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: «**Створення бібліотеки навчальних програм для моделі процесора DPM08**»

Захищено на засіданні
Атестаційної комісії № 40
протокол № __ від __.06.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ **СКОБ Ю.О.**

Виконав:
студент 4 курсу, групи КІ–41
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 – Комп'ютерна

інженерія.
ПАЛІЙ Світодар Сергійович

Керівник: к.т.н., доцент
РЕВА Сергій Миколайович

Рецензент: ст. викладач кафедри штучного
інтелекту та програмного

забезпечення

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, 3 розділів та висновків по них, загальних висновків, списку використаних джерел та додатків. Основний текст роботи викладений на 30 сторінках, містить 5 рисунків, список джерел з 10 найменувань. Загальний обсяг роботи, враховуючи додатки, складає 43 аркуші.

Метою роботи є вдосконалення навчального процесу на факультеті комп'ютерних наук та покращення знань студентів при вивченні основ мікропроцесорної техніки та основ програмування

Об'єктом дослідження є лабораторна модель цифрового процесора DPM08 та її використання у навчальному процесі.

Предметом дослідження є архітектура, програмні та апаратні можливості лабораторної моделі процесора DPM08.

Область застосування – Покращення навчального процесу. Розроблена бібліотека завдань допоможе викладачам в організації індивідуальної роботи студентів групи та простимулює серед студентів зацікавленість предметом та самостійний пошук нестандартних рішень.

Ключові слова: мікропроцесор, DPM08, машинні коди, Асемблер.

ABSTRACT

The explanatory note to the bachelor's thesis consists of an introduction, 3 chapters and their conclusions, general conclusions, a list of used sources and appendices. The main text of the work is laid out on 30 pages, contains 5 figures, a list of sources with 10 clauses. The total volume of work, including the appendices, is 43 pages.

The purpose of the work is to improve the educational process at the Faculty of Computer Sciences and improve the knowledge of students when studying the basics of microprocessor technology and the basics of programming

The **object** of research is the laboratory model of the DPM08 digital processor and its use in the educational process.

The **subject** of research is the architecture, software and hardware capabilities of the laboratory model of the DPM08 processor.

Application area - Improvement of the educational process. The developed task library will help teachers in organizing the individual work of group students and stimulate interest in the subject and independent search for non-standard solutions among students.

Keywords: microprocessor, DPM08, machine codes, Assembler.

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1. АРХІТЕКТУРА МОДЕЛІ ПРОЦЕСОРА DPM08 7

1.1. Історія створення, аналоги 7

1.2. Опис та функціонал архітектури 8

1.3. Наявні програмні емулятори 13

Висновки до розділу 1 14

РОЗДІЛ 2. АНАЛІЗ АПАРАТНИХ ТА ПРОГРАМНИХ

МОЖЛИВОСТЕЙ МОДЕЛІ 15

2.1. Апаратно-програмні можливості 16

2.2. Аналіз системи команд 17

Висновки до розділу 2 18

РОЗДІЛ 3. РОЗРОБКА БІБЛІОТЕКИ НАВЧАЛЬНИХ ПРОГРАМ 20

3.1. Систематизація завдань за цільовим використанням

та складністю 20

3.2. Розроблені завдання для програмування у машинних кодах 21

3.3. Розроблені завдання для програмування на Асемблері 24

Висновки до розділу 3 30

ВИСНОВКИ 31

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 33

ДОДАТКИ 34

ВСТУП

Актуальність роботи

Через високу популярність IT-напрямку серед молоді, велика кількість студентів очікувано вибирає пов'язані з комп'ютерними науками спеціальності, пріоритетною серед яких є вивчення різноманітних мов програмування для подальшого «швидкого старту» у даній галузі. Щоправда, такий підхід часто позбавляє можливості глибше зрозуміти принципи роботи сучасної комп'ютерної техніки на нижчих рівнях, починаючи з апаратного, що в результаті може викликати неочікувані проблеми, пов'язані з відсутністю розуміння основних принципів роботи тих пристроїв, для яких пишуться програми. Навчальні курси «Мікропроцесори та їх застосування», та «Основи комп'ютерної схемотехніки» покликані виправити цей недолік у системності навчання стосовно даної предметної області, Для вдосконалення навчального процесу, було прийнято рішення розробити повноцінний збірник навчальних програм, їх алгоритмічних рішень та методичних рекомендацій, що стане в нагоді як викладачу, який зможе скористатись великою кількістю підготовлених завдань для лабораторних чи практичних робіт і в разі потреби легко перевірити надані розв'язки, користуючись запропонованими у збірнику, так і студенту, який матиме доступ до набору чітко сформульованих задач, що значно полегшить процес виконання навчальних робіт та зекономить час.

Метою роботи є вдосконалення навчального процесу на факультеті комп'ютерних наук та покращення знань студентів при вивченні основ мікропроцесорної техніки та основ програмування.

Для досягнення поставленої мети у роботі необхідно вирішити такі **задачі**:

- провести аналіз архітектури моделі процесора DPM08 та її системи команд;
- проаналізувати апаратні та програмні можливості моделі процесора в

плані створення повчальних, цікавих та наочних програмних завдань

- розробити збірник завдань для програмування у машинних кодах та на мові Асемблер; та провести їх тестування;

- розробити методичні рекомендації щодо використання цих завдань та скласти довідник викладача зі зразками їх рішень.

Об'єктом дослідження є лабораторна модель цифрового процесора DPM08 та її використання у навчальному процесі.

Предметом дослідження є архітектура, програмні та апаратні можливості лабораторної моделі процесора DPM08.

Методами досліджень є аналіз, синтез, абстрагування, пояснення, класифікація, експеримент, опис.

Інформаційною базою досліджень є основні положення архітектури мікропроцесорної техніки, теорія алгоритмів, відкриті джерела Internet, наукові та технічні статті, документація.

Практичне значення отриманих результатів

Вирішення поставлених у роботі завдань дасть змогу використати отримані результати для підготовки та проведення лабораторних занять при вивченні основ мікропроцесорної техніки та основ програмування. Результати роботи можуть бути спрямовані на підвищення рівня зацікавленості та заохочення студентів, а також на покращення рівня їх професійних знань.

