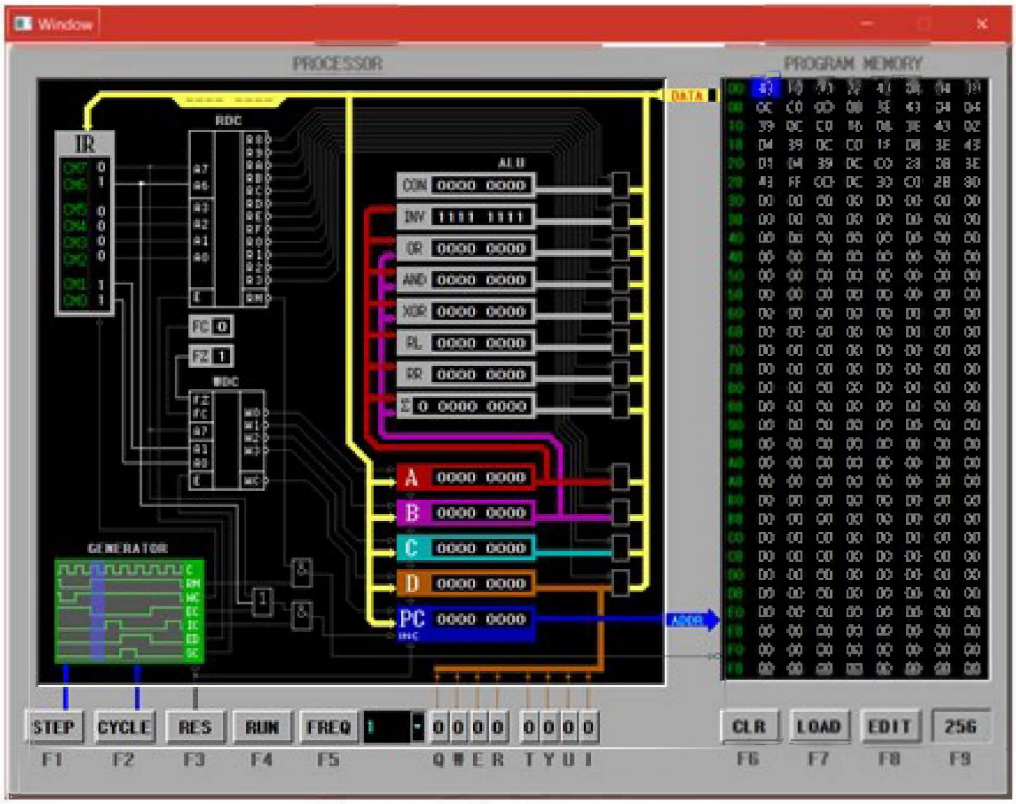

Рисунок В.3 – 3 етап циклу

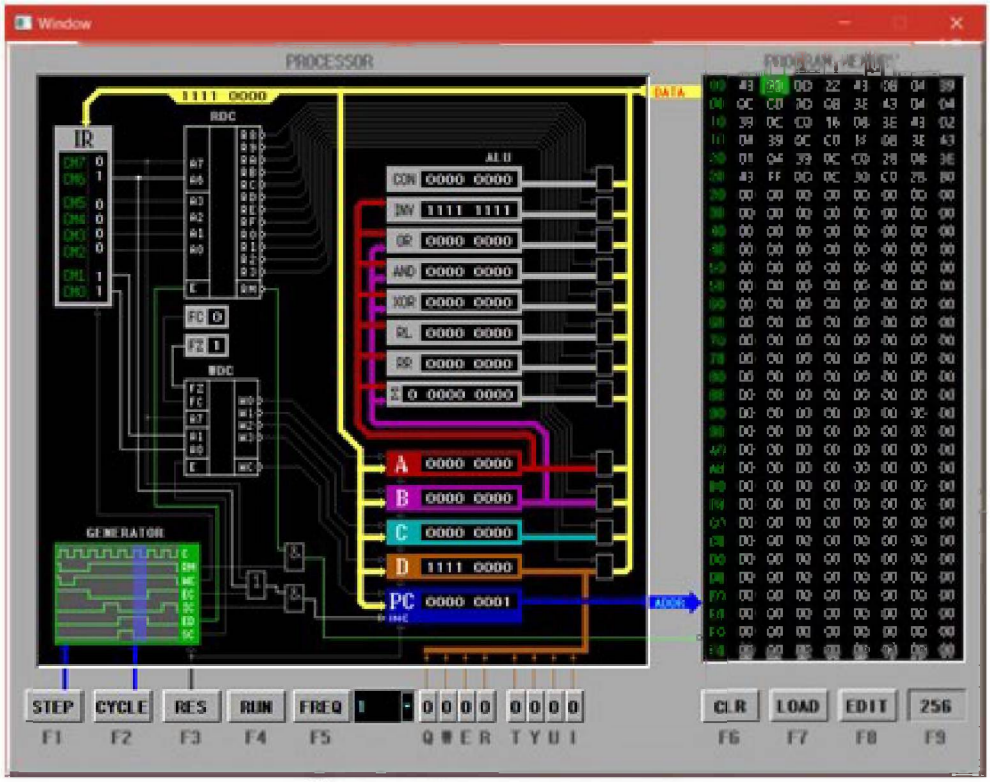

Рисунок В.6 – 6 етап циклу

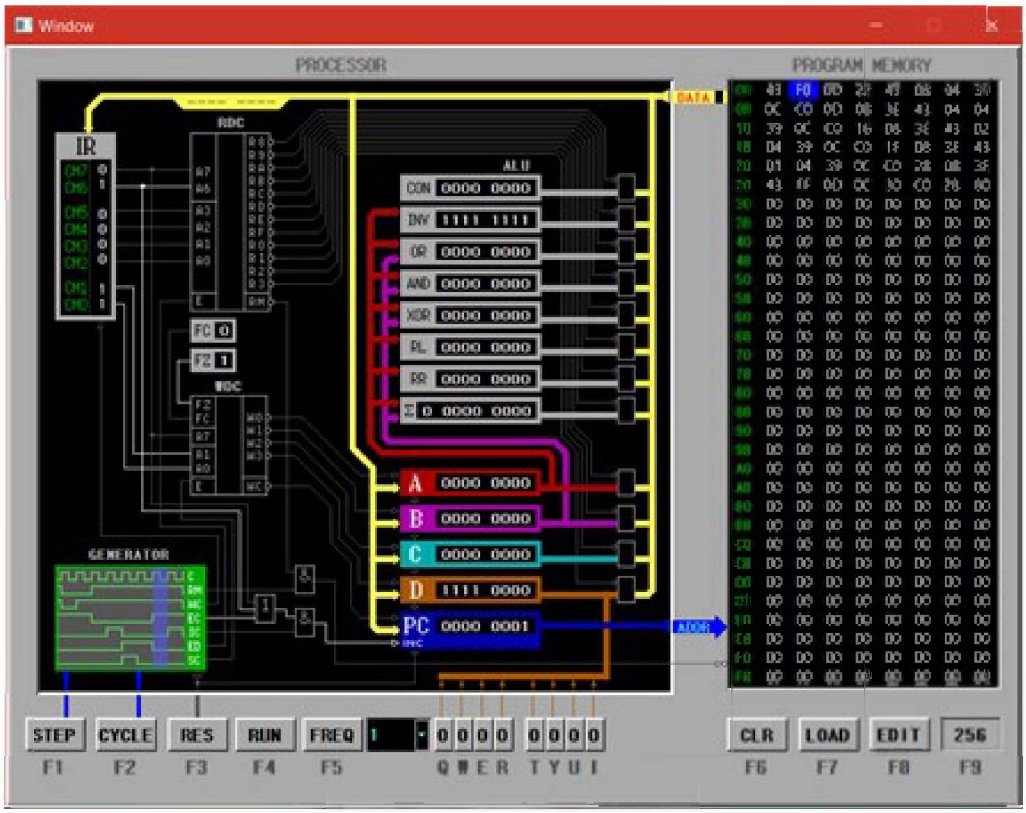

Рисунок В.7 – 7 етап циклу

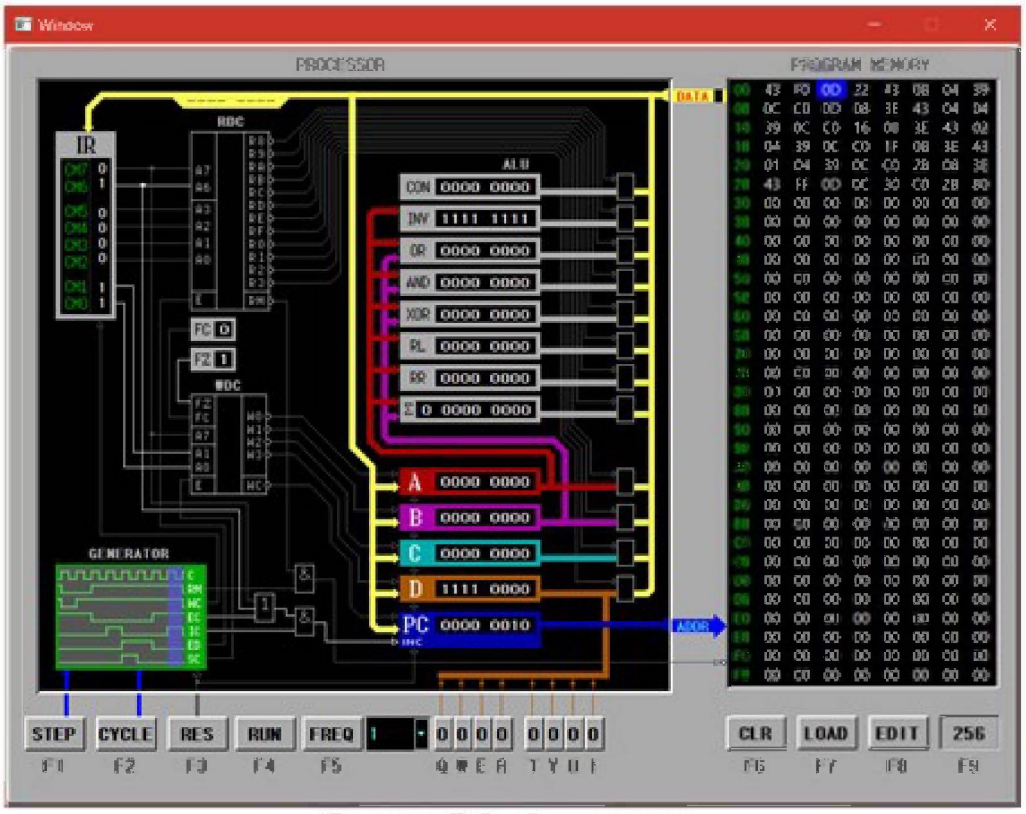

Рисунок В.8 – 8 етап циклу

Висновок: на наведених нижче рисунках можна простежити те, як відбувається виконання програми покроково. На 2 етапі можна побачити зміну кольору сигналу WC з генератору. 3 етап є проміжним, його виконання можна відстежити, якщо поглянемось на шину даних Data та на блок пам'яті. Там відбуваються деякі зміни, а саме: на шині даних з'являється такий собі бар'єр, а виділення лунки пам'яті змінюється на синій колір. На 4 етапі виконуються інкременті лічильника (регістр PC), але лише у випадку, якщо команда є двобайтною. В іншому випадку, нічого не відбувається. Відстежити, що це справді спрацювало, можна поглянувши на блок пам'яті, де вже інша лунка пам'яті буде виділена. На 5 етапі на шині даних виставляється значення (або з пам'яті, або з регістрів, в залежності від типу команди). На 6 етапі закінчується запис значення з шини даних в один з регістрів A,B,C,D, або регістр PC, якщо це команда переходу. 7 етап є також проміжним, лунка пам'яті змінює колір на синій. На 8 етапі змінюється значення регістра PC (якщо це не команда переходу) і перехід на іншу лунку пам'яті. Таким чином, тест 2 успішно виконан.

