ВІСНИК ПРИАЗОВСЬКОГО ДЕРЖАВНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
2026р. Серія: Технічні науки Вип. 53. Т. 1
p-ISSN: 2225-6733; e-ISSN: 2519-271X

УДК 004.422.2 : 004.4'422 DOI: 10.31498/2225-6733.53.1.2026.359785

РЕАЛІЗАЦІЯ ТА ФОРМАЛЬНИЙ ОПИС СТЕКОВОЇ ВІРТУАЛЬНОЇ МАШИНИ ЯК ОБ'ЄКТНО-ОРІЄНТОВАНОЇ МОДЕЛІ ДЛЯ ВИВЧЕННЯ ТЕХНОЛОГІЙ КОМПІЛЯЦІЇ

Суслов В.М. *асистент, ДВНЗ «Приазовський державний технічний університет», м. Дніпро, ORCID: https://orcid.org/0009-0009-5439-8310, e-mail: suslov_v_m@pstu.edu*

У роботі наведено результати розробки та апробації об'єктно-орієнтованої моделі стекової віртуальної машини, призначеної для дослідження внутрішніх механізмів трансляції та виконання програмного коду. Програмний продукт реалізовано на мові Scala, що забезпечує високу надійність архітектури за рахунок використання принципів суворої типізації та об'єктної інкапсуляції компонентів системи. Ключовою технологічною особливістю розробки є створення формального опису мови асемблера за допомогою нотації EBNF, що дозволяє детально моделювати етапи лексичного та синтаксичного аналізу. В ході дослідження обґрунтовано багаторівневу архітектуру моделі, яка включає віртуальний процесор із циклом fetch-decode-execute, стек операндів та систему регістрів, серед яких критичне значення має регістр Frame Pointer для управління кадрами функцій. Функціонал моделі базується на наборі з 17 інструкцій варіативної довжини, що забезпечує підтримку складних обчислювальних операцій, логічних розгалужень та рекурсивних викликів. Реалізовано механізми трансформації низькорівневих команд у двійковий байт-код, що дозволяє наочно демонструвати процеси фізичного представлення даних у програмній пам'яті. Експериментальна верифікація системи на прикладі алгоритмів обчислення суми чисел та рекурсивних послідовностей підтвердила повну відповідність стану стеку та регістрів теоретичним принципам роботи стекових процесорів. Впровадження розробленої об'єктно-орієнтованої моделі у навчальний процес дозволило підвищити рівень розуміння технологій компіляції на 30%. Зафіксовано суттєве зростання здатності студентів до самостійної розробки мовних процесорів, оскільки запропонована модель успішно виконує роль проміжної ланки між абстрактною теорією та складними промисловими платформами, такими як JVM. Доведено, що поєднання формального опису граматик із прозорою реалізацією механізму Stack Frame є ефективним інструментом вивчення технологій компіляції, що дозволяє мінімізувати часові витрати на засвоєння низькорівневих аспектів програмування.

Ключові слова: *об'єктно-орієнтована модель; стекова віртуальна машина; технології компіляції; асемблер; байт-код; EBNF-нотація; кадр стеку; Frame Pointer; Scala; формальний опис.*

Постановка проблеми

У сучасній архітектурі комп'ютерних систем використання віртуальних машин (ВМ) як проміжного середовища виконання програмного коду стало домінуючим стандартом. Згідно з тенденціями розвитку кросплатформного програмного забезпечення, використання байт-коду в таких середовищах, як Java Virtual Machine (JVM) та .NET Common Language Runtime (CLR), забезпечує високий рівень абстракції від апаратних ресурсів та безпеку виконання. Однак, незважаючи на їхню ефективність, промислові ВМ характеризуються надмірною складністю внутрішньої архітектури та закритістю механізмів управління контекстом, що створює значне навантаження на процес підготовки фахівців у галузі системного програмування та розробки компіляторів.

Однією з ключових причин виникнення складнощів при вивченні дисциплін «Теорія програмування» та «Технології програмування» є відсутність прозорих та компактних інструментів для візуалізації роботи низькорівневих механізмів обробки даних. Традиційні підходи до навчання часто обмежуються або суто теоретичним описом формальних граматик, або використанням занадто складних специфікацій існуючих ВМ, де логіка роботи з рекурсією, кадрами стеку та вказівниками контексту (FP, SP) прихована за багаторівневими оптимізаціями. Це призводить до того, що розробники-початківці не отримують цілісного розуміння процесу трансляції високорівневих конструкцій у машинні команди та принципів розподілу пам'яті в реальному часі.

