DOI: 10.18372/2225-5036.28.16948

ІМПЛЕМЕНТАЦІЯ НАБОРУ ІНСТРУКЦІЇ RISC-V

Максим Шабан

ШАБАН Максим Радуйович
Рік та місце народження: 1988 рік, м. Київ, Україна.
Освіта: Національний авіаційний університет, 2012 рік.
*Посада :*науковий співробітник Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова, Україна з 2021 року.
Наукові інтереси: інформаційна безпека, прикладне програмування, грід-обчислення, RISC-V.
Публікації: більше 30 наукових публікації, серед яких наукові статті, тези і матеріали доповідей на конференціях.
E-mail: maximsaban@gmail.com.
ORCID ID: 0000-0003-2706-8235.

Анотація *В 70-х роках минулого сторіччя, під час активного розвитку електронно-обчислювальної техніки, розвиток обчислювальних систем пішов двома шляхами: високопродуктивні обчислювальні системи (ОС); вузькопрофільні обчислювальні системи. Фактично, високопродуктивні системи – це універсальні ОС завданням яких є максимальна висока швидкість обчислень за одиницю часу. Вузькопрофільні ОС ставили за мету виконання певних типів завдань, де якраз швидкість обчислень не мала такого значення, а на перші ролі виходили інші технічні характеристики: енергоефективність, ергономіка виробу, необхідність виконання тільки певного роду завдань та інше. Саме вид завдань став вирішальним з точки зору архітектурної реалізації ОС. Розглянемо архітектуру RISC-V з метою оцінки перспективності впровадження її до масового сегменту.*

Ключові слова: *RISC-V, набір інструкцій, аналіз продуктивності апаратних засобів захисту, високопродуктивні обчислювальні системи, мікропроцесори.*

Постановка проблеми

Вимоги щодо захисту критичної інформаційної інфраструктури мають пряму залежність від інформаційних технологій. Існує два аспекти реалізації інформаційних технологій: програмний та апаратний. Характеристики програмної реалізації визначаються алгоритмічними особливостями конкретного механізму захисту, а апаратний аспект визначає загальну продуктивність. Але суттєве збільшення продуктивності може призвести до критичного впливу на надійність функціонування механізму захисту.

Прикладом цього є зростання довжини ключа шифрування внаслідок росту продуктивності електронно-обчислювальної техніки. Тому для збереження безпечного стану критичної інформаційної інфраструктури необхідно постійно приділяти увагу аналізу продуктивності апаратних засобів захисту інформації. Існує поняття абсолютної та практичної стійкості механізму захисту. В даній роботі приділяється увага масовому сегменту електронно-обчислювальної техніки, яка більше впливає саме на практичну стійкість механізму захисту. В 70-х роках минулого сторіччя, під час активного розвитку електронно-обчислювальної техніки, розвиток обчислювальних систем пішов двома шляхами: високопродуктивні ОС; вузькопрофільні обчислювальні системи.

Фактично, високопродуктивні системи – це універсальні ОС завданням яких є максимальна висока швидкість обчислень за одиницю часу. Інші технічні характеристики є похідними від цього. У свою чергу вузькопрофільні ОС ставили за мету виконання певних типів завдань, де якраз швидкість обчислень не мала такого значення, а на перші ролі виходили інші технічні характеристики: енергоефективність, ергономіка виробу, необхідність виконання тільки певного роду завдань та інше. Саме вид завдань став вирішальним з точки зору архітектурної реалізації ОС. Розглянемо архітектуру RISC-V з метою оцінки перспективності впровадження її до масового сегменту.

Основна частина

Будь-яка ОС має в своєму складі набор логік, що дозволяють виконувати обчислення. Цей набор, на апаратному рівні, виконуються у вигляді транзисторних схем окремого блоку (ядро) мікропроцесора. Сучасний мікропроцесор (див. рис.1) – інтегральна схема, в якій реалізовано багато блоків, що вкупі створюють єдину екосистему центрального процесора. Інструкція [1] – це імплементація цих логік в уніфікованому вигляді на першому рівні абстракції. Набор інструкції визначає архітектуру мікропроцесору. Мікропроцесори можуть бути «general purpose processor» та «special purpose processor».

