ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

« Побудова завадостійкої обчислювальної архітектури»

« Construction of an interference-resistant computing architecture»

Виконала: здобувачка заочної форми навчання

спеціальності 123 Комп'ютерна інженерія

(код, назва спеціальності)

Освітня програма ОП – Комп'ютерна інженерія

(назва)

Стокич Марина Олегівна

(прізвище, ім'я, по-батькові здобувача)

Керівник канд. ф-м. н., доц. Шугайло Ю.Б. _____

(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент д.т.н., проф. Гунченко Ю.О. _____

(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2024 р.

Завідувач кафедри
_____ Юрій ГУНЧЕНКО
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № _____
Протокол № __ від «___» _____ 2024 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК
_____ Алла КОБОЗЄВА
(підпис) (ім'я, прізвище)

Одеса - 2024

АНОТАЦІЯ

Цифрові процесори та обчислювальні системи постійно поліпшуються вже протягом багатьох років, вони отримують більшу обчислювальну потужність. Це стає можливим за рахунок зменшенню розмірів компонентів. Використання пристроїв такої складності вимагає низької напруги живлення, що збільшило дію фонових шумів, різноманітних завад та перешкод на ці пристрої. Питання надійності обчислювальних систем на сьогодні є актуальною задачею, особливо з урахуванням війни та широкому використанню систем радіоелектронної боротьби (РЕБ).

Системи з підвищеною надійністю вкрай потрібні у багатьох галузях, таких як військова, медична, космічна тощо. Відповідно, тема кваліфікаційної роботи – «Побудова завадостійкої обчислювальної архітектури», є **актуальною.** В роботі розглядаються питання побудови завадостійкої обчислювальної архітектури. Актуальність теми визначається зростаючими вимогами до надійності та безпеки обчислювальних систем. З одного боку, це пов'язано з поширенням критичних інформаційних систем, від яких залежить життя людей і функціонування цілих галузей економіки. З іншого боку, зростає складність та інтегрованість обчислювальних систем, що робить їх більш вразливими до збоїв та атак.

Метою даної роботи є підвищення завадостійкості обчислювальних систем.

Відповідно до поставленої мети в роботі виконані наступні завдання:

1) вивчити методи підвищення перешкодостійкості до відмови обчислювальних систем;

2) огляд існуючих завадостійких архітектур;

3) вибір кількох існуючих архітектур та їх детальний аналіз;

4) модифікувати вибрані архітектури для підвищення швидкодії;

5) запропонувати нову архітектуру, яка є комбінацією розглянутих;

6) обчислення показників кожної архітектури та їх порівняння;

7) аналіз отриманих результатів.

Об'єктом дослідження є обчислювальні архітектури.

Предметом дослідження є принципи будови завадостійких обчислювальних архітектур.

В роботі застосовані методи моделювання, математичні методи та екпериментальне дослідження.

ANNOTATION

Digital processors and computing systems have been continuously improving for many years, gaining greater computational power. This is made possible by the reduction in component sizes. The use of such complex devices requires low supply voltage, which increases the impact of background noise, various interferences, and disruptions on these devices. The issue of reliability of computing systems is currently a relevant task, especially considering the ongoing war and the widespread use of electronic warfare systems (EWS).

Highly reliable systems are critically needed in many fields, such as military, medical, space, and others. Accordingly, the topic of this qualification work – "Building a Noise-Resistant Computing Architecture" – is **relevant**. The work addresses the construction of a noise-resistant computing architecture. The relevance of the topic is determined by the growing requirements for the reliability and security of computing systems. On one hand, this is due to the proliferation of critical information systems, on which people's lives and the functioning of entire economic sectors depend. On the other hand, the complexity and integration of computing systems are increasing, making them more vulnerable to failures and attacks.

The aim of this work is to enhance the noise resistance of computing systems.

In accordance with the stated goal, the following tasks were performed in the work:

1) to study methods of increasing noise resistance to failures of computing systems;

2) to review existing noise-resistant architectures;

3) to select several existing architectures and perform their detailed analysis;

4) to modify the selected architectures to increase performance;

5) to propose a new architecture that combines the reviewed ones;

6) to calculate the indicators of each architecture and compare them;

7) to analyze the obtained results.

The object of the study is computing architectures.

The subject of the study is the principles of constructing noise-resistant computing architectures.

The work uses modeling methods, mathematical methods, and experimental research.

ЗМІСТ

ВСТУП

У сучасних обчислювальних системах часто гостро стоїть проблема надійності. При вирішенні цієї проблеми в основному розглядається три типи завдань:

1) підвищення надійності зберігання даних;

2) підвищення надійності передачі даних;

3) підвищення надійності обчислень (достовірність отриманих даних).

Завдання першого і другого типів спрямовані на виявлення і, можливо, усунення помилок, що виникають при передачі або зберіганні даних, і в основі своїй мають математичний або фізичний характер. Для їх вирішення існує безліч різних способів: використання контролю (біта) парності, кодів Хеммінга Хаффмана та інших, складніших кодів. Більш детально дані проблеми розглядаються в теорії кодування та радіоелектроніки.

Рішення завдань другого типу полягає у змінах архітектури обчислювальних пристроїв та самої системи в цілому.

Необхідність підвищення надійності обчислень найбільше відчувається при організації розподілених обчислень, а також побудові розподілених систем управління різноманітного призначення. Це стало можливо завдяки достатньому рівню розвитку сучасних комп'ютерних мереж, що дозволяють використовувати велику кількість віддалених комп'ютерів. Існуючі підходи зі створенням систем розподіленої обробки даних у мережевих ресурсах будуються як технологічні надбудови над стандартними операційними системами. В результаті виходять багатошарові програмні комплекси, управління якими потребує складного адміністрування.

Приклади спеціалізованих систем, що дають уявлення про масштаби та сфери застосування розподілених систем обробки даних, наведені в Таблиці 1.

Таким чином, ми бачимо, що проблема підвищення надійності обчислень на даний момент є актуальною, а її рішення затребувані.

Підвищення надійності обчислень за допомогою зміни архітектури може бути непрямим (наприклад, при побудові систолічних обчислювальних систем кожен обчислювальний елемент вирішує однотипні завдання, що дозволяє спростити його структуру, а в більш простій структурі легше досягається необхідна надійність). Так і безпосередньо впливатиме на надійність обчислень.

Таблиця 1

Приклади розподілених систем

Тип системи	Опис	Приклади
Веб-системи	Системи, які доступні через Інтернет і складаються з декількох серверів, що працюють разом.	Google Search, Amazon, Facebook, Wikipedia
Системи електронної комерції	Системи, які дозволяють користувачам купувати та продавати товари та послуги в Інтернеті.	Amazon, eBay, AliExpress
Соціальні мережі	Системи, які дозволяють користувачам спілкуватися один з одним, ділитися інформацією та створювати контент.	Facebook, Twitter, Instagram
Хмарні обчислення	Системи, які надають користувачам доступ до обчислювальних ресурсів, таких як сервери, сховища даних та програмне забезпечення, через Інтернет.	Amazon Web Services, Microsoft Azure, Google Cloud Platform
Системи обробки транзакцій	Системи, які обробляють фінансові транзакції, такі як банківські перекази та покупки з кредитної картки.	VISA, MasterCard, PayPal
Системи управління ланцюжками постачання	Системи, які допомагають компаніям відстежувати та керувати своїми ланцюжками постачання.	SAP Supply Chain Management, Oracle Supply Chain Management
Системи бронювання	Системи, які дозволяють користувачам бронювати квитки на літак, готелі та інші послуги.	Expedia, Kayak, Priceline
Ігри для кількох користувачів	Ігри, в які можуть грати одночасно декілька людей, використовуючи різні комп'ютери або пристрої.	World of Warcraft, League of Legends, Fortnite
Наукові дослідження	Системи, які використовуються вченими для проведення досліджень, таких як моделювання клімату та аналіз даних.	Folding@home, SETI@home

Джерело: складено автором

1. АРХІТЕКТУРА ПРОЦЕСОРА

1.1. Структура процесора

На рисунку 1.1 представлена блокова діаграма, що показує структурну схему процесора. На ній ми бачимо блок виконання команд 101. Він з'єднаний із лічильником команд 102, файл-регістром 103, буфером запису 104, блоком генерації сигнатури 105 і блоком контролю помилок 106. Лічильник команд 102, файл-регістр 103 і сигнатура 108, 109, 110.

Файл-регістр 103 має контрольний регістр 109. Контрольний регістр 109 може зберігати поточний стан файл-регістру (створення контрольної точки) і може відновлювати значення регістра файлу з контрольної точки.

Лічильник команд 102 має контрольний регістр 108, здатний зберігати поточне значення лічильника команд та відновлювати його значення.

Блок виконання команд 101 виконує інструкції, що містяться в пам'яті 111, на які вказує лічильник команд 102. Інструкції зчитуються і змушують блок виконання команд 101 оперувати вмістом файл-регістру 103. Інструкції також можуть змусити блок виконання команд 101 зберігати дані 112 і завантажувати 113 із пам'яті 114.

Буфер запису містить 104 дані, які повинні бути записані в пам'ять. Це виконує дві функції. По-перше, буфер запису 104 звільняє процесор від зупинок під час операцій запису. По-друге, буфер запису містить 104 дані, які можуть бути використані або відкинуті у разі виникнення помилки. Зауважимо, що буфер запису 104 не може бути джерелом даних, якщо адреса даних, що завантажуються, збігається з адресою даних, записаних у буфері запису 104. Це обмеження необхідне для досягнення відмовостійкості даної архітектури.

Блок генерації сигнатур 105 приймає всі результати, обчислені блоком виконання команд 101. До них також відносяться всі завантажені дані. Спосіб обчислення сигнатури не розглядатиметься. Блок генерації сигнатур 105 зберігає

обчислену сигнатуру відповідний регістр 107. Регістр сигнатури 107 має свій контрольний регістр 110, який записується поточне значення сигнатури. Контрольна копія сигнатури 110 ніколи не записується назад у регістр сигнатур 107.

Блок контролю помилок 106 контролює запис у контрольні регістри 108, 109, 110, контролює очищення буфера запису 104 і порівнює поточну сигнатуру 107 з попередньої 110. Псевдо-код для блоку контролю помилок 106 представлений у вигляді блок-схем.

