МАГІСТЕРСЬКА РОБОТА

МР.ІПм-17.00.00.000 ПЗ

Група ІПм-23-1

Войтович Степан

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Войтович Степан Олегович

(прізвище, ім'я, по батькові)

УДК 004.25

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі мікроархітектурних методів керування потоками протидії загрозам безпеки програмного забезпечення

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Войтович С.О.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Бандура Вікторія Валеріївна, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц.			Бандура В.В.
(посада)	(підпис)	(дата)	(ініціали та прізвище)

Нормоконтроль

доц.			Вовк Р.Б.
(посада)	(підпис)	(дата)	(ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**

доц. В.В. Бандура

“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Войтовичу Степану Олеговичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі мікроархітектурних методів керування потоками протидії загрозам безпеки програмного забезпечення ”

керівник проекту (роботи) Бандура Вікторія Валеріївна, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови мікроархітектурних моделей протидії загрозам безпеки програмного забезпечення

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд теоретичних основ мікроархітектурних безпекових методів
2. Аналіз сучасних концепцій та стандартів керування потоками інструкцій процесора
3. Огляд новітніх стратегій атак на пам’ять процесора
4. Розробка та реалізація інтегрованої моделі відстеження потоків

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Типова архітектура виконання інструкцій (рис. 1.6, ст. 26)
2. Конфігурація RISC процесорів (рис. 2.7, ст. 44)
3. Новітні атаки через побічні канали (рис. 3.5, ст. 51)
4. Загальна схема пропонованої моделі (рис. 4.2, ст. 63)
5. Потоки моделі на рівні вентилів (рис. 4.4, ст. 68)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Вивчення літератури з теми дослідження	01.10.2024	виконано
2	Огляд мікроархітектурних методів забезпечення безпеки	21.10.2024	виконано
3	Дослідження методологій керування потоками інструкцій процесора	30.10.2024	виконано
4	Формулювання вимог до нової архітектурної моделі	12.11.2024	виконано
5	Розробка моделі відстеження потоків, проведення робочої симуляції та аналіз ресурсної ефективності	25.11.2024	виконано
6	Оцінка моделі та огляд перспектив її застосування	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 87 с., 31 рис., 4 табл., 40 джерел.

Тема: Моделі мікроархітектурних методів керування потоками протидії загрозам безпеки програмного забезпечення.

Об'єкт дослідження: процеси забезпечення інформаційної безпеки програмних систем через управління потоками даних в сучасних процесорах.

Мета роботи: розробка інтегрованої методології відстеження інформаційних потоків для забезпечення безпеки виконуваних процесів.

Предмет дослідження: моделі відстеження потоків даних на рівні інструкцій і даних для протидії загрозам безпеки програмного забезпечення в архітектурі RISC-V.

Результати дослідження:

Проведено аналіз існуючих мікроархітектурних методів протидії загрозам, на основі якого запропоновано власну багаторівневу модель відстеження атакованих потоків на базі ядра RISC-V.

Висновок:

В результаті дослідження розроблено багаторівневу модель відстеження потоків даних, яка забезпечує комплекс механізмів безпеки програмного забезпечення та здійснює профілактику витоку даних.

ВІДСТЕЖЕННЯ ІНФОРМАЦІЙНИХ ПОТОКІВ, МОДЕЛІ ПРОТИДІЇ ЗАГРОЗАМ, ПРОЦЕСОРНІ АТАКИ, МІКРОАРХІТЕКТУРНІ МОДЕЛІ, БЕЗПЕКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ІНСТРУКЦІЇ ПРОЦЕСОРА

ABSTRACT

Master's thesis: 87 p., 31 fig., 4 tab., 40 sources.

Topic: Models of microarchitectural methods of flow control to counteract software security threats.

Object of research: processes of ensuring information security of software systems through the management of data flows in modern processors.

Purpose: to develop an integrated methodology for tracking information flows to ensure the security of processes.

Subject of research: models for tracking data flows at the instruction and data levels to counteract software security threats in the RISC-V architecture.

Research results:

An analysis of existing microarchitectural methods for countering threats is carried out, on the basis of which our own multilevel model for tracking attacked flows based on the RISC-V core is proposed.

Conclusion:

As a result of the study, a multilevel model for tracking data flows has been developed, which provides a set of software security mechanisms and prevents data leakage.

INFORMATION FLOW TRACKING, THREAT COUNTERACTION MODELS, PROCESSOR ATTACKS, MICROARCHITECTURAL MODELS, SOFTWARE SECURITY, PROCESSOR INSTRUCTIONS

ЗМІСТ

Стор.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AES – Advanced Encryption Standard

AES-NI – Advanced Encryption Standard New Instructions

ALU – Arithmetic Logic Unit

AMDU – Advanced Memory Dependent Unit

AMPU – Advanced Memory Processing Unit

ARM – Advanced RISC Machines

BHR – Branch History Register

CFI – Control Flow Integrity

CFG – Control Flow Graph

CGRA – Coarse-Grained Reconfigurable Architecture

CISC – Complex Instruction Set Computing

CPU – Central Processing Unit

CRAS – Certified Random Access Security

CSR – Control and Status Register

DDoS – Distributed Denial of Service

DMA – Direct Memory Access

DRAM – Dynamic Random Access Memory

EEPROM – Electrically Erasable Programmable Read-Only Memory

FESVR – Front-End Server

FPGA – Field Programmable Gate Array

GLIFT – Gate-Level Information Flow Tracking

GPP – General-Purpose Processor

GPR – General-Purpose Register

HTIF – Host-Target Interface

ICT – Indirect Control Transfer

IFT – Information Flow Tracking

IR – Intermediate Representation

IRM – Inline Reference Monitor

ISA – Instruction Set Architecture

LUT – Look-Up Table

MPK – Memory Protection Keys

NAND – Not AND (Logic Gate)

OSI – Open Systems Interconnection

PC – Program Counter

PYNQ – Python Productivity for Zynq

RAM – Random Access Memory

RISC – Reduced Instruction Set Computing

RoCC – Rocket Custom Coprocessor

ROM – Read-Only Memory

RSA – Rivest–Shamir–Adleman (Cryptographic Algorithm)

SDRAM – Synchronous Dynamic Random Access Memory

SFI – Software Fault Isolation

SGX – Software Guard Extensions

SRAM – Static Random Access Memory

TAGE – Tagged Geometric History Length Predictor

TEE – Trusted Execution Environment

TLS – Transport Layer Security

TSO – Total Store Ordering

VCT – Virtual Control Transfer

WMO – Write Memory Order

XOM – eXecute-Only Memory

XSS – Cross-Site Scripting

ВСТУП

Актуальність роботи

Нормальний розвиток людства неможливий без створення, накопичення та обробки інформації. Кілька тисяч років тому об’єм накопичених даних був відносно невеликим, і для їх обробки та зберігання достатньо було використовувати паперові носії інформації. Проте кількість даних постійно збільшувалась, відбулись друга (середина XIV ст.), третя (кінець XIX ст.) та четверта (70-і рр. XX ст.) інформаційні революції, які призвели до стрімкого росту світового обсягу даних. Сьогодні світ характеризується явищем інформаційного вибуху – швидким збільшенням кількості публікацій, що призводить до щорічного збільшення масиву світової інформації на 30%. В таких умовах, проблеми пошуку та обробки даних можуть вирішуватись тільки за допомогою високошвидкісних обчислювальних технологій, ключовими складовими яких залишаються центральні процесори.

Актуальність роботи обумовлена високим рівнем загроз інформаційного середовища, що вимагає від компаній, державних установ та навіть окремих користувачів приймати ряд заходів із забезпечення безпеки власного програмного й апаратного устаткування. Новітні технологічні досягнення, такі як штучний інтелект, а також зростання глобальної напруженості, зумовлюють неймовірно високий рівень комп’ютерних загроз.

За статистикою [1], світова кіберзлочинність принесла 8 трлн. дол. втрат компаніям у попередньому 2023 році. Ця цифра, як очікується, зросте майже до 24 трильйонів доларів у 2027 році.

Порівняння роботи з відомими розв’язаннями проблеми

Серед різнотипових моделей протидії загрозам безпеки програмного забезпечення особлива увага була приділена моделям відстеження інформаційних потоків. Ці моделі надають системний захист від атак, пов’язаних із пам’яттю, зокрема атак переповнення буфера та пошкодження даних.

FlexiTaint – програмований апаратний прискорювач розповсюдження плям, що реалізується без модифікації системної шини або основної пам'яті. Також відсутні значні модифікації критичного до продуктивності інтерфейсного конвеєра процесора або його механізму потоку даних. FlexiTaint зберігає дані про забруднення пам'яті у вигляді віртуального упакованого масиву. FlexiTaint характеризується низькими накладними витратами продуктивності навіть при використанні двобітових міток [2].

Апаратна реалізація HyperFlow [3] на базі RISC-V забезпечує надійний захист інформаційних потоків за допомогою механізмів статичної перевірки потоків ChiselFlow. HyperFlow підтримує складні безпекові політики трафіку даних, що налаштовуються користувачем в ході роботи. Дрібнозернисті інформаційні децентралізовані механізми забезпечують контрольовану комунікацію між процесами та системними викликами в різних доменах безпеки.

Мета і задачі дослідження

Метою магістерської роботи є розробка інтегрованої методології відстеження інформаційних потоків для забезпечення безпеки виконуваних процесів. Захист цілісності системи повинен забезпечуватись відстеженням інформації з ненадійних каналів зв'язку.

Нові підходи спрямовані на виявлення ненадійних джерел за допомогою механізмів присвоєння/відстеження тегів і моделювання інформаційних втрат окремих складових за допомогою «прихованої» логіки.

Досягнення мети охоплювало розв’язання таких **задач**:

1) ознайомлення з основами мікроархітетурних безпекових принципів;
2) дослідження сучасних методологій керування потоками процесора;
3) аналіз поширених атак на процесори;
4) формування вимог до пропонованої мікроархітектурної моделі;
5) розробка нової концепції моделі відстеження потоків даних;
6) проведення програмної симуляції та оцінка її результатів.

Об’єктом дослідження є процеси забезпечення інформаційної безпеки програмних систем через управління потоками даних в сучасних процесорах.

Предметом дослідження є моделі відстеження потоків даних на рівні інструкцій і даних для протидії загрозам безпеки програмного забезпечення в архітектурі RISC-V.

Методи дослідження

В роботі застосовувались: аналіз наукової літератури; порівняльний аналіз існуючих концепцій та стандартів; метод моделювання для відстеження потоків у процесорах RISC-V; емпіричні методи проведення тестів і робочих імітацій.

Наукова новизна одержаних результатів

Запропоновано мікроархітектурну модель керування потоками, що забезпечує багаторівневий захисний механізм обчислювальних потужностей, інтегрує грубо- та дрібнозернисті підходи відстеження інформаційних потоків.

Практичне значення одержаних результатів

Розроблено архітектурну модель, призначену для виявлення атак на пам’ять та витоків даних у важливих модулях. Вона може інтегруватись з існуючими апаратними системами на базі архітектури RISC-V.

Особистий внесок

Основні результати:

1) проаналізовано наявні методи керування потоків, визначено ряд проблем та перспектив їхнього подолання;

2) розроблено структурний опис нової моделі відстеження потоків з доданим модулем тегів;

3) проведено тестування та систематизовано отримані числові показники продуктивності, зокрема час виявлення загроз, кількість використаної площі процесора.

Структура магістерської роботи

Магістерська робота викладена на 87 сторінках друкованого тексту, який складається із вступу, чотирьох розділів, висновків, списку використаних джерел (40 найменувань). Робота містить 4 таблиці та 31 рисунок.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ МІКРОАРХІТЕКТУРНИХ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ

1.1 Основи мікроархітектури комп'ютерних систем

Мікроархітектура комп'ютерних систем — це комплексний рівень проектування центрального процесора, який визначає, як інструкції машинного рівня реалізуються у фізичному апаратному середовищі. Цей рівень є визначальним для продуктивності, енергоефективності та безпеки сучасних обчислювальних систем. З урахуванням постійно зростаючих вимог до обчислювальної потужності та захисту даних, мікроархітектурні рішення залишаються ключовою сферою досліджень і розробок у галузі комп'ютерної інженерії.

Одним із центральних елементів мікроархітектури є ієрархія пам'яті, яка забезпечує ефективний доступ до даних через використання кеш-пам'яті різних рівнів (L1, L2, L3). Кеш-пам'ять слугує проміжним буфером між основною пам'яттю та процесором, що дозволяє мінімізувати час затримки при доступі до часто використовуваних даних. Проте саме цей компонент є об'єктом багатьох атак сторонніх каналів, зокрема атак типу «Flush + Reload» і «Prime + Probe». Ці атаки дозволяють зловмисникам досягати конфіденційної інформації через аналіз часових характеристик доступу до кешованих даних [4-7]. Для протидії таким загрозам розробляються методи, які включають рандомізацію доступу до кешу, ізоляцію потоків даних, а також впровадження апаратних засобів захисту, таких як динамічна реструктуризація кеш-пам'яті [5].

Іншим важливим компонентом є механізм позачергового виконання інструкцій, який дозволяє процесору виконувати інструкції не за порядком їх надходження, а за логікою готовності операндів. Це підвищує загальну ефективність використання обчислювальних ресурсів. Проте позачергове виконання створює умови для атак, які експлуатують часові розбіжності між

виконанням інструкцій. Наприклад, атаки типу «Spectre» використовують механізми спекулятивного виконання для отримання доступу до пам’яті, яка не повинна бути доступною у рамках певного потоку виконання [8]. Вирішення цієї проблеми вимагає вдосконалення алгоритмів контролю спекулятивного виконання, а також впровадження бар’єрів безпеки, які перешкоджають витоку даних через сторонні канали.

Провідним аспектом сучасної мікроархітектури є механізми прогнозування розгалужень, які визначають, яким чином процесор обробляє можливі гілки виконання програми до того, як стане відомо, яка з них буде виконана [6]. Прогнозування розгалужень значно підвищує продуктивність, зменшуючи затримки в роботі конвеєра інструкцій. Однак цей механізм також може бути використаний для створення атак, які спрямовані на виконання шкідливих інструкцій під виглядом допустимих. Для мінімізації таких ризиків розробляються алгоритми, які динамічно адаптують процес прогнозування, враховуючи потенційні загрози, і запроваджують системи перевірки коректності виконання інструкцій [9].

Криптографічні інструкції, такі як AES-NI, реалізовані безпосередньо на рівні мікроархітектури, забезпечують апаратне прискорення криптографічних операцій. Це дозволяє значно зменшити час, необхідний для шифрування та дешифрування даних, і підвищує їхню безпеку. Проте навіть ці механізми можуть бути вразливими до атак сторонніх каналів, які використовують часові затримки або енергетичні характеристики виконання інструкцій для відновлення ключів шифрування [7]. Для запобігання таким атакам впроваджуються додаткові заходи, такі як рівномірний розподіл енергоспоживання та введення псевдовипадкових затримок під час виконання криптографічних операцій.

Інтеграція апаратних засобів захисту, таких як Intel SGX (Software Guard Extensions) і ARM TrustZone, є наступним важливим напрямком розвитку мікроархітектури. Ці технології забезпечують створення захищених середовищ для виконання критично важливих операцій, ізолюючи їх від основної операційної системи [5]. Це дозволяє мінімізувати ризики несанкціонованого

доступу до конфіденційної інформації навіть у разі компрометації системного рівня. Проте ці рішення також мають свої обмеження, які пов’язані з продуктивністю та складністю реалізації.

Intel SGX є набором інструкцій, що забезпечує захист користувацьких конфіденційних даних від несанкціонованого доступу, створюючи в пам’яті надійне середовище виконання. SGX шифрує розділи пам’яті з допомогою інструкцій безпеки, властивих CPU. Цей спосіб апаратного шифрування дозволяє користувачам захистити свої найважливіші дані, помістивши їх у середовище з високим рівнем безпеки. Перевагами SGX користуються в багатьох галузях, причому це не завжди стосується IT-додатків. Конфіденційними даними може бути будь-яка інформація, що завдасть шкоди при її розкритті. Щоб захиститися від уразливостей безпеки операційної системи, які можуть мати хост-пристрої, основні частини проекту розгортаються в певному довіреному середовищі виконання (TEE).

Дослідження ефективності мікроархітектурних рішень проводиться з використанням симуляційних інструментів, таких як Gem5, що дозволяє моделювати поведінку процесорів при різних умовах експлуатації [8]. Це сприяє виявленню потенційних вразливостей та розробці методів їхнього усунення ще на етапі проектування. Крім того, системний аналіз із використанням UML забезпечує формалізацію архітектурних рішень, дозволяючи розробляти оптимальні конфігурації процесорів для конкретних сценаріїв використання. Мікроархітектура є складним і багаторівневим механізмом, який визначає функціональність, продуктивність і безпеку комп’ютерних систем. Її вдосконалення вимагає інтегрованого підходу, який враховує як технічні, так і економічні аспекти. Це включає розробку нових алгоритмів обробки інструкцій, ізоляцію потоків даних, оптимізацію кеш-пам’яті та інтеграцію засобів криптографічного захисту, які не лише підвищують рівень безпеки, але й забезпечують високий рівень продуктивності систем. У сучасному цифровому світі вдосконалення мікроархітектури є критично важливим для забезпечення стійкості до зростаючих кіберзагроз [10].

На рисунку 1.1 представлено мікроархітектурну схему ядра процесора Skylake, яка демонструє взаємодію ключових компонентів, таких як блоки виконання, ієрархія кеш-пам’яті та модулі управління інструкціями.

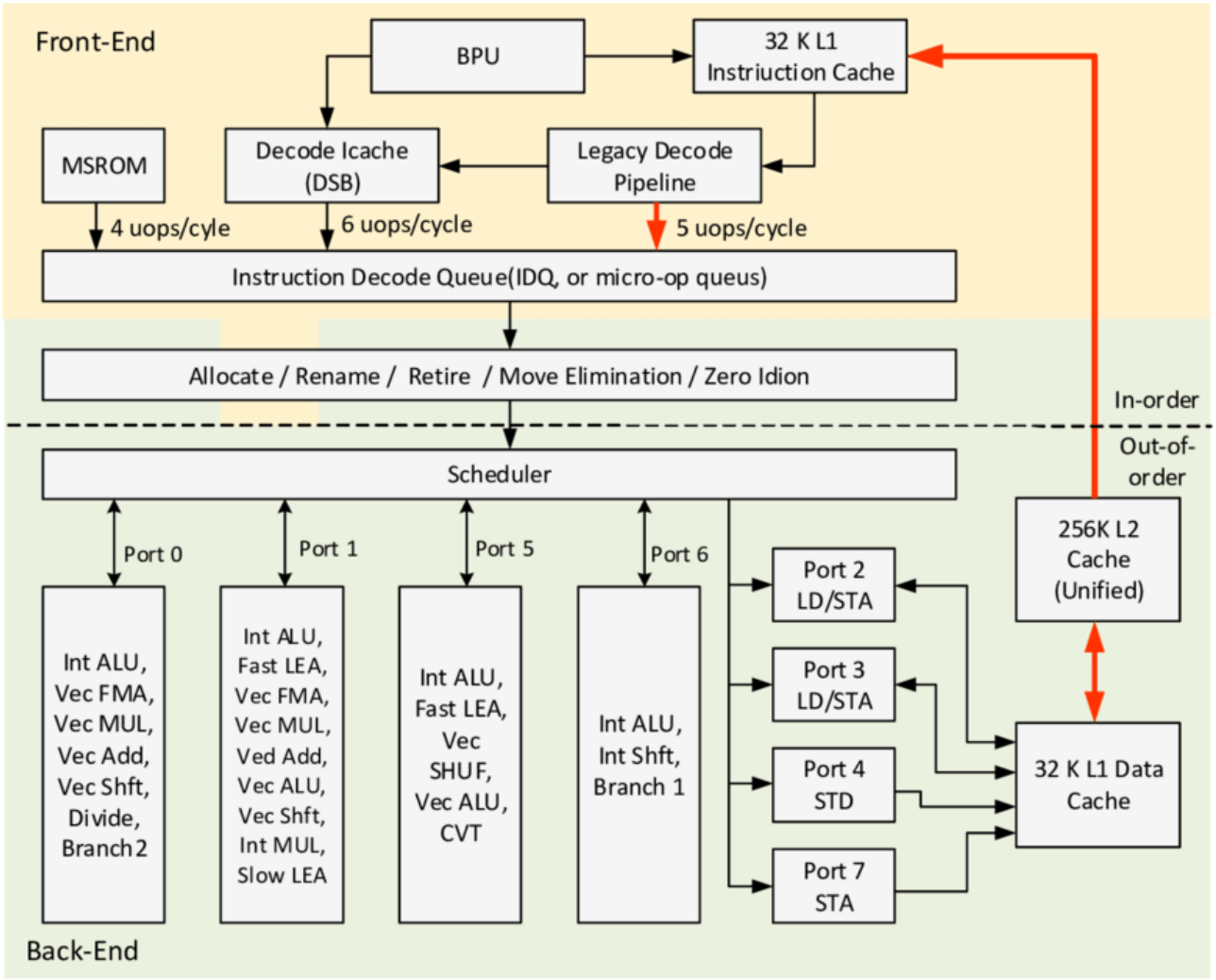

Рис. 1.1 Блок-схема ядра процесора Skylake

Рисунок ілюструє, як сучасні процесори оптимізують обчислення через використання механізмів позачергового виконання, спекулятивного прогнозування та багаторівневої організації пам’яті. Ця схема слугує основою для розуміння структурної організації процесорів та аналізу їх вразливостей у контексті сучасних кіберзагроз.

Архітектура процесора відзначається високою продуктивністю та енергоефективністю. Серед її особливостей – покращений виконуючий конвеєр, підтримка технологій енергозбереження та підвищена пропускна здатність.

1.2. Типологія загроз безпеки програмного забезпечення

Загрози безпеки програмного забезпечення є багатогранною проблемою, яка охоплює як вразливості програмного коду, так і особливості архітектури апаратного забезпечення. Систематизація та аналіз таких загроз дозволяють ефективніше розробляти як превентивні, так і реактивні заходи захисту [11]. У сучасному світі технологій, що характеризуються швидким розвитком, ця проблема ускладнюється зростанням обчислювальної потужності, динамічним розвитком інтернету речей (IoT), хмарних обчислень та штучного інтелекту [12].

Одна з ключових категорій загроз — атаки на конфіденційність даних. Серед найбільш відомих прикладів таких атак — «Data Exfiltration», які реалізуються через вразливості в каналах передачі даних або уразливості в коді програми [8]. Наприклад, неправильно налаштовані протоколи захисту, такі як HTTPS або TLS, дозволяють зловмисникам перехоплювати дані, які передаються мережею. Атаки, що використовують механізми ін’єкції, як-от SQL-ін’єкції чи атаки XSS (Cross-Site Scripting), є типовими прикладами експлуатації слабких місць у валідації введення даних. Ці атаки дозволяють отримати доступ до конфіденційних баз даних або маніпулювати веб-контентом, обходячи механізми автентифікації [13].

Загрози, спрямовані на цілісність програмного забезпечення, стають особливо небезпечними у критично важливих системах, таких як банківські системи, інфраструктури електропостачання чи медичні платформи. Маніпуляції із системними конфігураціями або модулями оновлення програмного забезпечення є типовими прикладами таких атак [14]. Наприклад, атаки типу «Supply Chain Attack» експлуатують ланцюги постачання, змінюючи компоненти програмного забезпечення ще до їх інтеграції у кінцеву систему. Атаки на доступність програмного забезпечення, такі як DDoS (Distributed Denial of Service), націлені на порушення роботи систем шляхом перевантаження її ресурсів. Це може включати виснаження пропускної здатності мережі, використання обмежень у пам’яті чи споживання процесорного часу. У сучасних

хмарних системах такі загрози стають особливо актуальними, оскільки вони здатні створювати суттєві фінансові втрати через перебої у роботі або зниження якості сервісу [15].

