№ 1(55) 2026
НАУКА і ТЕХНІКА СЬОГОДНІ

*УДК: 007:004*

*https://doi.org/10.52058/2786-6025-2026-1(55)-2501-2512*

**Рисований Олександр Миколайович** кандидат технічних наук, доцент, професор кафедри комп'ютерної інженерії та програмування Харківського технічного університету «Харківський політехнічний інститут», м. Харків, https://orcid.org/0000-0003-0616-1617

# ДОСЛІДЖЕННЯ ІНСТРУКЦІЇ БЕЗУМОВНОГО ПЕРЕХОДУ

**Анотація.** У роботі показано, що інструкція безумовного переходу (інструкція jmp в асемблері) показує, що ця команда переходу без аналізу умов перенаправляє виконання програми на іншу адресу пам'яті чи мітку. У цьому випадку здійснюється обхід потоку поточного виконання програми. Але у випадках багаторазового використання цієї інструкції створюються цикли та розгалуження, які призводять до не структурованого коду та збільшення часу виконання програми. Часто цю інструкцію спеціально застосовують для протидії реверсингу програми. Застосування цієї інструкції (якщо час виконання програми не критично) дозволяє сильно заплутати код виконання, так як в цьому випадку хакеру доступний тільки exe-файл і весь процес виконання коду він відстежує у налагоджувачі.

У роботі показано, що процес аналізу виконання коду у налагоджувачі, у випадку, якщо переходи виконуються від менших адрес програми до великим, є найчастіше застосовуваним і простим для аналізу, так як після коду операції розташовується кількість байтів переходу в прямому коді. Такий порядок байтів переходу є природним для аналізу. Але хакеру дуже часто доводиться аналізувати переходи, які здійснюються від великих адрес розташування коду до менших. Така ситуація виникає тоді, коли важливі байти програми (паролі, коди ініціалізації, прихована текстова інформація) записані в заголовки файлу (у DOS-заголовок, область DOS stub, PE-заголовок) і не видно навіть у налагоджувачі. І крім того, число байтів переходу, які розташовані після коду операції, записані не в додатковому, як прийнято в цифровій техніці при записі негативних чисел, а в зворотному коді. І цей факт теж ускладнює аналіз передбачуваних переходів.

Таким чином, у роботі наведено, що інструкція jmp застосовується для реалізації циклів та розгалужень, виходу з вкладених циклів або обробки помилок, створення неструктурованого коду, який складно читати та налагоджувати.

**Ключові слова:** інструкція jmp, PE-формат, переходи, розгалуження.

**Rysovanyi Oleksandr Mykolayovych** candidate of Technical Sciences, Associate Professor, Professor of the Department of Computer Engineering and Programming, Kharkiv Technical University "Kharkiv Polytechnic Institute", Kharkiv, https://orcid.org/0000-0003-0616-1617

## STUDY OF THE UNCONDITIONAL TRANSITION INSTRUCTION

**Abstract.** The paper demonstrates that the unconditional jump instruction (the jmp instruction in assembler) redirects program execution to another memory address or label without evaluating conditions. In this case, the current program execution flow is bypassed. However, repeated use of this instruction creates loops and branches, which lead to unstructured code and increased program execution time. This instruction is often specifically used to prevent program reversing. Using only this instruction (if program execution time is not critical) can significantly obfuscate the execution code, as in this case, the hacker only has access to the executable file and can monitor the entire code execution process in a debugger.

The paper demonstrates that the process of analyzing code execution in a debugger, for cases where jumps are made from lower to higher program addresses, is the most frequently used and easiest to analyze, as the number of jump bytes in the direct code follows the opcode. This jump byte order is natural for analysis. But a hacker very often has to analyze transitions that occur from high code addresses to low ones. This situation arises when important program bytes (passwords, initialization codes, hidden text information) are written in file headers (the DOS header, the DOS stub area, the PE header) and are not visible even in a debugger. Furthermore, the number of transition bytes located after the opcode is written not in two's complement, as is common in digital technology for negative numbers, but in one's complement. This fact greatly complicates the analysis of suspected transitions.

Thus, the work shows that the jmp instruction is used to implement loops and branches, exit nested loops or error handling, and create unstructured code that is difficult to read and debug.

**Keywords:** instructions jmp, PE-format, transitions, branches.

**Постановка проблеми.** Сучасні процесори з архітектурою x86-64 використовують конвеєрне виконання інструкцій, спекулятивне виконання та механізми передбачення переходів для підвищення продуктивності.