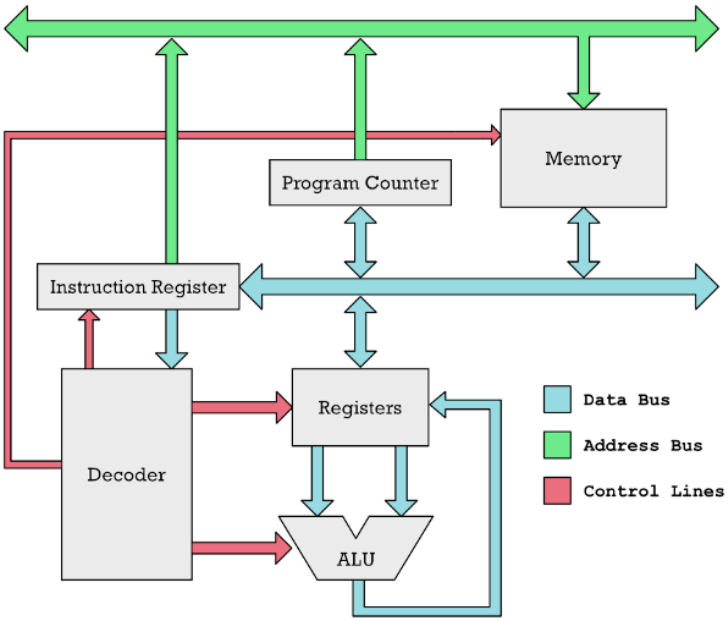

Рис.1. Організація роботи мікропроцесора

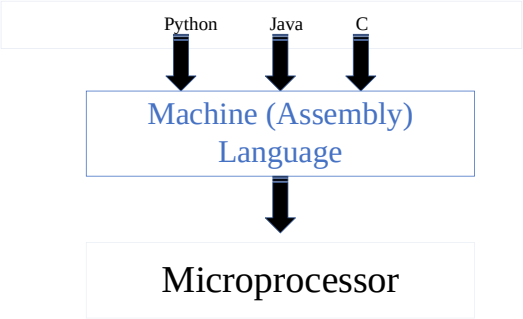

Рис. 2. Процесори загального користування

General purpose processor (процесори загального призначення) – це мікропроцесори (рис. 2), що не прив'язані до певної мови чи програмного забезпечення та не інтегрований з ним.

Special purpose processor (процесори вузького призначення) – це одноцільовий процесор, що створюється шляхом розробки спеціальної цифрової схеми. Переваги та недоліки [4] більш-менш протилежні процесору загального призначення: Переваги: продуктивність дуже хороша, малий розмір (точно підходить для одного рішення), споживають мало енергії. Організація перших моделей процесорів x86 - була спрямована, зокрема, на скорочення обсягу програм, критичного для систем того часу, що відрізнялися малою оперативною пам'яттю. Розширення спектра операцій, реалізованих системою команд, дозволило зменшити обсяг програм, і навіть трудомісткість їх написання та налагодження. Проте збільшення кількості команд підвищило трудомісткість розробки їх топологічних та мікропрограмних реалізацій. Останнє виявилося у подовженні термінів розробки CISC-процесорів, і навіть у прояві різних помилок у роботі. Крім того, нерегулярність потоку команд обмежила розвиток топології тимчасовим паралелізмом обробки інструкцій на конвеєрі "вибірка команди-дешифрація команди-вибірка даних-обчислення-запис результату".

Ці недоліки зумовили необхідність розробки альтернативної архітектури, націленої насамперед на зниження нерегулярності потоку команд зменшенням їх загальної кількості. Це було реалізовано в RISC-процесорах, назва яких означає "чіпи зі скороченою системою команд" (Reduced Instruction Set Computer). Одночасно "класичні" процесори одержали позначення CISC (Complex Instruction Set Computer) - комп'ютер зі складним набором інструкцій.

Скорочення нерегулярності потоку команд дозволило збагатити топологію RISC-процесорів просторовим паралелізмом, спеціалізованими апаратними АЛУ (ALU - блок логіки та арифметики = Arithmetic (and) Logic Unit), незалежними кешами даних і команд, роздільними шинами вводу-виводу. Останні зокрема збільшили довжину конвеєрів команд. Все це підвищило і продуктивність – збільшенням кількості операцій, що виконуються за один такт, та швидкодія - скороченням шляху транзактів - RISC-процесорів. У цьому термін розробки даних чіпів свідчить у тому, що її трудомісткість менше, ніж у разі CISC-процесорів.