Новий вимір у сфері кіберзагроз відкрили сучасні атаки на мікроархітектурному рівні. Атаки типу «Meltdown» та «Spectre» демонструють, як уразливості в механізмах прогнозування розгалужень або позачергового виконання інструкцій можуть бути використані для отримання доступу до конфіденційних даних. У цих атаках зловмисники використовують сторонні канали для обходу бар'єрів безпеки між потоками даних [16]. Ці загрози вимагають розробки нових архітектурних рішень, таких як динамічна ізоляція процесів, рандомізація кешів або впровадження спеціалізованих апаратних модулів безпеки. Окрему категорію становлять загрози, пов'язані зі сторонніми каналами. До таких загроз належать атаки, які експлуатують фізичні властивості обчислювальних систем, зокрема часові затримки, електромагнітні випромінювання або споживання енергії. Наприклад, атаки на основі аналізу енергоспоживання дозволяють відновлювати шифрувальні ключі, а атаки типу «Rowhammer» демонструють, як за допомогою маніпуляцій із динамічною пам'яттю DRAM можна змінювати дані без доступу до адміністративних привілеїв [17]. Окрім безпосередніх безпекових загроз, мікроархітектурні атаки генерують нові виклики для розробників, вимагають впровадження нових захисних механізмів, які б не погіршували продуктивність ПЗ. З метою протидії таким атакам активно досліджуються методи моніторингу активності на мікроархітектурному рівні, що виявляють підозрілу поведінку в режимі реального часі.

Особливістю цих загроз є їхня висока складність виявлення, оскільки зловмисники використовують низькорівневі апаратні процеси, що важко відслідковуються традиційними програмними засобами.

Для вирішення проблеми класифікації загроз та їхнього аналізу використовуються підходи, які базуються на формалізації характеристик загроз. Одним із таких підходів є моделювання загроз за допомогою діаграм UML, що

дозволяє візуалізувати механізми атак та їхні наслідки для системи. Крім того, використання системного аналізу дає змогу інтегрувати механізми захисту на етапі проектування системи [14,16].

У контексті глобальної цифрової трансформації загрози безпеки програмного забезпечення стають не лише технічною проблемою, але й соціальною. Порушення роботи критичних інфраструктур, зокрема енергетичних чи транспортних систем, може призводити до масштабних економічних втрат та створювати загрози для суспільства [18]. Таким чином, дослідження та вдосконалення типології загроз є важливим етапом для підвищення загального рівня безпеки цифрового середовища

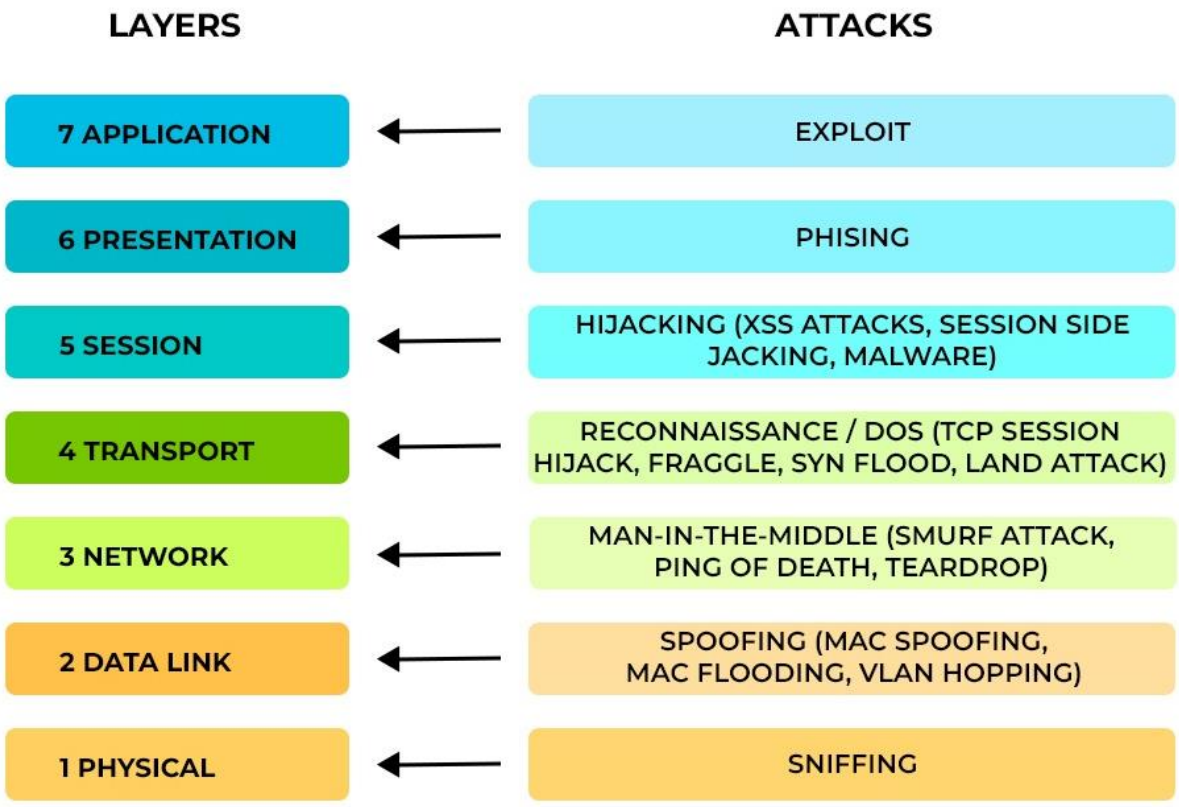

Рис. 1.2. Поширені безпекові атаки в моделі рівнів OSI

Схема на рисунку 1.2 систематизує різні види атак на безпеку, розподіляючи їх відповідно до рівнів моделі OSI, що дозволяє глибше зрозуміти, на яких етапах функціонування мережі можуть виникати певні загрози.

1.3. Аналіз існуючих моделей на мікроархітектурному рівні для захисту потоків даних

Існуючі мікроархітектурні моделі захисту потоків даних ґрунтуються на комплексному поєднанні апаратних і програмних рішень, спрямованих на зменшення ризиків несанкціонованого доступу до інформації, забезпечення конфіденційності та цілісності даних у реальному часі. Такі моделі розробляються з урахуванням сучасних викликів кібербезпеки, зокрема атак сторонніх каналів, спекулятивного виконання та маніпуляцій з пам’яттю [10].

Основною метою цих моделей є створення захищених середовищ для обробки даних, які гарантують ефективне функціонування систем навіть в умовах складних загроз. Апаратний рівень у цих моделях відіграє ключову роль, адже забезпечує основні засоби фізичної ізоляції потоків даних [8]. Наприклад, у багатоядерних процесорах сучасного покоління застосовуються архітектурні підходи, які дозволяють розподіляти ресурси (кеш-пам’ять і ін.), між потоками таким чином, щоб уникнути взаємного перетину. Це досягається за допомогою механізмів «Cache Partitioning», які розділяють кеш на ізольовані сегменти для кожного потоку, зменшуючи ризики атак типу «Flush + Reload» та «Prime + Probe». Крім того, впроваджуються технології динамічного керування кеш-пам’яттю, які адаптуються до змін у поведінці додатків, що мінімізує вразливості, пов’язані зі сторонніми каналами [10]. Спекулятивне виконання, що стало однією з основних причин атак типу «Spectre» та «Meltdown», у нових моделях захищене за допомогою удосконалених алгоритмів контролю прогнозування розгалужень. Наприклад, механізми «Branch Target Buffer Isolation» дозволяють ізолювати дані, що використовуються для прогнозування, від інших потоків у системі. Іншим важливим удосконаленням є запровадження інструкцій серійного бар’єру («Speculative Store Bypass Disable»), які блокують виконання шкідливих операцій, що могли бути викликані помилковими прогнозами [14].

На рівні управління пам’яттю сучасні моделі захисту інтегрують механізми контролю доступу, які забезпечують сегментацію даних і чітке визначення привілейованих зон пам’яті. Технології, такі як Memory Protection Keys (MPK), дозволяють розділяти доступ до пам’яті на основі ролей кожного процесу [19]. Це є особливо важливим для систем, що обробляють конфіденційну інформацію, оскільки зловмисники не зможуть отримати доступ до сегментів пам’яті без відповідних прав.

Ізоляція потоків даних також досягається через використання захищених середовищ виконання, таких як Intel Software Guard Extensions (SGX) або ARM TrustZone. Ці технології створюють ізольовані області пам’яті, які недоступні навіть для операційної системи [20]. Вони використовуються для зберігання та обробки ключів шифрування, біометричних даних або іншої критично важливої інформації. Однією з нових функцій, інтегрованих у ці середовища, є перевірка автентичності інструкцій перед їх виконанням, що значно знижує ризики маніпуляцій з кодом [21].

Моделі захисту потоків даних також активно використовують елементи машинного навчання для аналізу поведінки потоків і виявлення аномалій у реальному часі. Наприклад, алгоритми глибокого навчання здатні виявляти нетипову активність у потоках, що може вказувати на спроби компрометації системи. Такі підходи є перспективними, оскільки дозволяють не лише реагувати на вже відомі загрози, але й передбачати нові типи атак, які ще не отримали широкого розповсюдження [10].

У моделях для високонавантажених обчислювальних систем впроваджуються також механізми балансування навантаження з урахуванням вимог безпеки. Це дозволяє уникати «вузьких місць» у потоці даних, які можуть бути використані зловмисниками для атак на доступність системи. Наприклад, динамічне розподілення навантаження між ядрами процесора забезпечує стабільну роботу навіть у випадках пікових навантажень, одночасно мінімізуючи ризики перенасичення каналів передачі даних.

Дослідження існуючих мікроархітектурних моделей показує, що ефективний захист потоків даних є результатом інтегрованого підходу, який поєднує вдосконалення на рівні апаратного забезпечення, оптимізацію програмного забезпечення та використання методів машинного навчання. У сучасному світі, де загрози постійно еволюціонують, ці моделі є важливим інструментом забезпечення стабільності та безпеки цифрових систем. Їх подальший розвиток сприятиме створенню стійких до загроз платформ, які відповідають найвищим стандартам кібербезпеки [22].

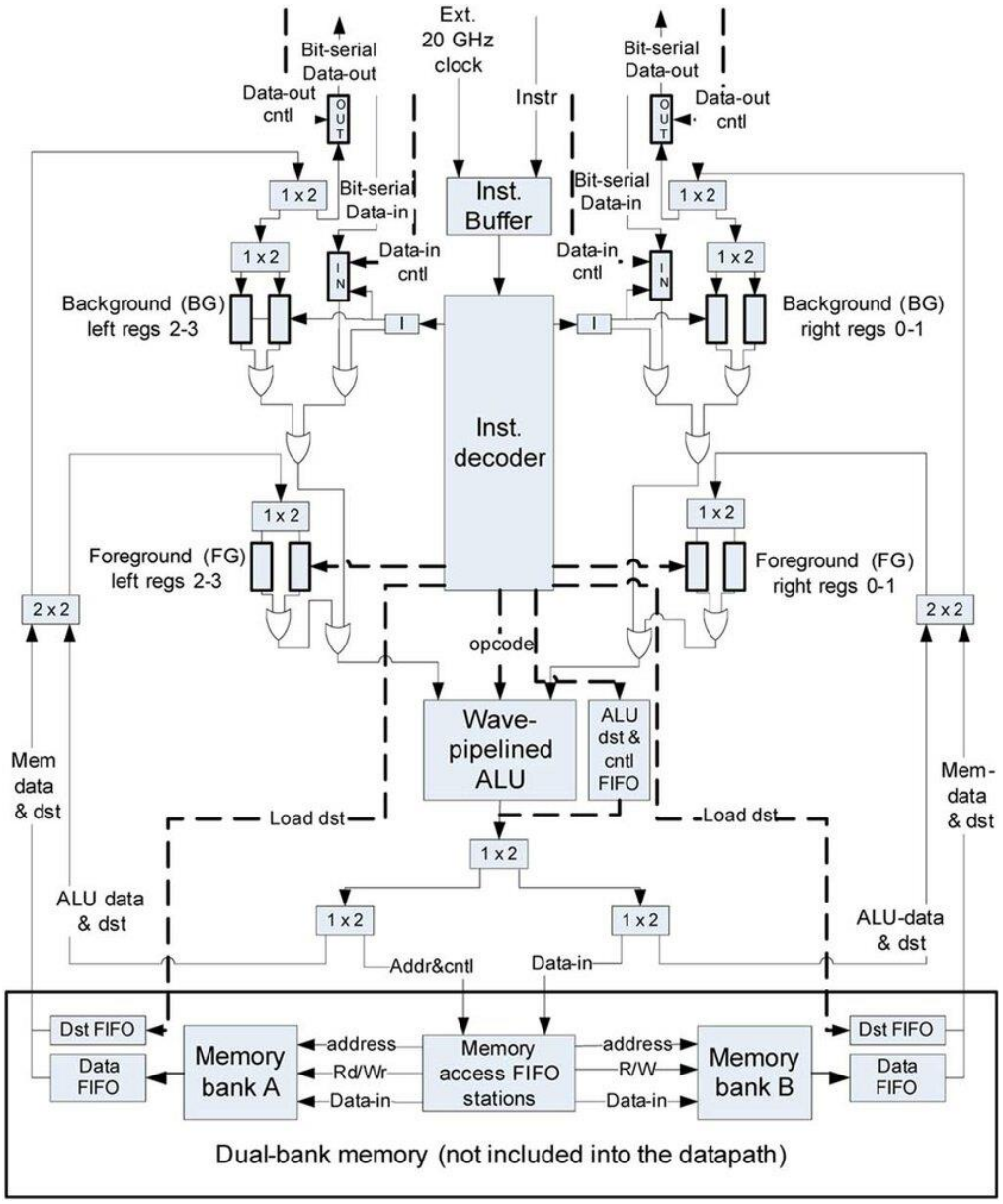

Рис. 1.3. Мікроархітектура обробки даних у процесорі з широкими шляхами передачі

На рисунку 1.3 зображено структуру мікроархітектури, яка забезпечує ефективну обробку потоків даних із використанням широких шляхів передачі. Двоканальний 32-бітний обчислювальний тракт Frontier [23] реалізований із використанням хвильового конвеєру, спільного потоку та інших механізмів синхронізації. Обчислювальний тракт складається з буферу інструкцій; декодеру; багатопортового регістр-файлу із хвильовою конвеєризацією; двох вхідних і двох вихідних 32-бітних портів; арифметико-логічного пристрою із хвильовою конвеєризацією; внутрішньо-процесорної з'єднувальної структури з хвильовою конвеєризацією.

Буфер інструкцій необхідний для точного моделювання схеми синхронізації між буфером видачі інструкцій та етапом декодування в обчислювальному тракті. Робота блоку організована у формі двоступеневого конвеєра із застосуванням механізму спільного тактування для досягнення швидкості декодування 20 ГГц [23].

Багатопортовий регістр-файл може виконувати дві одночасні операції читання з окремих банків регістрів; підтримує два одночасні записи з блоку пам'яті та один запис від арифметико-логічного пристрою (ALU), якщо використовуються окремі регістри. Поля джерел декодуються за допомогою 2-в-4 декодерів, створюючи сигнали читання регістрів у форматі "one-hot".

Хвильовий конвеєризований ALU використовує хвильову конвеєризацію для асинхронного виконання операцій; забезпечує ефективне використання всіх етапів без необхідності проміжних регістрів чи глобального тактування.

Внутрішньо-процесорна маршрутизація даних передбачає передачу результатів ALU до регістрів або портів вводу/виводу через хвильовий конвеєризований тракт. Контрольні сигнали маршрутизації випереджають хвилі даних для попередньої підготовки шляху передачі.

Наведений тракт здатний підтримувати відповідні цільові технологічні процеси при частотах 20 ГГц і 33 ГГц з високим рівнем захисту від безпекових атак на процесор [23].

На рисунку 1.4 представлено активну систему управління пам’яттю [24], яка включає процесорні модулі, інтерфейси кешу інструкцій і даних, а також модулі доступу до пам’яті. Така модель дозволяє зменшити затримки доступу до даних і запобігти стороннім каналам.

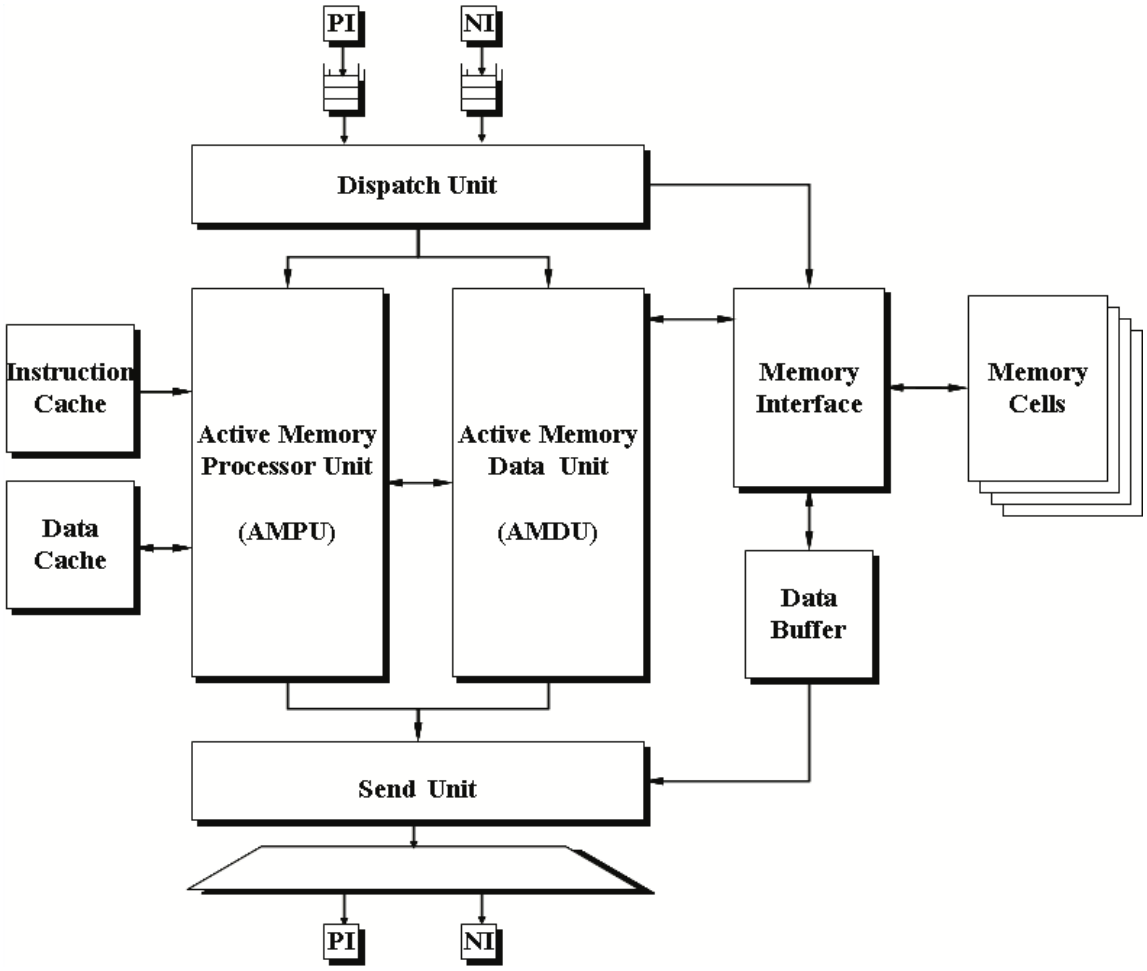

Рис. 1.4. Активна мікроархітектура управління пам’яттю

Двопотоковий активний процесор пам'яті (AMPU) виконує протоколи узгодження та характеризується наступними особливостями:

- не підтримує віртуальну пам'ять, виключення, операції з рухомою комою чи цілісні операції множення і ділення;
- має спеціалізовані інструкції для оптимізації протоколів когеренції кешу та активних операцій пам'яті;
- код і дані завантажуються з внутрішніх кешів інструкцій і даних, які підтримуються основною пам'яттю.

Функціональність блоку процесора активної пам’яті включає:

- виконання відповідного обробника протоколу для кожного запиту (звичайного чи активного);
- оновлює записи в директорії для збереження когеренції кешу;
- передає керівні повідомлення до AMDU для виконання активних операцій із даними.

Блок даних активної пам’яті AMDU є апаратним конвеєром для перевизначення адрес та прискорення складання/розбирання кеш-ліній. Він містить п’ять буферів розміру кеш-лінії – буфер базових адрес, буфер віртуальних адрес, буфер фізичних адрес, буфер адрес директорії, буфер даних; та три конвеєрні етапи – обчислення адреси, пошук у таблиці перекладу адрес, Обчислення адрес директорії та доступ до пам’яті [24].

Функціональність та роль AMDU:

- на кожному етапі обробляється один запис за цикл, а операції повністю конвеєризовані;
- найкраща затримка AMDU дорівнює затримці конвеєра збільшеній на 15 циклів;
- AMDU значно зменшує затримку обробників AMPU, оскільки працює паралельно з етапами доступу до пам’яті.

Функціональність інтерфейсу пам’яті (Memory Interface):

- з’єднує SDRAM із іншими частинами контролера;
- здійснює вибір запиту із черги довжиною до 16 записів і виконує операції завантаження/запису;
- повністю конвеєризований, що запобігає затримкам у AMDU, якщо черга запитів не переповнена.

Приведена на рисунку 1.4 архітектура системи управління пам’яттю забезпечує ефективну обробку активних операцій, дозволяє виконувати складні маніпуляції з даними та підтримувати когеренцію кешу з мінімальними затримками. Це сприяє підвищенню рівня безпеки процесора, шляхом унеможливлення залучення несанкціонованих каналів.

На рисунку 1.5 зображено процесор із підтримкою адаптивного кешу, який забезпечує зміну конфігурації залежно від потреб робочого навантаження. Включено механізми керування, такі як таблиці патернів і блоки конфігурації, що дозволяють динамічно змінювати параметри кешу для підвищення продуктивності та безпеки.

Кеш може змінювати налаштування динамічно (онлайн) після кожного інтервалу інструкцій або статично (офлайн) перед запуском іншої програми [25].