Інструкція безумовного переходу jmp, незважаючи на свою простоту, впливає на керування потоком виконання програми, роботу конвеєра та ефективність апаратних оптимізацій. Незважаючи на те, що інструкція jmp є безумовним переходом, її використання може знижувати продуктивність при неправильному розміщенні коду і призводити до надмірних переходів.

При виконанні втручання в exe-файл завжди необхідно знати особистості виконання розрахунків обчислення адрес переходу від більших до менших значень. Такі випадки виникають тоді, коли хакер заховує дані в заголовках PE-формату. А вислідити їх можна тільки спеціальними програмами, такими, як програми-аналізатори коду.

Тому аналіз поведінки інструкції jmp в різних випадках залишається актуальним для низькорівневого програмування, оптимізації критичних ділянок коду, аналізу поведінки процесора на мікроархітектурному рівні.

**Мета статті –** дослідження інструкції безумовного переходу з метою визначення кількості байтів при передачі управління як з менших адрес до великих, так і від великих адрес до менших. Такі дослідження необхідні при вибірці частин коду, захисту коду програми, при аналізі умов впровадження в чужий код, коли важлива інформація розташовується в заголовках PE-файлу та прихована навіть під налагоджувачем.

**Аналіз останніх досліджень і публікацій.** У роботі [1] розглянуто питання переупорядкування (reassembly) бінарного коду архітектур x86-64 із символізацією всіх покажчиків та з урахуванням jmp-інструкцій. Також показано, що можна надійно переписати виконувані файли зменшення часу виконання програми.

У роботі [2] пропонується метод для точного статичного дизассемблінгу та переупорядкування, що мінімізує помилкові шляхи через jmp-інструкції.

Правильне застосування інструкція jmp істотно впливає на захист програми під час контролю переходів [3].

Дослідженню траєкторії переходів при захисті графових нейронних мереж присвячено роботу [4].

При реверсі exe-файла завжди у налагоджувачі перевіряються інструкції розгалуження (переходів) [5-9]. А при впровадженні коду в різні секції файлу, що виконується, особлива увага приділяється інструкціям розгалуження.

Звісно, застосування інструкції переходу виконується мовою Асемблер.

**Виклад основного матеріалу.** Команда безумовного переходу (інструкція) jmp має важливе значення у програмах організації розгалужень (переходів). Вона дозволяє:

– здійснювати безумовний перехід у програмі;

– при великій їх кількості ускладнює аналіз коду (збільшує захист коду програми);

– у разі впровадження коду змінювати послідовність виконання коду програми.

Команда (інструкція) jmp має різні коди операцій та різну розмірність у байтах при короткому та довгому переході, при різній розмірності регістрів, а також при переходах до старших та молодших адрес.

У документації Microsoft вказують, що ближній перехід (стрибок) обмежений від –128 до +127 від поточного значення EIP.

Щоб це показати розглянемо блок коду

```
jmp @2
 mas2 db 127 dup(1)
@2:
```

У цьому блоці здійснюється короткий перехід на 127 = 7Fh байтів у бік збільшення адрес.

Адреса переходу відображається у налагоджувачі x64Dbg після мнемоніки команди jmp та імені exe-файла (в даному випадку це 03.exe) на рис. 1.

Адреса, на яку здійснюється цей перехід можна вирахувати, як:

DF1008h + 7Fh + 2 (EB 7F) = DF1089h

```
RIP RAX RDX R9  0000000000DF1000  C8 8000 00   enter 80,0
                0000000000DF1004  48:83EC 60   sub rsp,60
                0000000000DF1008  EB 7F        jmp 03.DF1089
                0000000000DF100A  0101         add dword ptr ds:[rcx],eax
                0000000000DF100C  0101         add dword ptr ds:[rcx],eax
```

Рис. 1. Код команди jmp ближнього переходу

Т.ч., команда jmp ближнього переходу займає 2 байти (перший байт – це код операції і має значення EB; другий байт – це кількість байтів, через яке здійснюється перехід і, в даному прикладі, має значення 7F).

Найзручніше вважати пропущені байти. Для цього необхідно взяти адресу першого байта після команди та додати до нього кількість байтів, через які здійснюється цей перехід:

DF100Ah + 7Fh = DF1089h.