RISC-V [2] (вимовляється як "RISC-FIVE") - це архітектура з відкритим набором команд (ISA), заснована на принципі обчислень зі скороченим набором команд (RISC). V представляє п'яте покоління RISC (комп'ютер зі скороченим набором команд), що означає, що було чотири покоління чіпа прототипу процесора RISC. Кожне покоління процесорів RISC створюється під керівництвом однієї людини, тобто професора Девіда Паттерсона з Каліфорнійського університету в Берклі. Його мета – стати відкритими в області архітектури набору команд з програмами, що охоплюють пристрої IoT [6] (Інтернет речей), настільні комп'ютери, високопродуктивні комп'ютери та багато інших областей. Це зв'язано з тим, що дослідники UCB виявили такі проблеми у поточної архітектури набору команд у процесі вивчення архітектури набору команд: більшість архітектур з набором команд захищені патентами, наприклад x86, MIPS та Alpha.

Використання цих архітектур потребує авторизації, що обмежує конкуренцію, а також стримує інновації; поточна архітектура набору інструкцій щодо складна, не підходить для академічних досліджень, а велика складність викликана неякісним дизайном або історичним багажем; поточна архітектура набору команд орієнтована на певну область. Наприклад, x86 в основному призначений для серверів [3], а ARM – в основному для мобільних терміналів. Тому у відповідній архітектурі набору інструкцій було зроблено багато оптимізації для даної області, і в ній відсутня Єдина архітектура може застосовуватися до кількох областей; архітектура комерційного набору команд легко залежить від розвитку підприємства. Наприклад, архітектура Alpha майже зникла з придбанням DEC; існуючі різні архітектури наборів команд незручні для розширень користувача для конкретних додатків.

З цією метою дослідники Крсте Асанович, Ендрю Вотерман та Юнсуп Лі з Каліфорнійського університету в Берклі вирішили розробити нову архітектуру на рівні інструкцій та вирішили відкрити вихідний код під ліцензією BSD, сподіваючись створити більш інноваційні процесори. Є більше процесорів з відкритим вихідним кодом і для здешевлення електронних продуктів.

З моменту свого офіційного випуску в 2014 році RISC-V був схвалений багатьма компаніями, включаючи Google, IBM, Oracle тощо, а також багатьма відомими університетами та дослідницькими інститутами, включаючи Кембриджський університет, ETH Zurich, Індійський технологічний інститут та Китайську академію наук. Увага та участь, екологічне середовище, що оточує RISC-V, поступово покращується, і багато процесорів з відкритим вихідним кодом та system-on-chip (SoC) приймають архітектуру RISC-V. Ці процесори мають як скалярні, і суперскалярні процесори, і навіть одноядерні процесори. Існують також багатоядерні процесори. Ця стаття коротко представить базовий дизайн архітектури RISC-V, а потім докладно опише процесори з відкритим вихідним кодом та SoC, які в даний час використовують архітектуру RISC-V.

Архітектура набору інструкцій RISC-V організована в групи інструкцій (розширення Instruction Set Architecture (ISA)). Ви можете змішувати і поєднувати їх, як хочете. Наприклад, ви можете мати процесор RISC-V, який реалізує мінімальний мінімум, або процесор RISC-V, який реалізує всі розширення архітектури набору інструкцій, залежно від потреб проектування.

У наступній таблиці наведено основні розширення архітектури набору інструкцій, які були ратифіковані RISC-V Foundation, і розширення архітектури набору інструкцій, які зараз розробляються.

Наведена таблиця буде розширена в майбутньому, оскільки буде додано більше розширень ISA.

Хоча список вже великий, може виникнути ситуація, коли не буде відповідного готового розширення ISA, яке б відповідало потребам дизайну. У цьому випадку специфікація RISC-V дозволяє додати власне розширення ISA. Це може бути «секретним соусом» компанії та ключовою відмінністю. Завдяки природі екосистеми RISC-V, спеціальні розширення ISA не порушують відповідність головній специфікації; навіть з додатковими інструкціями ваш процесор все ще повністю сумісний з RISC-V і може запускати загальний програмний стек, взятий з екосистеми.