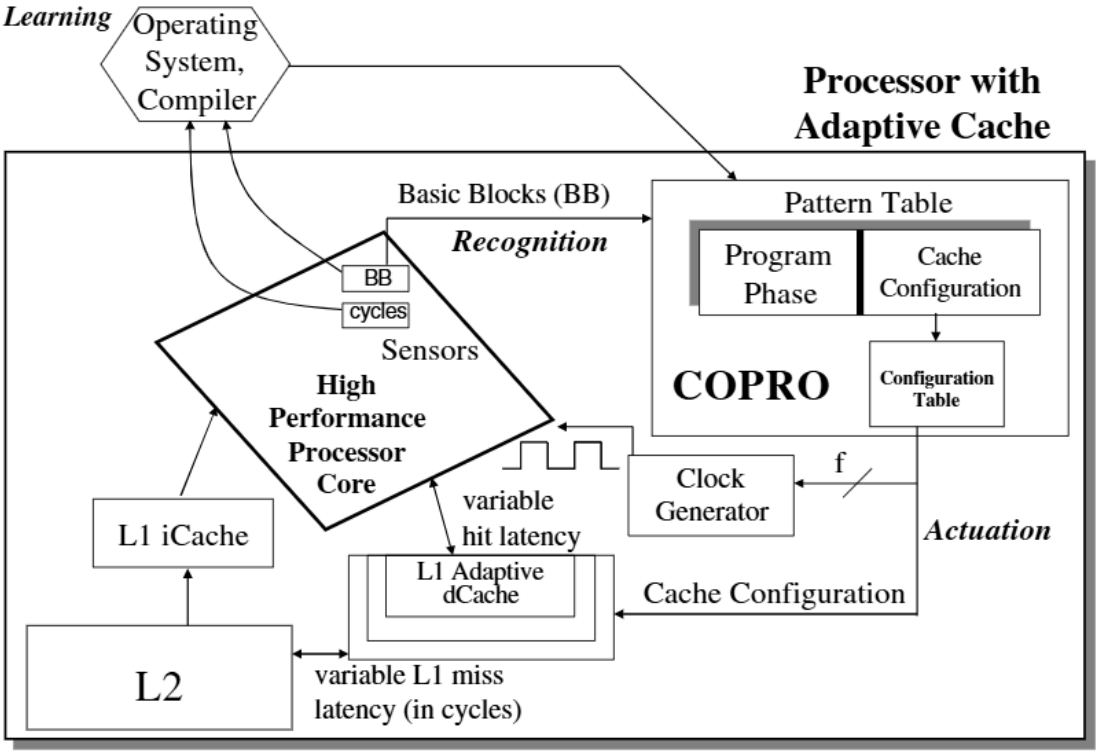

Рис. 1.5. Процесор із адаптивним кешем

Для функціонування процесора необхідні кілька внутрішніх сенсорів; інтегрований апаратний ко-процесор, що працює разом із процесором інструкцій і виконує завдання онлайн-контролю; адаптивна кеш-пам'ять представлена як набір модульних блоків на основі SRAM-мікросхем; механізм конфігурації кешу, яка визначається вбудованою пам'яттю конфігурації.

Така мікроархітектура дозволяє ефективно адаптувати кеш під різні умови роботи, забезпечуючи оптимальну енергоефективність, продуктивність та захист процесора від зловмисних дій щодо пам’яті чи атак типу Side-channel [25].

На рисунку 1.6 продемонстрована поширена структура [26] виконання інструкцій у сучасних процесорах, що використовує та вдосконалює ряд описаних раніше концепцій.

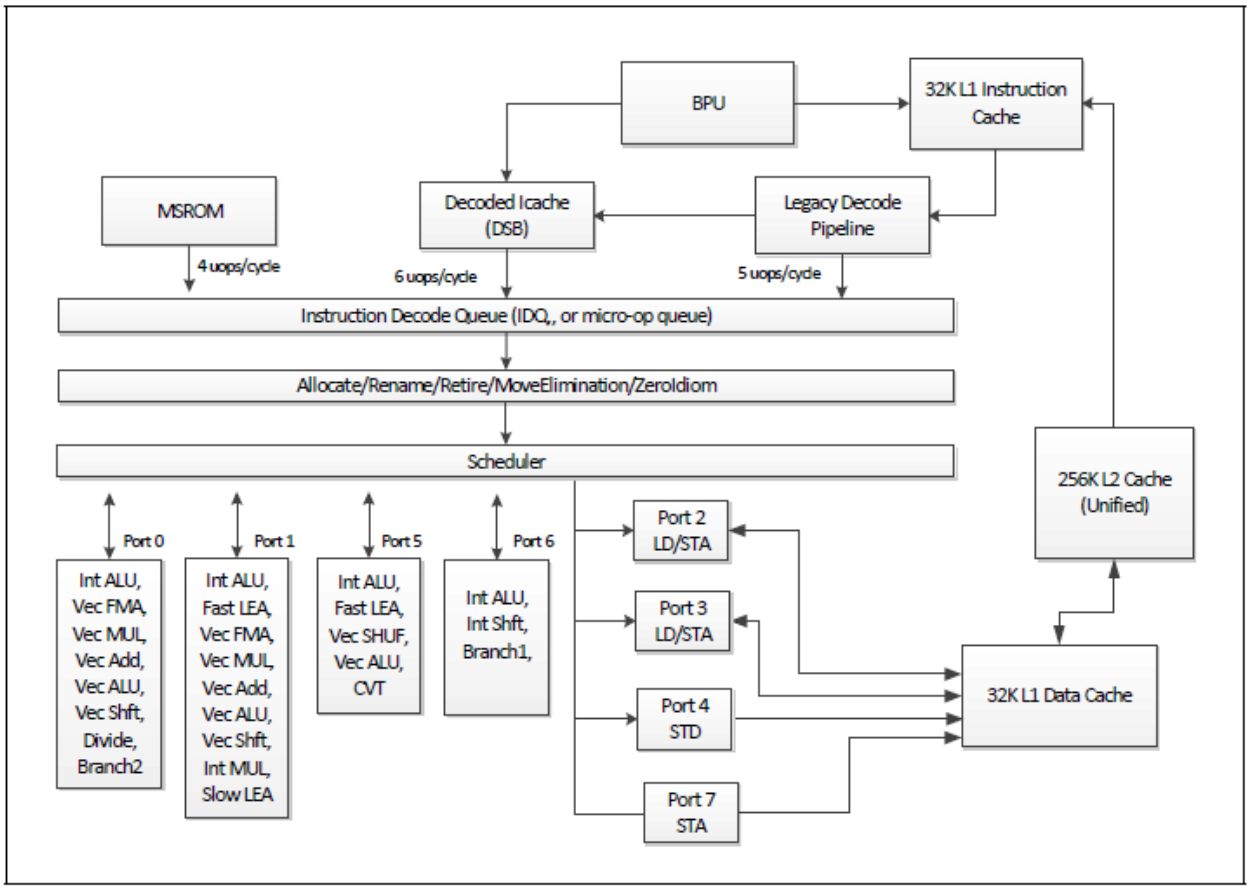

Рис. 1.6. Типова мікроархітектура виконання інструкцій

Мікроархітектура процесора характеризується наявністю блоку планування інструкцій, кешу першого та другого рівнів, модулів обчислень ALU. У порівнянні з іншими рішеннями така схема володіє більшим внутрішнім буфером та кращою пропускною здатність кешу; вдосконаленим предиктором переходів; покращеною пропускною здатністю та зниженою затримкою операцій ділення; зниженим енергоспоживанням; вищою одночасною багатопоточністю.

Модель підкреслює важливість ізоляції потоків для забезпечення стійкості до атак. Розділення обчислювальних ресурсів мінімізує можливість взаємодії між різними потоками, яка може використовуватись для атак.

1.4. Проблеми та обмеження сучасних підходів

Сучасні мікроархітектурні моделі захисту потоків даних стикаються із значними викликами, які впливають на їх ефективність і здатність протистояти новітнім загрозам. Основною проблемою є складність забезпечення одночасно високої продуктивності, сумісності з існуючими стандартами та стійкості до атак [15]. Розробка моделей, які враховують усі ці аспекти, потребує глибокого аналізу архітектурних особливостей, впровадження інноваційних технологій і адаптації до постійно змінюваного ландшафту кіберзагроз.

Однією з ключових проблем є атаки сторонніх каналів, які використовують часові, енергетичні або електромагнітні характеристики роботи процесора для отримання конфіденційної інформації. Наприклад, атаки типу «Flush + Reload» та «Prime + Probe» демонструють, як часові характеристики доступу до кеш-пам’яті можуть бути використані для відновлення даних. Для протидії цим атакам впроваджуються механізми рандомізації кеш-пам’яті, які змінюють розташування даних у пам’яті, роблячи доступ непередбачуваним для зловмисників [27]. Проте такі механізми можуть знижувати продуктивність, що особливо критично для високонавантажених систем.

Ще однією проблемою є загрози, пов’язані зі спекулятивним виконанням інструкцій, такі як «Spectre» та «Meltdown». Ці атаки використовують уразливості в механізмах прогнозування розгалужень і позачергового виконання для отримання доступу до даних, які мали б бути недоступними. Для боротьби з цими загрозами впроваджуються технології ізоляції буферів прогнозування розгалужень, які блокують можливість використання тимчасових проміжних результатів спекулятивного виконання. Проте такі підходи значно ускладнюють архітектуру процесора і вимагають значних ресурсів для забезпечення їхньої коректної роботи.

Важливим аспектом є управління кеш-пам’яттю [28]. Сучасні моделі використовують багаторівневу ієрархію кешів, яка забезпечує швидкий доступ до даних. Проте через спільне використання кешу різними потоками даних

виникає ризик, що дані з одного потоку можуть бути доступні іншому. Для вирішення цієї проблеми впроваджуються механізми розділення кешу (Cache Partitioning), які забезпечують логічну або фізичну ізоляцію даних різних потоків. Крім того, використовуються технології керованої когерентності кешів, які мінімізують можливість витоку даних через неконтрольовану синхронізацію між ядрами процесора [8,12].

Іншим важливим викликом є забезпечення захисту потоків даних у багатопоточному середовищі. У сучасних системах багатопоточність є стандартом, що підвищує продуктивність, але також створює додаткові ризики для безпеки. Наприклад, зловмисник може використовувати один потік для спостереження за поведінкою іншого через спільно використовувані ресурси, такі як регістри або кеші [29]. Для протидії таким загрозам розробляються моделі ізоляції потоків, які забезпечують контроль за доступом до ресурсів на апаратному рівні.

Окрім апаратних обмежень, важливим фактором є програмна сумісність. Багато мікроархітектурних моделей не забезпечують прозорості для програмного забезпечення, що ускладнює їх інтеграцію у реальні системи [6]. Наприклад, механізми ізоляції потоків можуть вимагати внесення змін до операційної системи або програмного забезпечення. Це створює додаткові складнощі для розробників та ускладнює впровадження захисних механізмів у масштабних системах.

Використання технологій машинного навчання для моніторингу та аналізу потоків також має даних характеризується певними лімітами застосування [28]. Незважаючи на високу ефективність виявлення аномалій, такі підходи потребують значних обчислювальних ресурсів і великих обсягів навчальних даних. Крім того, алгоритми машинного навчання можуть бути вразливими до атак на моделі, які змінюють їхню поведінку, створюючи хибні результати. Проблеми та обмеження сучасних підходів також стосуються їхньої масштабованості.

У міру зростання обчислювальних потреб і інтеграції нових компонентів, таких як криптографічні модулі або ізольовані середовища, існує ризик перевантаження системи [30]. Це вимагає від розробників детального аналізу архітектурних змін і пошуку компромісів між продуктивністю, безпекою та вартістю реалізації. Інтеграція апаратних і програмних рішень, використання машинного навчання, оптимізація управління ресурсами та розробка адаптивних систем є основними напрямками розвитку для забезпечення безпеки потоків даних у майбутньому. Це дозволить створити стійкі до загроз обчислювальні системи, які відповідають вимогам сучасного цифрового середовища [31-32].

1.5 Висновки до розділу

В даному розділі розкриті такі основні теоретичні базиси основ методів забезпечення безпеки на мікроархітектурному рівні, як основи компонування комп’ютерних систем, типологія сучасних безпекових загроз програмного забезпечення, існуючі мікроархітектурні моделі захисту потоків даних, аналіз переваг та недоліків наявних підходів.

Категоризація безпекових загроз сприяє ефективнішому розробленні превентивних моделей захисту програмного забезпечення. Одними з основних викликів залишаються атаки, спрямовані на конфіденційність даних.

Поточні мікроархітектурні стратегії захисту ґрунтуються на поєднанні апаратних і програмних рішень. Їх основна мета полягає у створенні захищених середовищ обробки даних, що підтримують функціонування систем у в умовах загроз. Однією з ключових прогалин існуючих підходів є незахищені сторонні канали, атаки на які призводять до витоку даних.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ СУЧАСНИХ МЕТОДІВ КЕРУВАННЯ ПОТОКАМИ ІНСТРУКЦІЙ НА РІВНІ МІКРОАРХІТЕКТУРИ

2.1 Характеристика технології передбачення розгалужень

У сфері сучасних обчислень ефективне виконання програмних інструкцій має першорядне значення та забезпечується технологією передбачення розгалужень – Branch prediction. Це критичний компонент ЦП, який передбачає результат умовних розвилок.

У сучасних процесорах прогнозування розгалужень є важливою технікою, яка спрямована на передбачення результатів гілок програми ще до завершення їхнього фактичного виконання. Такий підхід дозволяє уникнути затримок, пов’язаних із очікуванням розв’язання умов переходу, і забезпечує попереднє завантаження наступних інструкцій. Точність роботи предиктора переходів стає ключовим фактором, що значно впливає на загальну продуктивність процесорів.

Процесор працює зі послідовністю інструкцій, представлених у машинному коді, і звичайно виконує ці інструкції по черзі.

Етапами виконання інструкцій є: отримання інструкції, завантаження наступної інструкції з пам'яті на основі лічильника програм; збільшення значення лічильника; декодування інструкції, що передбачає визначення операції, що має бути виконана, виявлення операндів та їх адрес; прогнозування переходів на основі часової інформації або шаблонів поведінки; етап виконання; запис результату; кількість етапів може змінюватись залежно від архітектури процесора.

Ця послідовна, але рознесена обробка інструкцій є основою для більш ефективної роботи сучасних процесорів, хоча вона також створює потребу в механізмах, що дозволяють мінімізувати затримки.

Прогнозування переходів допомагає зменшити ці затримки, передбачаючи результат умовних переходів завчасно, що дозволяє процесору

завантажувати інструкції до того, як умова переходу буде повністю визначена.

Ключовими апаратними компонентами [33] для динамічного прогнозування переходів є:

- Лічильник програм (PC). Лічильник програм — це регістр, що зберігає адресу поточної інструкції, яка виконується в процесорі. PC є критично важливим для індексації структур прогнозування переходів, оскільки він допомагає визначати та відрізняти різні точки в програмі. Адреси, які зберігає PC, використовуються для здійснення точних прогнозів на основі історичної інформації про поведінку переходів у конкретних місцях програми.
- Реєстр історії переходів (BHR). Реєстр історії переходів реалізується у вигляді зміщувального реєстру, який оновлюється з кожним прийнятим рішенням про перехід. BHR зберігає історичні результати останніх переходів у процесорі, що дозволяє аналізувати послідовну поведінку переходів. Реєстр історії переходів використовується для індексації структур прогнозування переходів, оскільки він надає інформацію про шаблони поведінки переходів у минулому. Завдяки BHR здійснюється передбачення майбутніх результатів переходів на основі повторюваної поведінки, що спостерігалась раніше.

Ці два компоненти – PC та BHR – є основою динамічного прогнозування переходів, оскільки вони надають інформацію про поточний стан інструкцій і їх історію поведінки, що дозволяє здійснювати точні прогнози для мінімізації затримок у виконанні програм.

Предиктор TAGE (Tagged Geometric History Length), що наведений на рисунку 2.1 є одним із найефективніших блоків для передбачення переходів, які використовуються в сучасних процесорах, і він демонструє високу продуктивність з моменту свого впровадження.

Принцип роботи TAGE полягає у функціонуванні на основі збереження історичних даних про результати переходів у спеціальному часовому наборі даних. Основні особливості його дій полягають у поєднанні стандартного предиктора та спеціалізованих блоків. Це дозволяє ефективно адаптуватися до різноманітних шаблонів програмної поведінки [34].

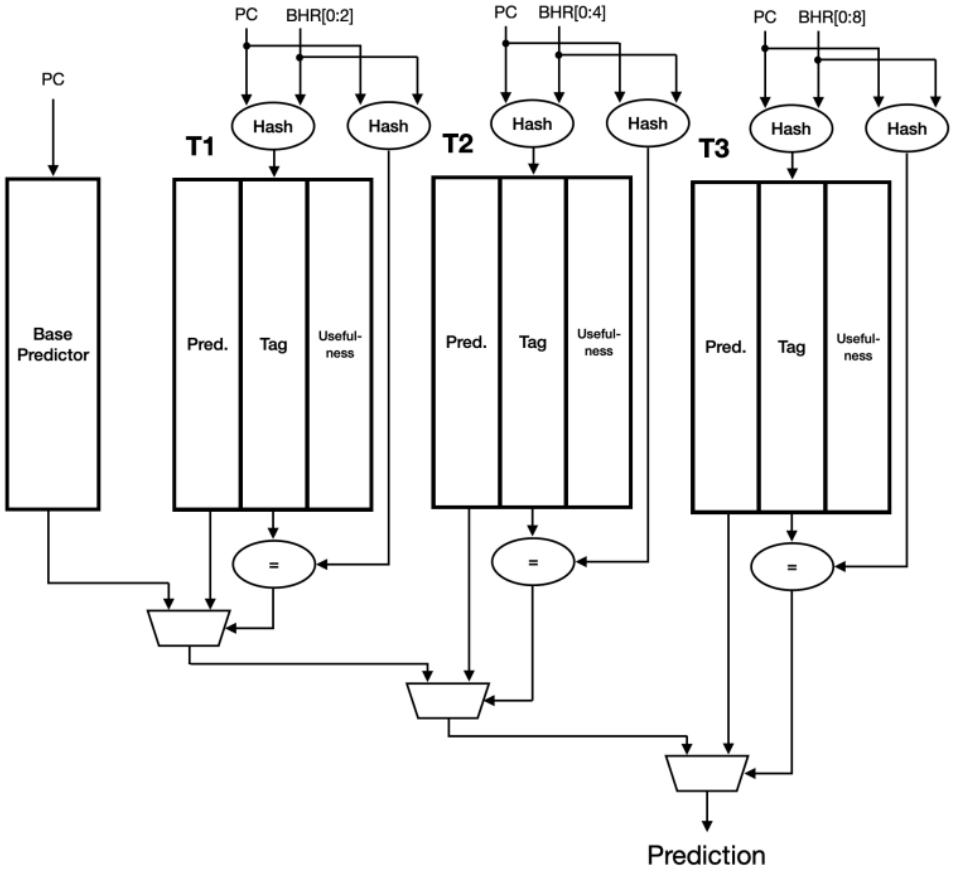

Рис. 2.1. Структурна модель предиктора TAGE

Етапи процесу прогнозування TAGE при надходженні нової інструкцій в процесор:

- аналіз історичної інформації, що зберігається у наборі даних;
- підбір найкращої відповідності серед наявних даних;
- прогнозування результату поточного переходу на основі знайденого найкращого збігу.

Перевагами TAGE є:

- адаптивність – завдяки використанню геометричної послідовності різних довжин історій, TAGE ефективно адаптується до різних програмних шаблонів і стратегій поведінки;
- точність – поєднання стандартного та спеціалізованих предикторів дозволяє блоку прогнозування TAGE забезпечувати високу точність прогнозування;
- ефективність – використання історичних даних забезпечує

зменшення кількості помилок у прогнозах, що позитивно впливає на швидкість роботи процесора.

TAGE використовує багатошаровий підхід, де стандартний предиктор надає базову інформацію, а кастомізовані предиктори дозволяють враховувати більш складні патерни та довші часові інтервали. Ця комбінація робить його одним із найефективніших методів динамічного прогнозування переходів [34].

На рисунку 2.2 наведена одна з повноцінних реалізацій архітектурної одиниці на основі технології Branch Prediction.

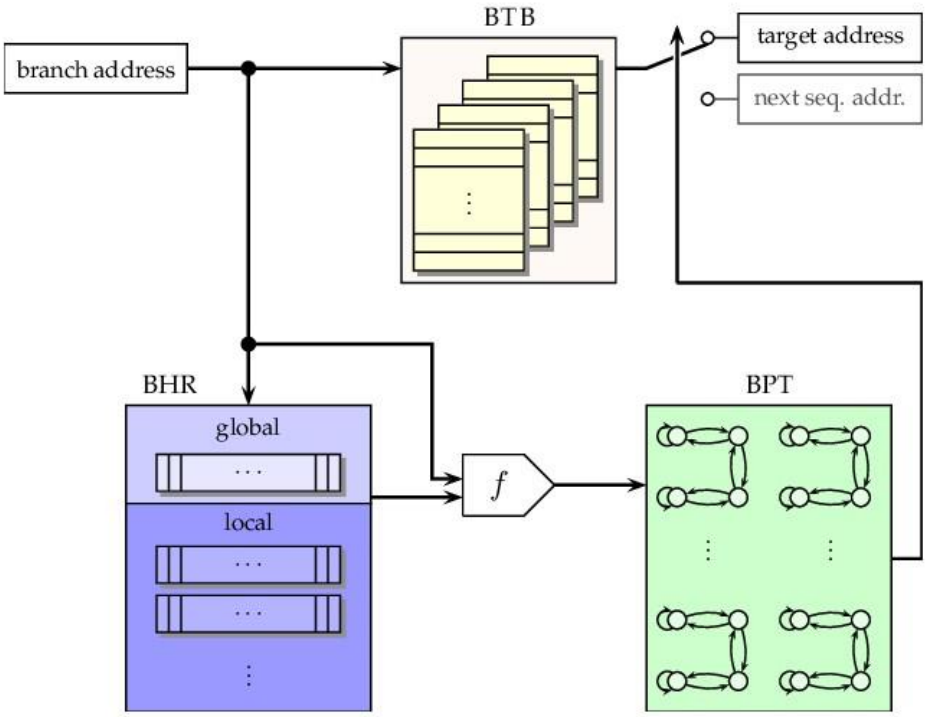

Рис. 2.2. Реалізація технології Branch Prediction

Реєстр історії переходів та предиктор працюють у тандемі, щоб забезпечити оптимальну ефективність процесора. Наявність набору історичної інформації зменшує затримки, зберігаючи цільові адреси переходів, а предиктор забезпечує точність у прогнозуванні, зменшуючи час очікування.

Ця комбінація мінімізує кількість помилок під час виконання умовних переходів, що підвищує загальну продуктивність й рівень безпеки процесора.

2.2 Сутність концепції цілісності потоку керування

Комп’ютери часто стають об’єктами зовнішніх атак, які спрямовані на зміну поведінки програмного забезпечення. Як правило, такі атаки надходять у вигляді даних через стандартні канали зв’язку і, потрапляючи в пам’ять програми, використовують попередньо існуючі вразливості. Вони можуть змінювати виконання програми, захоплюючи контроль над її поведінкою. Сукупний ефект таких атак робить їх однією з найважливіших проблем у сфері комп’ютерної безпеки.

Для комплексної протидії таким типам загроз була сформована технологія, яка спрямована на забезпечення інтеграції потоку управління (Control Flow Integrity, CFI). Політика безпеки CFI передбачає, що виконання програмного забезпечення має відповідати шляху в заздалегідь визначеному графі управління потоком (CFG). Граф може бути створений за допомогою аналізу коду, бінарного аналізу або профілювання виконання.

Метод CFI забезпечує захист навіть за умови повного контролю зловмисників над даними пам’яті. Реалізація CFI використовує поєднання статичної верифікації та переписування машинного коду з додаванням перевірок у час виконання. Ці перевірки динамічно забезпечують цілісність управління потоком та гарантують, що воно залишається в межах заданого графу CFG.

Впровадження CFI ефективно захищає від широкого спектра поширених атак, оскільки ненормальна модифікація потоку управління є ключовим етапом у багатьох експлойтах — незалежно від того чи використовуються переповнення буфера або інші вразливості. Це передбачає як класичні атаки з переповненням стеку, так і новіші атаки "jump-to-libc" на основі купи. Також забезпечується захист від порівняно нових атак типу "підміна вказівника", які обходять багато попередніх методів захисту. Впровадження CFI має свої слабкі місця – до експлойтів в межах дозволеного графу CFG не застосовуються механізми протидії. До них належать певні атаки, які використовують некоректний розбір аргументів рядка для запуску небезпечного виконуваного файлу [35].