Тест 3: циклічне виконання програми

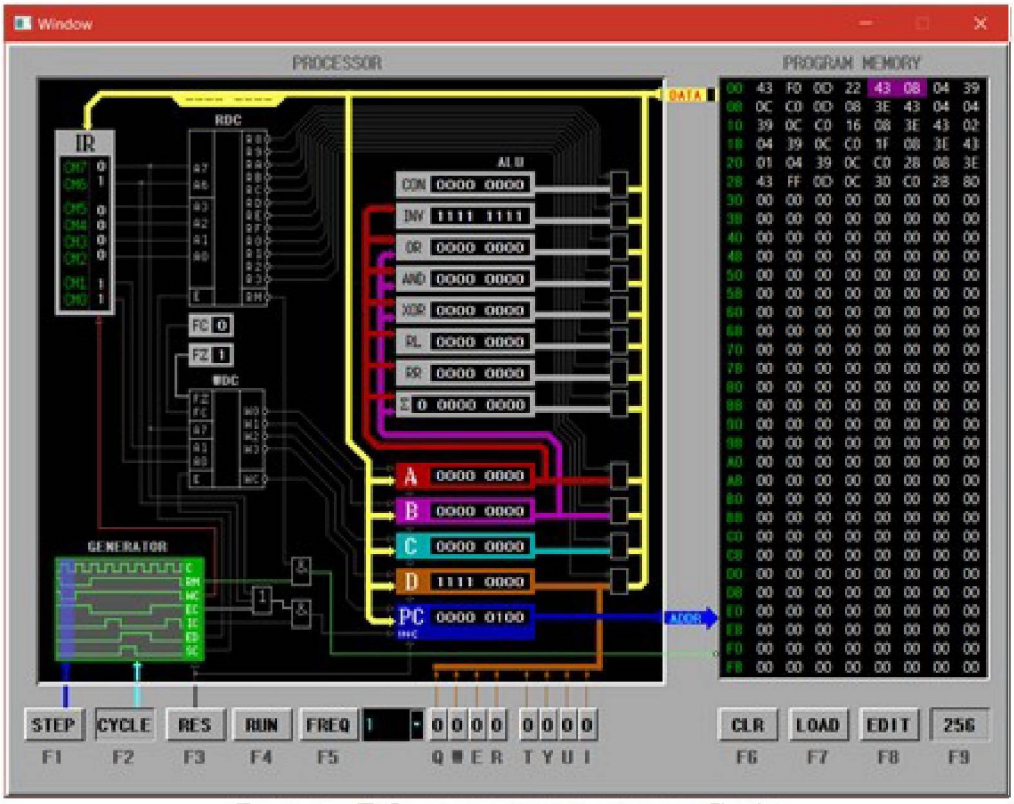

Рисунок В.9 – натискання кнопки Cycle

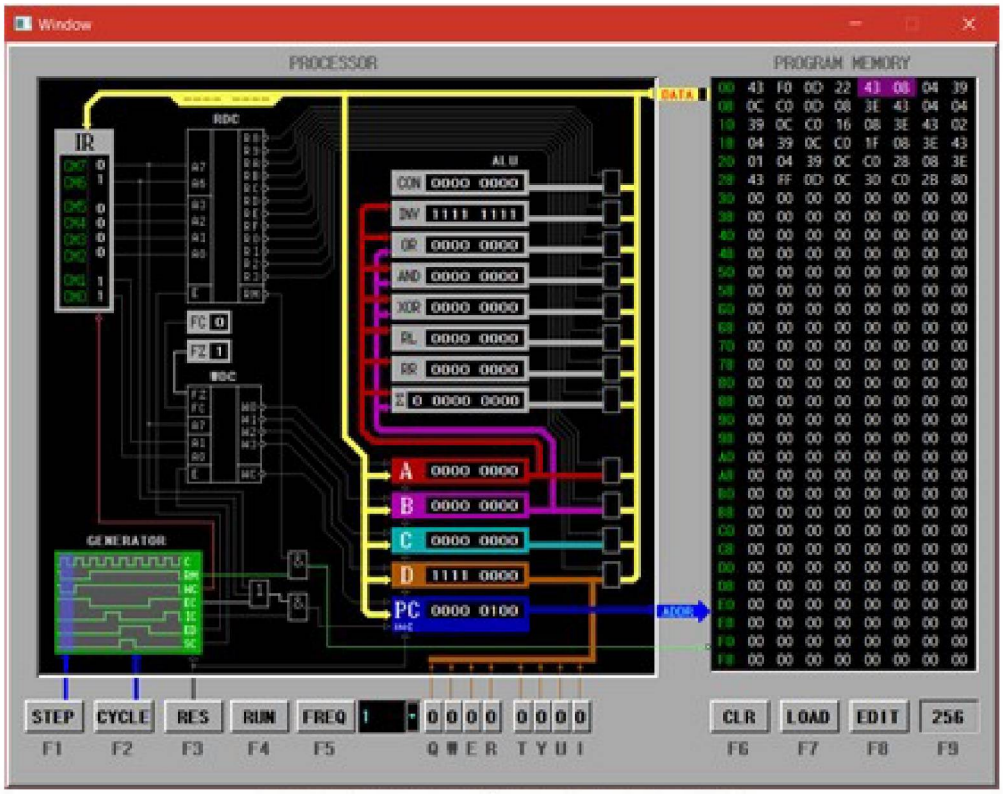

Рисунок В.10 – відпускання кнопки

Висновок: Тест 3 пройшов успішно, команда пройшла 8 циклів фон Неймана і перейшла на наступну лунку пам'яті до наступної команди (позначено фіолетовим).