На рис. 3 нижче показано, як користувацьке розширення ISA вписується в програмний стек. На найнижчому рівні є RISC-V-сумісний процесор із користувацьким розширенням ISA. Він працює під керуванням ОС, будь то "гола" або багата операційна система (ОС). Його можна скомпілювати з будь-яким компілятором, сумісним зі стандартним процесором RISC-V (без спеціальних розширень ISA). Поверх ОС є три програми. App1 — це загальна програма, яка не вимагає прискорення. Ви можете використовувати загальнодоступний, готовий компілятор (наприклад, GCC) для його компіляції, або навіть можливо використовувати попередньо скомпільований додаток; процесор RISC-V зможе запустити його. App2 і App3 є важливими, що мають працювати якомога швидше. Вони повинні бути скомпільовані компілятором, який спеціально знає про користувацьке розширення ISA.

Компілятор може використовувати нові інструкції, які прискорять App2 і App3.

Таблиця 1

Розширення ISA	Ухвалено	Пояснення
I/E	Так	Інструкції для основних цілочисельних операцій. Це єдине розширення, яке є обов'язковим. I потребує 32 регістри, E вимагає лише 16.
M	Так	Інструкція з множення та ділення
C	Так	Компактні інструкції, які мають лише 16-бітове кодування. Це розширення дуже важливе для програм, які потребують малого обсягу пам'яті.
F	Так	Інструкції з плаваючою комою одинарної точності
D	Так	Інструкції з плаваючою комою подвійної точності
A	Так	Інструкції з атомарної пам'яті
B	Ні	Інструкції з маніпуляції з бітами. Розширення містить інструкції, які використовуються для маніпуляцій з бітами, таких як обертання або інструкції встановлення/очистки бітів.
V	Ні	Векторні інструкції, які можна використовувати для високопродуктивних обчислень
P	Ні	Цифрова обробка сигналів і упаковані єдині інструкції декількох команд даних, необхідні для вбудованих процесорів цифрової обробки сигналів.

На рис. 4 нижче показано інший приклад RISC-V-сумісного процесора з користувацьким розширенням ISA. App1 не використовує спеціальне розширення ISA. App2 і App3 використовують загальний API. API реалізовано бібліотекою, яка знає про спеціальне розширення ISA, яке знову ж таки може прискорити App2 і App3. I App2, і App3 можна повторно використовувати в стандартному процесорі RISC-V. Все, що потрібно, це бібліотека, яка реалізує необхідний API. У цій системі переміщення App2 і App3 з RISC-V із користувацьким розширенням ISA на RISC-V без розширення є простим і не вимагає перенесення програм.

Користувацькі архітектури набору інструкцій або користувацькі розширення ISA не є новими і існують деякий час. Однак вони зазвичай вимагають великих зусиль. Спочатку потрібно визначити інструкцію. Потім вам потрібно додати їх до компілятора C, симуляторів, налагоджувачів та інших інструментів і переконатися, що зміни додають те саме до всіх цих різних інструментів. Додавання спеціальної інструкції також вимагає деяких зусиль вручну.

Як правило, вам потрібна команда, яка додасть нові інструкції до комплекта для розробки програмного забезпечення, щоб інструменти програмування могли передавати та компілювати інструкції. Вам також потрібно додати новий код до симулятора набору інструкцій. Нарешті, Register Transfer Level (RTL) має бути розширений, і будь-які зміни в RTL мають бути перевірені. Залежно від кількості ручних зусиль, розширення ISA можуть бути досить дорогими з точки зору часу та ресурсів.