РОЗДІЛ 1

АРХІТЕКТУРА МОДЕЛІ ПРОЦЕСОРА DPM08

У 2010-х роках, викладачами кафедри електроніки та управляючих систем за участі студентів-дипломників старших курсів був проведений всебічний аналіз проблеми і, відповідно, розроблена ця модель цифрового процесора, що призначена виключно для навчальних цілей. Модель є наочною, легкою для розуміння та зручною для виконання різнотипних навчальних завдань, тому продовжує застосовуватись в ході вивчення дисциплін «Мікропроцесори та їх застосування» й «Архітектура комп'ютерів». Аналогічні розробки оцінювати важко через вкрай малу поширеність застосування подібного методу вивчення мікропроцесорної техніки у навчальних закладах нашої держави (винятком є призначена для вивчення подібних курсів модель універсальної навчальної мікроконтролерної системи УНМС-2, розробки Харківського авіакосмічного університету, яка хоч і є повністю іншою за принципами роботи, однак призначена для подібних цілей), хоча цілком ймовірно, що у зарубіжних навчальних закладах застосовуються подібні за призначенням демонстраційні моделі.

Перейдемо до архітектури описуваної моделі. Для успішного використання в межах навчального процесу, модель цифрового процесора окрім функціональності також має бути наочною та чітко відображати візуально більшість внутрішніх процесів для кращого розуміння студентами загального принципу дії мікропроцесора. Виходячи з цих вимог, розробники моделі зіткнулися зі значною кількістю обмежень стосовно свободи вибору архітектурних принципів побудови моделі процесора.

Рис. 1.1 — Структурна схема моделі процесора.[1]

Вище наведена структурна схема моделі, на якій можна побачити взаємозв'язок основних її компонентів. Зокрема, був застосований принцип багататактного машинного циклу, при якому команда виконуватиметься процесором за кілька періодів тактової частоти, до того ж забезпечуючи виконання не більше ніж однієї конкретної дії за такт. Так як команди мають виконуватись послідовно, на завершальних етапах виконання попередньої команди відбувається інкремент лічильника команд, що дозволяє сформувати адресу наступної команди.

Враховуючи потребу у наглядному відображенні процесів, що відбуваються в ході виконання команд, модель ядра була сформована таким чином, що виконання будь-якої команди можна примітивно описати як перезапис певних даних з джерела у приймач, де в якості джерела можуть виступати регістри, пам'ять чи АЛП, а приймачем слугує регістр ядра. При такому підході, доволі зручно буде відображати усі зміни станів елементів процесора в ході покрокового виконання навчальної програми. Сам машинний цикл займає 8 тактів і містить наступні сигнали[1]:

Рис.1.2 — сигнали машинного циклу [1]

Як зображено на схемі, за 8 тактів має відбутись наступне:

Необхідно дозволити зчитування пам’яті задля отримання даних з тієї лунки, адреса якої виставлена на шині адрес. Для цього формується низький логічний рівень сигналу ReadMemory (RM).

Потрібно записати отриману інструкцію у регістр команд шляхом встановлення високого логічного рівня сигналу WriteCommand (WC). Він активується на один період пізніше через перехідні процеси на шині даних, що мають завершитись до його початку. Після цього, шину даних потрібно звільнити для подальших операцій з нею – тому RM повертається назад на високий рівень.

Відразу після запису команди, на третьому такті, необхідно дозволити передачу даних між джерелом та приймачем. Для цього існує сигнал

EnableCommand (EC), який активується низьким логічним рівнем.

Наступний сигнал відрізняється тим, що його поява залежить від типу команди. Якщо вона однобайтова, то сигнал IC маскується і не впливає на процес виконання команди протягом того часу, поки активний EC, а якщо двобайтова – то виконує інкремент лічильника команд, формуючи адресу наступної лунки пам'яті з командою на шині адрес.

Під час низького рівня сигналу EC, на п'ятому такті активуються два наступні сигнали – дозвіл передавати дані з джерела на шину даних EnableData (ED) та увімкнення дешифраторів сигналу запису в регістри-приймачі SC (Strobe Command). Перший діє на шостому такті теж, другий спадає по завершенню п'ятого.

У період між спадами SC та ED відповідно і відбувається сам процес перезапису даних, а наприкінці шостого такту спад ED звільняє шину даних.

Завершується цикл другим імпульсом (перший може бути замаскованим якщо команда однобайтова, другий – ніколи) IC, який проводить інкремент лічильника команд та по завершенню циклу виставляє адресу наступної команди на шину адреси, якщо виконана команда не була командою передачі управління.

Щоб коректно сформувати вищенаведені сигнали, необхідний такий елемент структури, як генератор машинного циклу – пристрій, подібний до мікропрограмного автомата, що пропускає через себе тактові імпульси і за допомогою лічильника, дешифратора, логічних елементів та тригерів формує відповідні сигнали у правильній послідовності.

Лічильник команд є восьмирозрядним і здатний до паралельного запису даних, формуючи своїми сигналами шину адреси. Регістр команд також восьмирозрядний і призначений для тимчасового утримування інструкцій, а системний дешифратор, як вже було згадано вище, аналізує отриману команду та направляє сигнали читання RD і запису WD до відповідних джерел та приймачів даних. Шина даних є двонаправленою та об'єднує лічильник команд,

зовнішні інтерфейси та регістри ядра, шина управління однонаправлено передає сигнали машинного циклу до всіх приймачів, а шина команд робить те ж саме з інструкцією, яку передає на регістри ядра, дешифратор та інтерфейс підключення персонального комп'ютера відповідно. Окремим пристроєм є генератор тактових імпульсів, що надходять на вхід генератора сигналів машинного циклу. Окрім, власне, генератора імпульсів, цей модуль має генератор сигналу перезапуску (Reset). Щоб керування пристроєм було зручним та наочним, генератор має кілька режимів. Одиничне натискання кнопки S1 може генерувати імпульс. Автоматичний генератор на основі логічних елементів може робити це постійно, а зовнішній сигнал CLK-IBM, може надходити з персонального комп'ютера. Вибір режиму роботи здійснюється за допомогою двох перемикачів, а інтерфейс взаємодії з ПК дозволяє контролювати усі сигнали процесора за допомогою зовнішнього комп'ютера.