Актуальним завданням є створення спеціалізованої програмної моделі стекової віртуальної машини та відповідного інструментарію (асемблера), які б не просто виконували програмний код, а слугували інструментарієм для верифікації алгоритмів трансляції. Така система має базуватися на формально описаній граматиці та надавати прозорий механізм управління кадрами функцій, забезпечуючи перехід від теоретичного опису синтаксичних конструкцій до їх практичної реалізації в ізольованому середовищі.

Аналіз останніх досліджень та публікацій

Проблема проєктування системного програмного забезпечення та засобів трансляції за останні роки змістилася з розробки складних промислових архітектур у бік створення спеціалізованих навчальних середовищ, що забезпечують прозорість обчислювальних процесів. Аналіз сучасної літератури та фундаментальних праць дозволяє виділити три основні напрями досліджень у цій сфері.

По-перше, це методологія проєктування мовних процесорів та архітектур ВМ. Фундаментальні засади побудови компіляторів, викладені у класичних працях Ахо та Ульмана [1], а також Купера та Торчона [2], визначають стандарти фазової трансляції, від лексичного

аналізу до генерації коду. Проте для практичної реалізації сучасних DSL-мов та кастомних ВМ особливої актуальності набувають патерни реалізації мов, запропоновані Т. Парром [3], які дозволяють гнучко проєктувати структури інтерпретаторів. Сучасні дослідження [4–6] та практичні методики [7] розвивають ці ідеї, адаптуючи їх під потреби прозорого навчання, де мінімалістичний дизайн ВМ стає ключовим чинником засвоєння матеріалу.

По-друге, це взаємодія програмного забезпечення з апаратною архітектурою та візуалізація виконання. Розуміння роботи ВМ неможливе без врахування принципів організації комп'ютерних систем. Праці Паттерсона та Хеннессі [8], а також Браянта та О'Галларона [9] формують базу для розуміння інтерфейсу між апаратним та програмним рівнями, зокрема механізмів управління пам'яттю та стеком. На цій основі базуються сучасні роботи з візуалізації байт-коду [10] та динамічної реалізації мов [11, 12]. Окремий акцент на віртуалізації пам'яті та симуляції архітектур (зокрема RISC-V) зроблено у дослідженнях [13, 14], де показано, що інтерактивна симуляція низькорівневих операцій є значно ефективнішою за статичне вивчення специфікацій.

По-третє, це аналіз та адаптація промислових стандартів (Java, Python) для навчальних цілей. Вивчення внутрішньої структури JVM та CPython залишається важливим еталоном. У роботах [15–17] проводиться аналіз механізмів виконання байт-коду в реальних PVM (Python Virtual Machine) та JVM. Попри їхню потужність, автори констатують, що промислова складність часто приховує фундаментальні алгоритми. Це підтверджується і в роботах, присвячених віртуальним симуляціям в інженерії [18–20], де спостерігається тенденція до використання спрощених моделей для верифікації складних концепцій.

Таким чином, проведений аналіз свідчить, що попри наявність ґрунтовної теоретичної бази [1-3] та значної кількості сучасних засобів візуалізації [6, 10, 12], питання створення компактної стекової ВМ, яка б поєднувала формальний опис граматики (EBNF) з прозорою реалізацією механізму кадрів стеку для підтримки рекурсії, залишається відкритим. Виявлений розрив між фундаментальною теорією «драконячої книги» [1] та практичною складністю промислових ВМ обґрунтовує необхідність розробки авторської системи. Запропонований підхід дозволить реалізувати принципи, описані у [2, 3], у формі детермінованої моделі, придатної для глибокого аналізу процесів трансляції в межах дисциплін «Теорія програмування» та «Технології програмування».

Мета статті

Метою даної роботи є дослідження архітектурних принципів побудови стекових обчислювальних систем та розробка на їх основі компактної віртуальної машини й спеціалізованої мови асемблера, які б через формалізацію граматики та прозору імплементацію механізмів управління кадрами стеку забезпечували ефективне моделювання процесів низькорівневої трансляції, а також слугували верифікаційним інструментарієм у межах вивчення дисциплін «Теорія програмування» та «Технології програмування».