Рис.3. Приклад взаємодії ISA з розширенням ISA

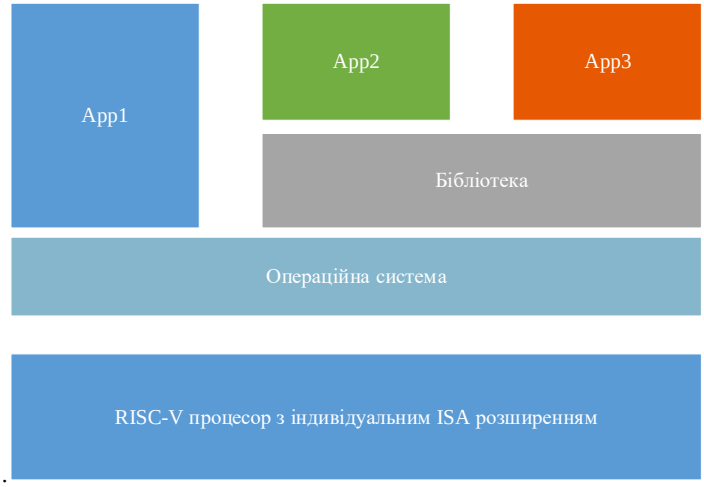

Рис.4. Приклад взаємодії ISA з розширенням ISA

Щоб знизити вартість розширень ISA, нам потрібно максимально автоматизувати – від визначення відповідних інструкцій до перевірки RTL. Це саме те, що Codasip може зробити дуже добре. Codasip надає інструмент - electronic design automation (EDA) під назвою Studio, який дозволяє налаштовувати готові процесори, які також пропонує Codasip. Ви можете розпочати свою роботу з повноцінним процесором, сумісним із RISC-V, і додати власні розширення ISA відповідно до ваших потреб, або ви можете створити свій власний процесор RISC-V з нуля. Індивідуальні інструкції можуть бути простими [5], наприклад, варіантами інструкцій, що множитимуться та накопичуються, або вони можуть бути користувацькими інструкціями керування, такими як нульові накладні цикли (апаратні цикли). Ви також можете мати спеціальні інструкції щодо завантаження/збереження з наступним або попереднім збільшенням.

Як бачите, користувацькі інструкції відрізняються за складністю. Це, серед іншого, впливає на можливості компілятора C і продуктивність результуючого процесора. Прості інструкції можуть використовуватися компілятором C без необхідності змінювати оригінальний код C. Іншими словами, у вас може бути складені з однієї програми, і ви можете зібрати її для x86 або RISC-V.

Якщо інструкція занадто складна, єдиний спосіб її використання — це вбудована збірка або «C intrinsic». Обмеження становить близько ~25 операцій і ~3 виходи.

З іншого боку, більш складні інструкції зазвичай покращують продуктивність, тому результат вартий зусиль. Існує простий спосіб інтегрувати вбудовану збірку або «intrinsic» в бібліотеку. Бібліотека також має загальну реалізацію.

Перевага такої бібліотеки полягає в тому, що ви можете мати одну реалізацію кінцевої програми, і ви можете скомпілювати її для кількох цілей. Кожна ціль може використовувати різну реалізацію. Додатку не потрібно знати про остаточну реалізацію.

```
// universal byteswap() function
inline uint32_t byteswap(const uint32_t word) {
#ifdef ISA_BYTE_SWAP
    // C compiler intrinsic
    return __byteswap(word);
#else
    return ((word >> 24) & 0x000000ff) |
           ((word <<  8) & 0x00ff0000) |
           ((word >>  8) & 0x0000ff00) |
           ((word << 24) & 0xff000000);

#endif
}
```

У наступному прикладі показано фрагмент коду такої бібліотеки [7].

Він являє собою просту функцію заміни байтів. Якщо вказаний макрос присутній, компілятор C має спеціальну інструкцію, яка виконує підкачку. В іншому випадку використовується стандартний підхід. Код, згенерований для першої частини, досить простий; лише одна інструкція

```
byteswap:
  byteswap x10, x10
  c.jr ra // return from function
```

Друга частина виконана в дванадцяти інструкціях на RV32IMC (див. нижче), X10 утримує значення слова і в кінці також зберігає значення, що повертається.

Таким чином, підвищується не тільки продуктивність програми; розмір коду також значно менший. Ви можете мати багато інших подібних інструкцій, включаючи підрахунок спливаючих даних, різні інструкції маніпулювання бітами, інструкції керування тощо.

```
byteswap:
  srl x15, x10, 8
  lui x14, %hi( 65280 )
  add x14, x14, 65280 & 0xfff
  c.and x15, x14
  srl x14, x10, 24
  c.or x15, x14
  sll x14, x10, 8
  lui x13, 16711680>>12 &0xfffff
  c.and x14, x13
  sll x13, x10, 24
  c.or x14, x13
  or x10, x14, x15
  c.jr ra // return from function.
```