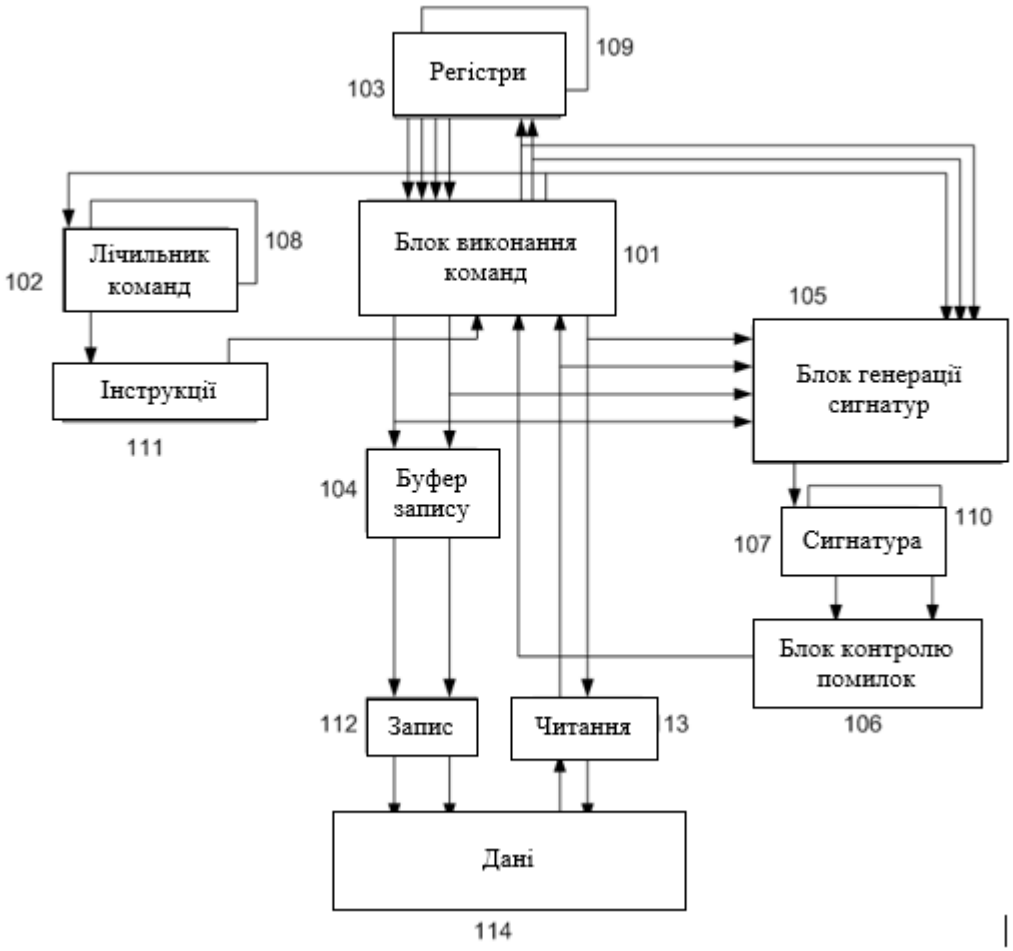

Рис. 1.1. – Структура процесора

1.2. Стан "0"

У стані "0" процесор виконує інструкції для формування сигнатури, яка буде використовуватися в наступному стані виявлення помилки.

Посилаючись на рисунки 1.1 і 1.2, на початку стану "0" відключається запис буферу запису 104 (блок 201). Буфер запису 104 міг містити дані, отримані під час попередніх команд. Під час стану "0" операції запису будуть поставлені в чергу запису буфер запису 104. Ці операції не проходять ніякої обробки після підрахунку числа таких операцій і збереження адрес і даних в блок генерації сигнатури 105.

Під час виконання блоку 202 інструкції виконуються доти, доки не буде виконано одну з двох умов. Виконання команд припиняється у разі:

а) якщо кількість дорівнює заданій постійній "N";

б) якщо лічильник у буфері запису 104 вказує, що кількість операцій запису буфер запису 104 дорівнює його розміру.

За цей час сигнатура або кілька сигнатур акумулюються на основі обчислених результатів, даних, що зберігаються, та їх адрес в регістрі сигнатури 107. Під час виконання блоку 202, якщо будь-яка інформація зберігалася в буфері запису 104, вона буде записана в пам'ять одночасно з виконанням інструкцій.

Після завершення попереднього блоку на досягненні однієї з двох умов (а) або (б), описаних вище, накопичена сигнатура копіюється з блоку сигнатури 107 контрольний регістр 110 (блок 203).

У блоці 204 блок сигнатури 107 ініціалізується, лічильник команд 102 відновлюється з контрольного регістра 108 і файл-регістр 103 відновлюється з контрольного регістра 109. До переходу в стан "1", блок 205 завершує виконання всіх операцій запису з буфера запису 104.

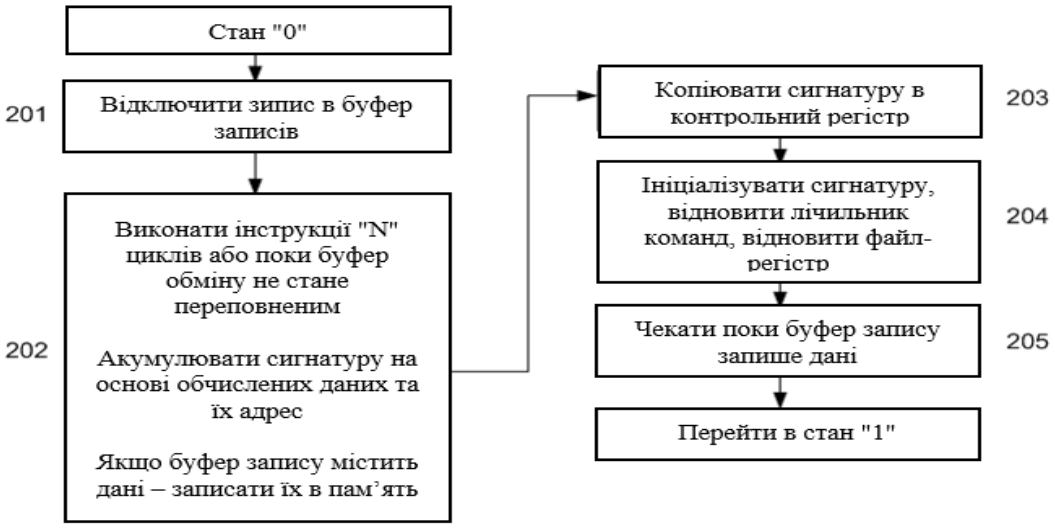
Стан "0"
201
Відключити запис в буфер записів
202
Виконати інструкції "N" циклів або поки буфер обміну не стане переповненим
Акумулювати сигнатуру на основі обчислених даних та їх адрес
Якщо буфер запису містить дані – записати їх в пам'ять
203
Копіювати сигнатуру в контрольний регістр
204
Ініціалізувати сигнатуру, відновити лічильник команд, відновити файл-регістр
205
Чекати поки буфер запису запише дані
Перейти в стан "1"

Рис. 1.2. – Стан "0"

1.3. Стан "1"

У стані "1" знову виконуються всі команди для визначення наявності помилки. На початку стану "1" блок 301 дозволяє запис у буфер запису 104. Блок 302 аналогічний блоку 202. Таким чином виконання інструкцій припиниться при досягненні однієї з двох умов:

а) якщо кількість тактів дорівнює заданій постійній "N";

б) якщо лічильник у буфері запису 104 вказує, що кількість операцій запису для зберігання буфера дорівнює його розміру.

За цей час сигнатура або кілька сигнатур акумулюються на основі обчислених результатів даних, що зберігаються, та їх адрес в регістрі сигнатури 107.

Після закінчення виконання інструкцій на виявленні однієї з двох умов (а) або (б), описаних вище, накопичена сигнатура порівнюється з контрольним регістром 110, записаним раніше блоком 203. Якщо сигнатури збігаються, тобто помилки не було, процес триває з виконання блоку 304. Процесор переходить у стан "0", копіюючи вміст файлу-регістру 103 в контрольний регістр 109, вміст лічильника команд 102 в резервний регістр 108 і ініціалізуючи регістр сигнатури 107. Після цих кроків процесор буде перебувати в стані "0". Якщо поточна сигнатура не збігається з попередньою, тоді було виявлено помилку (блок 305) і процесор переходить у стан "2". На Рис. 1.3 показано стан "1".

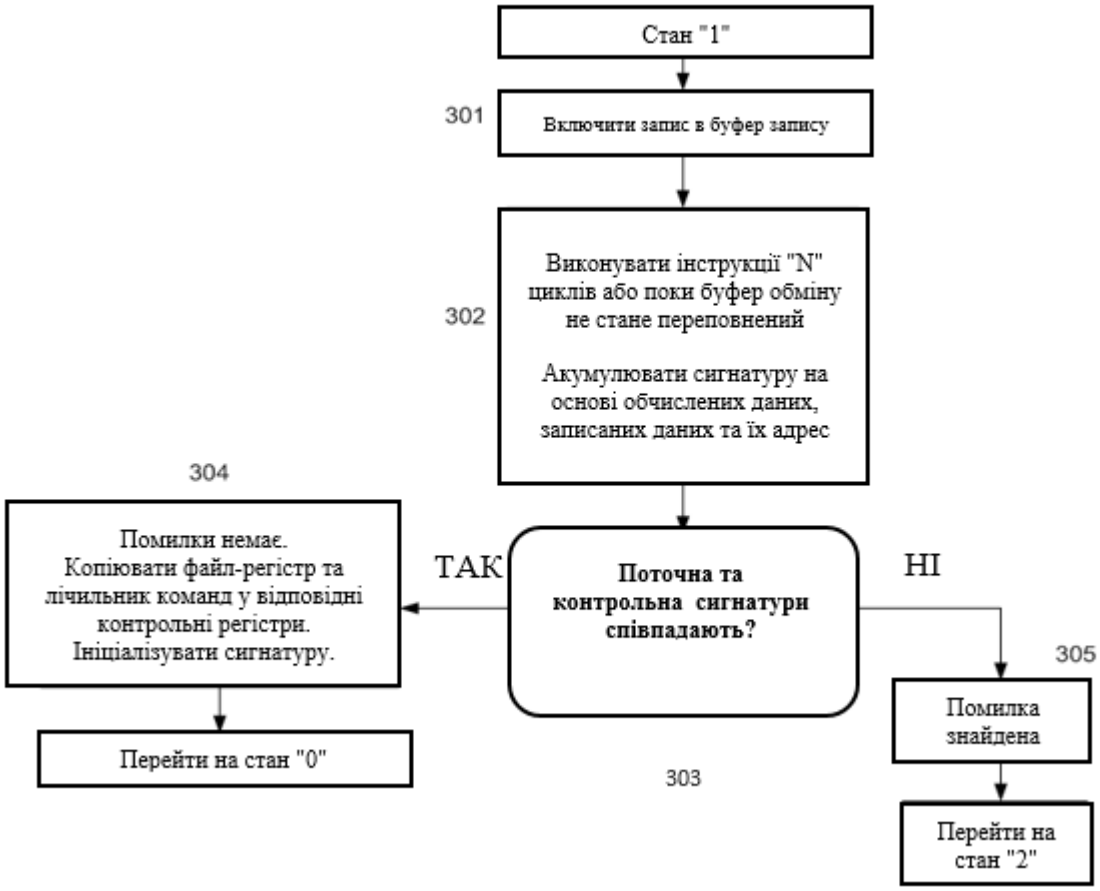

Рис 1.3. – Стан "1"

1.4. Стан "2"

Як видно з рисунку 1.4, стан "2" починається з очищення буфера запису 104 (блок 401). Це зроблено тому що була виявлена помилка та дані, що зберігаються в ньому можуть бути невірними. Далі зовнішній пристрій повідомляється про те, що була виявлена помилка (блок 402). Це повідомлення є надзвичайно корисним. Воно може бути використане для виявлення нестабільного середовища або виявлення пристрою, що наближається до відмови. Цей стан процесора ініціалізується для повторного входу в стан "0" без інформації від попереднього виконання команд у станах "0" і "1". Поточне значення лічильника команд 102 не копіюється в контрольний регістр 108 регістр (блок 403), поточне значення файл-

регістра 103 не копіюється в резервний регістр 109 (блок 404). Значення лічильника команд і файл-регістру відновлюється зі спеціальних регістрів 108 і 109 відповідно, регістр сигнатури 107 ініціалізується (блок 405). Таким чином, стан "0" буде починатися в тих же початкових умовах, що і в попередній раз. Після цього процесор перетворюється на стан "0" .