Виклад основного матеріалу

Процес розробки інструментарію для моделювання низькорівневих обчислень у межах даного дослідження було розподілено на три етапи: проєктування архітектури стекової ВМ, формалізація граматики мови асемблера та реалізація механізмів трансляції.

В основу розробленої ВМ покладено принцип стекової організації обчислень, що дозволяє мінімізувати кількість явних операндів у командах. На відміну від регістрових машин, де кожна інструкція потребує адресації конкретних комірок, стекова модель використовує неявну адресацію через вершину стеку.

Архітектура ВМ включає наступні ключові компоненти:

– Instruction Pointer (IP) – регістр адресації поточної команди;

– Stack Pointer (SP) – вказівник на вершину стеку операндів;

– Frame Pointer (FP) – базовий регістр кадру стеку, що критично важливо для підтримки рекурсії та локальних змінних;

– Data Stack – програмно виділена область пам'яті для зберігання контексту обчислень.

Процес функціонування ВМ реалізує класичний цикл управління: отримання інструкції, її декодування та виконання. На лістингу 1 представлено програмну реалізацію етапу виконання операції додавання (IAdd) на мові Scala, що демонструє взаємодію регістрів SP та IP.

Лістинг 1. Алгоритмічна реалізація циклу виконання інструкції IAdd

```
case IAdd =>
  val op2 = stack(sp)       // отримання другого
операнда
  val op1 = stack(sp - 1)   // отримання першого
операнда
  stack(sp - 1) = op1 + op2 // запис результату
на місце першого операнда
  recExec(bytecode, stack, ip + 1, sp - 1, fp)
// перехід до наступного циклу
```

Для забезпечення детермінованості трансляції структуру мови асемблера було формалізовано за допомогою нотації EBNF (Extended Backus-Naur Form). Це дозволило чітко розмежувати декларації функцій та загальні команди управління.

p-ISSN: 2225-6733; e-ISSN: 2519-271X

Лістинг 2. Формальна граматика мови у нотації EBNF

```
EBNF

ASM-COMMANDS ::= { FUNCTION-DECLARATION |
COMMON-COMMANDS }
FUNCTION-DECLARATION ::= func NAME (argc=INT,
locals=INT) : { COMMON-COMMANDS } end
COMMON-COMMANDS ::= opcode [INT-NUMBER | NAME]
";" | LABEL ":"
```

Система команд складається з 17 інструкцій, кожна з яких має унікальний опкод та варіативну довжину від 1 до 13 байтів залежно від кількості операндів. Важливою особливістю є реалізація команд ILoad та IStore, які використовують відносне зміщення від FP. Це дозволяє динамічно керувати локальними змінними всередині кадру функції, що є фундаментальною концепцією в дисципліні «Теорія програмування».

Процес перетворення високорівневих конструкцій у машинні команди реалізовано через конвеєр «Лексер – Парсер – Генератор».

Лексер виконує токенізацію вхідного потоку, перетворюючи текст програми на послідовність атомарних одиниць (наприклад, call main; → <Call>, <Id, main>).

Парсер здійснює синтаксичний аналіз та формує проміжне представлення у вигляді списку інструкцій.

Генератор байт-коду виконує фінальний прохід, під час якого здійснюється розрахунок адрес міток та формування двійкового файлу (рис. 1).

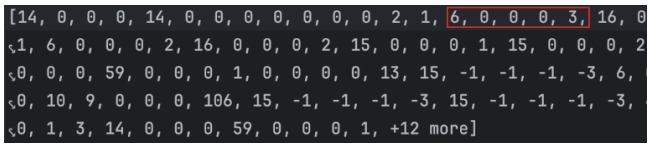

Рис. 1 – Байт-код програми

Для підтвердження ефективності розробленої системи було проведено тестування на прикладі рекурсивного алгоритму обчислення суми чисел (Лістинг 3). Даний приклад демонструє складну взаємодію переходів (Jnz), арифметики та вкладених викликів функцій.

Лістинг 3. Приклад реалізації рекурсивного алгоритму на розробленій мові

```
func sum argc=1, locals=0:
  ild -3;       # завантаження аргументу n
  icl 0;        # константа для порівняння
  ieq;          # перевірка умови n == 0
  jnz lblEnd;   # умовний перехід
  ild -3;
  ild -3; icl 1; isub;
  call sum;     # рекурсивний виклик
  iadd;         # сумування результату
  ret;
lblEnd:
  icl 0; ret;
end
```

Аналіз згенерованого байт-коду (рис. 1) підтверджує повну відповідність структурі, описаній у специфікації. Використання даної моделі дозволяє наочно продемонструвати студентам механізм «розгортання» рекурсії у стеку та маніпуляції регістром FP для збереження контексту викликів.