Для динамічних перевірок існує кілька можливих стратегій. За одною з них CFI забезпечується перевірками, які порівнюють цільову адресу кожного обчислюваного переходу з набором дозволених адрес. Такий підхід може реалізовуватись через еквівалент машинного коду до оператора switch над множиною константних адрес. Однак якщо набір дозволених адрес великий, це призводить до неприйнятних затримок.

Натомість використовуються ефективніші альтернативи. Застосовуваною є реалізація спеціальних інструкцій машинного коду із заданими ідентифікатором (ID), які перевіряють відповідність динамічної адреси потоку з відповідною еквівалентною групою. У разі порушення виконання блокується. Такий підхід знаходить застосування на існуючих процесорах, зокрема на архітектурі x86, із помірними витратами ресурсів.

Інструменталізація CFI змінює кожну інструкцію джерела та всі можливі інструкції-призначення для переходів відповідно до CFG. У кожному місці призначення інструменталізація додає шаблон біту, який позначає еквівалентний клас призначень. Перед кожним джерелом також додається динамічна перевірка (ID-перевірка), яка гарантує, що під час виконання вибрано правильне призначення.

Однією з важливих складових CFI є система ізоляції помилок програмного забезпечення (SFI) — один із типів механізму Inlined Reference Monitors (IRM), що призначений для емуляції традиційного захисту пам'яті. У SFI код вставляється в кожну інструкцію машинного коду, яка отримує доступ до пам'яті, щоб переконатися, що цільова адреса пам'яті знаходиться в певному діапазоні. CFI робить інструменталізацію SFI нездоланною, такою, якої неможливо уникнути. Крім того, CFI дозволяє зменшити накладні витрати SFI. Наприклад, гарантії щодо потоку керування усувають необхідність повторної перевірки адрес пам'яті в локальних змінних [35].

Хоча політики CFI, які враховують контекст, значно зменшують дозволені цільові адреси для кожного інструктивного переходу (ICT), більшість із них можуть забезпечити унікальну коректну ціль для кожного ICT.

На відміну від них, так звана, μCFI – одна з найсучасніших різновидів політики CFI, забезпечує властивість UCT (Unique Code Target), що гарантує, що кожна ICT-інструкція має лише одну унікальну ціль під час кожного кроку виконання програми.

μCFI досягає цієї властивості, визначаючи обмежуючі дані зі значень програмного коду та використовуючи ці дані як контекст під час застосування CFI. На рисунку 2.3 представлений огляд застосування μCFI [36].

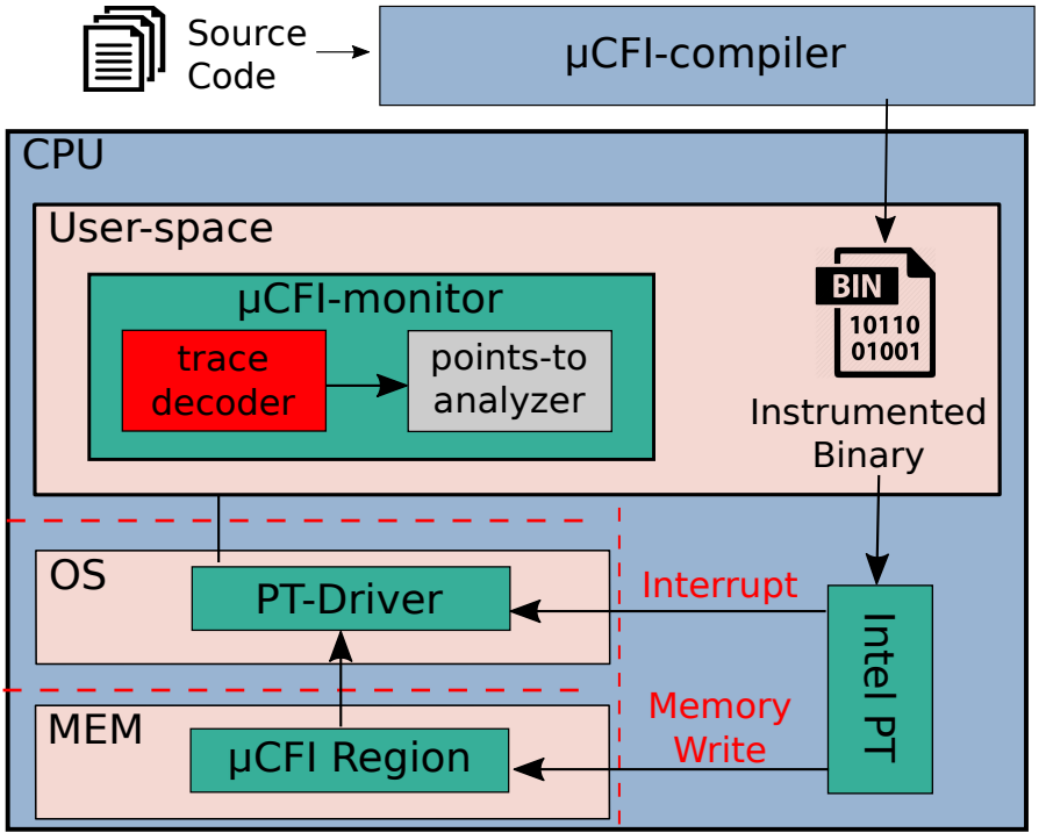

Рис. 2.3. Політика μCFI

μCFI складається з компілятора і динамічного монітора. Основними аспектами цієї методології є:

- μCFI-compiler: інструментує вихідний код програми, щоб ідентифікувати обмежуючі дані, і створює інструментований бінарний файл;
- (Performance Tracing): під час виконання інструментованого бінарного файлу здійснює запис закодованих трасувань до буфера трасування;

- керуючий драйвер (PT-Driver): копіює дані з буфера трасування до буфера ядра при його переповненні;

- µCFI-monitor: отримує закодовані трасування з буфера ядра через механізм сигналізації до PT-Driver, декодує ці трасування за допомогою модуля декодування, перевіряє цільові адреси ICT через модуль аналізу;

Ця структура дозволяє µCFI ефективно застосовувати механізм перевірки під час виконання програми, забезпечуючи коректність та безпеку виконання.

µCFI складається з двох компонентів:

- компілятор (µCFI-compiler), який інструментує програму для ідентифікації обмежених даних;

- монітор часу виконання (µCFI-monitor), що перевіряє ICT.

Для забезпечення захисту переходів у контрольному потоці, µCFI-monitor потребує збору даних, цільових адрес індексованих викликів та адрес, які використовуються для виконання чутливих повернень у робочій програмі.

На рисунку 2.4 наведена реалізація підходів µCFI на базі типового апаратного механізму PHMon.

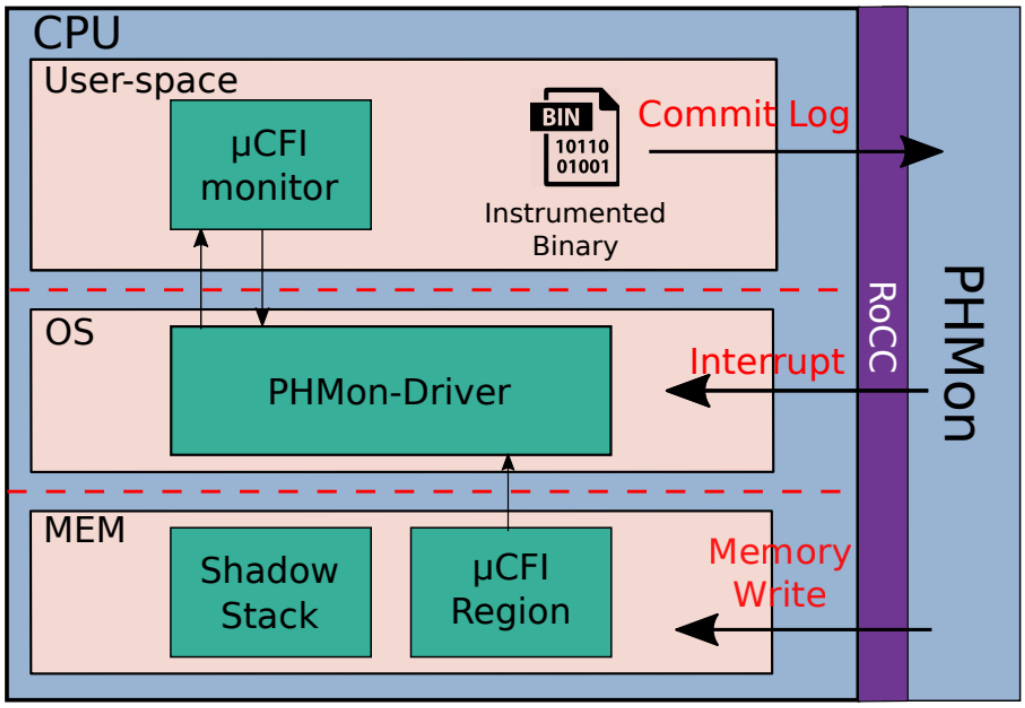

Рис. 2.4. Забезпечення політики µCFI з використанням монітора PHMon

Основні процеси виглядають наступним чином:

- компіляція коду та інструментування: програму спочатку компілюють за допомогою модифікованого μCFI-компілятора, який додає необхідний код для збору інформації; створюється інструментований двійковий код;
- забір даних монітором PHMon: перед виконанням інструментованого двійкового коду, PHMon налаштовується на збір необхідних даних; під час роботи програми, PHMon збирає дані через механізм RoCC (Rocket Chip Coherence Controller), а потім застосовує визначені правила моніторингу для аналізу цих даних;
- перезапис даних до буфера: зібрані дані передаються до буферу трасування, що позначений як μCFI Region; при наповненні буфера, PHMon ініціює переривання;
- керування через ядро: модуль ядра PHMon-Driver копіює дані до буфера ядра та інформує операційну систему, щоб вона могла продовжити виконання програми; PHMon-Driver також передає дані до μCFI-monitor, де відбувається перевірка інструкцій;
- захист зворотніх ребер за допомогою PHMon: захист здійснюється за допомогою механізму тіньового стеку; на відміну від підходу, що використовує лише програмний стек (рис. 2.3), в цій схемі присутній тіньовий стек на базі PHMon.

Мікроархітекура μCFI з використанням PHMon одночасно поєднує апаратну підтримку моніторингу із механізмами Control Flow Integrity для забезпечення високоефективного, гнучкого і безпечного захисту виконання програми. Цей підхід не лише мінімізує витрати продуктивності, а й дозволяє легко масштабувати CFI для великих програмних середовищ [36].

Використання тіньового стеку та контекстно-залежного моніторингу дозволяє ефективно захищати як прямі так і зворотні ребра робочого коду. Перевагою використання такої схеми є гнучкість, адже PHMon може проводити швидкий збір даних як для потреб CFI, так і для перевірок безпеки.

2.3 Принципи архітектури обчислень зі скороченим набором команд

Відокремлення апаратної архітектури від її реалізації дозволило створювати кілька варіантів фізичного втілення однієї й тієї ж архітектурної схеми. Це дозволило систематично переносити програми з однієї моделі на іншу, очікуючи, що вони даватимуть той самий результат. Така постановка дій визначає поняття архітектурної сумісності.

Проектування та контроль системної архітектури – постійний процес, основна мета якого полягає у видаленні невизначеностей при конструюванні архітектури та, у деяких випадках, коригуванні забезпечуваного функціоналу.

Особливе місце в комп'ютерній архітектурі займають обчислення зі скороченим набором команд (Reduced Instruction Set Computing або RISC). Підходи RISC були розроблені в результаті проекту, що розпочався в 1975 році та був завершений на початку 1980-х років.

За результатом аналізу стрічок програмного сліду, що містили мільйони інструкцій, які виконувались на комп'ютері під час роботи з репрезентативними програмами, виявили що протягом 90% часу в реальних програмах використовувалося лише близько 10 інструкцій з усього доступного набору інструкцій. Це дало привід для виникнення ідеї оптимізації лише обраних інструкцій, які повинні виконуватися в ході короткого циклу [37].

Архітектура RISC будується на основі малого набору найчастіше використовуваних інструкцій, що визначає структуру конвеєра процесора для забезпечення швидкого виконання інструкцій за один цикл.

Особливості RISC:

- один цикл на інструкцію досягається завдяки паралелізму, що реалізується через конвеєризацію;
- паралелізм є ключовою характеристикою архітектури, та надає основу для формування всіх інших її патернів;
- архітектура RISC орієнтована на підвищення продуктивності завдяки використанню такого паралелізму.

Кожна інструкція в RISC-архітектурах є простою та прямолінійною, що дозволяє скоротити час, необхідний для її виконання, а також зменшити кількість тактів, потрібних для обробки інструкції.

Час виконання інструкції, зазвичай, ділиться на п'ять етапів (машинних циклів), і як тільки обробка одного етапу завершена, машина переходить до виконання другого етапу. Однак, коли певний етап стає вільним, він використовується для виконання операції наступної інструкції. Ця методика дозволяє виконання інструкцій у конвеєрному режимі.

П'ять етапів конвеєра (pipeline) RISC:

- IF (Instruction Fetch) – завантаження інструкції з пам'яті;
- ID (Instruction Decode) – декодування інструкції: інструкція розшифровується, щоб зрозуміти, які дії з нею потрібно виконати;
- EX (Execute) – виконання основної операції інструкції;
- MA (Memory Access) – доступ до пам'яті: операції, які вимагають доступу до пам'яті, виконуються на цьому етапі;
- WB (Write Back) – зворотній запис результату: результати операції записуються у регістри.

Оскільки головною характеристикою RISC є архітектурна підтримка паралелізму на рівні інструкцій, всі визначальні характеристики RISC слід розглядати в контексті їх підтримки функціонування конвеєра. Архітектура RISC використовує принцип локальності: просторової та часової [37].

Просторова локальність означає, що дані, які були нещодавно використані, мають більшу ймовірність повторного використання. Це обґрунтовує наявність відносно великої універсальної множини регістрів у RISC-машинах, що відрізняє їх від CISC (обчислення зі складним набором команд).

Тимчасова локальність означає, що дані, які були використані в недавньому проміжку часу, найімовірніше будуть використані в найближчому майбутньому.

Ці принципи локальності є основою для використання кеш-пам'яті в моделі RISC.

Часто архітектуру RISC називають архітектурою завантаження й збереження (load/store). Альтернативною назвою основного типу операцій в її наборі інструкцій є операції між регістрами (Register-to-Register), детальна логіка яких наведена на рисунку 2.5.

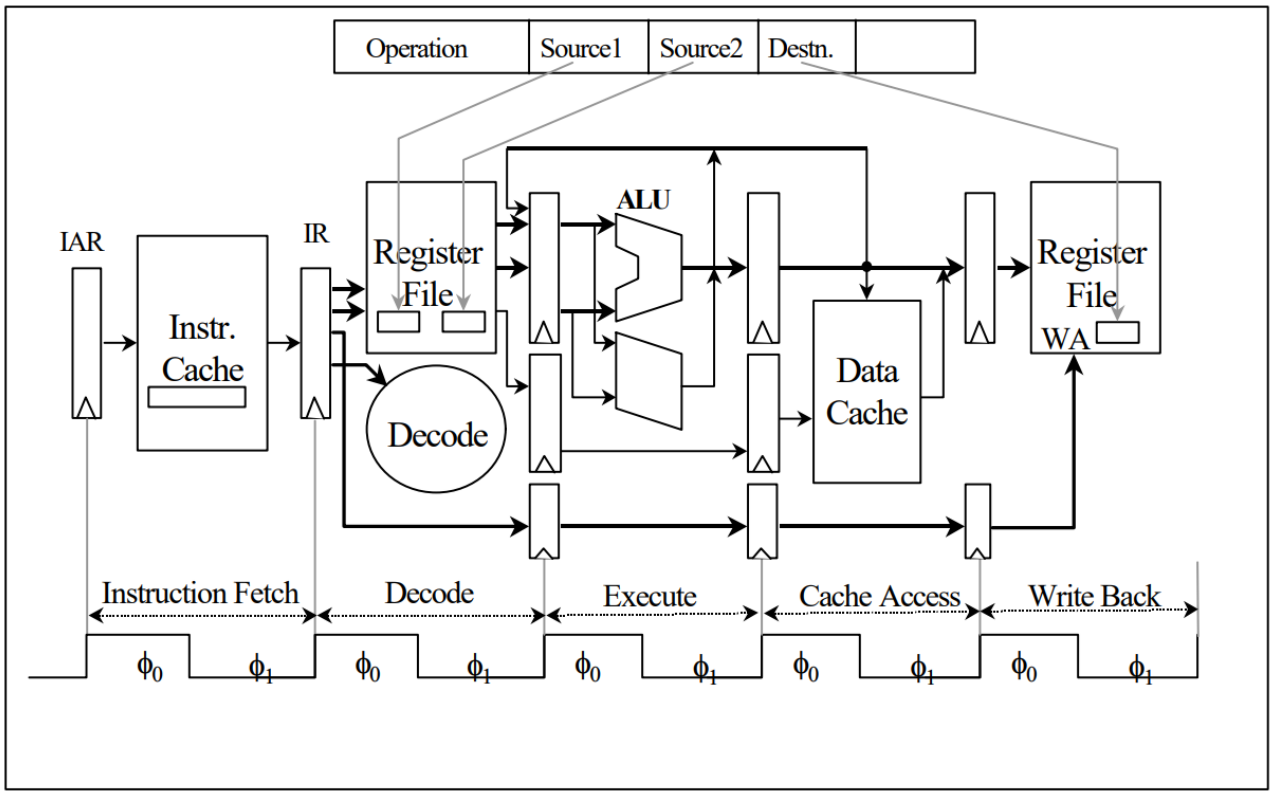

Рис. 2.5. Схема конвеєрного потоку операцій між регістрами

Всі операції в апаратному забезпеченні на базі RISC здійснюються між операндами, які знаходяться в універсальному регістрі (General Purpose Register File, GPR). Результат операції також зворотнім чином записується в GPR. Обмеження місць зберігання операндів лише в GPR дозволяє забезпечити детермінізм в операціях RISC. Таким чином, потенційно багатопроцесорний та непередбачуваний доступ до пам'яті був відокремлений від операцій.

В момент, коли операнди доступні в GPR, операція може виконуватись детермінованим способом. З високою ймовірністю, операція завершиться за кількість тактів, що визначаються глибиною конвеєра. Конфлікти між операндами вирішуються на апаратному рівні

Доступ до пам'яті здійснюється тільки за допомогою інструкцій завантаження та збереження, тому термін load/store архітектура часто використовується при описі RISC. Логіка різних етапів конвеєра для load/store операцій наведена на рисунку 2.6.

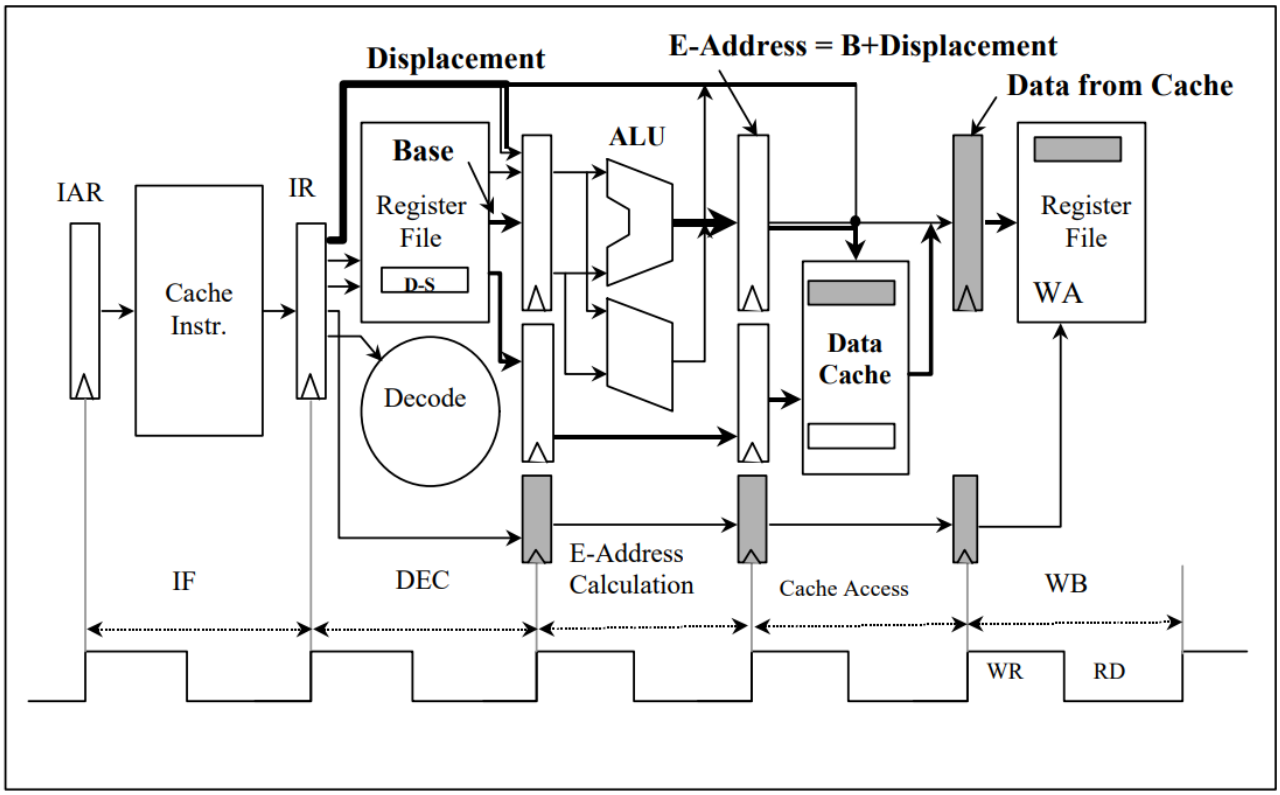

Рис. 2.6. Схема конвеєру операцій завантаження/зберігання

Конвеєр в архітектурі обчислень зі скороченим набором команд спроектований таким чином, щоб забезпечувати рівну ефективність як для операцій, так і для доступу до пам'яті.

Принцип локальності застосовується у всій архітектурі RISC. Найчастіше використання лише невеликого набору інструкцій визначає найбільш ефективну організацію конвеєра з метою раціонального використання паралелізму на рівні інструкцій [38].

Конвеєр RISC повинен ефективно обробляти три основні класи інструкцій: доступ до кешу (Load/Store), арифметичні та логічні операції, переходи (розгалуження).

З огляду на простоту конвеєра, контрольна частина архітектури RISC реалізована на апаратному рівні, на відміну від архітектури CISC. Однак, це не передбачає, що кількість інструкцій в RISC є малою. Наявний ряд сучасних RISC-машин, що володіють більшим набором інструкцій, ніж апаратне забезпечення обчислень зі складним набором команд.

На рисунку 2.7 наведена одна з реалізацій повного мікроархітектурного модуля за принципами RISC.