Відмовостійкий алгоритм представлений на рисунках 1.2, 1.3 та 1.4 працює наступним чином. Блок контролю помилок вимагає результати, що мають ті самі сигнатури. Вміст буфера запису 104, що залишився після попереднього періоду, вивантажується під час першого проходження наступного періоду.

Рис. 1.4. – Стан "2"

2. МОДИФІКАЦІЯ АРХІТЕКТУРИ

2.1. Модифікована архітектура

Внесемо невеликі зміни до вихідної архітектури, а саме: змінимо алгоритм у стані "2", як показано на рисунку 5.

Як видно з рисунка 2.1, відмінність від основної архітектури тільки в блоці 505 та кінцевому переході у стан "1". Таким чином, якщо перехід у стан "2" не був здійснений (не було помилки), то відмінностей між даною архітектурою та первісною не буде. Якщо ж перехід у стан "2" було здійснено (було помилку), то очищається буфер запису 104 (блок 501). Як і в блоці 401, це зроблено, тому що була виявлена помилка і дані, що зберігаються в ньому, можуть бути невірними. Далі зовнішній пристрій повідомляється про виявлення помилки (блок 502). Поточне значення лічильника команд 102 не копіюється в контрольний регістр 108 (блок 503), поточне значення файл-регістру 103 не копіюється резервний регістр 109 (блок 504). Значення лічильника команд і файл-регістру відновлюється зі спеціальних регістрів 108 і 109 відповідно, регістр сигнатури 107 копіюється в регістр 110 контрольний і ініціалізується, (блок 505). Таким чином, стан "1" буде починатися в тих же початкових умовах, що і в попередній раз за винятком того, що контрольний регістр сигнатури 110 буде зберігатися нове значення. Далі здійснюється перехід у стан "1". Як видно, внесена зміна призвела до того, що блок контролю помилок вважатиме результат правильним, якщо його сигнатура збігається із сигнатурою попереднього результату.

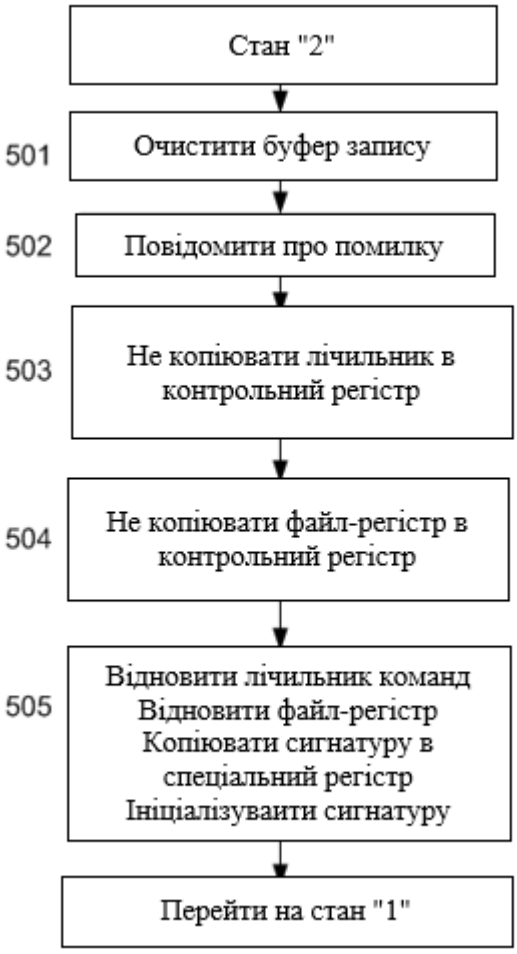

Рис. 2.1. – Стан "2"

2.2. Архітектура з використанням мажоритарного блоку

Внесемо невеликі зміни у вихідну архітектуру та назвемо отриману архітектуру "архітектура 3".

1) Змінимо структуру процесора додавши ще один контрольний регістр 615, назвемо його контрольний регістр сигнатури 2. Це показано рисунку 2.2.

2) Змінимо стани "1" і "2", як показано на рисунках 2.3 та 2.4.

На рисунку 2.3 демонструє алгоритм стану "1". Як видно з рисунка, блоки 701, 702 та 705 повністю ідентичні вихідним бокам 301, 302 та 305 відповідно. Після підрахунку сигнатури (блок 702) виконується її порівняння з сигнатурами, що зберігаються в контрольному регістрі 610 та контрольному регістрі 615. за

допомогою мажоритарного блоку (блок 703). У контрольному регістрі 610 на момент порівняння зберігатиметься сигнатура попереднього результату (внаслідок роботи блоку 203 при першому порівнянні і роботи блоку 805 при наступних порівняннях, які виникнуть тільки у разі виявлення помилки). Контрольний регістр 615 на момент першого порівняння буде ініціалізований (робота блоку 704), а при наступних порівняннях (якщо буде виявлена помилка) міститиме сигнатуру, отриману на попередньому кроці (результат роботи блоку 805).

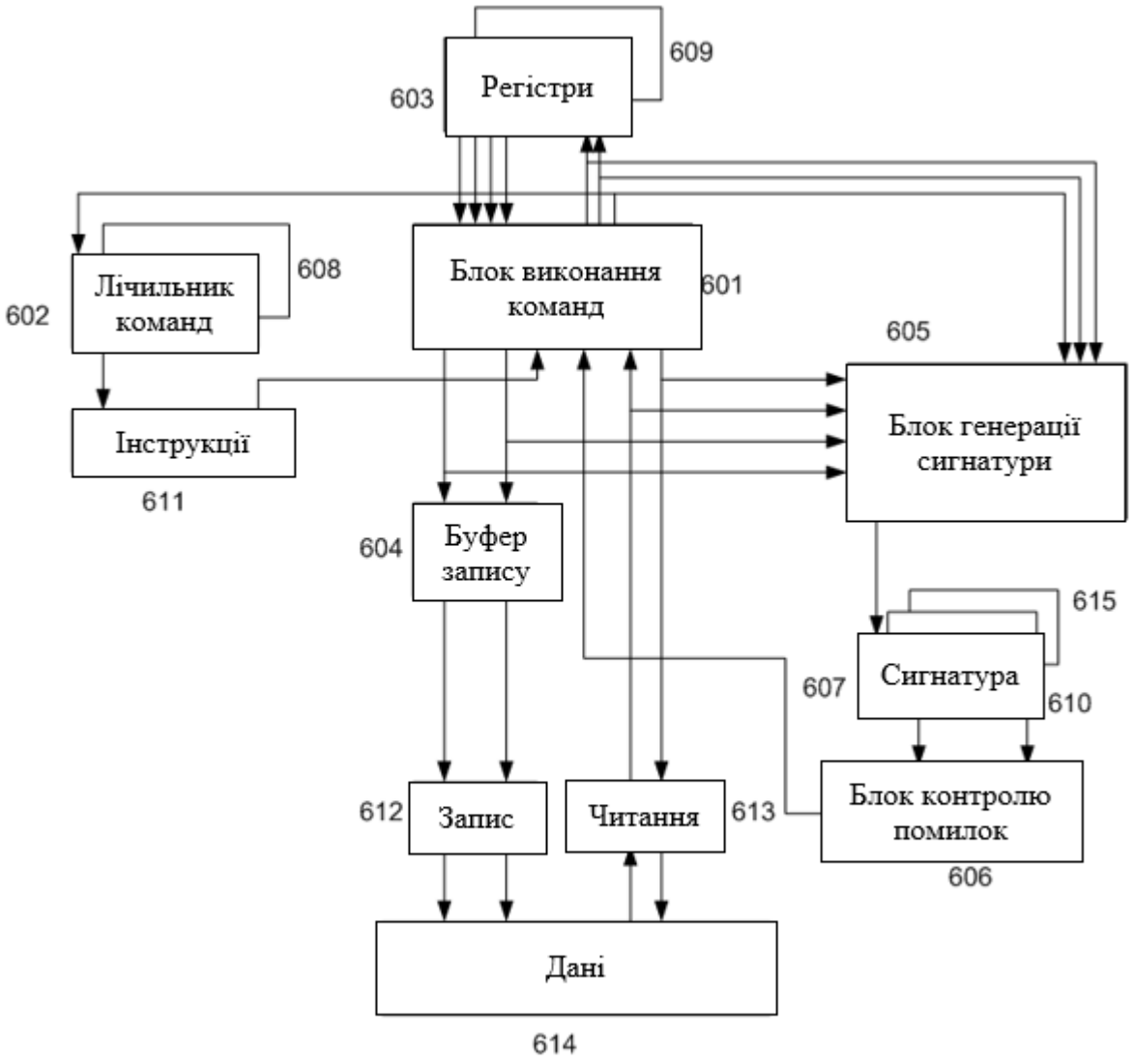

Рис. 2.2. – Структура процесора

Таким чином, блок контролю помилок вважатиме результат правильним, якщо його сигнатура збігається з однією з двох попередніх сигнатур.