Тест 4: очищення пам'яті

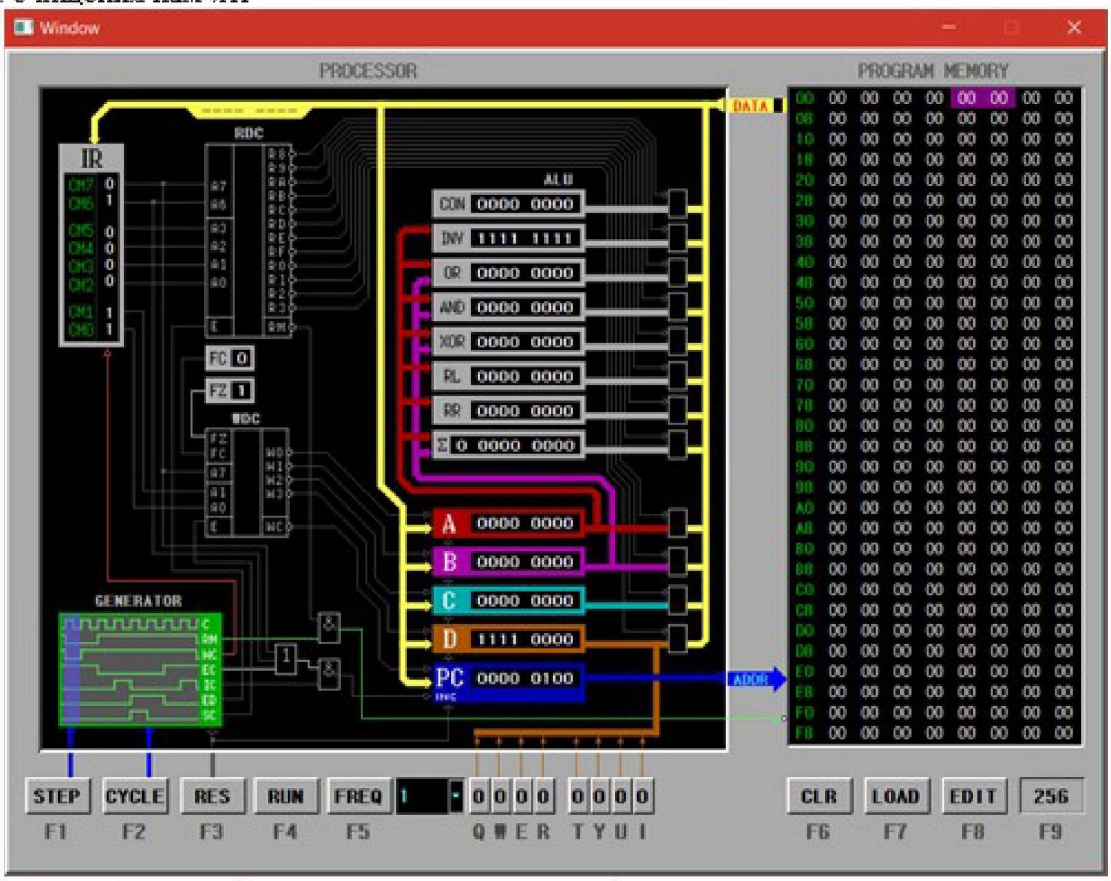

Рисунок В.11 – пам'ять очистилась після натискання кнопки CLR

Висновок: Було натиснуто кнопку CLR, після якої всі лунки пам'яті заповнюються нулями, що свідчить про те, що пам'ять була очищена. Тест 4 виконано успішно.

Виконавець
студент групи КІ41
Комерислий В.С.