Арифметичні та логічні операції забезпечуються блоком АЛП як процес перезапису даних з функціональних вузлів в певний регістр. Щоправда, він приймає дані виключно з регістрів A та B, до яких підключений, так як ці регістри мають додатковий вихід даних на спеціальні додаткові шини, по яких дані надходять в АЛП. Інформація з регістрів присутня на додаткових шинах весь час виконання програми, незалежно від стадії машинного циклу.

Щодо обмежень такої схеми, можна згадати про відсутність стеку та системи переривань, що унеможливлює роботу з ними та дещо ускладнює (але в той же час, спрощує шляхом невикористання складніших елементів процесора) процес вирішення поставлених задач.

Перейдемо до принципів утворення команд. Кожна командна інструкція є двійковим числом, у якому 8 розрядів, що визначають стани сигналів на командній шині. Кожен біт має власне призначення стосовно виконуваної операції. Більш докладно це показано у таблиці 1.3

Таблиця 1.3

Структура командного слова[1]

Командне слово має три функціональні частини. Два старші біти СМ7 і СМ6 (в табл. 1.3 виділені червоним кольором) вказують на тип виконуваної команди. Всього цих типів чотири: нульовий, перший, другий та третій.

Команди нульового типу призначені для перезапису даних між регістрами оперативного призначення. Адреса джерела визначається бітами СМ5..СМ2, а отримувача — бітами СМ1 та СМ0 відповідно. Цей тип команд розподіляється на дві групи:

1. Якщо біт командного слова СМ5=0, то джерелом даних буде один із регістрів оперативного призначення. Його адреса в цьому випадку вказується в бітах СМ3, СМ2,а біт СМ4 ігнорується дешифратором і його значення неважливе.

2. Якщо біт СМ5=1, то джерелом даних буде певний вузол АЛП. а отримувачем — регістр оперативного призначення. Номер потрібної функції АЛП, з восьми існуючих, вказується в цьому випадку в бітах СМ4, СМ3, СМ2.

Команди першого типу перезаписують значення параметра двобайтової команди до одного із регістрів оперативного призначення, адреса якого вказана бітами СМ1, СМ0, як і в командах нульового типу.

Команда другого типу лише одна, це команда безумовного переходу. Вона завжди передає управління на адресу, яка вказана в наступному байті — параметрі команди. Технічно вона записує певне число (другий байт) у лічильник команд.

Команди третього типу також є двобайтовими командами передачі управління і виконуються так само, як і команда другого типу, але тільки при виконанні умови, яка розміщена в бітах СМ1, СМ0:

– 00 — якщо вміст регістру А дорівнює нулю;

– 01 — якщо вміст регістру А відмінний від нуля;

– 10 — якщо встановлено біт переносу;

– 11 — якщо біт переносу дорівнює нулю.

Також модель забезпечена програмними емуляторами, що можуть бути запущені на ПК студента і повністю повторюють її принципи роботи а також надають візуалізацію процесів, які відбуваються у генераторі машинного циклу, що сприяє кращому розумінню принципів роботи моделі. Станом на тепер, існує два емулятори, розроблені викладачами та студентами кафедри – для DOS та Windows відповідно. Вони характеризуються чітким відображенням виконуваних процесів і загальною наочністю. Щоправда, емулятор для DOS має певні обмеження у вигляді відображуваного обсягу пам’яті (160 байт замість 256 наявних) а також не обладнаний захистом від некоректного вводу, через що може припиняти свою роботу в деяких випадках. Версія для Windows просуває наочність ще далі а також позбавлена недоліків попередника, а ще здатна регулювати частоту віртуального «процесора».

Рис.1.4 – емулятор Emy-v2

Рис.1.5 – емулятор Emy-v3

Висновки за розділом 1

Взявши до уваги попередню інформацію, зазначу, що модель процесора здатна виконувати 54 різні команди, серед яких прості команди перезапису даних, унарні логічні команди для операцій над вмістом регістру A, бінарні логічні команди для роботи з даними у регістрах A та B, єдина арифметична команда суми, команди безумовного та умовних переходів для передачі управління. Тим не менш, цих функціональних можливостей достатньо для виконання широкого спектру різних завдань в умовах обмеженої пам'яті, до того ж сам процес є наочним та зрозумілим. Усі шини та регістри обладнані світлодіодними індикаторами для відображення процесів, які в них відбуваються, а потенціал апаратних можливостей може бути додатково розширений зовнішніми пристроями, що можуть бути під'єднані до регістрів-портів C та D.

Отже, архітектура моделі та її програмні можливості є повністю спрямованими на досягнення максимально ефективного навчання студентів шляхом виконання ними різноманітних завдань. Ці завдання знайомитимуть з принципами роботи окремих елементів процесора, вироблятимуть розуміння побудови алгоритмічних рішень та навчатимуть створенню оптимальних алгоритмів для вирішення задач.

Підсумовуючи, можна зробити висновок, що наявна архітектура моделі цифрового процесора DPM08 відповідає поставленим перед нею задачам, а саме дозволяє використовувати її в якості навчального засобу, так як вона характеризується наочним принципом формування управляючих сигналів, має інтуїтивну та чітку структуру командного слова, що є зрозумілою для студента. Модель процесора може бути застосована для виконання програми як через операції з модулем пам'яті об'ємом 8 байт, так і з пам'яттю максимального об'єму (256 байт), щоправда в цьому випадку необхідно підключити

персональний комп’ютер через інтерфейсний модуль для завантаження програм у пам’ять. Також вона надає увесь необхідний спектр можливостей для освоєння навчальних дисциплін при мінімальних потребах для роботи.

РОЗДІЛ 2

АНАЛІЗ АПАРАТНИХ ТА ПРОГРАМНИХ МОЖЛИВОСТЕЙ МОДЕЛІ

Перш ніж аналізувати розроблені навчальні програми, необхідно чітко описати наявні можливості моделі процесора саме в світлі виконуваних задач і визначити, яким чином вони впливатимуть на процес створення програм. Почнемо з огляду апаратних можливостей пристрою.