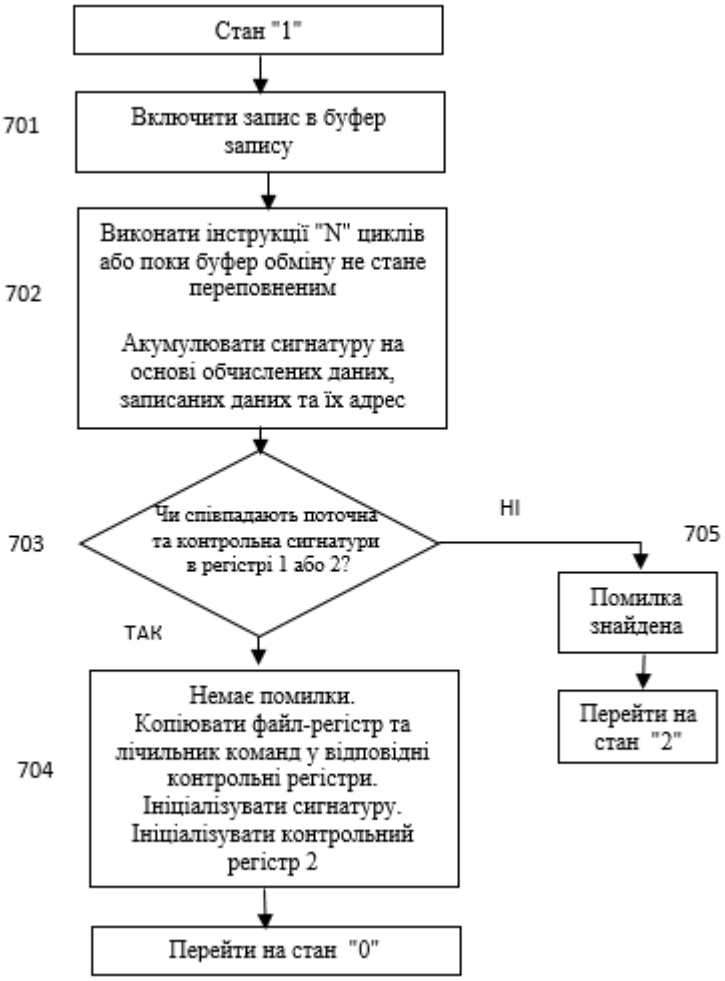

Рис. 2.3. – Стан "1"

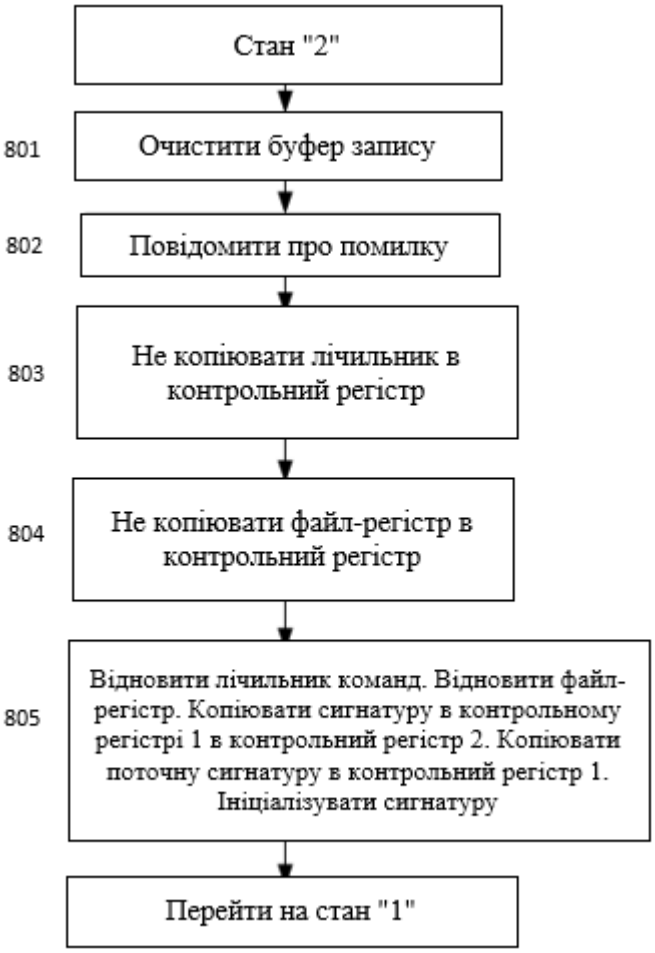
Стан "2"
801
Очистити буфер запису
802
Повідомити про помилку
803
Не копіювати лічильник в контрольний регістр
804
Не копіювати файл-регістр в контрольний регістр
805
Відновити лічильник команд. Відновити файл-регістр. Копіювати сигнатуру в контрольному регістрі 1 в контрольний регістр 2. Копіювати поточну сигнатуру в контрольний регістр 1. Ініціалізувати сигнатуру
Перейти на стан "1"

Рис. 2.4. – Стан "2"

3. АРХІТЕКТУРА З ВИКОРИСТАННЯМ ДВОХ ПРОЦЕСОРІВ

3.1. Загальна схема архітектури

Раніше було розглянуто різні варіанти архітектур обчислювальних систем з одним процесором. Проте є обчислювальні системи, до складу яких входять кілька процесорів. Такі архітектури застосовують метод FRC (functional redundancy checking) або функціональної перевірки надмірності, в якій зазвичай використовуються два процесори. Один із процесорів називається «головний», а другий – «перевірним». Обидва ці процесори отримують однакові вхідні дані і виконують одні й самі інструкції одночасно. Перевірочний процесор порівнює вихідні сигнали головного процесора зі своїми вихідними сигналами. Якщо вихідні сигнали не збігаються, процесор генерує сигнал про помилку.

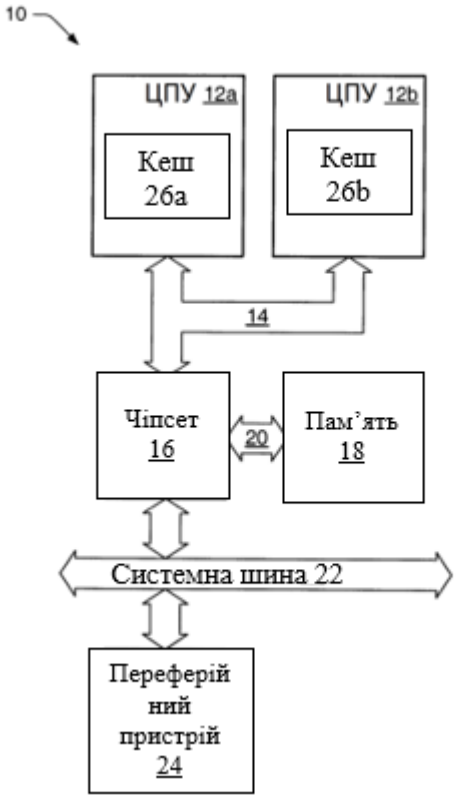

Рис. 3.1. – Архітектура з двома процесорами

На рисунку 3.1 представлено блок-схему типової архітектури з використанням FRC. Архітектура включає два ідентичні процесори 12a і 12b, міжпроцесорної шиною 14, чіпсета 16, блоку пам'яті 18, шини пам'яті 20, системної шини 22 і периферійних пристроїв 24. Процесорна шина 14 об'єднує обидва процесори один з одним, а так само. Чіпсет 16 є інтерфейсом між процесорами 12a і 12b та системною шиною 22, а також між процесорами 12a та 12b та блоком пам'яті 18. Системна шина 22 може підтримувати кілька периферійних пристроїв. Як такі пристрої можуть виступати пристрої відображення, принтер та інші. Блок пам'яті 18 виступає як сховища даних і з'єднаний з чіпсетів через шину пам'яті 20.

Під час ініціалізації системи процесор 12a або 12b налаштовується працювати як «головного», тоді як інший процесор стає «перевіреним». «Головний» процесор налаштовує свої виходи на функціонування в нормальному режимі, в той час, як "перевірочний" налаштовує свої виходи на роботу тільки в режимі читання. Після виконання дій, перевірочний процесор порівнює значення, які згенерував головний, зі значеннями, які він отримав сам. При їх невідповідності процесор перевірки генерує сигнал про помилку, який може бути повідомленням для зовнішніх пристроїв.

Головний процесор ініціює дані читання та запису. У відповідь на запит читання з пам'яті від головного процесора, чіпсет отримує дані з блоку пам'яті 18 через шину пам'яті 20 і передає ці дані на обидва процесори 12a і 12b через процесорну шину 14. Під час операції запису в пам'ять чіпсет 16 отримує дані від головного процесора і записує дані блок пам'яті 18 через шину пам'яті 20. При запиті читання їх адрес в діапазоні адрес, призначених периферійним пристроям 24, чіпсет 16 отримує дані від периферійного пристрою 24 по системній шині 22 і надає їх процесорам за допомогою процесорної шини 14. час операції запису в діапазон адрес, призначених периферійним пристроям 24, чіпсет 16 отримує дані від головного процесора процесорної шині 14 і записує їх в периферійний пристрій 24 по системній шині 22.

При використанні цієї архітектури можуть виникнути деякі проблеми. Основна проблема полягає в тому, що вихідні сигнали процесорів можуть не відображати поточний внутрішній стан процесора. Наприклад, може виникнути затримка в кілька системних тактів перш, ніж результати будуть подані на виходи, крім того процесори 12a і 12b можуть включати досить великий внутрішній кеш, в результаті чого процесори 12a і 12b можуть працювати тривалий час використовуючи дані, що зберігаються в них кеш-пам'яті 26a та 26b відповідно. У цей час будь-які помилки обчислень не відображаються на виходах процесорів 12a та 12b і, отже, не виявляються системою FRC. В результаті збільшується час, необхідний виявлення помилки.

3.2. Архітектура із використанням сигнатур

Проблеми, викладені вище, значною мірою вирішуються за допомогою системи FRC, у якій порівнюються сигнатури, що генеруються кожним процесором. Кожна сигнатура складається з відносно невеликої кількості сигналів, які є репрезентативними для внутрішнього стану кожного процесора. Така архітектура включає два процесори. Кожен процесор налаштований виконання команд і отримання вхідних сигналів. Обидва процесори однакові та виконують інструкції одночасно. Кожен процесор містить генератор сигнатури до створення сигнатури, що описує його внутрішній стан. Архітектура також включає блок порівняння отриманих сигнатур, згенерованих процесорами, який при нерівності отриманих сигнатур генерує сигнал про помилку. Кожен процесор може включати ряд функціональних блоків, у тому числі і блок шинного інтерфейсу (БШІ), який обробляє всі операції передачі даних для процесора відповідно до встановлених процедур.