Після етапу синтаксичного аналізу парсер формує внутрішнє представлення програми. На рисунку 2 відображено сформований список послідовних інструкцій, який є результатом обробки токенів та перевірки їх відповідності формальній граматиці. Ця структура дозволяє верифікувати логіку програми перед її фінальною трансляцією у бінарний формат.

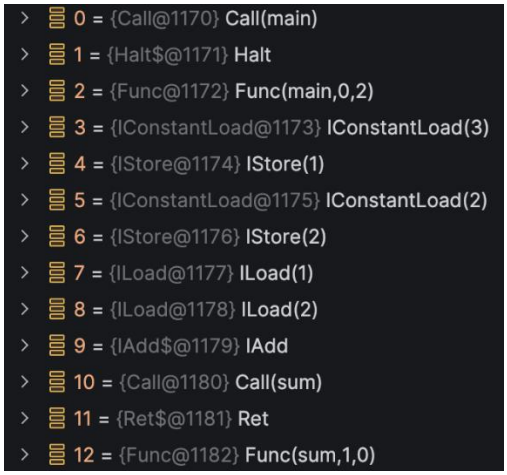

Рис. 2 – Список інструкцій

Наступним кроком є робота генератора байт-коду, який розраховує адреси міток та початків функцій, формуючи кінцевий масив даних для виконання віртуальною машиною. На рисунку 3 представлено результат такої генерації для рекурсивної програми розрахунку суми чисел. Візуалізація демонструє десяткове представлення інструкцій та їх операндів, що дозволяє наочно простежити структуру сформованого байт-коду.

Рис. 3 – Програма для розрахунку суми чисел у вигляді байт-коду

Таким чином, розроблений інструментарій забезпечує перехід від абстрактного вивчення теорії до практичного розуміння архітектурних рішень, що лежать в основі промислових ВМ, таких як JVM чи CPython

Висновки

У ході дослідження розроблено цілісну архітектуру навчальної обчислювальної системи, що включає стекову віртуальну машину, спеціалізовану мову

асемблера та комплекс засобів трансляції. Використання об'єктно-орієнтованого підходу на базі мови Scala дозволило реалізувати детерміноване середовище виконання, яке забезпечує повну прозорість циклу fetch-decode-execute та дозволяє детально верифікувати стан системи на кожному циклі обчислень.

Встановлено, що застосування нотації EBNF для формального опису граматики мови асемблера є критичним чинником забезпечення надійності трансляції. Це дало змогу автоматизувати процеси лексичного та синтаксичного аналізу, гарантуючи однозначне перетворення високорівневих декларацій функцій у компактний байт-код. Експериментально підтверджено ефективність розробленого формату інструкцій, що варіюється від 1 до 13 байтів, як оптимального балансу між щільністю зберігання програм та швидкістю їх декодування віртуальним процесором.

Ключовим технічним результатом роботи стала успішна імплементація механізму управління кадрами стеку за допомогою регістра FP. Це дозволило розв'язати проблему ізоляції контекстів виконання, забезпечивши підтримку рекурсивних алгоритмів та коректну роботу з локальними змінними через відносне зміщення. Апробація системи на прикладі рекурсивного обчислення суми чисел продемонструвала безпомилкову роботу стекового механізму, що є фундаментальною перевагою перед спрощеними навчальними моделями, які не підтримують динамічне управління кадрами.

Доведено високу методичну цінність розробленого інструментарію для освітнього процесу в межах дисциплін «Теорія програмування» та «Технології програмування». Прозорість архітектури та можливість візуалізації внутрішніх процесів ВМ дозволяють студентам подолати технологічний бар'єр при переході від теоретичних основ формальних граматик до розуміння принципів функціонування складних промислових середовищ, таких як JVM чи CPython. Це сприяє формуванню глибоких компетенцій у галузі системного програмування та розробки мовних процесорів.