Архітектура та детальний опис процесів, що відбуваються у вузлах моделі під час її роботи, були описані у попередньому розділі, тому перейдемо одразу до аналізу тих елементів, які впливають на процес виконання навчальних завдань. Перш за все, варто відзначити, що усі основні елементи моделі оснащені світлодіодними індикаторами з метою відображення внутрішніх процесів для студентів у наочній формі. Зокрема, усі чотири блоки регістрів оперативного призначення містять по 8 світлодіодів, під'єднаних таким чином, що їх стан відповідає значенню відповідного біта у регістрі (світиться – одиниця, не світиться – нуль). Шини даних мають спеціальний модуль індикації, який під'єднується через інтерфейсний порт. З його допомогою візуалізуються ті процеси, які відбуваються на конкретній шині під час роботи програми. Це дає змогу у режимі реального часу контролювати процеси, які відбуваються у регістрах оперативного призначення та усіх шинах, і на основі спостережуваних результатів відповідно вносити правки у програмний код. Також, саме на наявності світлодіодів у складі регістрів засновані ті варіанти програм, які мають на меті створення анімації, що виводитиметься на ці світлодіодні дисплеї у складі регістрів.

Регістри C та D виконані у вигляді двосторонніх портів, завдяки чому дозволяють підключати до них зовнішні пристрої. У поточній конфігурації, до регістру оперативного призначення D під'єднана клавіатура – невеликий пристрій, який дозволяє шляхом натискання будь-якої з восьми кнопок

сформувати та вивести на порт D певне значення. Варто зауважити, що для роботи з клавіатурою, ті біти, які в ході виконання програми матимуть можливість зміни стану з її допомогою, повинні бути завчасно встановлені у логічні одиниці. Одначе, до регістру-порту можна під'єднати не лише її, наприклад, існує свого часу розроблений студентами зовнішній модуль оперативної пам'яті об'ємом 32 кілобайти. Його можна під'єднати до моделі та використовувати за допомогою звернень до нього через певні процедури програми на Асемблері. Ще є можливість приєднати стандартний рідкокристалічний дисплей, на який можна виводити текстову та числову інформацію.

Окрім прямої роботи з пам'яттю шляхом встановлення окремих бітів на 8-байтному модулі, вводити програми, необхідні для виконання моделлю, та контролювати процес їх виконання, можливо і за допомогою персонального комп'ютера. Для цього призначено спеціальний порт, через який модель можна під'єднати до ПК, на якому має бути встановлена управляюча програма. Вона дозволить редагувати пам'ять у будь-якому об'ємі до 256 байт включно та контролювати роботу процесора.

Варто згадати і про апаратні обмеження стосовно можливих виконуваних програм. Так, враховуючи що АЛП під'єднаний лише до регістрів оперативного призначення A та B, очевидно, що арифметичні та логічні операції можуть виконуватись лише над даними у цих регістрах. В той час як значення регістрів C та D можуть лише перезаписуватись та очищатись. Слід зауважити, що унарні логічні операції (наприклад, лівий та правий зсув) можуть виконуватись ВИКЛЮЧНО над вмістом регістру A. Через необхідність створення спеціальних додаткових шин для дублювання цих функцій на регістр B, це викликало б цілий ряд ускладнень архітектури, зокрема зміни конструкції дешифратору команд, додавання нових шинних формувачів і тому подібне.

Перейдемо до програмних можливостей моделі процесора та їх

застосуванню. Використовуючи наявну систему команд, можливо побудувати доволі цікаві програми, навіть в межах 8-байтної пам'яті. Зокрема, правильно комбінуючи арифметичну та логічні команди з командами перезапису та передачі управління, засобами наявного функціоналу можливо виконати навіть такі, здавалося б, просунуті задачі як множення чисел, створення різноманітних анімацій на індикаторах та аналіз введених даних, наприклад на відношення чисел. Також, настільки жорстке обмеження у 8 байт привчає студентів до розробки чітких алгоритмічних рішень, за допомогою яких можна вписати у цей об'єм доволі самодостатні та цікаві програми, а часом стимулює нешаблонне мислення, яке призводить до появи унікальних варіантів розв'язку поставлених задач. До того ж, вищезгадані світлодіодні індикатори станів дають змогу відлагодити програму, визначивши, які кроки виконуються вірно і коли виникає (якщо виникає) помилка у виконанні.

Що ж стосується тієї частини завдань, яка має на меті роботу з мовою Асемблер та 256-байтним модулем пам'яті, перш за все варто зазначити, що для коректного завантаження таких програм у пам'ять, необхідно розуміти принципи трансляції та утворення об'єктних файлів, а також вміти працювати з транслятором (у розглянутому випадку це транслятор TASM). Таким чином, одна з основних переваг роботи з моделлю полягає у кращому розумінні взаємозв'язку між машинними кодами та позначеннями Асемблера. Так як студенти створюють свої власні системи позначень для підтримуваних процесором команд, це дає змогу транслятору працювати з програмами, написаними на тому варіанті Асемблера, що був створений студентом. Це сприяє кращому засвоєнню системи команд та її мнемонічних кодів. Отже, студент зможе отримати досвід розробки алгоритмів та програмування на Асемблері, використовуючи ним же створену систему мнемонічних позначень для команд процесора. Більше того – запропонований для використання в ході навчального процесу транслятор TASM підтримує роботу з макрокомандами. Завдяки цьому, студенти мають можливість отримати навички програмування на

мові Макроасемблер, усвідомлюючи зв'язок між низьким та високим рівнями програмування. Хоча система команд і не передбачає виклику процедур, однак засобами Асемблера це можливо зробити. Завдяки цьому, деякі завдання другої частини можуть мати цікаві програмні рішення з використанням технології процедурно-орієнтованого, або навіть об'єктно-орієнтованого програмування.

Висновки за розділом 2

Розглянувши відповідні програмні та апаратні можливості моделі, можна зробити висновок, що наявні функціональні можливості, незважаючи на певну обмеженість, цілком відповідають призначенню — допомогти в розумінні принципів дії пристроїв мікропроцесорної техніки та отриманні навичок програмування на мовах низького рівня, починаючи з машинних кодів, і закінчуючи рівнем Макроасемблера. Обмежена пам'ять в свою чергу вимагає нестандартного підходу та в цілому сприяє кращому розумінню алгоритмічних рішень поставлених задач. А широкі можливості по підключенню зовнішніх пристроїв та роботі з макротранслятором Асемблера відкривають шлях до нових можливостей, зокрема створення більш об'ємних програм з розгалуженими алгоритмами та уніфікованими процедурами для отримання додаткових зручностей у програмуванні.