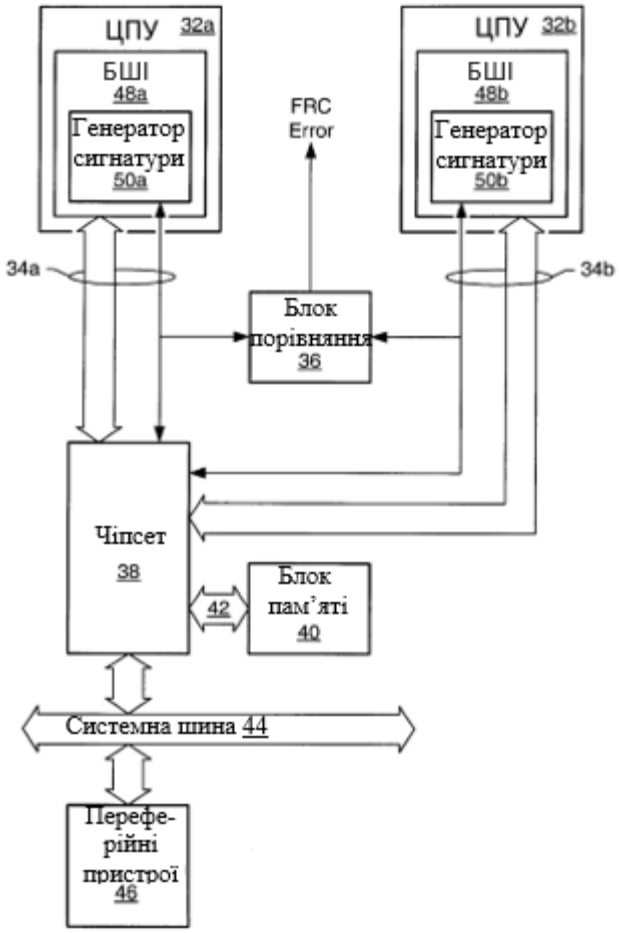

Рис. 3.2. – Схема архітектури з використанням сигнатур

На рисунку 3.2 представлено схему подібної архітектури. У неї входять два ідентичні процесори 32a і 32b, процесорні шини 34a і 34b, які з'єднують чіпсет 38 з процесорами 32a і 32b відповідно, блоку пам'яті 40, шини пам'яті 42, системної шини 44 і периферійних пристроїв 2 і процесорів 2 43. виконують інструкції одночасно. Блок порівняння 26 отримує сигнатури, згенеровані кожним процесором, і у разі їхньої нерівності генерує сигнал про помилку FRC error. Чіпсет 38 підключений до системної шини 44 і виконує функцію інтерфейсу між процесорами і системною шиною. Чипсет також пов'язаний з блоком пам'яті 40 через шину пам'яті 42 і може використовуватися як контролер пам'яті. Системна шина 44 адаптована для зв'язку з одним або декількома периферійними

пристроями. Периферійні пристрої 46 з'єднані із системною шиною 44.

Процесор 32a включає Блок шинного інтерфейсу (БШІ) 48a, процесор 32b включає свій блок шинного інтерфейсу (БШІ) 48b. Кожен БШІ обробляє всі операції передачі даних для відповідного процесора 32 відповідно до встановлених процедур. Кожен БШІ 48 отримує вхідні сигнали від входів відповідного процесора 32 та керує вихідними сигналами процесора. БШІ також включає генератор сигнатури 50, який генерує сигнатуру, що відображає внутрішній стан процесора 32.

Сигнатура включає кілька біт, число яких менше, ніж число вихідних сигналів процесора 32. Одна з реалізацій може включати, наприклад, чотири біти сигнатури. Перший біт може залежати від внутрішнього стану кеш даних, другий біт може залежати від внутрішнього стану кеш інструкцій, третій біт може залежати від внутрішнього стану буфера контролера, а четвертий може залежати від внутрішнього стану блоку обчислення з плаваючою точкою. Кожна сигнатура може бути логічною комбінацією (наприклад, що виключає АБО) сигналів, що генеруються у відповідному функціональному блоці.

4. КОМБІНОВАНА АРХІТЕКТУРА

4.1 Загальна схема архітектури

У перших аналізованих архітектурах використовувався процесор із запам'ятовуванням деякого числа попередніх сигнатур, а останніх – з кількома процесорами, але не запам'ятовування будь-яких даних. Далі буде розглянута архітектура, в якій буде два процесори, аналогічно останній архітектурі, але при цьому буде запам'ятовування попередніх сигнатур, аналогічно першим архітектурам.

Схема такої архітектури представлена рисунку 4.1.

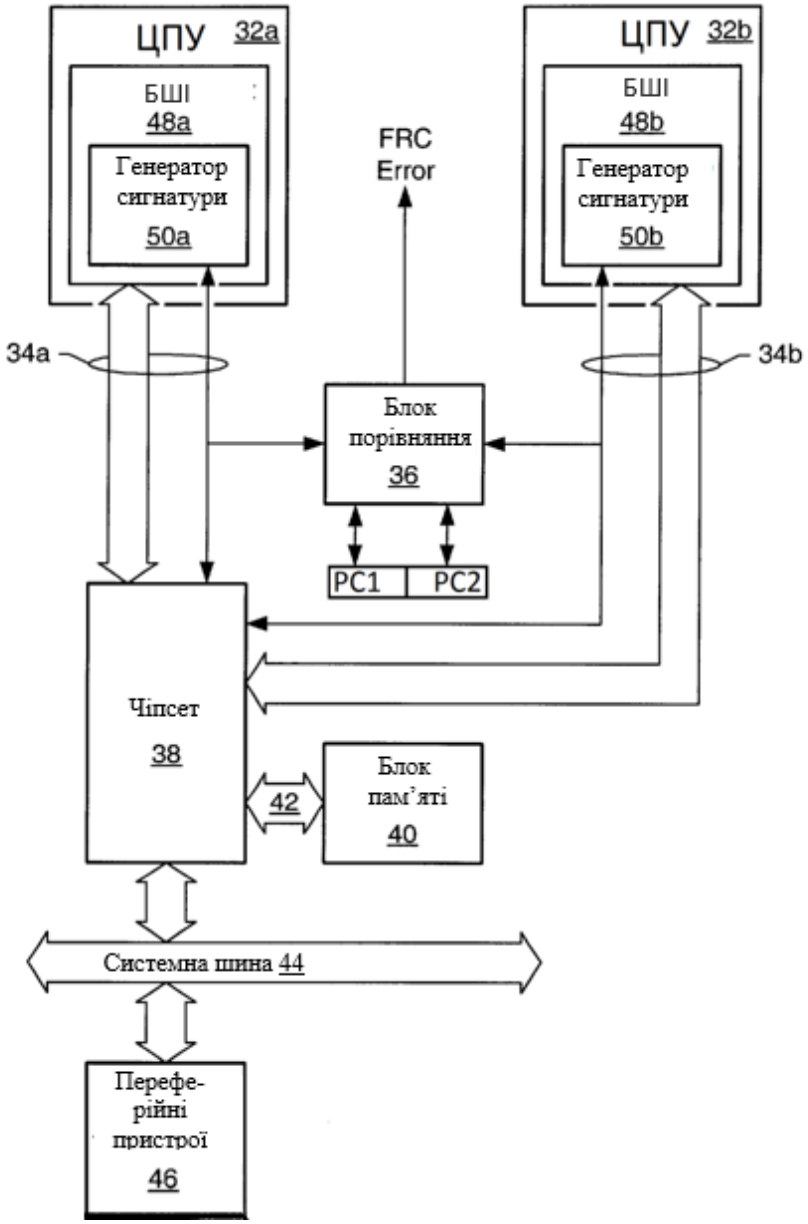

Рис. 4.1. – Схема комбінованої архітектури

Запропонована архітектура відрізняється від початкової наявністю двох регістрів сигнатури PC1 і PC2, в яких зберігаються сигнатури, що згенеровані на попередньому кроці першим і другим процесором відповідно. Також ця архітектура відрізняється поведінкою блоку порівняння. Відмінність у тому, що спочатку регістри PC1 і PC2 ініціалізуються початковими значеннями. Після того, як буде виконана команда або набір команд, згенеровані на підставі виконаних

операцій першим і другим процесором сигнатури, подаються на блок порівняння, якщо сигнатури рівні, процесор виконує наступну команду або набір команд. Якщо ж сигнатури не рівні, блок порівняння записує отримані від першого і другого процесора сигнатури в регістри PC1 і PC2 відповідно. Далі обидва процесори знову виконують команду або набір команд, після закінчення її або їх виконання блок порівняння порівнює знову отримані сигнатури з запам'ятованими. Якщо серед порівнюваних чотирьох сигнатур знайдуться хоча б дві однакові, тоді блок порівняння застосовує зміни, зроблені процесором, який видав правильний результат, заново ініціалізує регістри PC1 і PC2 і продовжує роботу, якщо ж серед порівнюваних сигнатур не знайшлося двох однакових, то тоді нові сигнатури записуються у відповідні їм регістри (від першого процесора до PC1, від другого – до PC2) і процесори заново виконують команду чи набір команд. Схема роботи такого алгоритму аналогічна до алгоритму роботи першої модифікації першого прототипу, описаного в першому підпункті другого розділу. Відмінність полягає в тому, що потрібно записувати та ініціалізувати одразу 2 сигнатури, порівнювати 4 і у разі успішного порівняння застосовувати результати лише того процесора, які видав правильний результат.

4.2. Алгоритми роботи

Алгоритм роботи першого та другого процесора мають бути ідентичними. Приклад алгоритму роботи, а також приклад алгоритму роботи блоку порівняння представлені у вигляді блок-схем на рисунках 4.2 і 4.3 відповідно. Зазначимо, що представлені алгоритми є обов'язковими використання, лише відображають загальні принципи роботи.