Додаток Г

Фрагменти коду розробленої програми-емулятора

```
Window::Window(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Window)
{
    ui->setupUi(this);
    QFontDatabase::addApplicationFont(":/resources/Undertale-Battle-Font.ttf");
    setStartInterfaceStyle();
    setTableProperties();
    secFunc = new SecondaryFunctions();
    svg_rend = new QSvgRenderer();
    connect(ui->editButton, &QPushButton::clicked, this, &Window::editItem);
    connect(ui->oneButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->twoButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->threeButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->fourButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->fiveButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->sixButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->sevenButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->eightButton, SIGNAL(clicked()), this, SLOT(numButtons_clicked()));
    connect(ui->memoryTable, &QTableWidget::itemClicked, this, &Window::handleItemClicked);
    connect(timer, &QTimer::timeout, this, &Window::runWorkOfProcessor);
    connect(ui->freqComboBox, SIGNAL(currentIndexChanged(int)), this, SLOT(onComboBoxIndexChanged(int)));
}

Window::~Window()
{
    delete ui;
}
```

```
void Window::setStartInterfaceStyle()
{
    //Встановлюю фіксований розмір вікна та його стиль
    ui->centralWidget->setMinimumSize(1024, 768);
    ui->centralWidget->setMaximumSize(1024, 768);
    ui->centralWidget->setStyleSheet(StyleHelper::getCentralWidgetStyle());

    //Встановлюю стиль для усіх кнопок
    ui->stepButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->cycleButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->resButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->runButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->freqButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->oneButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->twoButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->threeButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->fourButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->fiveButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->sixButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->sevenButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->eightButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->clearButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
```

```
    ui->loadButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->editButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
    ui->memoryButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());

    //Встановлюю стиль для таблиці
    ui->memoryTable->setStyleSheet(StyleHelper::getMemoryStyle());

    //Налаштовую надписи та встановлюю стилі для них
    ui->processLabel->setText("PROCESSOR");
    ui->processLabel->setAlignment(Qt::AlignCenter);
    ui->processLabel->setStyleSheet(StyleHelper::getDisplayAndMemoryTitleStyle());

    ui->memoryLabel->setText("PROGRAM MEMORY");
    ui->memoryLabel->setAlignment(Qt::AlignCenter);
    ui->memoryLabel->setStyleSheet(StyleHelper::getDisplayAndMemoryTitleStyle());

    ui->f1Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f2Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f3Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f4Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f5Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f6Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f7Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f8Label->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->f9Label->setStyleSheet(StyleHelper::getLabelsStyle());

    ui->qLabel->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->wLabel->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->eLabel->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->rLabel->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->tLabel->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->yLabel->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->uLabel->setStyleSheet(StyleHelper::getLabelsStyle());
    ui->iLabel->setStyleSheet(StyleHelper::getLabelsStyle());

    //Встановлюю стиль для віконця частоти
    ui->freqComboBox->setStyleSheet(StyleHelper::getFrequencyStyle());
    QListView* listView = qobject_cast<QListView*>(ui->freqComboBox->view());
    if (listView) {
        listView->setStyleSheet("background-color: black; "
                                "selection-background-color: blue;"