РОЗДІЛ 3

РОЗРОБКА БІБЛІОТЕКИ НАВЧАЛЬНИХ ПРОГРАМ

Ознайомившись з архітектурою моделі процесора, її можливостях стосовно проведення навчального процесу та функціоналом для реалізації програмних рішень тих чи інших задач, перейдемо до огляду розроблених завдань, їх класифікації та прикладів.

Перш за все, запропоновані в роботі завдання можна розподілити на дві основні секції. Перша з них є комплектом задач малого об'єму для знайомства з роботою процесора, програмуванням у машинних кодах та розробкою простих алгоритмічних рішень. Друга ж представлена завданнями для програмування на мові Асемблер та призначена для кращого розуміння системи команд, сприяє отриманню досвіду роботи з Асемблером (і, у певних випадках, Макроасемблером). Вона загалом спрямована на стимуляцію знаходження більш широких та складних алгоритмічних розв'язків деяких задач в умовах наявних апаратних і програмних обмежень.

Перейдемо до аналізу першої частини, що представлена завданнями для роботи з машинними кодами. Ці програми перш за все розподілені за складністю. Необхідно врахувати, що не кожен студент може відносно швидко і якісно набути досвіду складання алгоритмів та їх оптимізації в умовах обмеженої пам'яті. З іншого боку, для деяких студентів, які виявлять інтерес до більш глибокого вивчення дисципліни, базові завдання навпаки можуть здатися примітивними й нецікавими. Отже, враховуючи ці особливості, було розроблено диференційований підхід до створення завдань для забезпечення поступового розвитку вмінь студентів. Їх було розбито на три рівні складності.

Початковий включає в себе найпростіші операції над процесором, зокрема елементарні логічні та арифметичні дії над числами, створення простих анімацій і тому подібне, причому ці завдання вимагають мінімального розуміння

системи команд і без проблем можуть бути виконані тими студентами, які ще не мають досвіду та чіткого розуміння принципів роботи мікропроцесорної техніки.

Середній рівень містить найбільше варіантів завдань і призначений для виконання основною масою студентів, які відносно добре розуміють навчальний матеріал. Завдання цього рівня є досить різноманітними та іноді вимагають нестандартного алгоритмічного рішення. В цей блок входять завдання усіх напрямків, починаючи від складніших арифметичних, логічних та комбінованих задач і закінчуючи анімаціями. Щоправда, вони потребують пошуку більш цікавих алгоритмічних рішень, але в той же час не вимагають знання архітектури процесора на глибокому рівні.

Третій блок містить завдання високого рівня, рішення яких потребує досконалого знання принципу дії процесора, його взаємодії з оперативною пам'яттю та особливостей виконання окремих команд. Самі завдання часто не є складними з точки зору алгоритмічних рішень, однак очевидна реалізація неможлива по причині жорстких обмежень пам'яті (для роботи з 8-байтним модулем), тому ці завдання потребують критичної оптимізації та використання недокументованих можливостей процесора.

Для кращого розуміння подібної системи розподілу, наведу кілька прикладів розроблених завдань з їх поясненням:

Завдання 5с. Організувати поступове наповнення регістрів C та D одиницями (від молодшого біта до старшого), після остаточного заповнення – вміст регістру очищується, причому задача має виконуватись циклічно.

Це завдання належить до середнього рівня і має на увазі послідовне зростання значень двох регістрів таким чином, що кожне наступне значення займає на біт більше (відповідно, на моделі процесора мають вмикатися світлодіоди від одного молодшого розряду до усіх восьми, залежно від етапу виконання).

Так як математичні й логічні операції можливі лише над регістрами A та B, для виконання цього завдання необхідно організувати циклічне зміщення значення A та подальше додавання до нього записаної у B одиниці, що поступово формуватиме необхідну в умові анімацію у регістрі A. Для виконання завдання достатньо просто копіювати отриманий результат у C та D. Попри застосування виключно простих команд (запис числа, лівий зсув, арифметична сума, копіювання значення та безумовний перехід), завдання може здатись складнішим через неочевидність деяких команд. Команда лівого зсуву ще пустого під час виконання першого циклу регістру A на перший погляд є недоречною, одначе без цієї операції не вдасться реалізувати необхідний цикл заповнення, через що для вірного рішення задачі необхідне чітке алгоритмічне рішення (зсув значення A –> додавання до зсунутого значення одиниці –> копіювання результату у цільові регістри –> безумовний перехід на початок). Можливим варіантом розв'язку також може здатись використання логічної суми замість арифметичної, однак цьому випадку не виконуватиметься очищення регістру. Тому завданню було присвоєно середній рівень складності – воно потребує лише розуміння принципів формування бінарних чисел та вміння складати чіткий алгоритм рішення. В той же час, задача не вимагає застосування складних команд, наприклад умовних переходів, і тим більше не потребує широких знань особливостей архітектури навчальної моделі.

Запропонований розв'язок завдання у машинних кодах (та їх шістнадцяткових значеннях для роботи з емулятором) наведено у таблиці 3.1:

Таблиця 3.1

Програмне рішення завдання 5с

Зразок рішення:

Адреса	*Код (2)*	*Код (16)*	*Дія команди*
0	01 0000 01	41	B = 1
1	00 0000 01	01	
2	00 1101 00	34	RLA, A
3	00 1111 00	3C	A = A + B
4	00 0000 10	02	C = A
5	00 0000 11	03	D = A
6	10 0000 00	80	Go to 3
7	00 0000 10	02	

У якості другого прикладу стосовно першої частини розроблених завдань, наведемо одну задачу високого рівня:

Завдання 1в. Організувати переміщення світлодіодів у протилежних напрямках по регістрах C та D.

Без детального огляду, завдання може здатися представником середнього, якщо не початкового рівнів – воно не потребує для виконання ніяких арифметичних чи бінарних логічних операцій, умовних переходів і тому подібного. Очевидне рішення виконується за допомогою різнонаправлених зсувів та копіювання значень. Одначе, при спробі вирішити його, студент наштовхнеться на, здавалося б, нездоланну перешкоду – об'єм коду. Так, програмне рішення цієї задачі навіть при проведеній оптимізації займатиме мінімум 9 байтів, а не 8, через що його коректне виконання є на перший погляд неможливим. Щоправда, якщо ввести запропоноване нижче рішення з першою половиною команди безумовного переходу, яка виходитиме за межі пам'яті, то програма працюватиме.