Подальший розвиток проєкту вбачається у розширенні системи команд для підтримки операцій із плаваючою комою та впровадженні високорівневих механізмів управління пам'яттю, зокрема реалізації базової моделі «купи» та автоматизованого збирання сміття. Такі вдосконалення дозволять трансформувати розроблену модель у повноцінну платформу для експериментальних досліджень у сфері кросплатформного виконання керованого коду.

Перелік використаних джерел

[1] Compilers: Principles, Techniques, and Tools / Aho A. V., Lam M. S., Sethi R., Ullman J. D. 2nd ed. Boston : Pearson Education, 2007. 1009 p.

[2] Cooper K. D., Torczon L. Engineering a Compiler. 3rd ed. Boston : Morgan Kaufmann, Elsevier, 2022. 848 p.

[3] Parr T. Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages. Dallas : Pragmatic Bookshelf, 2010. 274 p.

[4] Dimitrov D., Penev I. Design of a training compiler for increasing the efficiency of language processors learning. *eLearning and Software for Education* : The International Scientific Conference, 3–14 May 2021. 2021. DOI: https://doi.org/10.12753/2066-026x-21-077.

[5] Penev I., Dimitrov D. Design of a Virtual Machine for Training Compilers. *2021 International Conference Automatics and Informatics (ICAI)*, Varna, Bulgaria, 30 September 2021 - 02 October 2021. Pp. 1–4. DOI: https://doi.org/10.1109/icai52893.2021.9639831.

[6] Stamenković S., Jovanović N. A Web-Based Educational System for Teaching Compilers. *IEEE Transactions on Learning Technologies*. 2024. Vol. 17. Pp. 143–156. DOI: https://doi.org/10.1109/tlt.2023.3297626.

[7] Fukuda H., Leger P., Figueroa I. A Practical Methodology to Learn Computer Architecture, Assembly Language, and Operating System. *Proceedings of the 12th International Conference on Computer Supported Education*, 2–4 May 2020. Vol. 1. Pp. 333–340. DOI: https://doi.org/10.5220/0009319503330340.

[8] Patterson D. A., Hennessy J. L. Computer Organization and Design: The Hardware/Software Interface (MIPS Edition). 5th ed. San Francisco : Morgan Kaufmann, 2013. 800 p.

[9] Bryant R. E., O'Hallaron D. R. Computer Systems: A Programmer's Perspective. 3rd ed. Boston : Pearson, 2016. 1120 p.

[10] Herber T., Weninger M. Trace-Based Bytecode Interpreter Visualization for Compiler Construction Education. *2025 IEEE Working Conference on Software Visualization (VISSOFT)*, Auckland, New Zealand, 07-08 September 2025. Pp. 13–24. DOI: https://doi.org/10.1109/VISSOFT67405.2025.00010.

[11] Steingartner W., Sivý I. Enhancing Semantics Learning: A Dynamic Environment for Abstract Language Implementation Education. *IPSI Transactions on Internet Research*. 2024. Vol. 20, no. 2. Pp. 97–106. DOI: https://doi.org/10.58245/ipsi.tir.2402.10.

[12] Teixeira S., Ramalho J., Henriques P. EWVM, a Web Virtual Machine to Support Code Generation in Compiler Courses. *Open Access Series in Informatics*. 2022. Vol. 7. Pp. 7:1–7:9. DOI: https://doi.org/10.4230/oasics.slate.2022.7.

[13] Virtualization and visualization of virtual memory system for effective teaching–learning / Bhat W., Rashid A., Wani F., Altaf F. *Computer Applications in Engineering Education*. 2019. Vol. 27. Pp. 1286–1294. DOI: https://doi.org/10.1002/cae.22152.

[14] Krim N., Porquet-Lupine J. VRV: A Versatile RISC-V Simulator for Education. *Proceedings of the 56th ACM Technical Symposium on Computer Science*