Якщо перехід здійснюється через 128 і більше байтів (далекий перехід), наприклад, як у блоці коду

```
jmp @2
 mas2 db 128 dup(1)
@2:
```

то код операцій команди jmp далекого переходу займає 5 байтів, і, в даному прикладі, дорівнює E9 80000000 (рис. 2).

```
RIP RAX RDX R9  0000000000A71000  C8 8000 00   enter 80,0
                0000000000A71004  48:83EC 60   sub rsp,60
                0000000000A71008  E9 80000000  jmp 03.A7108D
                0000000000A7100D  0101         add dword ptr ds:[rcx],eax
                0000000000A7100F  0101         add dword ptr ds:[rcx],eax
                0000000000A71011  0101         add dword ptr ds:[rcx],eax
```

Рис. 2. Код команди jmp дальнього переходу

Для виконання безумовного переходу в середовищі masm64 інструкція JMP застосовується найчастіше у формах:

№ 1(55) 2026
iТехніка Наука сьогодні

jmp метка ; перехід по мітці

jmp регистр ; перехід по вмісту регістру

В інструкції

jmp мітка

вказується мітка, до якої переходить програма після виконання. Розглянемо програму:

```
.code
entry_point proc
mov r15, ret1
jmp r15
www db 12
ret1:
;...
invoke ExitProcess,0
entry_point endp
end
```

Ця програма цікава тим, що в секції тексту обминаючи секцію даних оголошено дані

www db 12

Якби не було перед цими даними команди

jmp r15

то виникла помилка компіляції.

Процес налагодження разом із адресою переходу, який завантажений у регістр r15 командою

mov r15, ret1

відстежується у налагоджувачі x64Dbg (рис. 3).

```
0000000000531000    C8 8000 00              enter 80,0
0000000000531004    48:83EC 60              sub rsp,60
0000000000531008    49:BF 1610530000000000  mov r15,03.531016
0000000000531012    41:FFE7                 jmp r15
0000000000531015    0C 48                   or al,48
0000000000531017    C7C1 00000000           mov ecx,0
000000000053101D    FF15 DD0F0000           call qword ptr ds:[<ExitProcess>]
```

Рис. 3. Налагодження програми

Такий прийом розміщення даних у тексті програми зручно використовуватиме приховування цих даних, оскільки після компіляції вони відображаються як коди операцій. І зрозуміти зміст у програмі ускладняється.

Розглянемо команду безумовного переходу для випадків, коли переходи здійснюються від менших адрес рядків коду до великих (переходи здійснюються «вперед»):

```
.code
entry_point proc
jmp @1
mas1 db 256 dup(1)
@1:

jmp @2
@2:
nop
jmp @3
nop
@3: nop
ret
entry_point endp
end
```

Exe-файл цієї програми у налагоджувачі x64Dbg представлений на рис. 4.

На початку програми вставлено блок даних у 256 байтів:

```
jmp @1
mas1 db 256 dup(1)
@1:
```

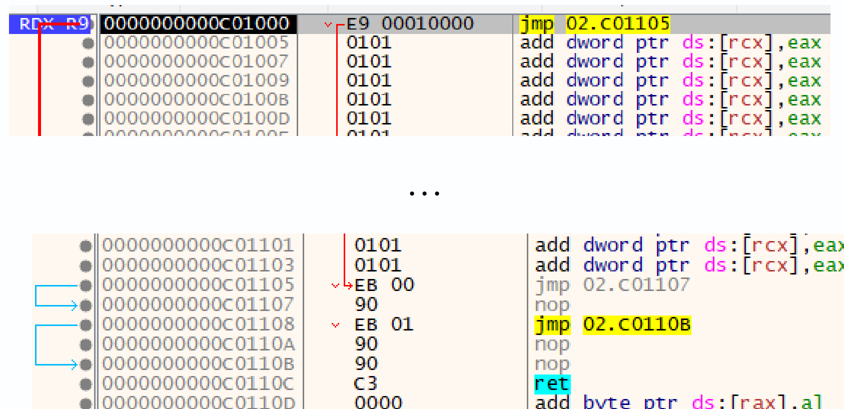


Рис. 4. Exe-файл програми

Такий блок даних (-128/127 байт щодо поточної інструкції) використаний для організації далекого переходу. При дальньому переході код операції дорівнює EB та інструкція має довжину 5 байтів.

Після коду операції EB розташовується не адреса переходу, а кількість байтів під час переходу. У блоці програми, що розглядається, інструкція jmp

@1 має розмір 5 байтів і формат E9 00010000h. Перехід здійснюється на рядок коду, який має адресу C01105h. Отже, число байтів, на яке здійснюється перехід, визначається як

C01105h - C01005h = 100h = 256d байтів.