А все тому, що процесор при виконанні команди 80 для зчитування

параметра (тобто, адреси переходу) додає в середині циклу одиницю до регістру PC і переходить з сьомої адреси (00000111) на восьму (00001000). Але оскільки 8-байтний модуль пам'яті під'єднаний до адресної шини всього трьома проводами, її значення мало б у такому разі бути (00001 000), але вона виставляє на шину даних число з нульової адреси (бо восьмої фізично немає). І в лічильник команд завантажується адреса 01000 011 (43h). А на початку наступного машинного циклу число 43h виставляється на шину адреси для читання наступної команди. Але, оскільки 5 старших розрядів не підключено, то ця адреса сприймається пам'яттю як адреса 00000 011, або ж 3h… що і є адресою, необхідною для правильного замикання циклу. У наведеному в таблиці 3.2 програмному рішенні можна це помітити.

Таблиця 3.2

Програмне рішення завдання 1в

Зразок рішення:

Адреса	*Код (2)*	*Код (16)*	*Дія команди*
0	01 0000 11	43	D = 1
1	00 0000 01	01	
2	00 0011 10	0E	C = D
3	00 0011 00	0C	A = D
4	00 1110 11	3B	D = RRA
5	00 0010 00	08	A = C
6	00 1101 10	36	C = RLA
7	10 0000 00	80	GOTO… (3)
(0)	*0(1) 0000 11*	*(03)*	

Отже, можна зробити висновок, що для правильного рішення цього завдання необхідно не лише побудувати правильне алгоритмічне рішення, а і з самого початку його формування взяти до уваги цю неочевидну особливість роботи процесора з адресами в умовах його апаратних обмежень, і з огляду на це реалізувати рішення програми.

Переходячи до другого розділу (програм на Асемблері), щоб завершити градацію по складності, наведу приклад простого завдання з початкового рівня, що призначене для тих студентів, які ще не до кінця розібралися у принципах роботи процесора:

Завдання 2п. Розробити анімацію «змійки» – ланцюжку з чотирьох світлодіодів, які рухатимуться по «краю» регістрів. Стартову позицію та напрям руху «змійки» (за/проти годинникової стрілки) обрати самостійно.

Це завдання є максимально примітивним, так як по суті являє вертикальне переміщення «змійки» у лівому секторі за допомогою команд зсуву, а в усіх інших – простим записом чисел. Так чи інакше, виконання завдання все ще потребує успішного створення власної мови Асемблера для роботи з моделлю процесора і хоча б базового розуміння принципів відображення двійкових чисел та роботи з мітками. Через це, задача підійде для тих студентів, які ще не

встигли розібратись у більш довершених алгоритмах і мають з чогось починати.

Запропонований розв'язок завдання на мові Асемблер (при умові, що змійка стартує у молодшому півбайті регістру А та рухається за годинниковою стрілкою) наведено далі:

Таблиця 3.3

Програмне рішення завдання 2п

```
            ORG     000h          ; Початок програми з нульової адреси пам'яті


            LD  A,  #0Fh          ; Встановлення 'змійки' у початкову позицію та рух
вгору
START:      RLA A
            RLA A
            RLA A
            RLA A
            LD  B,  #80h          ; Початок руху вправо
            LD  A,  #0Ch
            LD  C,  #80h
            LD  A,  #0Eh
            LD  D,  #80h
            LD  A,  #0F0h
            CLR A
            LD  D,  #0Ch
            CLR B
            LD  D,  #0Eh
            CLR C
            LD  D,  #0Fh           ; Початок руху вниз
            LD  D,  #78h
            LD  D,  #3Ch
            LD  D,  #1Eh
            LD  D,  #0Fh
            LD  C,  #1h            ; поворот наліво
            LD  D,  #7h
            LD  B,  #1h
            LD  D,  #3h
            LD  A,  #1h
            LD  D,  #1h
            LD  A,  #3h
            CLR D
            LD  A,  #7h
            CLR C
            LD  A,  #0Fh
            CLR B                  ;повернення в початкову позицію
            GOTO    START
```

Наступне розглянуте завдання, навпаки, відноситься до розряду складних та цікавих, і запропонована нижче реалізація була виконана за допомого макропроцедур:

Завдання 2в. Розробити програму для аналізу стану двох кнопок клавіатури (а саме, відповідних розрядам D.0 та D.1). Залежно від стану кнопок,

мають виконуватись наступні операції: натиснута перша – до вмісту регістру C додається 1, натиснута друга – до вмісту регістру C додається 2. Повинні оброблятися усі можливі комбінації та послідовності натиснених кнопок.

Програма може здатися складною на перший погляд, і якщо дивитися з точки зору стандартного Асемблера, так воно і є. Однак, вона має цілком послідовне та чітке рішення засобами Макроасемблера – необхідно розділити програму на дві частини. Перша міститиме виклики макрофункцій зчитування станів та подальші переходи між ними й виклики відповідних арифметичних операцій, а друга в свою чергу являтиме собою бібліотеку макропроцедур, у якій і міститимуться ці функції аналізу станів та додавання. Для вірної реалізації подібної задачі, необхідно сформулювати чіткий алгоритм аналізу, який представлено на рисунку 3.3:

Рис. 3.3 – Алгоритм рішення задачі 2в

Як бачимо, необхідно взяти

до уваги кожну можливу комбінацію кнопок та отримати три можливі результати. Однак, цього недостатньо для коректної реалізації додавання відповідних чисел, так як кнопки можуть бути відтиснуті або натиснуті в ході виконання програми. Для вірної реалізації суми, наведемо розмічений граф станів цих кнопок та відповідних їм операцій:

Рис. 3.4 – граф станів кнопок D.0 та D.1

Як бачимо, насправді можливих станів не три, а чотири (бо має значення порядок натиснутих кнопок), а додаване до вмісту регістра C число залежить і від послідовності зміни цих станів.

Взявши до уваги попередні умови, наведу наступне програмне рішення.