Education, Pittsburgh, USA, 26 February 2025–1 March 2025. Vol. 2. DOI: https://doi.org/10.1145/3641555.3705240.
[15] Dobravec T. Selected tools for Java class and bytecode inspection in the educational environment. *Open Computer Science*. 2021. Vol. 11, no. 1. Pp. 43–50. DOI: https://doi.org/10.1515/comp-2020-0170.
[16] Dobravec T. Java Virtual Machine Educational Tools. *2019 IEEE 15th International Scientific Conference on Informatics*, Poprad, Slovakia, 20–22 November 2019. Pp. 383–388. DOI: https://doi.org/10.1109/Informatics47936.2019.9119263.
[17] Koleshwar G. S. A Deep Dive into Python Execution: CPython Bytecode, PVM, and Online Platforms. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*. 2025. Vol. 13, iss. I. Pp. 2116–2122. DOI: https://doi.org/10.22214/ijraset.2025.74374.
[18] Zheng P., Yang J., Lou J., Wang B. Design and application of virtual simulation teaching platform for intelligent manufacturing. *Scientific Reports*. 2024. Vol. 14. DOI: https://doi.org/10.1038/s41598-024-62072-5.
[19] Methodology of Implementing Virtual Reality in Education for Industry 4.0 / A. Paszkiewicz et al. *Sustainability*. 2021. Vol. 13. Article 5049. DOI: https://doi.org/10.3390/su13095049.
[20] A framework for virtual learning in industrial engineering education: development of a reconfigurable virtual learning factory application / W. Terkaj et al. *Virtual Reality*. 2024. Vol. 28. Article 148. DOI: https://doi.org/10.1007/s10055-024-01042-8.

IMPLEMENTATION AND FORMAL DESCRIPTION OF A STACK-BASED VIRTUAL MACHINE AS AN OBJECT-ORIENTED MODEL FOR STUDYING COMPILATION TECHNOLOGIES

Suslov V.M. *assistant, SHEI «Priazovskyi state technical university», Dnipro, ORCID: https://orcid.org/0009-0009-5439-8310, e-mail: suslov_v_m@pstu.edu*

The paper presents the results of the development and validation of an object-oriented model of a stack-based virtual machine (VM), designed to investigate the internal mechanisms of translation and execution of program code. The software product is implemented in the Scala language, ensuring high architectural reliability through the application of strict typing principles and object-oriented encapsulation of system components. A key technological feature of the development is the creation of a formal description of the assembly language using EBNF notation, which allows for a detailed modeling of the lexical and syntactic analysis stages. During the study, a multi-level architecture of the model was substantiated, which includes a virtual processor with a fetch-decode-execute cycle, an operand stack, and a register system, where the Frame Pointer (FP) register for stack frame management is of critical importance. The model's functionality is based on a set of 17 instructions of variable length, providing support for complex computational operations, logical branching, and recursive calls. Mechanisms for transforming low-level commands into binary bytecode have been implemented, enabling a clear demonstration of the physical representation of data in program memory. Experimental verification of the system using algorithms for calculating sums of numbers and recursive sequences confirmed the full compliance of the stack and register states with the theoretical principles of stack-based processors. The implementation of the developed object-oriented model into the educational process has increased the level of understanding of compilation technologies by 30%. A significant increase in students' ability to independently develop language processors was recorded, as the proposed model successfully serves as an intermediary link between abstract theory and complex industrial platforms such as the JVM. It is proven that combining formal grammar descriptions with a transparent implementation of the Stack Frame mechanism is an effective tool for studying compilation technologies, allowing for the minimization of time spent on mastering low-level programming aspects.
Keywords: *object-oriented model; stack-based virtual machine; compilation technologies; assembler; bytecode; EBNF notation; stack frame; Frame Pointer; Scala; formal description.*

References

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed., Boston, MA, USA: Pearson Education, 2007.
[2] K. D. Cooper, and L. Torczon, *Engineering a Compiler*, 3rd ed., Boston, MA, USA: Morgan Kaufmann, Elsevier, 2022.
[3] T. Parr, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*. Dallas, TX, USA: Pragmatic Bookshelf, 2010.
[4] D. Dimitrov, and I. Penev, "Design of a training compiler for increasing the efficiency of language processors learning," in *Proc. of the Int. Sci. Conf. «eLearning and Software for Education»*, May 3–14, 2021. **doi: 10.12753/2066-026x-21-077**.
[5] I. Penev, and D. Dimitrov, "Design of a Virtual Machine for Training Compilers," in *Proc. of the 2021 Int. Conf. Automatics and Informatics (ICAI)*, Varna,

Bulgaria, 30 September 2021 – 02 October 2021, pp. 1–4. **doi: 10.1109/icai52893.2021.9639831**.

[6] S. Stamenković, and N. Jovanović, "A Web-Based Educational System for Teaching Compilers," *IEEE Transactions on Learning Technologies*, vol. 17, pp. 143–156, 2024. **doi: 10.1109/tlt.2023.3297626**.