                                "color: #55FFFF;"
                                "border: none");
    }
    ui->freqComboBox->addItem(QString("1"));
    ui->freqComboBox->addItem(QString("10"));
    ui->freqComboBox->addItem(QString("100"));
    ui->freqComboBox->addItem(QString("1000"));
}
```

```
void Window::setTableProperties()
{
    //Встановлюю кількість рядків та стовпців
    ui->memoryTable->setRowCount(32);
    ui->memoryTable->setColumnCount(8);
    //Вимикаю сітку таблиці
    ui->memoryTable->setShowGrid(false);
    //Встановлюю користувацький делегат для відображення елементів, а також для редагування комірок
    ui->memoryTable->setItemDelegate(new MyDelegate);
    //Встановлюю режим виділення комірок
    ui->memoryTable->setSelectionMode(QAbstractItemView::SingleSelection);
    //Налаштовую розміри горизонтального хедеру
    ui->memoryTable->horizontalHeader()->setVisible(false);
    ui->memoryTable->horizontalHeader()->setDefaultSectionSize(30);
    ui->memoryTable->horizontalHeader()->setHighlightSections(false);
    //Налаштовую розміри вертикального хедеру
    ui->memoryTable->verticalHeader()->setVisible(true);
    ui->memoryTable->verticalHeader()->setDefaultSectionSize(19);
    ui->memoryTable->verticalHeader()->setMinimumWidth(30);
    ui->memoryTable->verticalHeader()->setHighlightSections(false);
    //Змінюю стиль вертикального хедеру, а також встановлюю фіксований розмір
    QHeaderView *header = ui->memoryTable->verticalHeader();
    header->setStyleSheet(StyleHelper::getVerticalHeaderItemStyle());
    header->setSectionResizeMode(QHeaderView::Fixed);
    ui->memoryTable->setVerticalHeader(header);
    //Замінюю елементи таблиці на створені, а також змінюю елементи вертикального хедеру
    QTableWidgetItem *item;
    QString str = "";
    int value = 0;
    for(int row = 0; row < ui->memoryTable->rowCount(); row++){
        str = QString("%1").arg(value, 2, 16, QChar('0'));
        str = str.toUpper();
        item = new QTableWidgetItem(str);
        item->setTextAlignment(Qt::AlignCenter);
        ui->memoryTable->setVerticalHeaderItem(row, item);
        value += 8;
        for(int col = 0; col < ui->memoryTable->columnCount(); col++){
            item = new QTableWidgetItem("00");
            item->setTextAlignment(Qt::AlignCenter);
            if(row == 0){
                item->setForeground(QColor(QString("#FFFFFF")));
            }else item->setForeground(QColor(QString("#555555")));
            item->setBackground(Qt::black);
            item->setFlags(item->flags() & ~Qt::ItemIsEditable);
            ui->memoryTable->setItem(row, col, item);
        }
    }
}
```

```
void Window::changeInMemory(int &row, int &col)
{
    col = secFunc->getPc() % 8;
    row = secFunc->getPc() / 8;
    if(!secFunc->getIsBigMemory() && (row > 0)){
        row = 0;
    }
    ui->memoryTable->setCurrentCell(row, col);
}
```

```
void Window::on_stepButton_clicked()
{
    if(ui->memoryTable->selectedItems().isEmpty() || resClicked || secFunc->getRunClicked())
        return;
    else ui->memoryTable->setCurrentItem(ui->memoryTable->selectedItems().at(0));
    stepClicked = !stepClicked;
    bool ok;
    std::bitset<8> bits;
    QTableWidgetItem *item = ui->memoryTable->currentItem();
    int rows = item->row();
    int columns = item->column();
    QPalette palette = ui->memoryTable->palette();
    if(stepClicked){
        secFunc->setValOfCell(item->text().toInt(&ok, 16));
        ui->stepButton->setStyleSheet(StyleHelper::getButtonClickedStyle());
        secFunc->setColors(46, QColor(QString("#55FFFF")));
        switch(step){
        //1 крок циклу: виділяю необхідні провідники, на шину даних виставляю значення з пам'яті
        case 1:
            for(int i = 0; i < 46; i++){
                if(i == 7 || i == 26){
                    secFunc->setColors(i, QColor(QString("#55FF55")));
                } else if(i == 8 || i == 13){
                    secFunc->setColors(i, QColor(QString("#FFFFFF")));
                } else if(i == 29){
                    secFunc->setColors(i, QColor(QString("#FF5555")));
                } else {
                    secFunc->setColors(i, QColor(QString("#555555")));
                }
            }
            secFunc->setColors(62, QColor(QString("#FFFF55")));
            secFunc->setBusText(QString("%1").arg(secFunc->getValOfCell(), 8, 2, QChar('0')));
            value = 0;
            // Встановлюємо колір тексту виділених комірок на білий
            palette.setColor(QPalette::HighlightedText, Qt::white);