Таблиця 3.4

Основний файл програми для завдання 2в

```
;_____________________________________________________________________________

        .ORG          000h      ; Призначення адреси трансляції
;-------------------------------------------------------------------------------
            #Include MACROS.ASM ; Під'єднання бібліотеки макропроцедур
;-------------------------------------------------------------------------------

        LD   D,       #003h     ; Готуємо регістр D для зчитування кнопок M(D.0) і
N(D.1)

; ========================  КОНТРОЛЬ МОЖЛИВИХ СТАНІВ ПРИСТРОЮ
========================

ST00: Stat00()                  ; Контроль стану N=0, M=0
            GOTO P1_01          ; Якщо натиснуто <M> -> додаємо C=C+1 та переходимо в
стан 01
            GOTO P2_10          ; Якщо натиснуто <N>   -> На додавання 2
            GOTO P3_11          ; Якщо натиснуті <N+M> -> На додавання 3

ST01: Stat01()                  ; Контроль стану N=0, M=1
            GOTO ST00           ; Якщо відпущено <M>, переходимо до стану 00
            GOTO P2_11          ; Якщо натиснуто <N>, C=C+2 та переходимо до стану 11
            GOTO P2_10          ; Якщо змінились <N+M>, C=C+2 і переходимо до стану 10

ST10: Stat10()                  ; Контроль стану N=1, M=0
            GOTO P1_11          ; Якщо натиснуто <M>, C=C+1 та переходимо до стану 11
            GOTO ST00           ; Якщо відпущено <N>, переходимо до стану 00
            GOTO P1_01          ; Якщо змінились <N+M>, C=C+1 і переходимо до стану 01

ST11: Stat11()                  ; Контроль стану N=1, M=1
            GOTO ST10           ; Якщо відпущено <M>, переходимо до стану 10
            GOTO ST01           ; Якщо натиснуто <N>, переходимо до стану 01
            GOTO ST00           ; Якщо змінились <N+M>, переходимо до стану 00

; ================================  ЗМІНА СТАНІВ ПРИСТРОЮ
================================

P1_01:      AddC(1)             ; Додавання C=C+1
            GOTO ST01           ; Перехід до першого стану
P2_10:      AddC(2)             ; Додавання C=C+2
            GOTO ST10           ; Перехід до другого стану
P1_11:      AddC(1)             ; Додавання C=C+1
            GOTO ST11           ; Перехід до третього стану
P2_11:      AddC(2)             ; Додавання C=C+2
            GOTO ST11           ; Перехід до третього стану
P3_11:      AddC(3)             ; Додавання C=C+3
            GOTO ST11           ; Перехід до третього стану
;---------------------------
        .END
;---------------------------
```

Таблиця 3.5

Бібліотека макропроцедур для завдання 2в

```
;       .NOLIST
;
;   ОЧІКУВАННЯ ЗМІНИ ДАНИХ ДЛЯ STAT00

;----------------------------------------------------------------------------------------

#DEFINE  Stat00()          MOV    B,    D           ; Читаємо в B стан обох кнопок
#DEFCONT \                 LD     A,    #00000010b  ; Готуємо маску для аналізу <N>
#DEFCONT \                 LD     A,    AND         ; Виділяємо стан кнопки <N>
#DEFCONT \                 JZ     *+9               ; Якщо <N> не натисн.->аналіз <M>
#DEFCONT \                 LD     A,    #00000001b  ; Готуємо маску для аналізу <M>
#DEFCONT \                 LD     A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                 JNZ    *+13              ; Якщо <M> натиснута-> <Push NM>
#DEFCONT \                 GOTO   *+9               ; Інакше -> вихід з <Push N>
#DEFCONT \                 LD     A,    #00000001b  ; Готуємо маску для аналізу <M>
#DEFCONT \                 LD     A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                 JZ     *-16              ; Якщо <M> не натисн.->на початок
                                                    ; Інакше -> вихід з <Push M>
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;
;   ОЧІКУВАННЯ ЗМІНИ ДАНИХ ДЛЯ STAT01

;----------------------------------------------------------------------------------------

#DEFINE  Stat01()          MOV    B,    D           ; Читаємо в B стан обох кнопок
#DEFCONT \                 LD     A,    #00000010b  ; Готуємо маску для аналізу <N>
#DEFCONT \                 LD     A,    AND         ; Виділяємо стан кнопки <N>
#DEFCONT \                 JZ     *+9               ; Якщо <N> не натисн.->аналіз <M>
#DEFCONT \                 LD     A,    #00000001b  ; Готуємо маску для аналізу <M>
#DEFCONT \                 LD     A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                 JZ     *+13              ; Якщо <M> натиснута -> <Push NM>
#DEFCONT \                 GOTO *+9                 ; Інакше -> вихід з <Push N>
#DEFCONT \                 LD     A,    #00000001b  ; Готуємо маску для аналізу <M>
#DEFCONT \                 LD     A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                 JNZ    *-16              ; Якщо <M> не натисн.->на початок
                                                    ; Інакше -> вихід з <Push M>
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;
;   ОЧІКУВАННЯ ЗМІНИ ДАНИХ ДЛЯ STAT10

;----------------------------------------------------------------------------------------
```

```
#DEFINE  Stat10()              MOV     B,    D           ; Читаємо в B стан обох кнопок
#DEFCONT \                     LD      A,    #00000010b  ; Готуємо маску для аналізу
<N>
#DEFCONT \                     LD      A,    AND         ; Виділяємо стан кнопки <N>
#DEFCONT \                     JNZ     *+9               ; Якщо <N> не натисн.->аналіз
<M>
#DEFCONT \                     LD      A,    #00000001b  ; Готуємо маску для аналізу
<M>
#DEFCONT \                     LD      A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                     JNZ     *+13              ; Якщо <M> натисн. -> <Push
NM>
#DEFCONT \                     GOTO *+9                  ; Інакше -> вихід з <Push N>
#DEFCONT \                     LD      A,    #00000001b  ; Готуємо маску для аналізу
<M>
#DEFCONT \                     LD      A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                     JZ      *-16              ; Якщо <M> не натисн -> на
початок
                                                         ; Інакше -> вихід з <Push M>
;_____________________________________________________________________________________________
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
;______________________________________________________________________________________
;ШШШШШШШШШШШШШШШШШШШШШШШШШ  ОЧІКУВАННЯ ЗМІНИ ДАНИХ ДЛЯ STAT11
ШШШШШШШШШШШШШШШШШШШШШШШШШШШ

;-------------------------------------------------------------------------------------

#DEFINE  Stat11()              MOV    B,    D           ; Читаємо в B стан обох кнопок
#DEFCONT \                     LD     A,    #00000010b  ; Готуємо маску для аналізу
кнопки <N>
#DEFCONT \                     LD     A,    AND         ; Виділяємо стан кнопки <N>
#DEFCONT \                     JNZ    *+9               ; Якщо <N> не натисн.->аналіз
<M>
#DEFCONT \                     LD     A,    #00000001b  ; Готуємо маску для аналізу
<M>
#DEFCONT \                     LD     A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                     JZ     *+13              ; Якщо <M> натиснута -> <Push
NM>
#DEFCONT \                     GOTO *+9                 ; Інакше -> вихід з <Push N>
#DEFCONT \                     LD     A,    #00000001b  ; Готуємо маску для аналізу
<M>
#DEFCONT \                     LD     A,    AND         ; Виділяємо стан кнопки <M>
#DEFCONT \                     JNZ    *-16              ; Якщо <M> не натисн.->на
початок
                                                        ; Інакше -> вихід з <Push M>
;______________________________________________________________________________________
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;______________________________________________________________________________________
;ШШШШШШШШШШШШШШШШШШШШШШШШШШШ  ДОДАВАННЯ ЧИСЛА ДО РЕГІСТРА С
ШШШШШШШШШШШШШШШШШШШШШШШШШШШШШ

;   Функція AddC(N) додає до значення регістра С число N.  При роботі процедури
використо-
; вуються регістри А та В.
;-------------------------------------------------------------------------------------

#DEFINE  AddC(N)               LD     B,    #N     ; Завантажуємо число N до регістра В
#DEFCONT \                     MOV    A,    C      ; Читаємо в регістр А стан регістра
С
#DEFCONT \                     LD     C,    SUM    ; Та завантажуємо результат А+В в С
;______________________________________________________________________________________
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;       .LIST
;*************************************************************************************
```