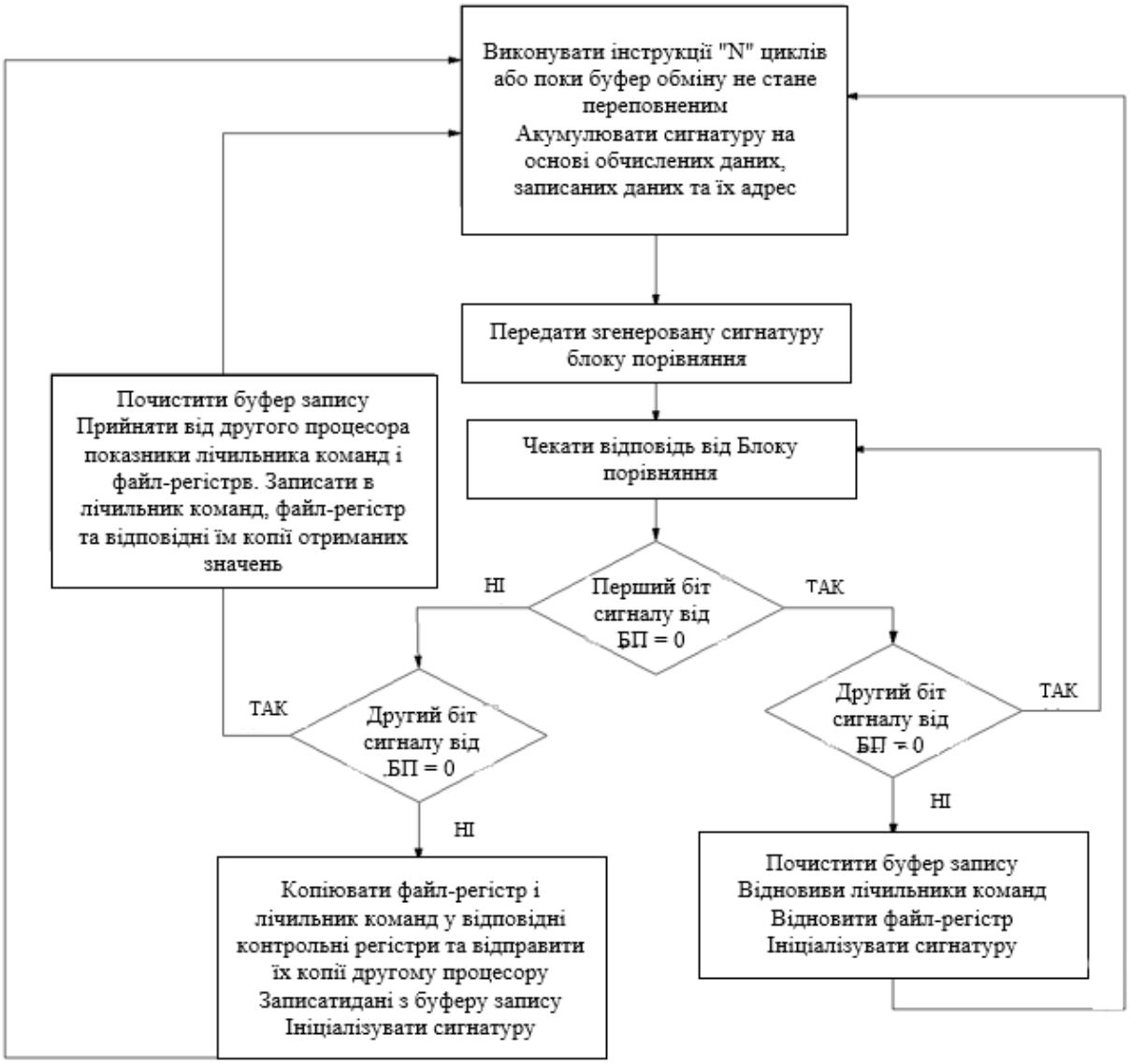

Рис. 4.2. – Алгоритм роботи процесора

Як очевидно зі схеми, кожен процесор виконує команду чи набір команд, генеруючи сигнатуру. Після закінчення виконання кожен процесор передає отриману сигнатуру блоку порівняння і чекає відповіді від нього. Відповідь складається з двох біт, де перший біт вказує на те, чи був отриманий правильний результат на поточному кроці, а друга - як вчинити процесору в кожній з двох ситуацій.

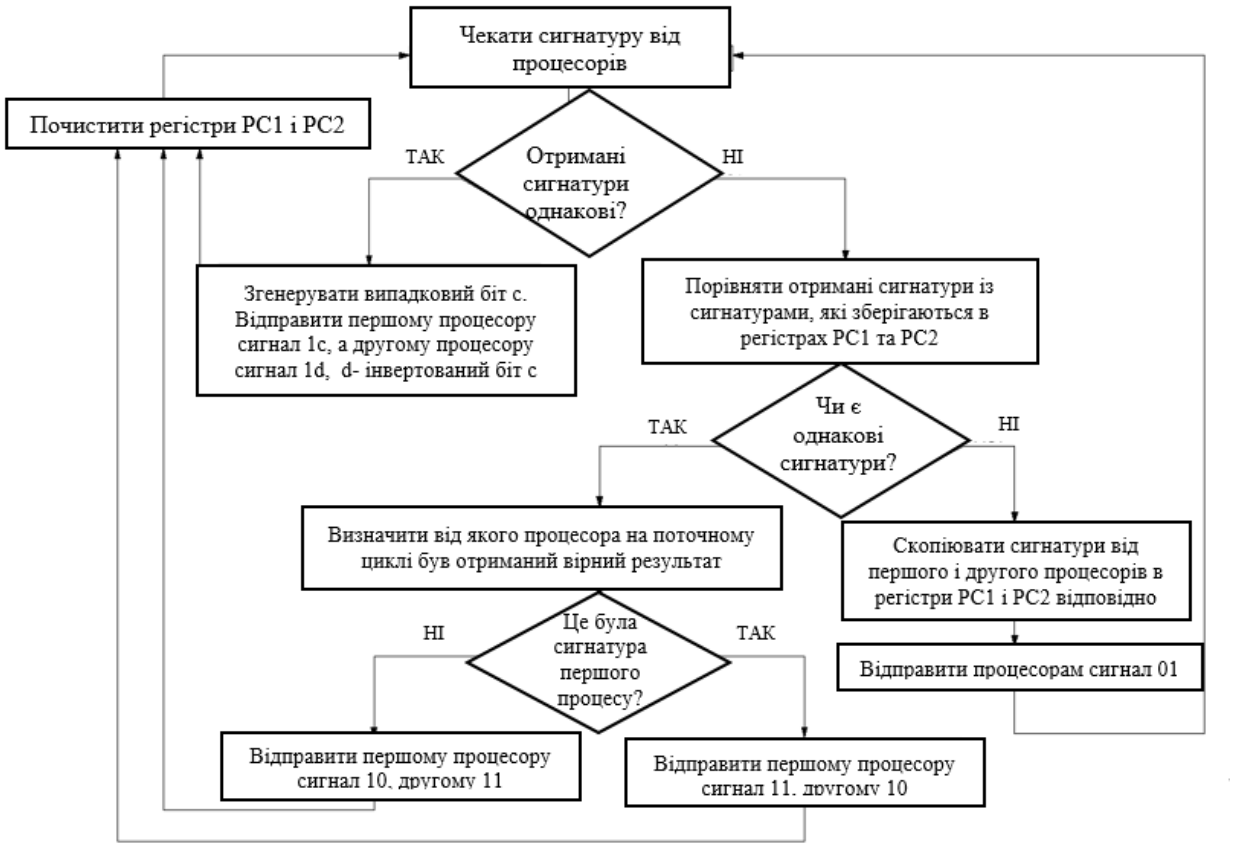

Рис. 4.3. – Алгоритм роботи блоку порівняння

Як видно зі схеми, блок порівняння чекає, поки обидва процесори передадуть йому його сигнатури, після чого порівнює їх. Якщо вони рівні, він вважає їх правильними і випадковому процесору відправляє сигнал 11, а іншому 10. Якщо отримані сигнатури не рівні, тоді вони порівнюються з сигнатурами, що зберігаються в тимчасових регістрах PC1 і PC2. Якщо серед порівнюваних сигнатур не було двох однакових, то блок порівняння записує сигнатури від першого і другого процесора в регістри PC1 і PC2 відповідно, після чого посилає процесорам сигнал 01. Якщо ж серед порівнюваних сигнатур були дві однакові, то блок порівняння визначає, який з процесорів на поточному кроці видав правильний результат і відправляє йому сигнал 11, а іншому процесору відправляється сигнал 10. Зазначимо також, що при прийнятті рішення, регістри PC1 і PC2 очищаються мініціалізуються деякими різними початковими значеннями.

5. ПОРІВНЯННЯ АРХІТЕКТУР

5.1. Розрахунок ймовірностей

Приймемо ймовірність отримання хибної сигнатури на кожному кроці рівноймовірної та рівної p, тоді ймовірність отримання вірної сигнатури q = 1 – p. Виконання команди менш ніж за два цикли неможливе в жодній із трьох розглянутих архітектур. Позначимо через Pi(n) можливість виконання команди на n-тому кроці в і-й архітектурі. Для першого прототипу, а також для його модифікацій виконання команди менш ніж за 2 кроки неможливе і тому Pi(n)=0, при n<2 для будь-якого і. Також відзначимо, що P(2)= q2 для кожної з перших трьох аналізованих архітектури, т.к. відмінності в них спостерігаються тільки при виявленні помилки, а прийняття результату на другому етапі свідчить про її відсутність. Для другого прототипу архітектури, як і для комбінованої, P(1)=q2,оскільки для цього необхідно, щоб обидва процесори видали правильний результат.

5.1.1. Вихідна архітектура

Розглянемо вихідну архітектуру: У ній результат приймається лише на парному числі циклів (2, 4, 6…). На кожній парі кроків (1 і 2, 3 і 4, …) шанс прийняти результат дорівнює q2, отже, шанс відкинути результат дорівнює (1-q2). Для прийняття результату на n-тому кроці необхідно, щоб:

а) Останні два результати були вірними

б) Результат не було ухвалено на попередніх (n-2) кроках. тобто. був відкинутий (n-2)/2 раз.

Таким чином, P_ (n=2k)=q^2 (1-q^2)^(k-1)

5.1.2. Модифікована архітектура

Розглянемо архітектуру 2: У ній результат приймається, якщо він збігається з попереднім результатом. Для прийняття результату на n-тому кроці необхідно що:

Останні два результати були вірними

Попередній результат був невірний (інакше результат був би прийнятий на попередньому кроці)

Результат був прийнятий на (n-3)-х перших кроках.

Таким чином P(n)=pq^2 (1-F(n-3)) , де - функція розподілу, що дорівнює ймовірності отримання правильного результату за n або менше циклів.

5.1.3. Архітектура з використанням мажоритарного блоку

Розглянемо архітектуру з використанням мажоритарного блоку: у ній результат прийматиметься, якщо він збігається з хоча б одним із двох попередніх. Зауважимо, що поточний результат не може дорівнювати одночасно двом попереднім результатам, інакше результат був би прийнятий на попередньому кроці. Нехай результат був прийнятий на n-тому кроці і n>4. Припустимо, що поточний результат дорівнює попередньому:

На n-тому і (n-1)-му кроці було отримано правильний результат

На (n-2)-му та (n-3)-му кроці були отримані невірні результати, інакше результат був би прийнятий на попередньому кроці

За перші (n-4) кроки результат прийнято не було.

Припустимо, що поточний результат дорівнює отриманому на (n-2) кроці:

На n-тому і (n-2)-му кроці було отримано правильний результат

На (n-1)-му (n-3)-му та (n-4)-му кроках були отримані невірні результати, інакше результат був би прийнятий на (n-2)-му або (n-1)- м кроку.

За перші (n-5) кроків результату прийнято не було.

Таким чином, ймовірність прийняти результат на n-му кроці при n>4

P(n)=(pq)^2 (1-F(n-4))+p(pq)^2 (1-F(n-5))

P(n)=(pq)^2 (1-F(n-4)+p-pF(n-5)

Де – функція розподілу, що дорівнює ймовірності отримання правильного результату за n або менше циклів.

5.1.4. Архітектура з використанням двох процесорів

Розглянемо архітектуру другого прототипу, у якому порівнюються сигнатури, згенеровані двома процесорами. На кожному кроці шанс прийняття результату дорівнює q2, так як для прийняття результату необхідно, щоб обидва процесори видали правильний результат.

результат. Для прийняття рішення на n-тому кроці необхідно, щоб

Поточні результати були правильними

Результат не було прийнято на попередніх n-1 кроках

Таким чином: P(n)=q^2 (1-q^2)^(n-1),

5.1.5. Комбінована архітектура

Розглянемо комбіновану архітектуру, у ній результат приймається, якщо серед двох поточних та двох останніх сигнатур будуть хоча б дві однакові. Зауважимо, що можливі такі випадки ухвалення рішення:

На останньому кроці обидва процесори видали однакову правильну відповідь, а на попередньому кроці обидва процесори видали неправильні відповіді.