            // Встановлюємо колір фону виділених комірок на зелений
            palette.setColor(QPalette::Highlight, QColor(QString("#00AA00")));
            // Встановлюємо палітру
            ui->memoryTable->setPalette(palette);
            update(32, 35, 740, 680);
            step++;
            break;
        //2 крок циклу: змінюю необхідні провідники, заношу значення з шини даних до регістра IR, підсвічую необхідні провідники
        //білим кольором.
        case 2:
            secFunc->setColors(29, QColor(QString("#555555")));
            secFunc->setColors(48, QColor(QString("#FF5555")));

            bits = std::bitset<8>(secFunc->getValOfCell());
            secFunc->lineSelection(bits);
            secFunc->setIrVal(secFunc->getValOfCell());
            secFunc->setIrText(QString("%1").arg(secFunc->getValOfCell(), 8, 2, QChar('0')));
            value += 15.5;
            // Встановлюємо колір тексту виділених комірок на білий
            palette.setColor(QPalette::HighlightedText, Qt::white);
            // Встановлюємо колір фону виділених комірок на зелений
            palette.setColor(QPalette::Highlight, QColor(QString("#00AA00")));
            // Встановлюємо палітру
            ui->memoryTable->setPalette(palette);
            update(32, 35, 740, 680);
            step++;
            break;
        //3 крок циклу: всі провідники, окрім тих, що йдуть від регістру IR, залишаю без виділення, очищую шину даних.
        case 3:
            for(int i = 0; i < 46; i++){
                if((i < 7) || (i > 9 && i < 12) || (i > 15 && i < 18))
                    continue;
                else
                    secFunc->setColors(i, QColor(QString("#555555")));
            }
```

```
        secFunc->setColors(62, QColor(QString("#000000")));
        secFunc->setBusText(secFunc->getEmptyBus());
        rdcFlag = wdcFlag = true;
        value += 15.5;
        // Встановлюємо колір тексту виділених комірок на білий
        palette.setColor(QPalette::HighlightedText, Qt::white);
        // Встановлюємо колір фону виділених комірок на блакитний
        palette.setColor(QPalette::Highlight, Qt::blue);
        // Устанавливаем палитру
        ui->memoryTable->setPalette(palette);
        update(32, 35, 740, 680);
        step++;
        break;
    //4 крок циклу: починаю аналізувати число в регістрі IR: якщо команда двобайтна, то збільшую значення регістра PC
    //(лічильник команд), тим самим виділяю наступну комірку пам'яті. В іншому випадку, лише виділяю лінію IC генератора.
    case 4:
        if((secFunc->getIrVal() >= 64 && secFunc->getIrVal() < 68) || (secFunc->getIrVal() == 128) ||
                (secFunc->getIrVal() >= 192 && secFunc->getIrVal() < 196)){                                    //Якщо команда двойбайтна
            secFunc->incPC();
            changeInMemory(rows, columns);
            secFunc->setColors(13, QColor(QString("#FFFFFF")));
            secFunc->setColors(28, QColor(QString("#FFFFFF")));
            secFunc->setColors(61, QColor(QString("#FF55FF")));
        }
        secFunc->setColors(9, QColor(QString("#FFFFFF")));
        value += 15.5;
        update(32, 35, 640, 680);
        step++;
        break;
    //5 крок циклу: продовжую аналіз числа з регістра IR: якщо команда двобайтна, то на шину даних виставляю значення з комірки
    //пам'яті та змінюю кольори необхідних провідників; окрім цього перевіряю, до якого типу команд відноситься обрана команда.
    //Якщо ж команда однобайтна, то аналізую, що буде джерелом даних, а що отримувачем.
    case 5:
        if((secFunc->getIrVal() < 64 && secFunc->getIrVal() >= 32) || secFunc->getIrVal() < 16){
            addrRD = (int)(secFunc->getIrVal() / 4);
            secFunc->setBusText(secFunc->analyzeSource(0, addrRD));
        } else {
            // Встановлюємо колір тексту виділених комірок на білий
            palette.setColor(QPalette::HighlightedText, Qt::white);
            // Встановлюємо колір фону виділених комірок на зелений
            palette.setColor(QPalette::Highlight, QColor(QString("#00AA00")));
            // Встановлюємо палітру
            ui->memoryTable->setPalette(palette);
            if(secFunc->getIrVal() >= 64 && secFunc->getIrVal() < 68){
                secFunc->setBusText(secFunc->analyzeSource(1, secFunc->getValOfCell()));
            } else if(secFunc->getIrVal() == 128){
                secFunc->setBusText(secFunc->analyzeSource(2, secFunc->getValOfCell()));

            } else if(secFunc->getIrVal() >= 192 && secFunc->getIrVal() < 196){
