КОМП'ЮТЕРНІ НАУКИ

УДК 004.75
DOI https://doi.org/10.37734/2518-7171-2024-1-7

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

О. В. ОЛЬХОВСЬКА, кандидат фізико-математичних наук, доцент;
О. О. ЧЕРНЕНКО, кандидат фізико-математичних наук, доцент;
Т. О. ПАРФЬОНОВА, кандидат фізико-математичних наук, доцент;
Є. І. АВРАМЕНКО, магістр напрямку «Комп'ютерні науки»
(Полтавський університет економіки і торгівлі)

Анотація. Мета статті *– дослідження методів програмної реалізації тренажеру, який дозволить ефективно навчати та вдосконалювати навички використання арифметичних команд мови Assembler.* ***Методика дослідження.*** *Засоби для створення, редагування сучасних вебзастосунків і програм для хмарних систем Visual Studio Code, відкрита JavaScript бібліотека для створення інтерфейсів користувача React, середовище розробки Create React App.* ***Результати.*** *У ході розробки тренажеру з вивчення арифметичних команд мови Assembler було проведено значне дослідження алгоритмів та методів реалізації, яке включало в себе аналіз існуючих рішень, визначення вимог до програми, розробку архітектури, вибір технологій, програмування та тестування.*

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

Розроблено архітектурну частину тренажеру. Враховуючи специфіку тренажеру для навчання арифметичним командам мови Assembler, до архітектурної структури включено наступні компоненти: користувацький інтерфейс, модуль візуалізації, систему зберігання прогресу користувача, модуль навчання, модуль тестування, модуль виконання завдань. Детально описано реалізацію основних компонентів, які були визначені в архітектурній структурі. ***Практична значущість результатів дослідження.*** *Використання навчального продукту дозволить студентам та фахівцям отримати практичний досвід роботи з мовою низького рівня, розширити свої знання у галузі архітектури обчислювальних систем та підвищити рівень володіння програмуванням.*

Ключові слова: *Assembler, навчальний інструмент, арифметичні операції, програмна реалізація.*

Постановка проблеми в загальному вигляді та її зв'язок з найважливішими науковими та практичними завданнями. У сучасному світі інформаційних технологій, де обчислювальні системи відіграють важливу роль у всіх сферах життя, розуміння архітектури комп'ютерів та операційних принципів є критично важливим для професійного зростання та успіху в галузі програмування, системного адміністрування та інженерії програмного забезпечення. Практичні навички роботи з низькорівневими операційними системами та мовами програмування, зокрема мовою Assembler, відіграють ключову роль у розробці швидкодіючих, ефективних та надійних програм [1–9].

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

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

Виклад основного матеріалу дослідження з повним обґрунтуванням отриманих наукових результатів. При реалізації тренажеру для навчання арифметичним командам мови Assembler важливо детально описати реалізацію основних компонентів, які були визначені в архітектурній структурі.

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

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

Використовується import для завантаження необхідних бібліотек та компонентів LearningMode, TestingMode, TaskMode. Також підключається таблиця стилів App.css.

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

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

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

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

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

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

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

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

В залежності від значення changeButton відображається або одна кнопка "Розпочати", або набір кнопок для навчання, тестування та завдань. Рендеринг також включає текстові елементи, які виводять інформацію про тему та розробника.

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

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

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

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

У цьому фрагменті коду відбувається рендеринг JSX для компонента Result. Компонент виводить інформацію про результати тестування, включаючи час, кількість правильних та неправильних відповідей. Також виводиться рекомендація в залежності від відсотка правильних відповідей. Кнопки «Розпочати знову» та «Вихід» дозволяють користувачеві перейти знову до тестування чи завдань або повернутися до початкового екрану.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Компонент повертає JSX, який відображає час у форматованому вигляді.

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

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

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

У цьому розділі використовується хук стану useState для створення змінних стану компоненту.