Висновки за розділом 3

Підсумовуючи, можна зауважити, що наявні завдання із самого початку розроблялися для максимально ефективного використання під час навчального процесу, через що розподілені по кільком рівням складності і, з огляду на запропоновані варіанти розв’язків та різноманітність очікуваних навичок, необхідних для їх вирішення, є досить наочними та більш ніж здатні цілком забезпечити матеріал для лабораторного практикуму курсу «Мікропроцесори та їх застосування».

ВИСНОВКИ

При виконанні дипломної роботи, були виконані усі сформульовані завдання, зокрема проведено аналіз архітектури моделі процесора DPM08 та її системи команд, проаналізовано апаратні та програмні можливості моделі процесора, і на їх основі розроблено два повноцінних блоки навчальних завдань. Також було розроблено методичні рекомендації щодо використання цих завдань та складено довідник викладача зі зразками їх рішень.

В результаті, було досягнуто мети роботи – успішна розробка комплекту завдань надасть можливість організувати самостійну роботу студентів, підвищити наглядність лекційних занять та зробити більш цікавими та повчальними лабораторні роботи. Це беззаперечно посприяє підвищенню якості підготовки студентів при вивченні роботи з мікропроцесорною технікою, напрацюванню навичок розробки алгоритмів та розумінню основ низькорівневого програмування. Завдяки виконанню поставлених завдань, була створена бібліотека, що надасть можливість викладачам забезпечити студентів групи індивідуальними завданнями та посприяти їх самостійній роботі. Створені завдання сприятимуть зацікавленості студентів у виконанні практичних робіт та стимулюватимуть дослідження матеріалу своїми силами. Також розроблена бібліотека повною мірою використає демонстраційні здатності моделі або її емуляторів для детального пояснення принципів дії мікропроцесорної техніки. Різноманітність та креативність запропонованих завдань сприятимуть отриманню навичок розробки алгоритмів та програмування на мовах низького рівня.

Щоправда, розроблені завдання не включають програм для роботи з зовнішніми пристроями, підключення яких підтримується моделлю – а саме, 32-кілобайтним модулем пам’яті або рідкокристалічним дисплеєм. Через відсутність можливості емулювати ці пристрої наявними програмними засобами та тимчасову недоступність реальної моделі, задачі, що мають на меті роботу з

цими пристроями відсутні у запропонованому збірнику. Однак, наявність цих апаратних можливостей дає змогу реалізувати завдання з використанням вищезазначених пристроїв у майбутньому, що надає широкі перспективи подальшого розвитку створеної бібліотеки. Через високу наочність задач та наявність програмного емулятора, розроблені завдання можуть бути застосовані в ході навчального процесу в інших навчальних закладах з метою популяризації серед молоді вивчення основ комп'ютерної техніки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Журавель Ю.А, Рева С.М «Модель цифрового процесора» Харківський національний університет імені В.Н. Каразіна, 2011 р.
2. Лекційні матеріали курсу «Мікропроцесори та їх застосування»
3. Vlad Pirogov «The Assembly Programming Master Book», A-List, LLC, 2004, с.7-25, 91-103.
4. Новацький А. О. Мікропроцесорні та мікроконтролерні системи : підручник.. – Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2020.
5. Поджаренко В.О., Кучерук В.Ю., Севастьянов В.М. Основи мікропроцесорної техніки. Навчальний посібник. - Вінниця: ВНТУ, 2006.
6. Ю. П. Колонтаєвський; Конспект лекцій з дисципліни «Мікропроцесорна техніка» – Харків, ХНУМГ ім. О. М. Бекетова, 2016.
7. Borland Turbo Assembler – User Guide [Електронний ресурс]. – режим доступу: URL: http://bitsavers.informatik.uni-stuttgart.de/pdf/borland/turbo_assembler/Turbo_Assembler_Version_5_Users_Guide.pdf (дата звернення – 20.04.2023).
8. Flat Assembler – overview [Електронний ресурс]. – режим доступу: URL: https://flatassembler.net/docs.php?article=fasmg (дата звернення – 20.04.2023).
9. Ivan Mezei - Evolution of an educational microprocessor [Електронний ресурс] – режим доступу: URL: https://www.researchgate.net/publication/342851306_Evolution_of_an_educational_microprocessor (дата звернення – 20.05.2023).
10. Wenbing Wu - Analysis of several difficult problems in Assembly Language Programming [Електронний ресурс] – режим доступу: URL: https://www.researchgate.net/publication/368091370_Analysis_of_Several_Difficult_Problems_in_Assembly_Language_Programming (дата звернення – 20.05.2023).