                secFunc->setBusText(secFunc->analyzeSource(3, secFunc->getValOfCell()));
            }
        }
        if(secFunc->getFc()){
            secFunc->setColors(14, QColor(QString("#FFFFFF")));
        }
        if(secFunc->getFz()){
            secFunc->setColors(15, QColor(QString("#FFFFFF")));
        }
        secFunc->setColors(18, QColor(QString("#55FF55")));
        secFunc->setColors(19, QColor(QString("#FF5555")));
        secFunc->setColors(9, QColor(QString("#555555")));
        value += 15.5;
        update(32, 35, 725, 680);
        step++;
        break;
    case 6:
        addrRD = secFunc->getIrVal() % 4;
        if((secFunc->getIrVal() < 64 && secFunc->getIrVal() >= 32) || secFunc->getIrVal() < 16){
            secFunc->analyzeReceiver(0, addrRD, secFunc->stringToIntReg(secFunc->getBusText()));
        } else if(secFunc->getIrVal() >= 64 && secFunc->getIrVal() < 68){
            secFunc->analyzeReceiver(1, addrRD, secFunc->getValOfCell());
        } else if(secFunc->getIrVal() == 128){
            secFunc->analyzeReceiver(2, addrRD, secFunc->getValOfCell());
        } else if(secFunc->getIrVal() >= 192 && secFunc->getIrVal() < 196){
            secFunc->analyzeReceiver(3, addrRD, secFunc->getValOfCell());
        }
```

```
            numOne = secFunc->stringToIntReg(secFunc->getRegA());
            secFunc->setFz(((numOne == 0) ? true : false));
            numTwo = secFunc->stringToIntReg(secFunc->getRegB());
            secFunc->changeALU(numOne, numTwo);
            secFunc->setColors(19, QColor(QString("#555555")));
            value += 15.5;
            update(32, 35, 640, 680);
            step++;
            break;
        case 7:
            for(int i = 32; i < 61; i++){
                if(i == 32 || (i > 34 && i < 46)){
                    secFunc->setColors(i, QColor(QString("#555555")));
                } else if(i > 48){
                    secFunc->setColors(i, QColor(QString("#000000")));
                }
            }
            secFunc->setSourceReg(!secFunc->getSourceReg());
            changeInMemory(rows, columns);
            secFunc->setBusText(secFunc->getEmptyBus());
            secFunc->setColors(8, QColor(QString("#FFFFFF")));
            secFunc->setColors(12, QColor(QString("#555555")));
            secFunc->setColors(13, QColor(QString("#FFFFFF")));
            secFunc->setColors(18, QColor(QString("#555555")));
            secFunc->setColors(26, QColor(QString("#555555")));
            secFunc->setColors(62, QColor(QString("#000000")));
            secFunc->setColors(63, QColor(QString("#000000")));
            wdcFlag = rdcFlag = false;
            value += 15.5;
            // Встановлюємо колір тексту виділених комірок на білий
            palette.setColor(QPalette::HighlightedText, Qt::white);
            // Встановлюємо колір фону виділених комірок на блакитний
            palette.setColor(QPalette::Highlight, Qt::blue);
            // Встановлюємо палітру
            ui->memoryTable->setPalette(palette);
            update(32, 35, 725, 680);
            step++;
            break;
        default:
            if(secFunc->getIrVal() >= 192 && secFunc->getIrVal() < 196){
                addrRD = secFunc->getIrVal() % 4;
                if(!((addrRD == 0 && secFunc->getFz()) || (addrRD == 1 && !secFunc->getFz()) ||
                     (addrRD == 2 && secFunc->getFc()) || (addrRD == 3 && !secFunc->getFc()))){
                    secFunc->incPC();
                }
            } else if(secFunc->getIrVal() != 128){
                secFunc->incPC();
            }

                changeInMemory(rows, columns);
                secFunc->setColors(9, QColor(QString("#FFFFFF")));
                secFunc->setColors(28, QColor(QString("#FFFFFF")));
                value += 15.5;
                update(32, 35, 640, 680);
                step = 1;
                break;
            }
        }else{
            ui->stepButton->setStyleSheet(StyleHelper::getButtonDefaultStyle());
            secFunc->setColors(46, QColor(QString("#0000FF")));
            secFunc->setColors(48, QColor(QString("#000000")));
            for(int i = 57; i < 61; i++){
                if(secFunc->getColors(i) == QColor(QString("#FF5555"))){
                    if(secFunc->getSourceReg())
                        secFunc->setColors(i, QColor(QString("#00AA00")));
                    else secFunc->setColors(i, QColor(QString("#000000")));
                }
            }
            secFunc->setColors(61, QColor(QString("#000000")));
            update(32, 35, 640, 710);
        }
    }
```