На останньому кроці обидва процесори видали однакову правильну відповідь, а на останньому кроці була отримана одна правильна відповідь і одна неправильна (через симетричність системи не має значення, який із процесорів на попередньому кроці видав правильну відповідь, а яка неправильна).

На останньому кроці була отримана одна правильна відповідь і одна

неправильна, а на попередньому кроці також була отримана одна правильна і одна неправильна

Також зауважимо, що неможливе виникнення ситуації, в якій на попередньому кроці було отримано дві правильні відповіді, оскільки в цьому випадку результат було б прийнято на попередньому кроці.

Розрахуємо ймовірність виникнення кожної із ситуацій:

Ситуацію 1 і 2 можна об'єднати в одну з того, що для прийняття результату необхідно, щоб було як мінімум два правильні результати, так як на останньому кроці були отримані два правильні результати, то значення на попередньому кроці ніяк не вплинуть на роботу процесора. Для прийняття результату на n тому таким способом необхідно:

Що б на останньому кроці було отримано два правильні результати

Чи не попередніх (n-1)-н кроках результат не був прийнятий

Розглянемо ситуацію 3. Для прийняття результату на n-тому кроці в такий спосіб необхідно, щоб:

На останньому кроці була отримана одна правильна і одна неправильна відповіді

На (n-1)-му кроці була отримана одна правильна і одна неправильна відповідь

На (n-2)-му кроці були отримані дві неправильні відповіді, тому що в іншому випадку результат був би прийнятий на попередньому кроці

На попередніх (n-3) кроках результат не був прийнятий.

Таким чином: $P(n)=q^2\ (1-F(n-1))+4q^2\ p^4\ (1-F(n-3))$, де – функція розподілу, що дорівнює ймовірності отримання правильного результату за n або менше циклів.

5.2. Обчислення середньої кількості кроків

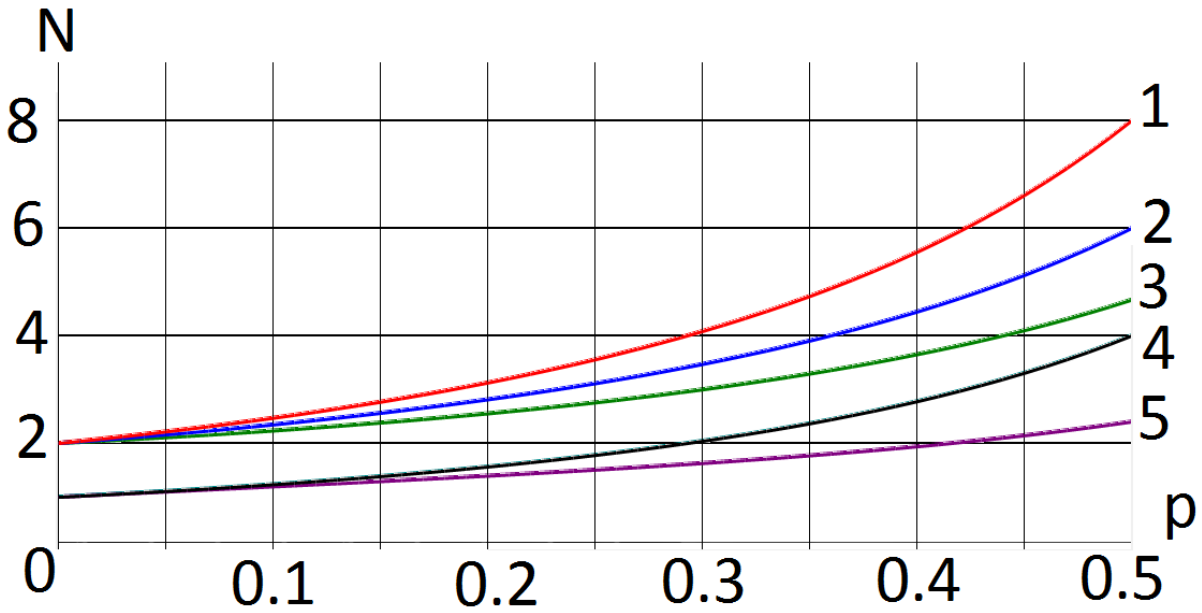

Рис. 5.1. – Середня кількість кроків при p<0,5

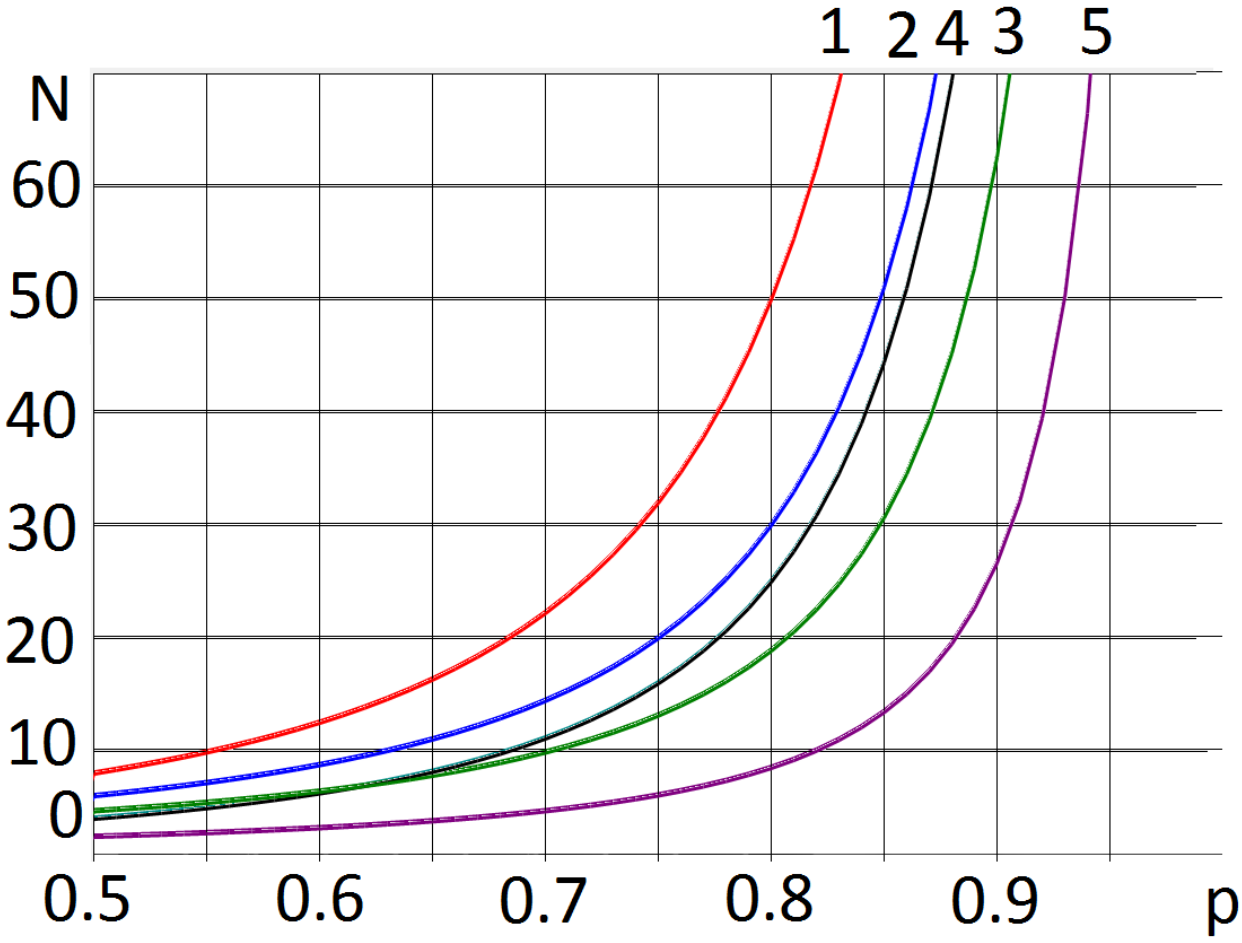

Рис. 5.2 – Середня кількість кроків при p>0,5

Середня кількість кроків, за яку буде виконана команда для будь-якої архітектури, що дорівнює математичному очікуванню функції P(n), відповідної архітектури і залежить від ймовірності помилки p.

На рисунках 5.1 та 5.2 наведено графіки залежності середньої кількості кроків від ймовірності помилки p. Першому прототипу відповідає графік під номером 1, його перша модифікація — під номером 2, модифікація з використанням мажоритарного блоку — під номером 3, архітектура з двома процесорами — під номером 4, а комбінована архітектура — під номером 5.

5.3. Порівняння отриманих результатів

Як видно з рисунків 14 і 15, за будь-якої ймовірності помилки p для комбінованої архітектури середня кількість кроків для отримання результату мінімальна, а для першого прототипу - максимально. Рисунок 16 наочно показує відносне зменшення середньої кількості кроків кожної з архітектур щодо першого прототипу.

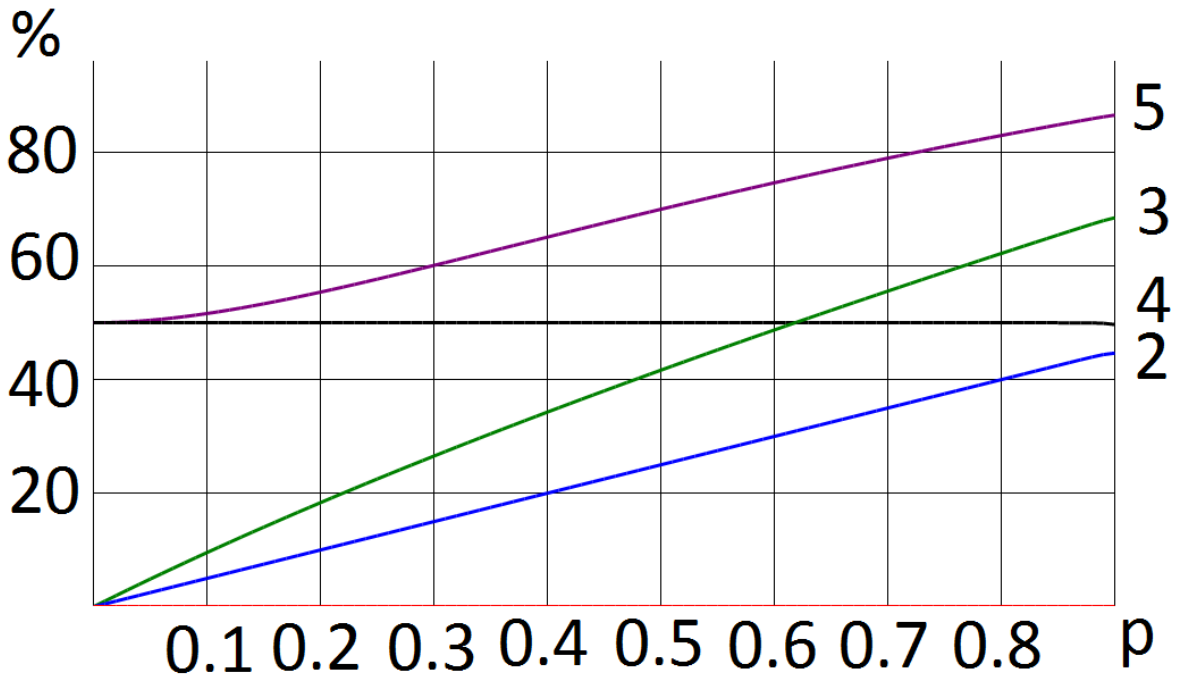

Рис. 5.3. – Відносне зменшення кроків

На рисунку 5.3 кожній архітектурі відповідають графіки під тими самими номерами, що й рисунках 5.1 і 5.2. Для обчислення використовувалася формула (N1-N)/Ni*100%, де N1 – Середня кількість кроків для вихідної архітектури, N – середня кількість кроків для архітектури, що порівнюється.