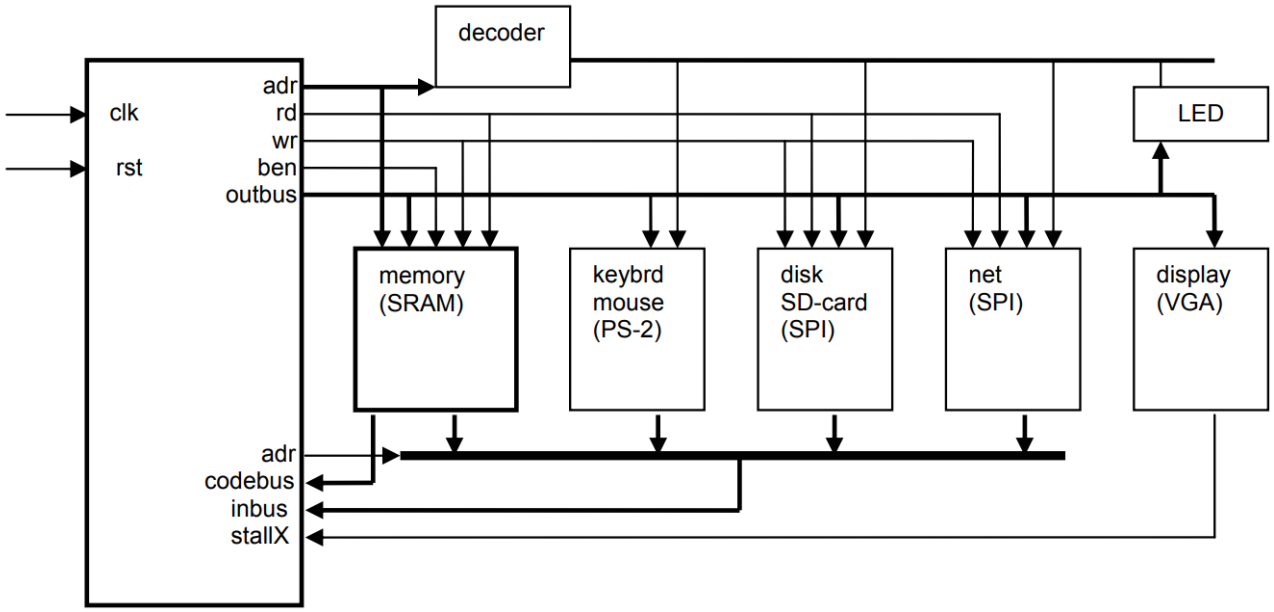

Рис. 2.7. RISC конфігурація модуля

Процесор RISC вбудований у середовище (модуль RISCTop.v), яке забезпечує його зв'язок з елементами, що знаходяться зовні чипу, але є доступними на материнській платі [38].

Середовище складається з:

- декодувальника адрес,
- мультиплексора даних,
- інтерфейсів до пам'яті та периферійних пристроїв.

Основними керуючими сигналами схеми, що оточує статичну оперативну пам'ять (SRAM) є: сигнал читання (SRoe), сигнал дозволу на запис (SRwe). В даному модулі SRAM мультиплексує лінії даних для вводу та виведення.

2.4 Висновки до розділу

В даному розділі проведено дослідження сучасних підходів до управління потоками інструкції, серед яких Branch prediction, Control Flow Integrity, комп'ютерна архітектура Reduced Instruction Set Computing. Ефективне виконання процесорних інструкцій має першорядне значення в забезпеченні безпеки мікроархітектурного рівня.

Технологія передбачення розгалужень дозволяє уникнути часових затримок, що виникають внаслідок очікування прийняття процесором рішень про здійснення переходу.

Концепція цілісності потоку керування гарантує комплексну протидію атакам, що спрямовані на зміну поведінки програмних систем. Політика безпеки реалізує виконання тільки тих інструкцій, що відповідають визначеному графу управління потоком.

В основі архітектури RISC лежить конвеєр процесора, що забезпечує швидке виконання інструкцій та визначається малим набором часто використовуваних машинних команд.

РОЗДІЛ 3
АНАЛІЗ ТА ФОРМУЛЮВАННЯ ВИМОГ ДО НОВОЇ МОДЕЛІ КЕРУВАННЯ ПОТОКАМИ ДАНИХ

3.1 Дослідження безпечних процесорів та середовищ виконання

В останні десятиліття різко зросла необхідність реалізації алгоритмів шифрування (криптографічних алгоритмів) у програмному та апаратному забезпеченні. Ефективні реалізації криптографічних алгоритмів, примітивів та протоколів на машинному рівні отримали назву криптографічних апаратних двигунів (hardware cryptographic engines).

За рівнем гнучкості двигуни можна класифікувати за декількома типами, а саме: налаштовані процесори загального призначення, криптографічні апаратні ко-процесори (апаратні криптографічні прискорювачі), криптопроцесори та криптографічні архітектури грубозернистої реконфігурації (CGRA) [39].

Криптопроцесор — це спеціалізований CPU, який виконує криптографічні алгоритми на апаратному рівні, прискорюючи процеси шифрування та дешифрування, що покращує безпеку даних та захист секретних ключів. На ринку доступно багато комерційних рішень, таких як IBM 4758, DeepCover, CryptoAuthentic, безпечні процесори SafeNet. Наявні архітектури зосереджуються на покращенні трьох ключових характеристики: безпеки, гнучкості та пропускної здатності.

Налаштовуваний процесор загального призначення (GPP) модифікується для реалізації визначених криптографічних алгоритмів. Він виконує різні функціональні модулі криптографічних алгоритмів, такі як операція модульного піднесення до ступеню для алгоритму RSA, операція S-Box для AES тощо. Основне налаштування таких процесорів — розширення набору криптографічних інструкцій для застосунків. Загальна архітектура процесорів наведена на рисунку 3.1 .

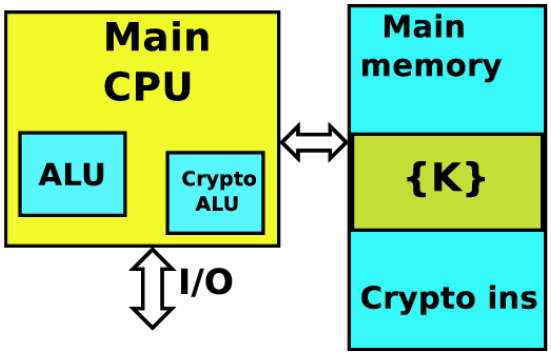

Рис. 3.1. Архітектура налаштовуваного процесора GPP

Секретні ключі зберігаються в основній пам'яті й використовуються як звичайні дані. Внаслідок цього, програмні атаки на архітектуру не вимагають серйозної складності, оскільки одна пам'ять використовується для зберігання як ключів, так і даних.

Криптографічний ко-процесор — це модуль поза межами процесора загального призначення, який прискорює виконання криптографічних обчислень. Криптографічні функції модуля реалізуються з дуже високою ефективністю.

Секретні ключі зазвичай не зберігаються в пам'яті криптографічного ко-процесора, а розташовуються як дані в регістрах або в основній пам'яті. Хостовий GPP може керувати ко-процесором або задавати його параметри, але не може змінювати чи втручатися в потік виконання його інструкцій [39].

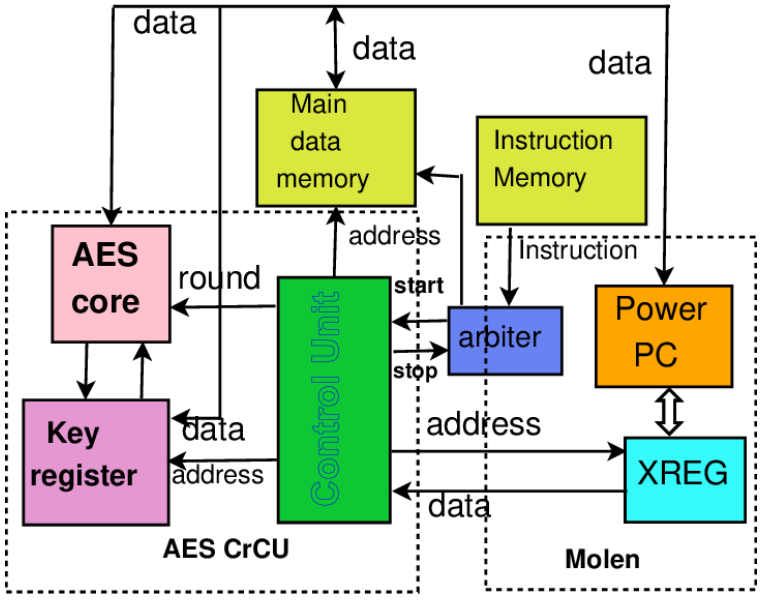

Рис. 3.2. Архітектура процесора AESCrCU

Такі системи підтримують багатозадачність: криптографічні функції виконуються у ко-процесорі, тоді як GPP одночасно виконує інші завдання.

На рисунку 3.2 наведена архітектура AES ко-процесора AESCrCU, розроблений із використанням GPP та декількох реконфігураційних блоків криптографічних обчислень (Cryptographic Computation Units, CrCU). Секретні ключі зберігаються в регістрі ключів основної пам'яті, яка підключена до AESCrCU через стандартну шину.

Іншими прикладами криптографічних співпроцесорів є багатоядерний співпроцесор AESTHETIC, одноядерний AES-співпроцесор, а також апаратна реалізація з подвійним AES-ядром для AES MultiStream (AES-MS).

Схеми CGRA, що з'єднуються з процесорами загального призначення, реалізують задані алгоритми, використовуючи криптографічні архітектури грубозернистої реконфігурації. На рисунку 3.3 наведена загальна структура криптографічної схеми з використанням GPP.

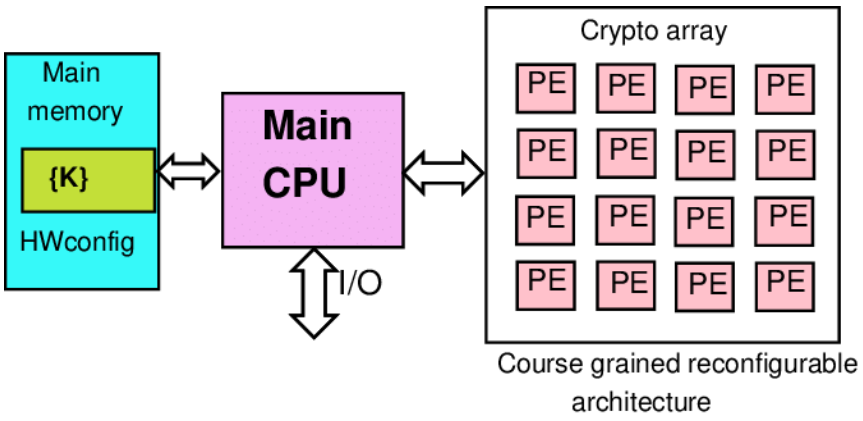

Рис. 3.3. Архітектура систем із взаємодією GPP та CGRA

Архітектура Celator, яка базується на матриці станів AES (AES-128) і реалізована з використанням систолічного масиву з 4×4 8-бітних обчислювальних елементів, також належить до схем такого типу.

Ще однією відомою архітектурою типу криптографічної CGRA є COBRA, що дозволяє реалізувати такі функції, як модульні операції, операції зсуву/обертання, множення в полі Галуа, інверсію, підстановки.

Середовища виконання є програмними шарами, які працюють поверх апаратного забезпечення. У кожному пристрої існує фізичний поділ між довіреною та недовіреною областями зі своїми середовищами виконання TEE. Код, який виконується в TEE, захищений від втручання завдяки окремому рівню захисту, який забезпечується апаратною частиною.

TEE може реалізовуватися різними способами, проте загальні концепції, наведені на рис. 3.4 залишаються незмінними.

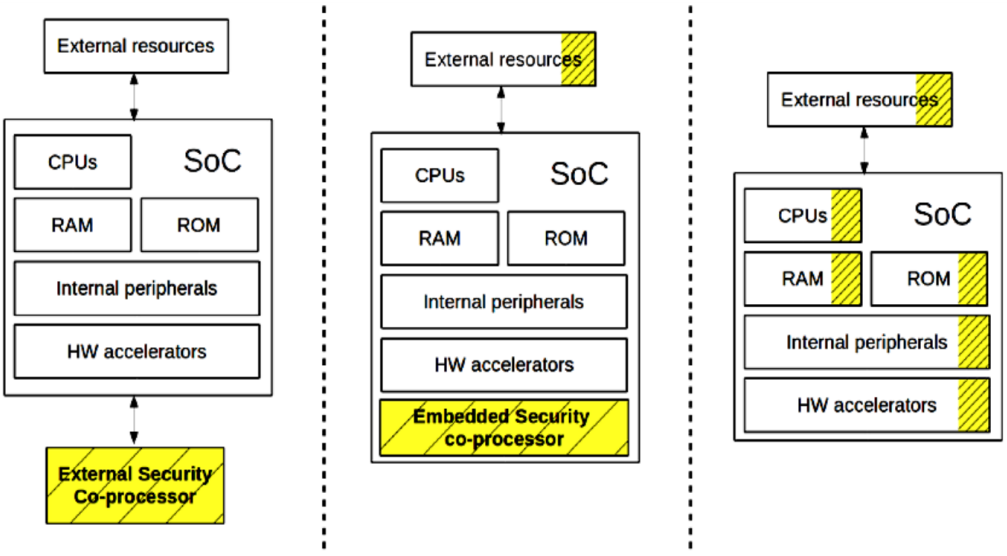

Рис. 3.4. Основні архітектурні концепції TEE

Цими архітектурними патернами є:

- на основі ко-процесора; перевагами є можливість повної ізоляції операцій, та їх виконання одночасно з основним ядром; недоліком - додаткові витрати, пов’язані з передаванням даних до ядра;
- так зване, «захищене середовище процесора» – одне ядро підтримує кілька віртуальних взаємно незалежних, що не працюють одночасно;
- підхід XOM (eXecute-Only-Memory) – досягає створення TEE за допомогою архітектурного розподілу, що запобігає витоку інформації з XOM-додатків шляхом використання «компартментів» (контейнерів);
- гібридні підходи, що поєднують можливості апаратного й програмного забезпечення; однією з таких технік є Intel Software Guard.

3.2 Огляд найбільш проблемних атак на процесори RISC

Протягом тривалого часу мікроархітектурні атаки продовжують становити загрозу для системної безпеки. У останні роки для архітектур x86 та ARM виявлено безліч побічних каналів, які використовувалися для атак на криптографічні реалізації або для стеження за поведінкою користувачів. Традиційні атаки, зокрема, атаки на пам’ять, також залишаються важливою загрозою, оскільки часто ігноруються під час проектування процесорів. На відміну від фрагментів часто коду, повністю захистити все ПЗ неможливо.

Мікроархітектурні атаки використовують певні ефекти, які виникають у ра рівні процесорної логіки. «Побічні» атаки через мікроархітектуру вилучають дані, використовуючи побічний канал, що штатно експлуатується системою. Сучасні атаки можна класифікувати за архітектурними елементами, на які вони націлені.

Процеси кешування створюють потенціал для побічних каналів, оскільки існує різниця у часі доступу до значень залежно від того, чи були вони кешовані. Найпоширеніші атаки на кеш [40]:

- Flush+Reload використовує інструкції обслуговування кешу для видалення значень зі спільної пам’яті. Атакувальник може визначити, чи були використані певні інструкції, виконавши їх завантаження зі спільної пам’яті та провівши вимірювання часу. Flush+Reload використовується для атак на криптографічні реалізації та стеження за поведінкою користувача, а також як основа для атак транзитного виконання.
- Flush+Flush — це варіант, який експлуатує поведінку інструкції очищення кешу, вимірюючи її часові характеристики виконання.
- Evict+Reload виконується, якщо інструкції обслуговування кешу є недоступними. Замість очищення кешу конкретний рядок видаляється шляхом повторного доступу до адреси у тому ж наборі кешу.
- Prime+Probe дозволяє атакувати процеси жертви, які не мають спільної пам’яті з атакувальником. У цьому методі зловмисник вивільняє набір

кешу, а потім вимірює час доступу до набору, щоб визначити чи було здійснено доступ до відстежуваної області.

Кеш трансляції адрес, також може бути використаний для мікроархітектурних атак. Різниця у часі доступу до кешованих і некешованих адрес у підсистемі пам’яті дозволяє атакувати криптографічні реалізації або проводити злом захисних механізмів компоненту рандомізації розташування адресного простору ядра .

Атаки на основі передбачення використовують різницю у часі, залежно від того чи предиктори розгалужень правильно передбачили той чи інший перехід. Якщо адреси процесів жертви та атакувальника потрапляють у той самий запис спільного предиктора, вони впливають одне на одного. Побічні канали передбачення гілок використовують затримки, пов’язані з некоректним навчанням предикторів.

Атаки на транзитне виконання або транзитні атаки базуються на виконанні потоку інструкцій, який було передбачено. Атака Spectre є однією з таких типів загроз.

На рис. 3.5 наведено загальний огляд RISC-V ядра U74 з трьома новітніми концепціями потужних атак: «Cache + Time», «Flush + Fault», «CycleDrift».

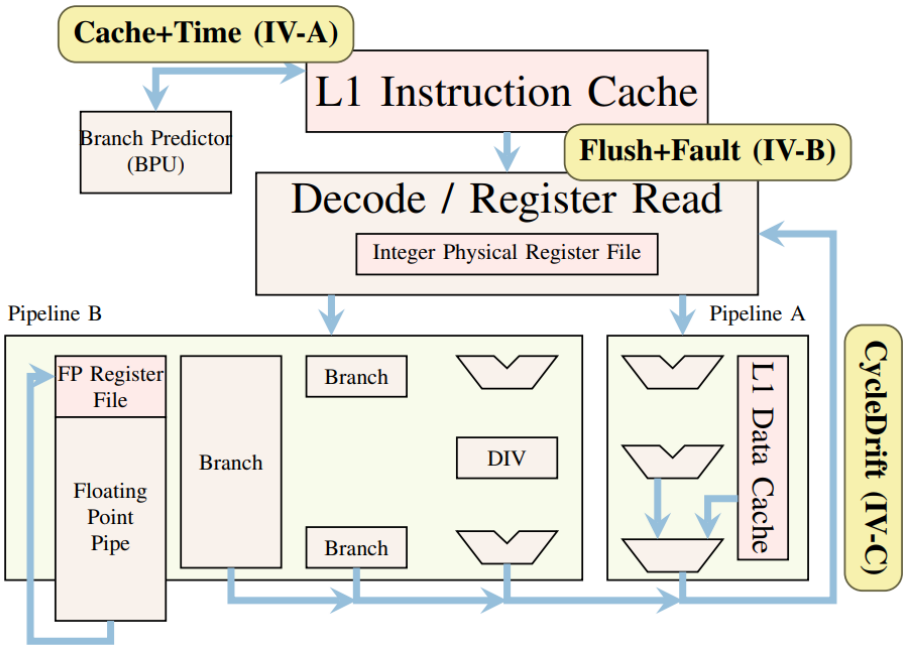

Рис. 3.5. Новітні атаки RISC-V подібних процесорів

Атака «Cache+Time» використовує прогнозування гілок у поєднанні з можливістю скидання кешу інструкцій (I-Cache) на процесорах типу RISC-V. Cache+Time володіє гранулярністю та може атакувати весь рівень кеш-ліній без необхідності використання спільної пам'яті.

Основна ідея атаки полягає в тому, щоб приховати та використати принцип іншої атаки «Evict+Time». В її ході атакуючий евакуює кеш-лінію, що його цікавить, і вимірює, чи змінився час виконання програми. При збільшенні часу виконання, приймається висновок про використанні жертвою цієї лінії. В іншому випадку лінія не використовується при атаці. Механізм атаки наведений на рисунку 3.6.

Атака використовує два основних елементи. По-перше, «Cache+Time» експлуатує можливість здійснити скидання всього кешу інструкцій непривілейованому учаснику в середовищі процесорів RISC-V. По-друге, атаці сприяє можливість обману предикторів (наприклад, на процесорах C906 та U74). Передбачуючі блоки можуть завантажувати довільні кеш-лінії з адресного простору жертви в кеш після використання зловмисником вбудованого гаджету попереднього вибору спекулятивних інструкцій [40].

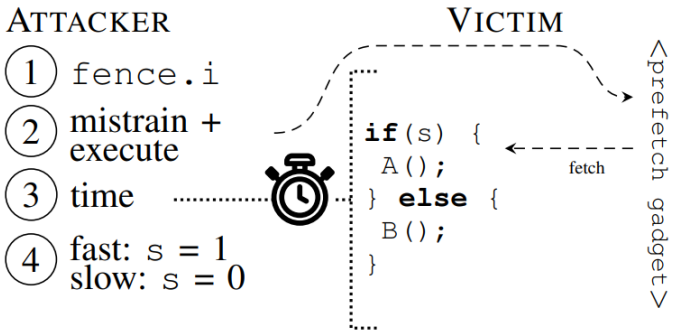

Рис. 3.6. Механізм атаки «Cache + Time»

У схемі на рис. 3.6 зловмисник скидає кеш інструкцій (1); використовує спекулятивний гаджет попереднього вибору для завантаження кеш-лінії функції A (2); вимірює час виконання програми-жертви (3); дізнається секретне логічне значення s, що є індикатором виконання потрібної інструкції (4).

Атака «Flush + Fault» є варіантом атаки «Flush + Reload», яка може бути проведеною за відсутності уніфікованих кешів. Сценарії атак «Flush+Reload» для впровадження в код жертви, залежні від результату виконання цільової кеш-лінії, що призводить до її завантаження в кеш процесора. Припускається, що кеші інструкцій (I-cache) і даних (D-cache) є уніфікованими або синхронізованими, що характерно для більшості рівнів L2 та L3 в процесорах Intel.

Ряд реалізацій RISC-V не завжди підтримують таку синхронізацію та використовують архітектуру з розділеними кешами. Атака «Flush+Fault» обходить дану властивість процесора. Основна ідея полягає в заміні кроку завантаження даних на помилковий перехід до спільного коду або негайний вихід з нього. Ці випадки носять назви «Fault-based» та «Return-based» відповідно.

В першому випадку зловмисник викликає помилку, намагаючись здійснити перехід на неіснуючу або некоректну адресу в коді жертви. Другий варіант використовує інструкцію ret – спеціальну інструкцію, яка повертає процес виконання до місця виклику, завершуючи поточну функцію. Оскільки при обидвох сценаріях код жертви виконується і завантажується, показники часу виконання відрізняються, якщо інструкції знаходяться в I-cache процесора [40].

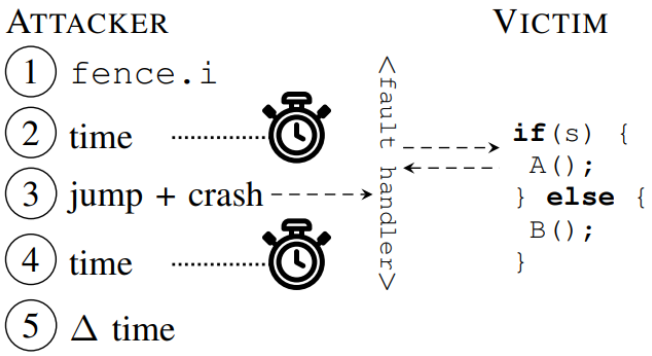

Рис. 3.7. Механізм атаки «Flush + Fault»

На рисунку 3.7 наведена логіка атаки, що описується. Зловмисник очищує кеш інструкцій (1); за допомогою гаджета попереднього вибору завантажує кеш-лінію, пов’язану з функцією A, в I-cache (2); вимірює часові характеристики виконання коду (3); визначає, чи виконувалась відслідковувана функція (4).

«CycleDrift» є атакою, яка використовує доступ до кількості виконаних інструкцій за відсутності привілейованих прав.