Наступний блок коду

```
jmp @2
@2:
nop
```

не має взагалі стрибка (переходу), тому що наступний вільний осередок має адресу C01107h, а за цією адресою розташовується команда, на яку здійснюється перехід:

C01107h - C01007h = 0 байтів.

Блок коду

```
jmp @3
nop
@3: nop
```

здійснює перехід через

C0110Bh - C0100Ah = 1 байт.

Мітка @3: стоїть вже на іншому рядку коду і не враховується у розрахунках.

Розглянемо команду безумовного переходу для випадків, коли переходи здійснюються від великих адрес рядків коду до менших (переходи здійснюються «назад»):

```
.code
entry_point proc
@1: mov rax,1
mov r15,1
jmp @1
@2: nop
mas1 db 256 dup(1)
jmp @2
@3: nop
jmp @3
ret
entry_point endp
end
```

При організації впровадження у загальний заголовок файлу завжди використовуються переходи "назад".

Exe-файл цієї програми у налагоджувачі x64Dbg представлений на рис. 5.

```
0000000000B21000   48:C7C0 01000000   mov rax,1
0000000000B21007   49:C7C7 01000000   mov r15,1
0000000000B2100E   EB F0              jmp <02.OptionalHeader.Addre
0000000000B21010   90                 nop
0000000000B21011   0101               add dword ptr ds:[rcx],eax
0000000000B21013   0101               add dword ptr ds:[rcx],eax
0000000000B21015   0101               add dword ptr ds:[rcx],eax
...
0000000000B21105   0101               add dword ptr ds:[rcx],eax
0000000000B21107   0101               add dword ptr ds:[rcx],eax
0000000000B21109   0101               add dword ptr ds:[rcx],eax
0000000000B2110B   0101               add dword ptr ds:[rcx],eax
0000000000B2110D   0101               add dword ptr ds:[rcx],eax
0000000000B2110F   0101               add dword ptr ds:[rcx],eax
0000000000B21111   E9 FAFEFFFF        jmp 02.B21010
0000000000B21116   90                 nop
0000000000B21117   EB FD              jmp 02.B21116
0000000000B21119   C3                 ret
0000000000B2111A   0000               add byte ptr ds:[rax],al
```

Рис. 5. Exe-файл програми з переходами до менших адрес

Розглянемо блок коду у налагоджувачі x64Dbg:

@1: mov rax,1

mov r15,1

jmp @1

У цьому блоці здійснюється короткий перехід. Код операції команди jmp залишився EB.

А кількість байтів записано у зворотному коді.

Розрахуємо кількість байтів переходу. Найпростіше з адреси останнього байта команди jmp відняти значення адреси команди, на яку вказує цикл та інвертувати цей результат.

Для прикладу код операції EB записаний за адресою B2100E. При короткому переході число байтів записується у наступному адресі – B2100F. Отже, кількість байтів буде дорівнювати:

$$B2100F - B21000 = Fh \rightarrow інв = 0_{пр.к.}$$

Перше число байта – завжди Fh, т.к. запис у зворотному коді. Отже, число байтів переходу записується як F0.

Розглянемо блок програми з переходом назад на 256 байтів усередині блоку:

@2: nop

mas1 db 256 dup(1)

jmp @2

Блок коду має довгий перехід. Тому код операції – E9. Записано код E9 за адресою B21111h. Перше число байта завжди Fh.

Останній байт операнда записано на адресу B21115h.

Отже, кількість байтів має формат 32 розряду і дорівнює:

B21115h – B21010h = 105h = …0000 0001 0000 0101 = FFFF FE FA$_{зв.к.}$

Запис здійснюється з молодших адрес до старших:

FAFE FFFF.

У блоці програми:

```
@3: nop
jmp @3
```

кількість байтів буде дорівнювати:

B21118h – B21116h = 2h = 0020 = Dh

Перше число байта – завжди Fh, тому запис у зворотному коді.