RISC-V ISA [8] пропонує широкий вибір груп ISA на вибір. Обов'язковою є лише базова група (I/E), решта необов'язкова. Концепція дозволяє дизайнерам вибирати саме те, що їм потрібно. Якщо цього все ще недостатньо для бажаних результатів, специфікація RISC-V дозволяє додавати додаткові інструкції, зберігаючи RISC-V-сумісність. Це значна перевага, що дозволяє повторно використовувати програмний стек спільноти та прискорювати лише важливі частини. Додавання користувальницьких інструкцій можна виконати вручну, але це завдання, яке схильне до помилок, яке також вимагає значної кількості ресурсів. Codasip націлений на це за допомогою своєї EDA tool Studio. Процесор RISC-V і будь-які додані користувацькі інструкції описані мовою опису архітектури високого рівня CodAL. Studio використовує опис для створення комплекту для розробки програмного забезпечення та реалізації процесора. Високий рівень автоматизації дозволяє додавати нові інструкції протягом декількох годин, днів або тижнів.

Висновки. Проведений аналіз в статті показує, що RISC-V — це багаторівневий і розширюваний ISA, тому розробники можуть легко впроваджувати мінімальний набір інструкцій, чітко визначені розширення та спеціальні розширення для створення спеціальних процесорів для найсучасніших робочих навантажень. Зменшує кількість інвестиції, дозволяючи розробникам використовувати усталені та загальні блоки інтелектуальної власності за допомогою зростаючого набору спільних інструментів і ресурсів розробки спільноти. Забезпечує гнучкість для створення тисяч можливих нестандартних процесорів. Оскільки реалізація визначається не на рівні ISA, а скоріше складом SoC та іншими атрибутами дизайну, інженери можуть зробити великий, маленький, потужний або легкий процесор із своїми проектами.

Прискорює час виходу на ринок завдяки співпраці та повторному використанню інтелектуальної власності з відкритим кодом. RISC-V не тільки скорочує витрати на розробку, але й дозволяє розробникам швидше виводити свої проекти на ринок.

ЛІТЕРАТУРА

[1] Instruction sets should be free: The case for risc-v/ Krste Asanović and David a Patterson. // EECA Department, University of California, Berkeley – 2014. – P. 4.

[2] Design and implementation of adynamic information flow tracking architecture to secure a RISC-V core for IoT applications/Christian Palmiero, Giuseppe Di Guglielmo, Luciano Lavagno, and Luca P Carloni. //HighPerformance extreme Computing Conference (HPEC) – 2018. – P. 2.

[3] The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA / Waterman, Andrew; Asanović, Krste // EECA Department, University of California, Berkeley – 2019. – P. 2.

[4] Accelerating Multimedia with Enhanced Microprocessors"/Lee, Ruby B.// CiteSeerX Berkeley – 1995. – P. 22-32.

[5] RISC-V Vector Extension Proposal / Schmidt, Colin; Ou, Albert; Lee, Yunsup; Asanović, Krste // Workshop c – 2015. – P. 19.

[6] 3.6.2 Ultra-Low-Power Co-Processor // Espressif Systems. – 2021. – P. 11.

[7] The RISC-V Compressed Instruction Set / Waterman Andrew //Berkeley – 2015. – P. 5-14.

[8] The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA version 2.1 / Waterman, Andrew; Asanović, Krste // University of California, Berkeley – 2016 – P. 21.

УДК 004.056:004.75

Shaban M.R. Implementation of the RISC-V instruction set.

Abstract. *In the 70s of the last centuries, during the active development of electronic computing technology, the development of computing systems went in two ways: high-performance computing systems (OS); narrow-profile computing systems. In fact, high-performance systems are universal operating systems whose task is to maximize the high speed of computing per unit of time. Narrow-profile operating systems aimed at performing certain types of tasks, where just the speed of calculations did not matter so much, and other technical characteristics were obtained for the first roles: energy efficiency, product ergonomics, the need to perform only certain types of tasks, and so on. Specifically, the type of tasks has become decisive in terms of the architectural implementation of the OS. Let's consider the RISC-V architecture to assess the prospects of its introduction into the mass segment.*

Keywords*: RISC-V, set of instructions, performance analysis of hardware protection devices, high-performance computing systems, microprocessors.*

Шабан Максим Радуйович, науковий співробітник Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова, Україна.

Maksym Shaban, researcher at the Institute of Modeling Problems in Energy named after G.E. Pukhova, Ukraine.

Отримано *7 липня 2022 року,* затверджено редколегією *14 листопада 2022 року*