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

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

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

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

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

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

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

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

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

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

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

У цьому розділі використовуються хуки useEffect для виконання певних дій при зміні значень currentQuestion та feedback. А саме, автоматична прокрутка до останнього питання та відображення зворотного зв'язку.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Висновки із зазначених проблем і перспективи подальших досліджень у поданому напрямі. Розробка тренажеру для вивчення арифметичних команд мови Assembler є лише початком у великій галузі навчання обчислювальних систем та програмування. Дана робота відкриває широкий простір для подальших досліджень та розвитку.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Строкань О. В., Прийма С. М., Литвин Ю. О.. *Комп'ютерна схемотехніка та архітектура комп'ютерів.* Мелітополь: ТДАТУ, 2019. 186 с.
2. Матвієнко М. П. *Комп'ютерна логіка: навч. посібник*. К.: Видавництво Ліра-К, 2012. 288 с.
3. *Інформатика. Комп'ютерна техніка. Комп'ютерні технології: Підручник для ВНЗ* / За ред. О.І. Пушкаря. К.: Академія, 2003. 704 с.
4. *Дистанційний курс «Архітектура обчислювальних систем»*. Сайт дистанційного навчання, Полтавський університет економіки і торгівлі. URL: https://el.puet.edu.ua/
5. Волошин В. В. *Основи асемблерної мови: Навчальний посібник.* Національний технічний університет України "Київський політехнічний інститут". 2012. 257 с.
6. Богданов О. В., Гусєва Н. В. *Основи програмування на асемблері.* Видавництво "Літера ЛТД". 2010. 146 с.
7. React documentation. URL: https://reactjs.org/docs/getting-started.html
8. JavaScript MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript
9. W3Schools tutorials. URL: https://www.w3schools.com/

REFERENCES

1. Strokan, O. V., Pryima, S. M., Lytvyn, Yu. O. (2019). *Kompiuterna skhemotekhnika ta arkhitektura kompiuteriv [Computer circuitry and computer architecture].* Melitopol: TDATU. 186 p. [in Ukrainian].
2. Matviienko, M. P. (2012). *Kompiuterna lohika: navch. Posibnyk [Computer logic: education. manual].* K.: Vydavnytstvo Lira-K. 288 p. [in Ukrainian].
3. *Informatyka. Kompiuterna tekhnika. Kompiuterni tekhnolohii: Pidruchnyk dlia VNZ (2003) [Informatics. Computer Engineering. Computer technologies: Textbook for universities].* / Za red. O.I. Pushkaria. K.: Akademiia. 704 p. [in Ukrainian].
4. *Dystantsiinyi kurs «Arkhitektura obchysliuvalnykh system» [Distance course "Architecture of computer systems"].* Sait dystantsiinoho navchannia, Poltavskyi universytet ekonomiky i torhivli. Retrieved from: https://el.puet.edu.ua/ [in Ukrainian].
5. Voloshyn, V. V. (2012). *Osnovy asemblernoi movy: Navchalnyi posibnyk [Basics of assembly language: Study guide].* Natsionalnyi tekhnichnyi universytet Ukrainy "Kyivskyi politekhnichnyi instytut". 257 p [in Ukrainian].
6. Bohdanov, O. V., & Husieva, N. V. (2010). *Osnovy prohramuvannia na asembleri [Basics of assembly language programming].* Vydavnytstvo "Litera LTD". 146 p. [in Ukrainian].
7. React documentation Retrieved from: https://reactjs.org/docs/getting-started.html
8. JavaScript MDN Web Docs. Retrieved from: https://developer.mozilla.org/en-US/docs/Web/JavaScript
9. W3Schools tutorials. Retrieved from: https://www.w3schools.com/

O. Olkhovska**, PhD., Associate Professor; **O. Chernenko**, PhD., Associate Professor; **T. Parfonova**, PhD., Associate Professor; Eugene Avramenko, master's degree in Computer Science (Poltava University of Economics and Trade).* ***Software implementation of the educational tool on the topic "Assembler language arithmetic commands" of the discipline "Computer systems architecture"

***Abstract. The purpose** of the article is to study the methods of software implementation of the simulator, which will allow to effectively teach and improve the skills of using arithmetic commands of the Assembler language. **Research methodology**. Tools for creating and editing modern web applications and programs for cloud systems Visual Studio Code, an open JavaScript library for creating React user interfaces, the Create React App development environment. **Results**. During the development of the simulator for learning the arithmetic commands of the Assembler language, a significant study of algorithms and implementation methods was carried out, which included the analysis of existing solutions, the definition of program requirements, the development of architecture, the selection of technologies, programming and testing.*

During the analysis of existing simulators for learning the Assembler language, it was found that most of them provide an opportunity to learn and test the general concepts of the Assembler language, but are somewhat limited in the variety of exercises and tasks, especially in the mode of execution of arithmetic commands. Also, many existing solutions have an outdated interface and limited interactivity capabilities.

The architectural part of the simulator has been developed. Taking into account the specifics of the simulator for teaching Arithmetic commands of the Assembler language, the following components are included in the architectural structure: user interface, visualization module, user progress storage system, training module, testing module, task execution module. The implementation of the main components, which were defined in the architectural structure, is described in detail.

Practical significance of research results. The use of the educational product will allow students and professionals to gain practical experience with a low-level language, expand their knowledge in the field of computer system architecture, and increase their programming skills.

***Key words:** Assembler, educational tool, arithmetic operations, software implementation.*