На відміну від x86 та ARM, RISC-V надає непривілейований лічильник виконаних інструкцій. Це створює побічний канал, оскільки інструкції, які займають більше одного такту, викликають розбіжності між кількістю виконаних інструкцій і числом тактів процесора. Зловмисник, який зчитує обидва лічильники може визначити їх кількість.

Перелічені інструкції визначаються на рівні ядра процесора і містять статистику щодо інструкцій різних програм та доменів безпеки, включно з операційною системою. Таким чином, цей побічний канал дозволяє шпигувати за кодом ядра.

Методами роботи атаки є такі виклики, як виклик вбудованого середовища для отримання ідентифікатора виробника платформи (SBI_EXT_BASE_GET_MVENDORID) і складнішого виклику для виведення символу в консоль (SBI_EXT_0_1_CONSOLE_PUTCHAR). Спостерігається чітка різниця в часі виконання цих інструкцій.

На процесорі C906 ці виклики тривають у середньому 613 та 85 109 тактів відповідно, а на U74 – 963 та 85 507 тактів. Також спостерігається різниця у виконаних інструкціях: на C906 їх завершується 265 і 4505 відповідно, а на U74 – 267 і 7050 [40].

Оскільки лічильник виконаних інструкцій та лічильник тактів є непривілейованими і не розрізняють користувацький і привілейований простір, CycleDrift дозволяє зловмиснику атакувати як непривілейовані, так і привілейовані додатки. Припускається, що цільовий код містить потік виконання, залежний від секретних даних, який можна відрізнити за співвідношенням між кількістю виконаних інструкцій і витраченими тактами процесора.

При спостереженні кількості виконаних інструкцій зловмисник може визначити, яке конкретне розгалуження було виконано. Таким чином, доступний користувачеві лічильник rdinstret відкриває додаткові вразливості для атак і дозволяє обходити механізми захисту

3.3 Формування вимог до пропонованої моделі

У контексті сучасних викликів інформаційної безпеки мікроархітектурні моделі керування потоками даних мають бути орієнтовані на забезпечення інтегративного підходу до вирішення проблем захисту інформації. Визначення вимог до таких моделей потребує високого рівня формалізації із врахуванням особливостей їх впровадження в умовах змінного середовища кіберзагроз. Пріоритетність цих вимог визначається критеріями забезпечення продуктивності, енергетичної ефективності, сумісності з існуючими стандартами та адаптивності до нових викликів [20, 22].

Однією з основних вимог до мікроархітектурних моделей є досягнення ізоляції потоків даних для запобігання витоку конфіденційної інформації через сторонні канали [21]. Сучасні дослідження підтверджують, що багатоядерні системи зі спільними ресурсами, такими як кеш-пам’ять або регістри, є особливо вразливими до атак. Моделі керування потоками повинні забезпечувати сегментацію кешу або впровадження динамічних політик управління, що дозволяють мінімізувати ризики перехресного доступу між потоками [11].

Заходи безпеки для захисту інтерфейсів і поширення даних через канали є критичними для забезпечення конфіденційності та цілісності даних. Пропонована модель протидії загрозам використовуватиметься для виявлення атак на дані та пошкодження пам’яті. Модель повинна відстежувати потоки даних, що надаються через ненадійні канали зв’язку. Новий підхід маркуватиме шкідливі дані тегами для позначення користувацьких політик безпеки та відстежувати їх розповсюдження, це дозволить оцінювати достовірність даних у реальному часі. Розроблені функції, серед іншого, повинні використовуватись для статичної перевірки на етапі проектування, та забезпечення динамічної перевірки на етапі виконання.

Модель протидії загрозам реалізовуватиме новий підхід, який підтримує як безпечне розповсюдження даних на рівні інструкцій, так і на рівні вентилів, на відміну від існуючих методологій, що охоплюють тільки одну з двох областей.

Архітектурне рішення повинне забезпечувати точність для обох рівнів деталізації, що, зі свого боку, посилює рівень безпеки.

Апаратний рівень моделі базуватиметься на основі процесорів з архітектурою RISC-V. Завдяки відкритій ліцензії та гнучкості ядра, концепція RISC-V знайшла широке застосування в науковій і економічній сферах. Даний архітектурний стандарт володіє більшою опірністю до атак, ніж відомі рішення. Процесори RISC-V з ходом часу можуть стати більш безпечною альтернативою старішим схемам, таким як x86 та ARM.

Рівень тіньової логіки повинен забезпечувати структурну перевірку з урахуванням властивостей безпеки. Універсальна модель синтезуватиме логіку двох концепцій

- виявлення витоків на рівні логічних вентилів;
- відстеження тегів для кількісного виявлення витоку інформації; в цьому випадку повинні враховуватись обмеження площі мікропроцесора.

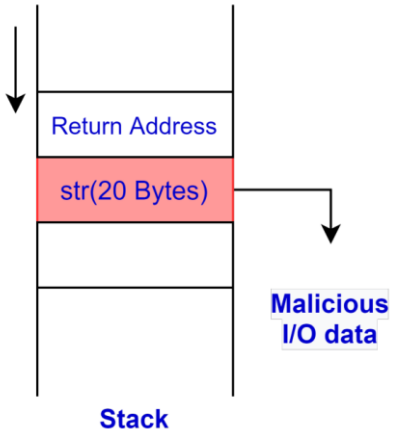

Рис. 3.8. Атака зворотньої адреси

В процесі проектування основна увага приділяється пошкодженню пам’яті через атаки переповнення буфера та атаки на зворотні адреси (рис. 3.8). Крім того, запропоновані методи оцінюватимуть виявлення витоків на рівні даних у функціональних модулях. Пропонована модель інтегрує апаратне і програмне забезпечення шляхом трансляції архітектурних розширень у моделі симуляції, орієнтовані на компілятор.

3.4 Висновки до розділу

В даному розділі проведено дослідження ряду застосовуваних моделей безпечних процесорів та середовищ виконання TEE; охарактеризовано структури налаштовуваного процесора, криптографічного ко-процесора та схем CGRA; проведений аналіз архітектурних патернів побудов TEE.

Було оглянуто механізми найбільш проблемних типів мікроархітектурних атак; досліджено логіку новітніх загроз «Cache+Time», «Flush+Fault», «CycleDrift», що є посиленням існуючих атак на RISC процесори.

Охарактеризовано важливість та сформовано вимоги до пропонованої мікроархітектурної моделі протидії загрозам програмного забезпечення. Модель повинна надавати можливість відстеження потоків даних через ненадійні канали. Реалізація нового підходу підтримуватиме безпеку поширення даних як на грубозернистому, так і на рівні логічних вентилів.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ТА ОЦІНКА ПЕРСПЕКТИВ ЗАСТОСУВАННЯ ІНТЕГРОВАНОЇ ТЕХНІКИ ВІДСТЕЖЕННЯ ПОТОКІВ ДАНИХ

4.1 Математична формалізація безпеки апаратного забезпечення

Об’єктом формалізації є система забезпечення безпеки, яка містить процесор (зокрема, блок шифрування), оперативну пам’ять (RAM), постійну пам’ять (ROM) та пам’ять EEPROM, яка зберігає дані навіть при вимкненні живлення (рисунок 4.1).

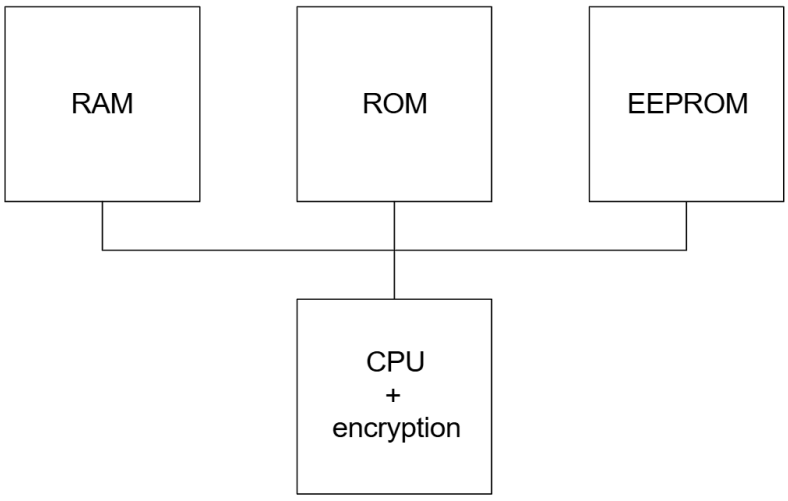

Рис. 4.1. Логіка апаратного забезпечення

Збереження безпеки інформації, що знаходиться в компонентах пам’яті, забезпечується шляхом дотримання наступних принципів:

- дані, що зберігаються в будь-якому з компонентів пам’яті, мають бути захищені від несанкціонованого доступу або модифікації;
- функції, що стосуються безпеки, і зберігаються в будь-якому з компонентів пам’яті, мають бути захищені від несанкціонованого доступу або модифікації;
- виконання будь-якої апаратної тестової процедури без авторизації має бути неможливим.

В ході формалізації, процесор визначається як система переходу станів. Системна модель визначається як варіант автоматів переходів станів над нескінченними структурами. Автомат M – це кортеж з п’яти компонентів:

$$M = (State, In, Out, S_0, \delta) \tag{4.1}$$

Тут $State$ позначає множину станів, In – множину можливих вхідних даних, Out є множиною можливих вихідних даних, $S_0 \subseteq State$ це множина початкових станів, $\delta \in State \times In \rightarrow P(State \times Out)$ функція переходу станів.

Автомат, визначений таким чином, поводиться недетерміновано, що враховує неповну визначеність поведінки. M визначається як множина можливих реакцій, що характеризуються послідовністю вхідних і вихідних даних.

Поведінка автомата M визначається як трійка:

$$b = (s, in, out) \in State^{\omega} \times In^{\omega} \times Out^{\omega} \tag{4.2}$$

Тут $(s_i, out_{i-1}) \in \delta(s_{i-1}, in_{i-1})$ для всіх $i \in N, 1 \leq i \leq \#b$.

Множина можливих поведінок автомата наступна:

$$\|M\| = \{b | b \text{ є поведінкою } M\} \tag{4.3}$$

Використання цих автоматів необхідне для збереження моделі простою й абстрактною, а також для моделювання та доведення відповідних властивостей.

Val позначає множину значень, що не визначені детально та відображають інформацію, яка міститься в об’єктах, або може бути з них отримана.

Формалізація цілей безпеки процесора стосується поведінки автомата, що описує системну модель, і задається у вигляді вимог.

Вимога 1: Обладнання має бути захищене від несанкціонованого розголошення функцій забезпечення безпеки.

Припускається, що $b = (s, in, out) \in \|M\|$. Тоді, для всіх $j < \#b$ повинна виконуватись імплікація:

$$out_j \in Val_{s_j}^{F_{sec}} \Longrightarrow (subject(in_j) = Pm\ f \vee s_{j+1} = e) \tag{4.4}$$

Вимога 2. Апаратне забезпечення повинно бути захищене від несанкціонованої модифікації функцій забезпечення безпеки. Додатково вимагається, щоб видалення функцій безпеки у станах, відмінних від стану помилки, було дозволеним тільки при фазових переходах.

Припускається, що $b = (s, in, out) \in \|M\|$. Для всіх $i \in N$, де $s_i \neq e$, повинно виконуватись:

$$f \in F_{Sec} \cap fct(s_i) \Longrightarrow Val_{s_i}^F(f) = Val_{s_0}^F(f) \tag{4.5}$$

А для $s_{i+1} \neq e$:

$$ph(s_i) = ph(s_{i+1}) \Longrightarrow fct(s_i)|F_{Sec} \subseteq fct(s_{i+1})|F_{Sec} \tag{4.6}$$

Вимога 3. Не повинно відбуватися несанкціонованого виконання тестових функцій. Такі запити на виконання відхиляються.

Нехай $b = (s, in, out) \in \|M\|$. Тоді для всіх $i \in; j = 0{,}1;\ sb \in Sb$ повинно виконуватись:

$$in_i = exec(sb, f) \wedge f \in F_{Testj} \Longrightarrow sb = Pm\ f \vee \begin{cases} s_{i+1} \in \{s_i, e\} \\ out_i = \text{"no"} \end{cases} \tag{4.7}$$

Математична безпекова модель складена у відповідності до критеріїв ITSEC згідно з рівнем забезпечення якості E4. Формалізація забезпечила максимальну простоту та абстрактність моделі без втрати її змістовності. Модель охоплює три специфікації безпекових вимог.

4.2 Розробка мікроархітектурної моделі протидії загрозам

Зі зростанням кількості електронних пристроїв, інтеграції сторонньої інтелектуальної власності у ланцюжок постачання напівпровідникової промисловості, та участі все більшої кількості ненадійних програмних середовищ, виник ряд проблем із безпекою апаратного забезпечення, які відкривають можливості для потенційних атак, таких як несанкціонований доступ до даних, ін’єкція збоїв та порушення конфіденційності. Різні методи безпеки були запропоновані для забезпечення стійкості пристроїв проти потенційних вразливостей, проте жоден із них не можна охарактеризувати як універсальне рішення.

В магістерській роботі розроблена та запропонована інтегрована техніка відстеження інформаційних потоків (Information Flow Tracking, або IFT), яка забезпечує безпеку під час виконання, захищає цілісність системи шляхом відстеження потоків даних із ненадійних каналів зв’язку. Існуючі апаратні схеми відстеження потоків є або занадто вузько спрямованими, або ресурсозатратними моделями. Їх рівень деталізації та точності обробки даних є мінімальним, забезпечуючи лише цілісність потоків керування або інформації. Жодна з поточних моделей безпеки не забезпечує багаторівневої деталізації через складність одночасного балансу між гнучкістю та апаратними витратами.

Запропонована апаратна техніка IFT являє собою інтегровану грубозернисту техніку відстеження даних, доповнену точним модульним блоком IFT на рівні логічних елементів для критичних з точки зору безпеки шляхів передачі даних. Це є новаторським підходом, який інтегрує підтримку інструментального ланцюга для розширень безпеки RISC-V разом із багаторівневою деталізацією для досягнення кращої точності та запобігання хибним спрацюванням системи.

Архітектурна модель продемонстрована на платформі RISC-V, доповнена розширеннями безпеки, які дозволяють виявляти та зупиняти програмні атаки на процесорну пам’ять. Безпекові концепції захищатимуть систему від ненадійних

джерел різних типів. Ці контрзаходи забезпечуються шляхом захисту зворотної адреси за допомогою розширюваного механізму тегів. У свою чергу, механізм тегів реалізує модульний підхід – призначає біти тегів та відстежує їх розповсюдження під час виконання програми для виявлення некоректних даних.

Процесор RISC-V підтримує як 32-бітні, так і 64-бітні варіанти з інструкціями фіксованої довжини 32 біти та кодуванням змінної довжини для налаштовуваних застосувань. Запропонована схема інтегрує функції безпеки та розширює їх до рівня набору інструкцій.

Як вказано в розділі 2, RISC-V є архітектурою типу load–store, у якій доступ до пам’яті здійснюється лише через інструкції завантаження та збереження. Модель IFT зосереджується на цих інструкціях для здійснення перевірки стану цілісності вмісту пам’яті. Інструкції завантаження закодовані у форматі I-type, а інструкції збереження — у форматі S-type. Для забезпечення перевірок безпеки механізму тегів з 1-бітними тегами в набір інструкцій (ISA) додано два нових елементи. Виходячи з кодування інструкцій завантаження та збереження, специфікованих у ядрі RISC-V, нові інструкції володіють наступними характеристиками:

- у кодуванні інструкції завантаження: LDTCHECK;
- у кодуванні інструкції збереження : SDTCHECK.

Нове кодування інструкції збереження використовується для збереження даних і призначення бітового тега адресі даних: якщо дані оцінюються як підозрілі, теговий біт встановлюється як 1 або 0. Нове кодування інструкції завантаження використовується для завантаження даних і перевірки значення бітового тега (1 чи 0). Також розроблено модуль тегів, в якому для їх зберігання призначається окрема таблиця-масив. Вона отримує окремий віртуальний доступ із кешем тегів замість основної пам’яті. Це підвищує рівень захисту доступу до таблиці.

Реєстр управління та стану (Control and Status Register, або CSR) в інструкційному наборі ISA пропонує спеціальний простір для невикористаних адрес, який використовується для призначення нового статусу для тегів і умов

невідповідності, які призводять до винятків. Ядро процесора було розширено, щоб додати теговий біт. Також для його підтримки розроблені нові запити для читання й запису. Модуль тегів використовується для перевірки адрес інструкцій завантаження та збереження з метою зменшення навантаження на архітектуру.

Модуль тегів складається з трьох компонентів: ініціалізації тегів, поширення тегів та перевірки тегів. Використання 1-бітних тегів зменшує навантаження, а доступ до пам’яті відбувається окремо у кеші тегів.

Модуль ініціалізації призначає теги всім адресам даних із ненадійних джерел за допомогою спеціально доданих інструкцій завантаження та збереження.

Модуль поширення тегів відстежує біти тегів усіх адрес даних і зберігає інформацію у кеші тегів. Мінімальна інформація про шлях зберігається у кеші тегів для уникнення надмірного навантаження і покращення продуктивності моделі.

Модуль перевірки тегів використовується для перевірки тегових бітів усіх адрес даних після виклику процедури, і при наявній невідповідності у біті тегу, генерується виняток.

Блок перевірки верифікує всі властивості, пов’язані із політиками безпеки для даних. За цими принципами дані заздалегідь класифікуються як підозрілі та повністю відстежуються для захисту від змін або зловмисного впливу на пам’ять. На рисунку 4.2 показана загальна схема системи з ядром RISC-V, механізмом тегів та рівнем логічних вентилів IFT із тіньовою логікою, де окремі блоки використовуються для захисту пам’яті від програмних атак і витоків даних.

Структура пам'яті є важливим аспектом архітектури, де стеки відіграють важливу роль у збереженні тимчасових змінних, створених за допомогою функцій. В ході виконання програми інформація зберігається разом з параметрами, адресою повернення та базовим вказівником. Дана схема дозволить покращити захист та не допустити можливості експлуатації зловмисником механізму користувацького введення без перевірки меж, що може призвести до перезапису адрес повернення й інших областей пам'яті.

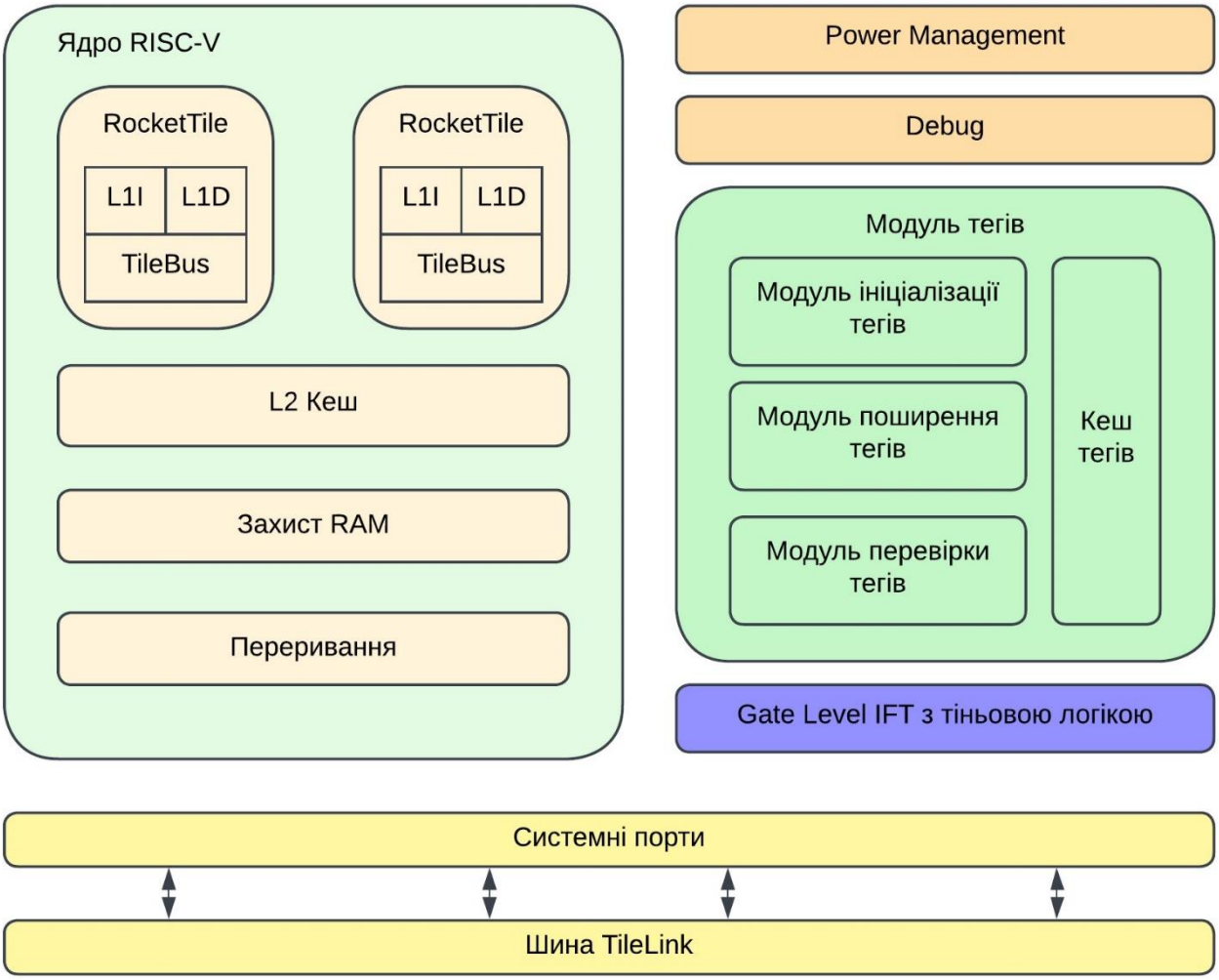

Рис. 4.2. Загальна схема системи з ядром RISC-V

Модель відстеження інформаційних потоків на рівні ISA інтегрована на мікроархітектурному рівні і включає безпечні інструкції, зокрема, завантаження та збереження. Запропонована технологія змінює обчислювальний процес в основному для шляху даних процесора (datapath) і його логіці керування. Інтегрований рівень вентилів IFT використовує логіку на рівні даних тільки для критичних, з точки рівня безпеки, модулів. Обидві моделі взаємодіють через інтерфейс TileLink – це конекторний стандарт на рівні чипа, який забезпечує кілька компонетів-майстрів з когерентним доступом до пам'яті та інших підлеглих пристроїв. TileLink призначений для використання в системах на чипі, щоб з'єднувати багатоцільові мультипроцесори, ко-процесори, акселератори, двигуни DMA та інші пристрої різного рівня складності.