Адресація ближнього переходу на -128/+127 байт щодо поточної інструкції відбувається аналогічно до попереднього прикладу. Для цього розглянемо програму:

```
.code
entry_point proc
mov rax,1
@1: nop
mas1 db 124 dup(1)
jmp @1
@2: nop
mas2 db 126 dup(1)
jmp @2
 ret
entry_point endp
end
```

Отриманий після компіляції exe-файл необхідно проаналізувати у налагоджувачі x64Dbg. Exe-файл цієї програми у налагоджувачі x64Dbg представлений на рис. 6.

```
RDX R9> 0000000000401000   48:C7C0 01000000   mov rax,1
        0000000000401007   90                 nop
        0000000000401008   0101               add dword ptr ds:[rcx],eax
        000000000040100A   0101               add dword ptr ds:[rcx],eax
        000000000040100C   0101               add dword ptr ds:[rcx],eax
...
        0000000000401080   0101               add dword ptr ds:[rcx],eax
        0000000000401082   0101               add dword ptr ds:[rcx],eax
        0000000000401084 ^ EB 81              jmp 02.401007
        0000000000401086   90                 nop
        0000000000401087   0101               add dword ptr ds:[rcx],eax
        0000000000401089   0101               add dword ptr ds:[rcx],eax
...
        0000000000401103   0101               add dword ptr ds:[rcx],eax
        0000000000401105   0101               add dword ptr ds:[rcx],eax
        0000000000401107 ^ E9 7AFFFFFF        jmp 02.401086
        000000000040110C   C3                 ret
        000000000040110D   0000               add byte ptr ds:[rax],al
```

Рис. 6. Exe-файл програми з переходами до менших адрес

У вікні налагоджувача x64Dbg видно, що число байтів для ближнього переходу до початку адрес програми записується в один байт у зворотному коді і один байт коду операції (всього 2 байти), а при 126 байтах переходу (без урахування поточного значення EIP) відбувається вже далекий перехід – 4 байти для запису коду операції (всього 5 байтів). Число байтів записується починаючи з молодших адрес та у зворотному коді. Таким чином, ближній перехід обмежений від –128 до +127 від поточного значення EIP (покажчик адреси наступної інструкції).

**Висновки.** У роботі наведено результати досліджень використання інструкції jmp при передачі управління як з менших адрес до великих, так і від великих адрес до менших. Досліджені різні варіанти використання інструкції на конкретних прикладах блоків програмного тексту.

Використання великої кількості команд переходу до менших адрес вносить труднощі в аналіз коду для тих, у кого відсутній текст програми.

Подальші плани: дослідження процесу втручання, особливо в PE-заголовок з метою приховування важливої інформації, в тому числі, та завдяки використанню інструкції jmp.

***Література:***


1. Hyungseok, Kim., Soomin, Kim., Sang, Kil Cha. (2025) *Towards Sound Reassembly of Modern x86-64 Binaries* – ASPLOS '25: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 Pages 1317 - 1333 https://doi.org/10.1145/3676641.3716026
2. Brian, Zhao., Yiwei, Yang., Yusheng, Zheng., Andi, Quinn. (2025) *Exploiting Control-flow Enforcement Technology for Sound and Precise Static Binary Disassembly* — UC Santa Cruz, USA https://doi.org/10.48550/arXiv.2506.09426.
3. Zhanyu, Sha., Carlton, Shepherd., Amir, Rafi., Konstantinos, Markantonakis. (2024) *Control-Flow Attestation: Concepts, Solutions, and Open Challenges* Computers & Security Volume 150, March 2025, 104254 https://doi.org/10.1016/j.cose.2024.104254.
4. Marco, Chilese., Richard, Mitev., Meni, Orenbach., Ahmad, Atamli., Ahmad-Reza, Sadeghi. (2024) *One for All and All for One: GNN-based Control-Flow Attestation for Embedded Devices* – 2024 IEEE Symposium on Security and Privacy (SP) | 979-8-3503-3130-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SP54263.2024.00251.
5. Методичні вказівки щодо виконання практичних та лабораторних робіт з курсу «Реверсне програмування. Антиналагоджувальні прийоми захисту від реверсу. Середовище програмування masm64: для студентів спеціальності 123 - «Комп'ютерна інженерія» всіх форм навчання [електронне видання] / укладач О.М. Рисований – Х.: НТУ «ХПІ», 2024 – 132 с. https://repository.kpi.kharkov.ua/handle/KhPI-Press/75508.
6. Реверсне програмування. Захист коду. Середовище програмування masm64: навчально-методичний посібник для студентів спеціальності 123 "Комп'ютерна інженерія" всіх форм навчання [електронне видання] / О.М. Рисований. – Харків : НТУ «ХПІ», 2024. – 214 с. https://repository.kpi.kharkov.ua/handle/KhPI-Press/79627.
7. Реверсне програмування. Впровадження коду. Середовище програмування masm64: навчальний посібник для студентів спеціальностей 123 "Комп'ютерна інженерія",

125 "Кібербезпека" / О.М. Рисований. – Харків: «Слово», 2021. – 250 с. https://repository.kpi.kharkov.ua/handle/KhPI-Press/70274.