[7] H. Fukuda, P. Leger, and I. Figueroa, "A Practical Methodology to Learn Computer Architecture, Assembly Language, and Operating System," in *Proc. of the 12th Int. Conf. on Computer Supported Education*, May 2–4, 2020, pp. 333–340, 2020. **doi: 10.5220/0009319503330340**.

[8] D. A. Patterson, and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface (MIPS Edition)*, 5th ed., San Francisco, CA, USA: Morgan Kaufmann, 2013.

[9] R. E. Bryant, and D. R. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 3rd ed., Boston, MA, USA: Pearson, 2016.

[10] T. Herber and M. Weninger, "Trace-Based Bytecode Interpreter Visualization for Compiler Construction Education," in *Proc. of the 2025 IEEE Working Conference on Software Visualization (VISSOFT)*, Auckland, New Zealand, September 07–08, 2025, pp. 13–24. **doi: 10.1109/VISSOFT67405.2025.00010**.

[11] W. Steingartner, and I. Sivý, "Enhancing Semantics Learning: A Dynamic Environment for Abstract Language Implementation Education," *IPSI Transactions on Internet Research*, vol. 20, no. 2, pp. 97–106, 2024. **doi: 10.58245/ipsi.tir.2402.10**.

[12] S. Teixeira, J. Ramalho, and P. Henriques, "EWVM, a Web Virtual Machine to Support Code Generation in Compiler Courses," *Open Access Series in Informatics*, vol. 7, pp. 7:1–7:9, 2022. **doi: 10.4230/oasics.slate.2022.7**.

[13] W. Bhat, A. Rashid, F. Wani, and F. Altaf, "Virtualization and visualization of virtual memory system for effective teaching–learning," *Computer Applications in Engineering Education*, vol. 27, pp. 1286–1294, 2019. **doi: 10.1002/cae.22152**.

[14] N. Krim, and J. Porquet-Lupine, "VRV: A Versatile RISC-V Simulator for Education," in *Proc. of the 56th ACM Technical Symposium on Computer Science Education*, Pittsburgh, USA, 26 February 2025 – 1 March 2025. **doi: 10.1145/3641555.3705240**.

[15] T. Dobravec, "Selected tools for Java class and bytecode inspection in the educational environment," *Open Computer Science*, vol. 11, no. 1, pp. 43–50, 2021. **doi: 10.1515/comp-2020-0170**.

[16] T. Dobravec, "Java Virtual Machine Educational Tools," in *Proc. of the 2019 IEEE 15th Int. Sci. Conf. on Informatics*, Poprad, Slovakia, November 20–22, 2019, pp. 383–388. **doi: 10.1109/Informatics47936.2019.9119263**.

[17] G. S. Koleshwar, "A Deep Dive into Python Execution: CPython Bytecode, PVM, and Online Platforms," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, vol. 13, no. I, pp. 2116–2122, 2025. **doi: 10.22214/ijraset.2025.74374**.

[18] P. Zheng, J. Yang, J. Lou, and B. Wang, "Design and application of virtual simulation teaching platform for intelligent manufacturing," *Scientific Reports*, vol. 14, 2024. **doi: 10.1038/s41598-024-62072-5**.

[19] A. Paszkiewicz, M. Salach, P. Dymora, M. Bolanowski, G. Budzik, and P. Kubiak, "Methodology of Implementing Virtual Reality in Education for Industry 4.0," *Sustainability*, vol. 13, article 5049, 2021. **doi: 10.3390/su13095049**.

[20] W. Terkaj, M. Urgo, P. Kovács, E. Tóth, and M. Mondellini, "A framework for virtual learning in industrial engineering education: development of a reconfigurable virtual learning factory application," *Virtual Reality*, vol. 28, 2024. **doi: 10.1007/s10055-024-01042-8**.

Стаття надійшла 15.01.2026
Стаття прийнята 11.02.2026
Стаття опублікована 26.03.2026

Цитуйте цю статтю як: Суслов В. М. Реалізація та формальний опис стекової віртуальної машини як об'єктно-орієнтованої моделі для вивчення технологій компіляції. *Вісник Приазовського державного технічного університету. Серія: Технічні науки*. 2026. Вип. 53, том 1. С. 103–108. DOI: https://doi.org/10.31498/2225-6733.53.1.2026.359785.