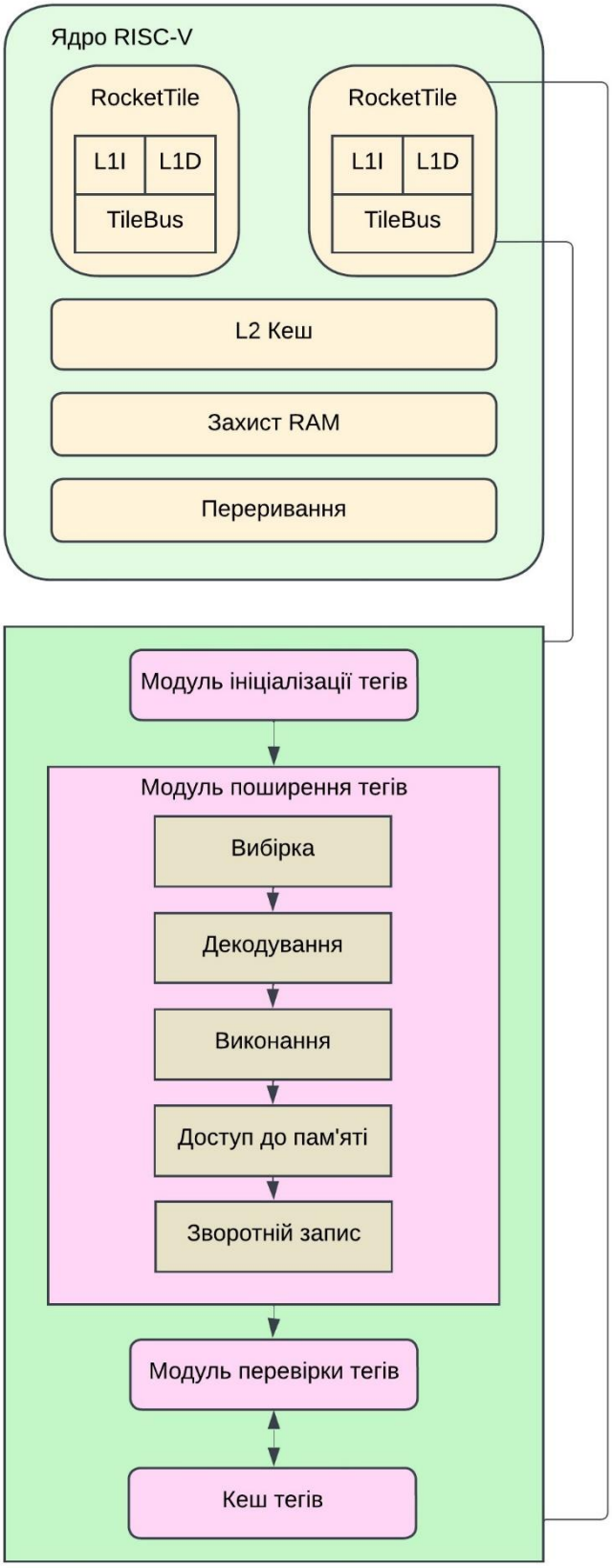

Рис. 4.3. Структура модуля тегів

Як показано на рисунку 4.3, модуль тегів на рівні ISA в ядрі RISC-V оновлюється в процесі поширення сигналу через процесорний шлях даних на

рівні, так званого, грубого гранулярного контролю. Цикл шляху даних процесора модифікується з додаванням ініціалізації тегу, поширення тегу та перевірки тегу для безпечних інструкцій завантаження й збереження.

Модуль ініціалізації тегу додає 1-бітовий тег до нової інструкції і переходить до наступного процесу поширення тегу. Наступний модуль реалізує механізм поширення тегу через цикл інструкцій: вибір, декодування, виконання, доступ до пам'яті та зворотній запис для продовження безпечних інструкцій завантаження й збереження. Наостанок, модуль перевірки оцінює відповідність біту тегу, взаємодіючи з кешем тегів, та, за потреби, викликає переривання.

Потенційно шкідливі канали введення/виведення для системи ідентифікуються через стандартизовані політики безпеки. Зосереджуючись на пошкодженні пам'яті, вони асоціюються з інструкціями завантаження й збереження та адресами повернення. Усі дані, надані користувачем, позначаються як шкідливі, оскільки вони використовують інструкції load/store з адресами джерела та призначення.

Також відслідковується значення лічильника програм (PC) як адреса повернення, яку потрібно захищати від модифікації. Правила перевірки тегів застосовуються до безпечних користувацьких інструкцій з тегами. Якщо значення тегів не збігаються, викликається переривання, що повинне зашкодити потенціалу атаки. При виконанні умовної інструкції переходу, модуль перевірки тегів перевіряє адресу і дозволяє виконання тільки в разі співпадіння тегів з таблицею. В іншому разі виклик припиняється – таким чином підвищується точність моделі з грубим гранулярним рівнем. Політики безпеки для захисту адреси повернення визначають правила перевірки для відслідковування всіх викликів процедур, а також введених користувачем даних.

Компоненти моделі на рівні gate-level відслідковують шляхи, чутливі до безпеки, використовуючи детальну інформаційну рівня логічних блоків. Детальний IFT процесі реалізує відслідковування неявних, явних та прихованих потоків інформації на рівні даних. У інтегрованій схемі CF-IFT дизайн на рівні елементів схеми зосереджений на специфічному безпековому критичному модулі

та його відповідному потоці даних, щоб мінімізувати накладні витрати безпеки й продуктивності. Спеціальні критичні модулі, такі як криптодвигуни та акселератори, які обмінюються секретними ключами або безпековою інформацією, вважаються логічними блоками, до яких застосовується IFT на рівні елементів схеми.

Без будь-яких змін у апаратному дизайні, логіка рівня gate-level інтегрується з ядром RISC-V як окремий модуль для виконання інформаційних політик критичних модулів розробленої моделі. Кожен вхід і вихід усього модуля позначений як забруднений, і для забруднених вентилів реалізується бібліотека тіньової логіки. Політики безпеки застосовуються для перевірки забрудненої інформацією модуля. Шкідливі входи для вентилів визначаються шляхом виявлення довільних змін на входах, які впливають на вихід, адже це призводить до витоку даних та збоїв, а також активує приховані функції.

Бібліотека тіньових вентилів формується для всіх основних компонентів gate-level рівня і порівнюється з основною логікою для позначення забруднених вихідних бітів. Модуль перевірки інформаційного потоку використовується для його відслідковування на основі реалізованих політик безпеки. Точність логіки вказується на основі забрудненої інформації. Якщо політика безпеки порушена, вихід вважається спотвореним з витоком інформації.

Політики безпеки для логічних вентилів і відповідних забруднених бітів реалізуються для всіх вентилів порівняно з бібліотекою тіньової логіки. Спочатку користувач вибирає модуль для відслідковування. Потім цей компонент або програма перетворюються в схему на рівні вентилів для відслідковування інформаційного потоку окремих даних і поділяються на підмодулі. Модуль IFT на рівні вентилів отримує інформацію та політики безпеки з тіньової логіки та модуля перевірки політик IFT для визначення забруднених входів, точного місця витоку даних і спотворених входів. Модуль відслідковує ці ненадійні шляхи інформації та викликає переривання, якщо відбувається зміна виходу та інформаційного потоку, що призводить до виявлення шкідливих даних.

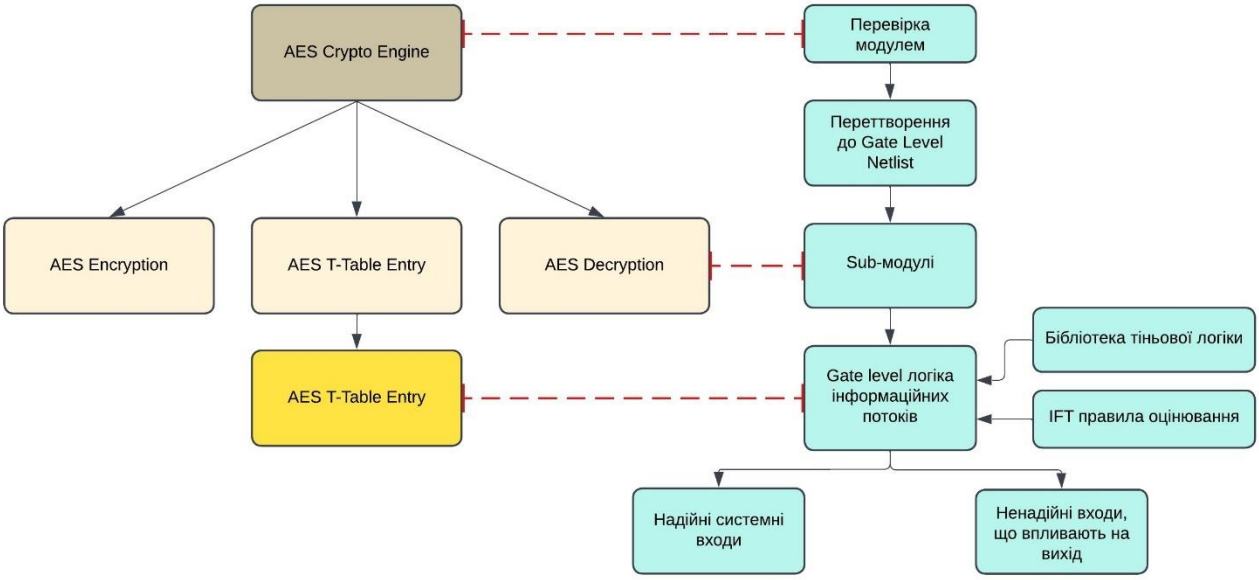

Рис. 4.4. Потоки моделі IFT на рівні вентилів

На рисунку 4.4 демонструється високорівневий потік автоматизації інтеграції IFT на рівні вентилів для безпекових критичних потоків даних до криптографічних ядер. Криптографічні ядра є складними і можуть виконувати кілька раундів, здійснюючи обчислювально витратні операції. Ядро оцінюється за властивостями безпеки та вибирає підмодулі, які є критичними для виконання аналізу під час роботи.

Як показано на схемі, модуль AES (Advanced Encryption Standard) розділяється на підмодулі ядер шифрування та дешифрування, які використовують Т-таблиці для заміни байтів. Т-таблиці вразливі до атак на кеш, які можуть шкідливо оновлювати заміну і/або спричиняти витік інформації. Підмодуль Т-таблиць застосовується при інтеграції детальної IFT на рівні вентилів для моніторингу сигналів за допомогою тіньової логіки. Модуль IFT на рівні вентилів відслідковує ненадійні шляхи інформації, викликає переривання, якщо відбувається зміна виходу, та виявляє поширення шкідливих даних під час виконання.

Запропонований підхід є ефективним без змін в апаратному забезпеченні та інтегрується разом з IFT на рівні архітектури, що дозволяє створити точну і ефективну модель з точки зору продуктивності. Залежно від програми або

архітектурних концепцій, доданих до системи, IFT на рівні вентилів виконується на будь-яких критично чутливих модулях. Ключі в модулі шифрування AES можуть бути відслідковані за допомогою IFT на рівні вентилів для перевірки цілісності та витоку даних із системи.

Таблиця 4.1

Таблиця істинності тіньової логіки на рівні вентилів при входах a і b

Вхід b Trusted/Untrusted	Результат операції OR
0/T	0, при a = 0 1, при a = 1
0/U	0/1, при a = 0 1/0, при a = 1
1/T	1, при a = 0 1, при a = 1
1/U	1/0, при a = 0 1/0, при a = 1

В таблиці 4.1 наведено спрощену тіньову логіку для вентиля з двома входами a і b, що реалізує операцію логічного АБО. У цьому прикладі вхід b вважається ненадійним, а таблиця істинності тіньової логіки демонструє, як забруднений вхід впливає на вихід. Подібним чином бібліотека тіньової логіки містить таблицю істинності для кожного окремого вентилю, що дозволяє відслідковувати забруднені дані і знаходити точне місце, що призводить до витоку даних.

Моделі виявлення витоків на рівні вентилів з парсерами та логічними модулями для формальної верифікації виявляють витікання шляхів, але спричиняють високу обчислювальну складність при роботі з великими проектами. Модель відстеження багатобітних міток кількісно виявляє витоки інформації проте є залежними від площі. Більш доцільною є об'єднана модель для поширення на рівні вентилів, що генерує синтезовану логіку поширення, яка може бути використана в інструментах автоматизації проєктування електронних схем. Запропонована технологія інтегрує механізм міток і забезпечує як грубий, так і більш детальний рівень гранулярності для відстеження даних.

4.3 Проведення симуляції та оцінювання моделі

4.3.1 Тестування архітектурної моделі

Для побудови та проведення тестуванні розробленої безпекової моделі використовувався додаток Spike — це симулятор промислової стандартної архітектури (ISA) для RISC-V, який інтегрує специфікацію ISA та виступає як еталонна модель для RISC-V. В ході роботи розширено можливості симулятора та додано нові розширення безпеки для виявлення атак під час виконання у моделі з RISC-V. Вона розширює модель програмної симуляції, дозволяючи її кореляцію з апаратною моделлю для моніторингу виконання IFT.

Spike є еталонним симулятором програмного забезпечення RISC-V ISA на C++. Він забезпечує повну емуляцію системи або емуляцію через проксі за допомогою HTIF/FESVR. Spike служить відправною точкою для запуску програмного забезпечення на цілі RISC-V та підтримує такі основні функції: кілька ISA через розширення RV32IMAFDQCV; кілька моделей пам’яті – Weak Memory Ordering (WMO) і Total Store Ordering (TSO); привілейовану специфікацію – режим машинного обладнання, супервізора, користувача; відлагодження; одноетапне налагодження з підтримкою перегляду вмісту пам'яті/реєстру; можливість роботи з декількома процесорами; можливість додавання й перевірки нових інструкцій.

Мета модифікації компілятора/асемблера полягає у покращенні ISA за рахунок кількох розширень набору інструкцій. Симулятор ISA Spike використовується для тестування модифікацій ISA на програмному рівні та розділений на різні модулі. Кожен модуль відповідає за симуляцію певного блоку архітектури, і додавання нових блоків у симулятор дозволяє включати нові розширення безпеки в модель. Початковий крок — оголошення інструкції у форматі інструкції для генерації значень адреси відповідності та маскування для нової інструкції. Наступний крок — опис довжини інструкції, кількості операндів і функціональності, а також додавання відповідних та основної адреси разом з апаратною моделлю.

На рисунку 4.5 наведений формат поля для архітектури RISC-V, що дозволяє додавати нові інструкції до структури кодів операцій RISC-V. Характеристика полів: назва інструкції, довжина, клас інструкції, операнди, механізм відповідності, маска, відповідність інформаційному коду, інформація про інструкцію.

Рис. 4.5. Формат поля інструкції в RISC-V

Виділені червоним поля змінюються для розміщення нових інструкцій у ланцюгу інструментів. Функції перевірки та збереження тегу реалізовані так, щоб забезпечити виконання умов його перевірки для адреси повернення разом із політиками безпеки для визначення потоку тегів.

Симуляція нових інструкцій передбачає додавання до модуля тегів Spike функцій перевірки та збереження бітів тегу. Для проведення симуляції розробленої моделі IFT була, серед іншого, реалізована атака переповнення буфера, коли дані, введені користувачем, вважаються ненадійними. Безпекова модель повинна виявляти модифіковану адресу повернення та запобігати атаці. Місце розташування адреси повернення, збереженої в стеку, отримується в результаті атаки. Підтримка інструментального ланцюга забезпечує гнучкість для додавання політик безпеки та розробки тестових середовищ виконання аналізу безпеки ISA RISC-V з мінімальними апаратними накладними витратами та більш точною логікою.

У модулі тегів обробляються мітки, призначені для адрес повернення та даних, а інформація зберігається у кеші міток. Подібно до базових інструкцій завантаження/збереження, інструкції з покращеною безпекою, що базуються на мітках, використовуються для додавання тегів лише до адрес повернення та даних, визначених користувачем.

В лістингу 4.1 наведений псевдокод модуля міток разом із кешем тегів, який створює клас для кешу міток із їх різними параметрами:

Лістинг 4.1

```
//1. Створення класу для розширення модуля тегів за допомогою
пакетних параметрів
class Tagcache extends Tagmodule
     {
          val index, matchbit, tagbit …
     }
//2. Функції перевірки умов відповідності тегів та валідності
when tagbit == matchbit
     {
          update the tag fields
     }
//3. Оновлення записі та масиву в кеші тегів
class Counter extends Tagcache
     {
          tags updated with direct mapping entries
     }
//4. Політики безпеки, використовувані для перевірки функцій
     {
          security policy functions to fetch the values and
          check the validity (store tag and check tag)
     }
```

Були реалізовані функції перевірки для зіставлення міток, що включає лічильник для підтримки масиву тегів у кеші, який оновлюється на основі записів. Визначення політик безпеки потрібні для проведення зіставлення бітів тегів і їх валідації.

Для створення прототипу та запуску модифікованого RISC-V Rocket Chip за допомогою Vivado використовувалась плата Xilinx Artix-7 Field Programmable Gate Array (FPGA). Щоб протестувати архітектуру, модифіковано користувацький додаток, який реалізує атаку переповнення буфера. Кількість використаних таблиць пошуку (LookUp Table, LUT) та тригерів (Flip-Flops) для запропонованої моделі IFT порівнюються із кількістю засобів, необхідних для моделей на базі криптографічного стеку адрес повернення (Cryptographic Return Address Stack, CRAS).

Результати тестування наведені в таблиці 4.2.

Таблиця 4.2

Тестування ресурсної ефективності моделі

Архітектура	Кількість LUT	Кількість FF
Розроблена модель IFT	10 528	7 113
Моделі на базі CRAS	11 468	6 130

Запропонована модель використовує 10528 елементів LUT та 7113 тригерів Flip-Flops для своєї реалізації на базі архітектури RISC-V, що сумарно є на 43 одиниці меншим, за ту кількість елементів, яка потрібна для моделі CRAS. У моделі CRAS використовується окремий стек для зберігання адрес повернення, щоб виключити витік даних із загальної пам’яті. Тестування показує, що загальне зростання використання ресурсів FPGA не перевищує 1%.

4.3.2 Тестування на рівні логічних елементів

В ході тестування моделі на рівні логічних вентилів, проводився аналізуються властивостей безпеки та верифікація на схемах ISCAS-85 Benchmark із використанням критичних шляхів даних.

Комбінована схема C432, що є контролером переривань, використовується для ілюстрації реалізації IFT на рівні вентилів у апаратному забезпеченні. Схема складається з 36 входів і 7 виходів із загальною кількістю 160 вентилів, у яких кожен канал активується за пріоритетом на основі запитів шини. Початкова схема перетворюється в netlist на рівні вентилів за допомогою реалізованого алгоритму і верифікується як модуль для відстеження. Модуль класифікується на підмодулі на основі інформації та критичних даних. Наостанок, логіка IFT на рівні вентилів використовується для порівняння відстежуваного підмодуля з бібліотекою тіньової логіки. Застосовуються політики безпеки для відстеження вентилів, на вихід яких впливає вхід. Це надає набір надійних та ненадійних вхідних вентилів, які потребують відстеження.

Модель IFT на рівні вентилів відстежує ненадійні вентилі та виявляє витік інформації, аналізуючи дані всіх ненадійних входів. Під час цього процесу спостерігається зміна потоку даних, що викликає переривання, вказуючи на прихований шкідливий потік інформації в абстракції ISA. Ця модель надає точну логіку для прогнозування ненадійних даних. Реалізована як окрема автоматизована модель, вона забезпечує менші накладні витрати на площу в порівнянні з іншими існуючими моделями GLIFT, що подвоюють вентилі тіньової логіки.

На рисунку 4.6 наведена схема моделі IFT на базі плати розробки PYNQ.

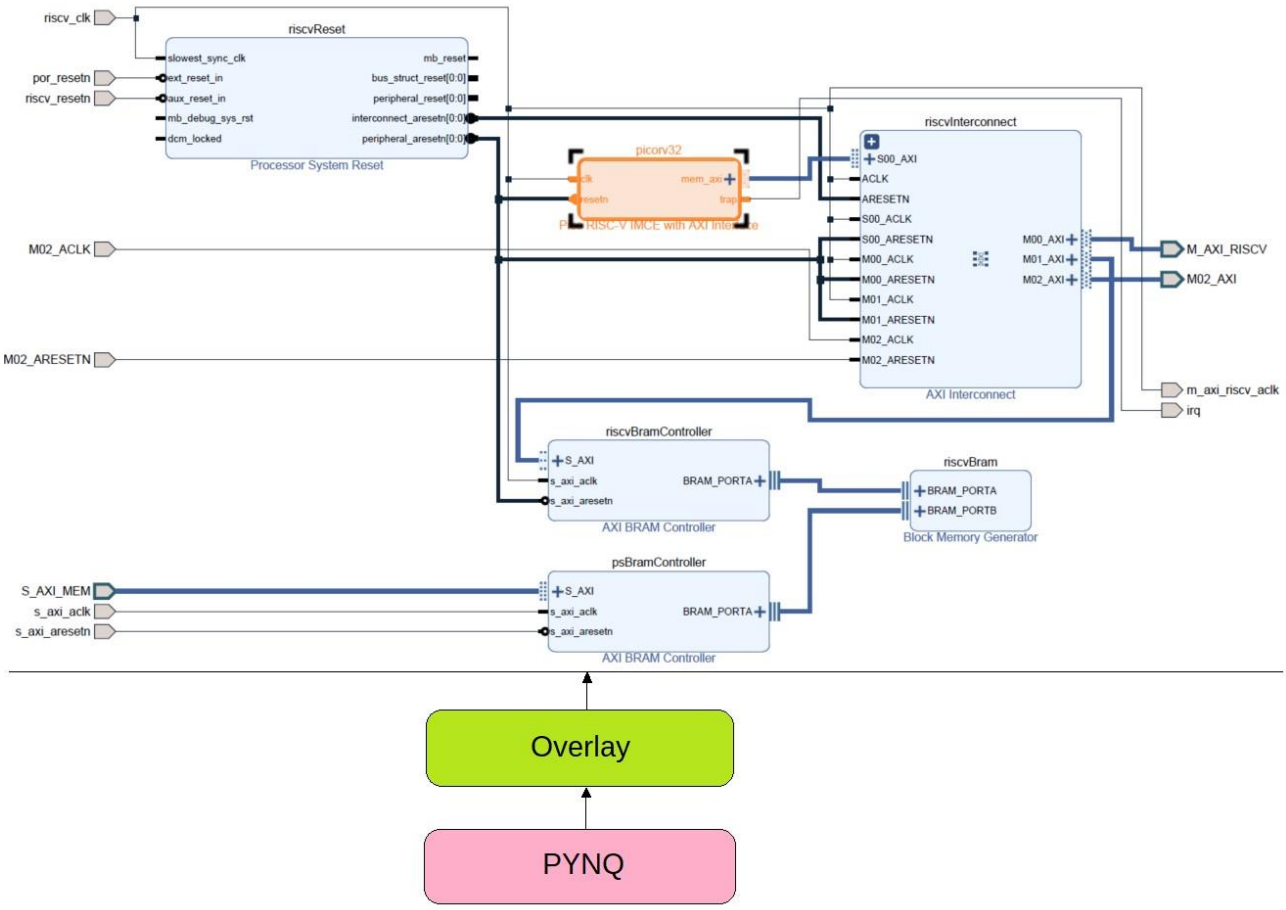

Рис. 4.6. Потік PYNQ з RISC-V IP та інтерфейсом AXI

Xilinx PYNQ FPGA плата використовується для демонстрації моделі IFT на рівні вентилів з схемами ISCAS-85. Оскільки специфікація RISC-V є надзвичайно гнучкою, різні варіанти м'яких процесорів можуть бути використані як інструмент для вивчення моделі загроз. Створення гнучких варіацій Inter та Intra-ISA для порівняння з родиною RISC-V допомагає в аналізі розширень безпеки. RISC-V IP реалізовано за допомогою інструмента Vivado для плати PYNQ, що підтримує «Overlay» – міст між спроектованим IP та FPGA. GLIFT інтегровано з IP для відстеження шляху даних безпеки-критичних модулів.