8. Реверсне програмування. Формат файлу PE. Середовище програмування masm64: навчально-методичний посібник для студентів спеціальності 123 "Комп'ютерна інженерія" всіх форм навчання [електронне видання] / О.М. Рисований. – Харків: НТУ «ХПІ», 2025. – 116 с. https://repository.kpi.kharkov.ua/handle/KhPI-Press/95746

9. Реверсне програмування. Хеш-функції та їх використання. Середовище програмування masm64: навчальний посібник для студентів спеціальності F7 "Комп'ютерна інженерія" всіх форм навчання [електронне видання] / О.М. Рисований. – Харків: НТУ «ХПІ», 2026. – 170 с.

***References:***

1. Hyungseok, Kim., Soomin, Kim., Sang, Kil Cha. (2025) *Towards Sound Reassembly of Modern x86-64 Binaries* – ASPLOS '25: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 Pages 1317 - 1333 https://doi.org/10.1145/3676641.3716026

2. Gibert, D., Mateu, C., & Planes, J. (2020). *The rise of machine learning for detection and classification of malware: Research developments, trends and challenges*. J. Netw. Comput., 153, 102526.

3. Li, X., & Li, Q. (2021). *An IRL-based malware adversarial generation method to evade anti-malware engines.* Comput. Secur., 104, 102118.

4. Qiao, Y., Zhang, W., Tian, Z., Yang, L.T., Liu, Y., & Alazab, M. (2022). *Adversarial malware sample generation method based on the prototype of deep learning detector*. Comput. Secur., 119, 102762.

5. Rysovanyi, O.M. (2024). *Metodichni vkazivki shodo vikonannya praktichnih ta laboratornih robit z kursu «Reversne programuvannya. Antinalagodzhuvalni prijomi zahistu vid reversu. Seredovishe programuvannya masm64: dlya studentiv specialnosti 123 - «Komp'yuterna inzheneriya» vsih form navchannya [Methodological instructions for performing practical and laboratory work on the course "Reverse programming. Anti-debug techniques for protection against reverse. Programming environment masm64: for students of specialty 123 - "Computer Engineering" of all forms of study]* – Kharkiv: NTU "KhPI" [in Ukrainian].

6. Rysovanyi, O.M. (2024). *Reversne programuvannya. Zahist kodu. Seredovishe programuvannya masm64: navchalno-metodichnij posibnik dlya studentiv specialnosti 123 "Komp'yuterna inzheneriya" vsih form navchannya [Reverse programming. Code protection. Programming environment masm64: teaching and methodological manual for students of specialty 123 "Computer Engineering" of all forms of study ]* – Kharkiv: NTU "KhPI" [in Ukrainian].

7. Rysovanyi, O.M. (2021). *Reversne programuvannya. Vprovadzhennya kodu. Seredovishe programuvannya masm64: navchalnij posibnik dlya studentiv specialnostej 123 "Komp'yuterna inzheneriya", 125 "Kiberbezpeka"* [*Reverse programming. Code implementation. Programming environment masm64: a textbook for students of specialties 123 "Computer Engineering", 125 "Cybersecurity"* ] – Kharkiv: "Slovo" [in Ukrainian].

8. Rysovanyi, O.M. (2025). *Reversne programuvannya. Format fajlu PE. Seredovishe programuvannya masm64: navchalno-metodichnij posibnik dlya studentiv specialnosti 123 "Komp'yuterna inzheneriya" vsih form navchannya* [*Reverse programming. File format PE. Programming environment masm64: a textbook for students of specialty 123 "Computer Engineering" of all forms of study*] – Kharkiv: NTU "KhPI", 2025. – 116 p. [in Ukrainian].

9. Rysovanyi, O.M. (2026). *Reversne programuvannya. Hesh-funkciyi ta yih vikoristannya. Seredovishe programuvannya masm64: navchalnij posibnik dlya studentiv specialnosti F7 "Komp'yuterna inzheneriya" vsih form navchannya* – Kharkiv: NTU "KhPI", 2026. – 170 p. [in Ukrainian].