На рис. 4.7 показано результат виявлення моделлю ненадійного вентилю в підмодулі m2 схеми C432.

```
Enter the module no:m2
Information flow tracking for module: Priority B


WHEN INPUTS ARE TRUSTED
Output of module2 when the inputs are 110 is :
 10
Output of all the gates in the module is:
 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0


WHEN INPUTS ARE NOT TRUSTED
Output of all the gates using shadow logic is:
 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

EXECUTING THE MODULE
Output of module2 when the inputs are 110 is:
 00

Fault detected at module: nand
None
```

Рис. 4.7. Вентильна логіка виявляє ненадійний шлюз (NAND) пріоритету підмодуля схеми C432

Схема розділена на підмодулі, з яких для тестування обраний модуль 2, що реалізує енкодер пріоритету B. Енкодер вибрано для відстеження з ненадійних входів, і модель виявила помилку на вентилі NAND, застосовуючи бібліотеки тіньової логіки, та знаючи послідовності входів. Вона класифікує вентилі, використовуючи бібліотеку, і вказує на ненадійний елемент. Аналогічно, з різними наборами входів модель виявляє ненадійні вентилі та відстежує їх впродовж виконання для виявлення витоку критичних даних та модифікації вхідних даних, викликаючи функцію безпеки, якщо виявлені шкідливі дані.

Підсумки аналізу часу виявлення ненадійних вентилів та ефекивності використання ресурсів розробленою моделлю наведені в таблицях 4.3 та 4.4 відповідно.

Таблиця 4.3

Час виявлення загроз моделлю IFT

Тестовий набір	Кількість входів	Кількість виходів	Кількість вентилів	Час виконання тіньової логіки
C432	36	7	160	186
S398	3	6	119	177
Adder	256	129	2162	556
Alu+crtl	7	26	306	373
Memory-crtl (submodule)	1204	1231	8956	984

Таблиця 4.4

Використання ресурсів для модуля GLIFT

Архітектура	LUT	FF	LUTRAM	BBRAM
Чипові системи з інтеграцією моделі IFT	53 200	106 400	17 400	140
Накладні витрати лише на модель IFT	4506	4753	188	16
Відсоткові накладні витрати інтеграції IFT	8,5 %	4,5 %	1,1%	11%

Архітектура RISC-V, що базується на GLIFT, використовує лише на 4506 таблиць пошуку, 4753 елементи FF, 188 LUTRAM та 16 BBRAM більше, порівняно з базовою комплектацією інтегрованого RISC-V. Загальна кількість покритих IFT-інтегрованих шляхів даних та контрольних шляхів оптимізована, оскільки критичні шляхи безпеки визначені для детального IFT, а інші шляхи контролюються на рівні мікроархітектури. Ця інтегрована схема IFT знижує накладні витрати на площу без шкоди для контролю над шляхами даних безпеки.

4.3.3 Проведення симуляції

Розроблений фреймворк IFT тестувався за допомогою симулятора Spike з реалізацією атаки переповнення буфера. Нові інструкції використовуються разом з операціями над стеком, де зберігається адреса повернення. Для адреси повернення та даних, що зберігаються в стеці, призначаються тегові біти. При відстеженні невідповідності, викликається переривання, і виконання програми

припиняється шляхом нейтралізації атаки переповнення буфера. Додана інструкція SDTCHECK використовується для призначення тегового біту для адреси повернення, а LDTCHECK використовується для перевірки тегу, що відповідає адресі повернення в кеші. Пропонована модель захищає стек, використовуючи впроваджені інструкції, та, забороняючи завантаження в стек нової зворотньої адреси, що була піддана зловмисному впливу.

В лістингу 4.2 викладений фрагмент коду (мовою С), що імітує атаку переповнення буфера з помилкою сегментації, в якому вразлива функція має аргумент str1, переданий до буфера розміру 5 за допомогою копіювання рядка. Це призведе до помилки сегментації при перевищенні виділеної адреси пам'яті. Проте, якщо існуватиме можливість обійти дані обмеження, зловмисник може отримати доступ до обмежених ділянок пам'яті, зокрема до адреси повернення. Навіть, якщо переривання при помилці сегментації призведе до завершення виконання програми та унеможливить доступ до захищених ділянок пам'яті, що зберігаються в стеці, це все ще може спричинити витік критично важливої інформації.

Лістинг 4.2

```
void target() {
    printf("Buffer overflow successfully occured");
    printf("\n");
    exit(0);
}

void vulnerable(char* str1) {
    char buf[5];
    strcpy(buf, str1);
}

int main() {
    vulnerable("abcdefghijklmnopqr");
    printf("This only prints in normal control flow execution");
}
```

В момент, коли помилку сегментації пройдено, типова програма виконується з дотриманням принципів стандартної архітектури RISC-V, успішно

протидіє атаці переповнення буфера, проводячи зміну зворотньої адреси.

```
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ riscv64-unknown-elf-gcc main.c -o segmentation
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ spike pk segmentation
bbl loader
z  0000000000000000 ra 0000000000007271 sp 000000007f7e9b40 gp 000000000001eca8
tp 0000000000000000 t0 0000000000010268 t1 000000000000000f t2 0000000000000000
s0 706f6e6d6c6b6a69 s1 0000000000000000 a0 000000007f7e9b28 a1 000000000001c480
a2 000000007f7e9b38 a3 0000000000000000 a4 0000000000000072 a5 0000000000000000
a6 ffffffffffffffff a7 0000000000000000 s2 0000000000000000 s3 0000000000000000
s4 0000000000000000 s5 0000000000000000 s6 0000000000000000 s7 0000000000000000
s8 0000000000000000 s9 0000000000000000 sA 0000000000000000 sB 0000000000000000
t3 0000000000000000 t4 0000000000000000 t5 0000000000000000 t6 0000000000000000
pc 0000000000007270 va 0000000000007270 insn       ffffffff sr 8000000200046020
User fetch segfault @ 0x0000000000007270
```

Рис. 4.8. Реакція системи на загрозу атаки переповнення буферу

Такий сценарій призведе до атаки на пам'ять і може бути нейтралізований за допомогою запропонованої моделі IFT, яка викличе переривання для завершення процесу. Модель виявляє модифікацію адреси повернення і припиняє виконання програми, викликаючи виняток для усунення атаки переповнення буфера, як показано на рисунку 4.9.

```
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ riscv64-unknown-elf-gcc -o nonIFT main.c
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ spike pk nonIFT
bbl loader
Buffer overflow successfully occured
```

Рис. 4.9. Атака переповнення буфера в звичайній архітектурі RISC-V

```
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ spike pk IFT
bbl loader
z  0000000000000000 ra 0000000000010150 sp 000000007f7e9b10 gp 000000000001eca8
tp 0000000000000000 t0 000000000001026a t1 000000000000000f t2 0000000000000000
s0 000000007f7e9b40 s1 0000000000000000 a0 000000007f7e9b28 a1 000000000001c480
a2 000000007f7e9b38 a3 0000000000000001 a4 0000000000000000 a5 0000000000000000
a6 ffffffffffffffff a7 0000000000000000 s2 0000000000000000 s3 0000000000000000
s4 0000000000000000 s5 0000000000000000 s6 0000000000000000 s7 0000000000000000
s8 0000000000000000 s9 0000000000000000 sA 0000000000000000 sB 0000000000000000
t3 0000000000000000 t4 0000000000000000 t5 0000000000000000 t6 0000000000000000
pc 0000000000010186 va 0000000002011457 insn       02011457 sr 8000000200046020
Bufferflow Exception by IFT
```

Рис. 4.10. Модуль IFT, що усуває атаку через переривання

Рисунок 4.9 демонструє виконану атаку переповнення буфера на RISC-V, натомість на рисунку 4.10 показане успішне відбиття атаки, за допомогою модуля IFT, який викликає переривання при зміні адреси повернення.

4.3.4 Аналіз безпекових можливостей

6. Аналіз безпеки

Розширення безпеки, запропоновані в рамках моделі відстеження інформаційних потоків призначена для усування атак, що базуються на використанні пам’яті.

Модель RISC-V є вразливою до атак, таких як переповнення буфера, атаки на адресу повернення, ін’єкції помилок тощо. Зловмисник може змінити біти або адресу повернення програми, що призводить до компрометації даних. Потоки критично важливих даних аналізуються за допомогою модуля тегів, який забезпечує захист цілісності даних під час виконання програми. Покращена модель RISC-V запобігає цим вразливостям завдяки багаторівневому відстеженню потоків інформації, що дозволяє виявити та усунути загрозу на ранніх етапах, тобто до того, як шкідливі інструкції чи дані зможуть вплинути на вихідні дані або керування потоками.

Перехоплення керування потоком у критично важливих модулях, що призводить до витоку даних або модифікації IP-ядер, є важливим аспектом підримки безпеки. Це завдання вирішується на рівні логіки вентилів за допомогою IFT, яка обмежує інтеграцію лише критичними шляхами, забезпечуючи мінімальне перевантаження порівняно зі схемами, що підтримують повне охоплення. Такий підхід забезпечує точність відстеження інформаційних потоків через усі часові канали.

Модель симуляції з розширенням інструментального ланцюжка підтримує налаштування користувацьких ISA для додавання нових політик безпеки. Це дає змогу проектувати та перевіряти розширення безпеки процесора RISC-V. Модель симуляції прискорює перевірку безпеки від рівня архітектури до рівня компілятора, що робить процес верифікації швидшим і сприяє інтеграції.

Запропонована модель захищає систему від витоку даних за допомогою gate-level логіки вентилів і забезпечує цілісність потоків даних; з високою точністю усуває атаки, пов’язані з пошкодженням пам’яті. Таким чином, модель забезпечує безпеку на різних рівнях абстракції.

Гнучка архітектура дозволяє інтегрувати еталонну модель Common Evaluation Platform (CEP) для захисту системи під час виконання. Це сприяє подальшому вдосконаленню сучасних методів перевірки безпеки платформи RISC-V і підтримує дослідження в галузі апаратної безпеки, такі як аналіз побічних каналів.

4.4 Перспективи застосування моделі відстеження потоків

Поточний прототип моделі IFT зосереджений на модифікації шляхів даних мікроархітектурного рівня. Він забезпечує багаторівневу модель, яка демонструє гнучкість використання різних варіантів архітектури RISC-V для вивчення моделей загроз і аналізу розширень безпеки. Пропонована модель може використовуватись при реалізації простих застосунків з акцентом на зменшенні виробничої площі процесора та ефективнішого використання ресурсів. Модель забезпечує ряд механізмів безпеки, таких як захист стеку, з одночасною профілактикою витоку інформації.

Майбутні дослідження можуть бути пов’язані з інтеграцією моделі IFT з об’ємними обчислювальними програмами, застосуванням тестових еталонів для оцінки її коректності та ефективності.

Потенційні вразливі місця моделі наступні:

- таблиця тегів: дві нові інструкції, які використовуються для механізму тегів, залежать від таблиці, яку можуть змінювати різні програмні атаки;
- експлуатація купи: модель приділяє увагу лише захисту стеку, тоді як переповнення в динамічно виділеній пам’яті (купі) може використовуватися для перезапису даних;
- механізм перевірки: механізми IFT у режимі виконання можуть відстежувати потоки інформації реального часу, проте це вимагає більш високого перевантаження площі процесора та зниження продуктивності, баланс між точністю перевіркою та перевантаженням залишається викликом;

- перевищене поширення тегів: поширення тегів може призводити до високої кількості хибних позитивних результатів, оскільки теги можуть забруднювати інші дані під час виконання.

Подальші дослідження і вдосконалення моделі IFT забезпечить більш надійний і ефективний підхід до покращень безпеки та зниження потенційних загроз.

4.5 Висновки до розділу

В зазначеному розділі проведено математичний опис безпекових вимог апаратного забезпечення, спираючись на прогнозовану поведінку якого, було розроблено та реалізовано мікроархітектурну модель відстеження інформаційних потоків. Пропонована технологія дозволяє виявляти та переривати програмні атаки на пам’ять процесора.

Здійснено розробку архітектурної схеми системи на базі ядра RISC-V, деталізовано структуру потоків моделі на рівні тіньової логіки. Тестування охоплювало як аналіз архітектурного та gate-level рівнів, так і проведення симуляції за допомогою програмного засобу Vivado. Визначено перспективи застосування моделі та подальші напрямки досліджень.

Запропонована концепція підвищуватиме захищеність системи від витоку даних, забезпечуватиме цілісність їх потоків та усуватиме атаки пошкодження пам’яті.

ВИСНОВКИ

В ході виконання магістерської роботи було проаналізовано основні концепції сучасних мікроархітектур комп’ютерних систем. Провідним аспектом сучасної мікроархітектури виступають механізми прогнозування розгалужень, що значно підвищують продуктивність процесора. Зі свого боку, криптографічні інструкції, такі як AES-NI, реалізовані безпосередньо на рівні мікроархітектури, також підтримують апаратне прискорення операцій.

Огляд типології безпекових загроз виявив ланцюг викликів, з якими стикаються розробники апаратного й програмного забезпечення. Атаки на конфіденційність даних стали одними з ключових кіберзагроз. Вони дозволяють отримувати доступ до прихованої інформації та обходити механізми автентифікації. Існуючі мікроархітектурні моделі протидії загрозам концентруються на створенні захищених середовищ обробки даних, що гарантують сталість системи, яка піддається атакам.

Наступний етап роботи полягав у дослідженні сучасних методологій керування потоками інструкції процесора. Стовпами сучасної галузі кіберзахисту залишаються теорія CFI та архітектура RISC.

Політика цілісності потоку керування вимагає виконання тільки інструкцій, які є частиною заздалегідь визначеного графа. Ця умова забезпечує захист, навіть при повному контролі над пам’яттю жертви. Передова концепція μCFI гарантує, що кожна ICT-інструкція має тільки одну ціль.

Підходи RISC ґрунтуються на принципі зменшення необхідної кількості часто використовуваних інструкцій, що дозволяє процесорному конвеєру виконувати їх за один цикл. Доступ до пам’яті здійснюється тільки через операцій завантаження/збереження. Висока надійність стандарту зумовила вибір RISC ядра як апаратної бази розроблюваної моделі.

Підготовчою стадією до розробки нової моделі керування потоками стало дослідження застосовуваних безпечних процесорів, найбільш проблемних атак на процесори RISC та формулювання вимог до пропонованого рішення.

Хоча більшість RISC-V процесорів останніх поколінь успішно протидіють усталеним і поширеним атакам на пам’ять – «Flush+Reload», «Flush+Flush», «Evict+Reload», «Prime+Probe», вони не можуть забезпечити захист від атак «другого рівня», що були удосконалені – «Cache+Time», «Flush+Fault», «CycleDrift». Виходячи з висновку, що атаки на пам’ять через побічні канали є основною загрозливою тенденцією процесорів RISC, сформовано вимоги до розроблюваної моделі відстеження потоків даних.

Запропонована техніка захищає цілісність системи шляхом відстеження потоків із ненадійних каналів зв’язку та викликає переривання при виявленні атаки. Ядро процесора розширене на один теговий біт. Доданий модуль тегів здійснює перевірку адрес інструкцій завантаження/збереження. Потенційно шкідливі канали безпеки ідентифікуються у відповідності до заданих безпекових політик.

Оцінювання ефективності моделі проводилось шляхом багатьох різнотипових симуляцій. Тестування показало успішне перехоплення тестових атак. Пропонована концепція IFT є на незначну величину ресурсно ефективнішою за моделі CRAS, а охоплення рівня тіньової логіки збільшує використовувану площу процесора на 1,1 % – 11 % для різних видів обчислювальних елементів.

Нова модель відстеження потоків забезпечує низку механізмів безпеки з одночасною профілактикою витоку даних. Майбутні дослідження можуть концентруватись на спробах інтеграції розробленої моделі з об’ємними обчислювальними програмами; намагатись застосовувати тестові еталони для оцінки коректності та ефективності моделі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mike McLean Cyberattack statistics 2024. Embrocer.com. URL: embroker.com/blog/cyber-attack-statistics/.
2. Venkataramani, G.; Doudalis, I.; Solihin, Y.; Prvulovic, M. FlexiTaint: A programmable accelerator for dynamic taint propagation. In Proceedings of the 2008 IEEE 14th International Symposium on High Performance Computer Architecture, Lake City, UT, USA,16–20 February 2008; pp. 173–184.
3. Andrew Ferraiuolo, Mark Zhao, Andrew C. Myers, and G. Edward Suh. 2018. HyperFlow: A Processor Architecture for Nonmalleable, Timing-Safe Information Flow Security. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (Toronto, Canada) (CCS '18). Association for Computing Machinery, New York, NY, USA, 1583–1600.
4. Андрощук, Г. О. Технологічна безпека: прогнозні оцінки трендів у розвитку науки і техніки // Науково-технічна інформація. — 2022. — № 1. — С. 5–12.
5. Бондаренко, С. В. Методи та системи виявлення несанкціонованого доступу в сучасних інформаційно-комунікаційних системах // Збірник наукових праць. — 2011. — № 2. — С. 45–52.
6. Василенко, І. М. Комплексні системи захисту інформації: підходи та перспективи // Вісник Вінницького національного технічного університету. — 2018. — № 3. — С. 118–125.
7. Гончаренко, О. П. Захист програмного забезпечення та обладнання: сучасні виклики // Інформаційна безпека. — 2012. — № 1. — С. 23–30.
8. Іванов, П. О. Комплексний захист інформації в умовах сучасних кіберзагроз // Сучасний захист інформації. — 2020. — № 4. — С. 10–18.
9. Ковальчук, А. В. Методи захисту від атак сторонніх каналів у сучасних процесорних архітектурах // Журнал інформаційної безпеки. — 2020. — № 2. — С. 33–40.
10. Дмитренко, В. О. Аналіз сучасних наукових публікацій з тематики кібербезпеки IoT технологій // Науковий вісник. — 2023. — № 2. — С. 15–22.

11. Козак, О. Г. Аналіз ефективності механізмів рандомізації кешу для запобігання атакам типу «Flush+Reload» // Вісник Національного технічного університету України «КПІ». Серія: Інформатика та обчислювальна техніка. — 2021. — № 1. — С. 27–34.
12. Зайцев, Д. М. Рекомендації щодо розробки моделі порушника інформаційної безпеки // Інформаційна безпека. — 2018. — № 2. — С. 45–52.
13. Кравець, І. П. Використання алгоритмів глибокого навчання для виявлення мікроархітектурних вразливостей // Журнал кібербезпеки та інформаційних технологій. — 2019. — № 4. — С. 56–63.
14. Назаренко, В. Г. Вплив атак сторонніх каналів на продуктивність процесорів та методи їх нейтралізації // Вісник Дніпровського національного університету. Серія: Інформатика. — 2021. — № 3. — С. 37–44.
15. Мельник, С. М. Інноваційні підходи до тестування безпеки мікроархітектурних систем // Технічні науки та технології. — 2020. — № 4. — С. 29–36.
16. Омельченко, Л. П. Захист мікроархітектурних компонентів від атак типу «Spectre» та «Meltdown» // Комп’ютерні науки та інформаційні технології. — 2019. — № 2. — С. 61–68.
17. Петров, О. В. Методи захисту інформації від витоку через побічні електромагнітні випромінювання // Вісник Харківського національного університету радіоелектроніки. — 2020. — № 2. — С. 55–62.
18. Шевченко, М. І. Адаптивні методи управління ресурсами для підвищення безпеки мікроархітектурних систем // Системний аналіз та інформаційні технології. — 2021. — № 3. — С. 27–34.
19. Полотай, О. І., Федінець, Н. І., Кухарська, Н. П. Дослідження загроз інформаційної безпеки та способів їх вирішення в комп’ютерних мережах на канальному рівні // Науково-технічна інформація. — 2023. — № 1 — С. 5 -12.
20. Ракобовчук, Л. М., Дзяний, Н. Р., Антоневич, М. С. Дослідження ефективності захисних покриттів для запобігання витоку акустичної

інформації // Кібербезпека та захист критичної інформаційної інфраструктури. — 2022. — № 2. — С. 33–40.

21. Сорока, В. М. Інтеграція машинного навчання для підвищення безпеки мікроархітектурних компонентів // Комп’ютерні технології та кібербезпека. — 2021. — № 3. — С. 39–46.
22. Хоменко, О. В. Інноваційні підходи до тестування безпеки мікроархітектурних систем // Технічні науки та технології. — 2021. — № 4. — С. 33–40.
23. M. Dorojevets, C. L. Ayala, and A. K. Kasperek, “Data-flow microarchitecture for wide datapath RSFQ processors: Design study,” IEEE Trans. Appl. Supercond., vol. 21, no. 3, pp. 787–791, Jun. 2011.
24. D. Kim, M. Chaudhuri and M. Heinrich. Leveraging Cache Coherence in Active Memory Systems. In Proceedings of the 16th ACM International Conference on Supercomputing, 2002.
25. Benítez, Domingo, Juan Carlos Moure, Dolores Isabel Rexachs, and Emilio Luque. “Performance and Power Evaluation of an Intelligently Adaptive Data Cache.” Lecture Notes in Computer Science, 2005, 363–75.
26. Hebbar, Ranjan. (2018). SPEC CPU2017: Performance, energy and event characterization on modern processors. University of Alabama, Huntsville, 2018.
27. Смирнов, М. С. Захист від атак сторонніх каналів у багатоядерних процесорах // Вісник Національного технічного університету України «КПІ». Серія: Інформатика та обчислювальна техніка. — 2020. — № 2. — С. 47–54.
28. Жуков, А. І. Проблеми безпеки та перспективи розвитку сучасних процесорів // Матеріали науково-методичної конференції. — 2018. — С. 21–28.
29. Кириленко, М. С. Інтеграція машинного навчання для підвищення безпеки мікроархітектурних систем // Комп’ютерні технології та кібербезпека. — 2019. — № 3. — С. 45–52.
30. Євтушенко, М. С. Засоби захисту інформації від витоку акустичними каналами // Технічні науки та технології. — 2018. — № 4. — С. 91–103.

31. Щербак, А. С. Використання штучного інтелекту для виявлення мікроархітектурних вразливостей // Журнал кібербезпеки. — 2019. — № 5. — С. 52–59.
32. Юрченко, О. М. Оптимізація енергоспоживання в умовах застосування механізмів захисту від атак сторонніх каналів // Радіоелектронні та комп’ютерні системи. — 2020. — № 2. — С. 42–49.
33. Avaneesh Deleep, 2024. Augmenting Dynamic Branch Predictors with Static Transformer Guided Clustering. Imperial College London, Department of Computing.
34. Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. 2006. Predicting secret keys via branch prediction. In Topics in Cryptology–CT-RSA 2007: The Cryptographers’ Track at the RSA Conference 2007, San Francisco, CA, USA,February 5-9, 2007. Proceedings. Springer, 225–242.
35. Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2009. Control-flow integrity principles, implementations, and applications. ACM Trans. Inf. Syst. Secur. 13, 1, Article 4 (October 2009), 40 p.
36. Sadullah Canakci, Leila Delshadtehrani, Boyou Zhou, Ajay Joshi, and Manuel Egele. Efficient context-sensitive cfi enforcement through a hardware monitor. In Proc. DIMVA, 2020
37. Oklobdzija, V. G., 1999. Reduced Instruction Set Computers. University of California, Department of Electrical and Computer Engineering, Berkeley.
38. Malti Bansal and Harsh, “Reduced Instruction Set Computer (RISC): A Survey”, 2021 J. Phys.: Conf. Ser. 1916 012040 pp. 1-14.
39. S. Sau, J. Haj-Yahya, M. Wong, K. Lam, and A. Chattopadhyay, “Survey of secure processors,” in 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Jul. 2017, pp. 253–260.
40. Lukas Gerlach et al. "A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs." 2023 IEEE Symposium on Security and Privacy (SP) (2023): 2321-2338. doi.org/10.1109/SP46215.2023.10179399.