Очевидно, що для кращої стійкості до відмов і швидкодії системи краще використовувати комбіновану архітектуру, однак для її використання необхідна наявність двох процесорів, що збільшує вартість. Зазначимо, що з використанні архітектури з двома процесорами ми отримуємо приріст швидкодії вдвічі щодо першого прототипу. Таким чином, якщо для системи необхідна максимальна швидкодія та стійкість до відмов, то оптимальним буде використання комбінованої архітектури. Якщо ж стійкість до відмови важлива, але ймовірність помилки при роботі системи буде низькою (система буде працювати в середовищі з низькою зашумленістю), то переважним є використання архітектури з двома прцеосорами, так як при її використанні використовується більш простий алгоритм, не потрібно встановлення додаткових регістрів і при Малої ймовірності помилки середня кількість кроків, необхідних їй задля досягнення правильного результату, практично не відрізняється від середньої кількості кроків, необхідних комбінованій архітектурі. Якщо ж швидкодія не така важлива і система працюватиме в середовищі з невеликою зашумленістю, то оптимальним буде використання першої модифікації першого прототипу, так як при невеликій ймовірності помилки середня кількість кроків, необхідна їй для досягнення правильного результату практично не відрізняється від середньої кількості кроків, необхідного архітектурі з використанням мажоритарного блоку, але при цьому не потрібна наявність додаткового регістру, а також необхідність другого копіювання сигнатури у разі помилки. Якщо ж швидкодія не така важлива, але система працюватиме в середовищі з великою зашумленістю, то оптимальним буде використання архітектури з мажоритарного блоку. Також зазначимо, що з ймовірності помилки більшої, ніж 0.6, архітектура з допомогою мажоритарного блоку працює швидше, ніж з використанням двох процесорів. Таким чином, при

значення помилки меншої, ніж 0,6, але досить близьких до неї, використання архітектури з мажоритарним блоком є кращим, оскільки при невеликій втраті швидкодії ми позбавляємося другого процесора, що значно зменшує вартість всієї обчислювальної системи. Якщо позначити через K(p) співвідношення середньої кількості кроків, необхідних архітектурі з мажоритарним блоком до середнього числа кроків, необхідних комбінованої архітектури при ймовірності p, то $K(p<0.5)<2$, $K(p=0.5)=2$, $K(0.5)>2$. Таким чином, вибір підходящої архітектури залежить від того, наскільки дуже важливою є швидкодія для обчислювальної системи, зашумленість середовища, в якому працюватиме обчислювальна система, а також можливий бюджет, який можна виділити на її проектування.

6. ПЕРЕВІРКА ВИКОНАННЯ ІНСТРУКЦІЙ

Усі аналізовані раніше архітектури визначали помилки виходячи з отриманих результатів чи сигнатур. Така схема перевірки ефективна, якщо помилка відбулася при обчисленнях, проте існують випадки, коли помилково тлумачить і розшифровує інструкції, що спричиняє помилкові дії відповідної обробки. Існують пристрої виявлення та виправлення помилок, що забезпечують виявлення та усунення помилок, які з'являються у даних, що зберігаються у пам'яті. Зазвичай за допомогою перевірки послідовності кадрів (FCS – Frame Checking Sequence). Однак вони не в змозі виявити помилкове виконання отриманої процесором команди.

Таким чином, знадобився пристрій, що забезпечує виявлення помилок декодування і виконання даних інструкцій процесором.

Розглянута система обробки призначена для інтерпретації та виконання набору пов'язаних операцій, що зберігаються у програмі. Передбачається виконання мікрокоманд за один елементарний цикл.

Рис 6.1 показує варіант процесора, в якому команда виконується один елементарний цикл. Процесор має регістр команд 100 (РК). Вміст РК може бути передано до блоку декодування команди (БДК) 102 по шині 170. БДК містить логічну схему для подання. Обмежувач 103 дозволяє передавати дані з шини Z 125 на вхід регістра А 105 сигналу мікрокоманди 114 "Z bus to A". Так само обмежувач 104 забезпечує пересилання вмісту шини Z в регістр В по команді 115 "Z bus to B". Обмежувачі 107 і 108 забезпечують передачу вмісту регістру А або на перший або другий вхід АЛУ за сигналом “A to OP” або “B to OP” відповідно. АЛУ виконує арифметичні та логічні операції під управлінням БДК через сигнали мікрокоманд 117-120. При активації лінії “ADD” 117 АЛУ Виконує додавання виходів обмежувачів 107 та 108, а результат подається на вхід обмежувача 110, який за сигналом 121 “OP to Z” поміщає результат у регістр Z/ При активації лінії “SUB” 118 значення, поданого на 1-й вхід АЛУ віднімається значення подане на 2-й вхід

АЛУ а результат подається на обмежувач 110. При активації інших ліній відбуваються аналогічні дії – при активації лінії 119 “AND” над значеннями відбувається операція логічного І, а при активації лінії 120 “XOR” – виключає або. За сигналом “OP to Z” результат обчислення АЛУ передається через обмежувач 110 регістр Z. Вихід регістра Z з'єднаний з входом обмежувача 112, значення якого за сигналом 122 Z to bus подається на шину Z.

Правильне виконання інструкцій, які у РК, виконується шляхом активації відповідних ліній Блоком ДК. Наприклад, якщо вміст регістру А повинен бути доданий до вмісту регістру, необхідно активувати лінії 116, 115, 117, 121, в той час, як інші лінії не повинні бути активованими. Якщо хоча б одна лінія не перебуватиме у вказаному стані, то процесор найімовірніше вважатиме невірний результат. Як говорилося раніше помилки такого роду, що виникають під час виконання операції, не можу бути виявлені пристроями виявлення помилок.

Для виявлення помилок під час виконання команди процесор містить блок перевірки 130 (БП), який з'єднаний з частиною РК, що містить сигнатуру команди (поле S). Розмір поля залежить від кількості помилок, які користувач хоче виявити. БП з'єднаний з лініями, що управляють 113-122. При декодуванні та виконанні команди БП обчислює сигнатуру на підставі дійсного стану керуючих ліній. За результатами останнього порівняння БП подає сигнал лінії 131, що вказує на відбулася чи ні помилка при виконанні.

На рисунку 6.2 показано приклад реалізації БП. У цьому прикладі сигнатура обмежується одним бітом для простоти. Блок реалізує операцію виключає або над реальним станом керуючих ліній і сигнатурою, що зберігається в полі S. Таким чином, якщо сигнатура сама є результатом виключає або над необхідними станами ліній, при коректному стані ліній на виході буде логічний 0. Лог. 0 так само буде за будь-якої парної кількості помилок. Для більш точного результату можна збільшити розмір поля сигнатури.

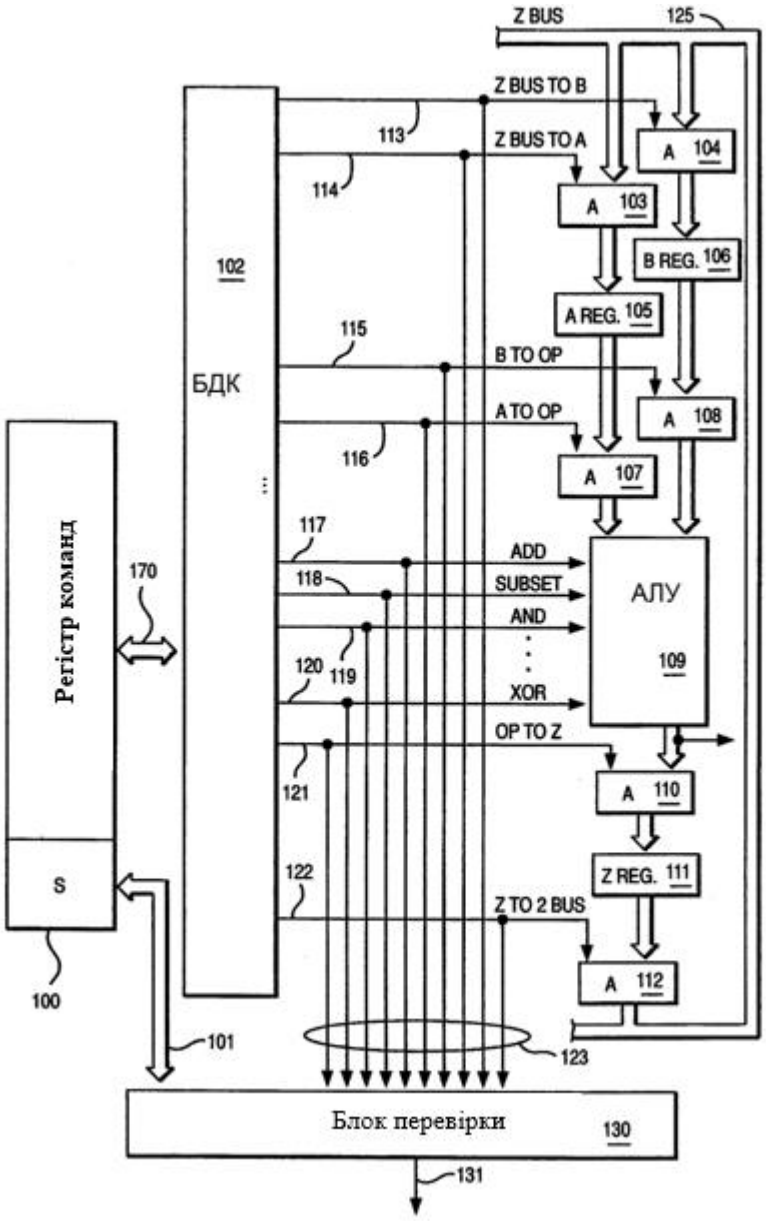

Рис. 6.1. – Схема процесора

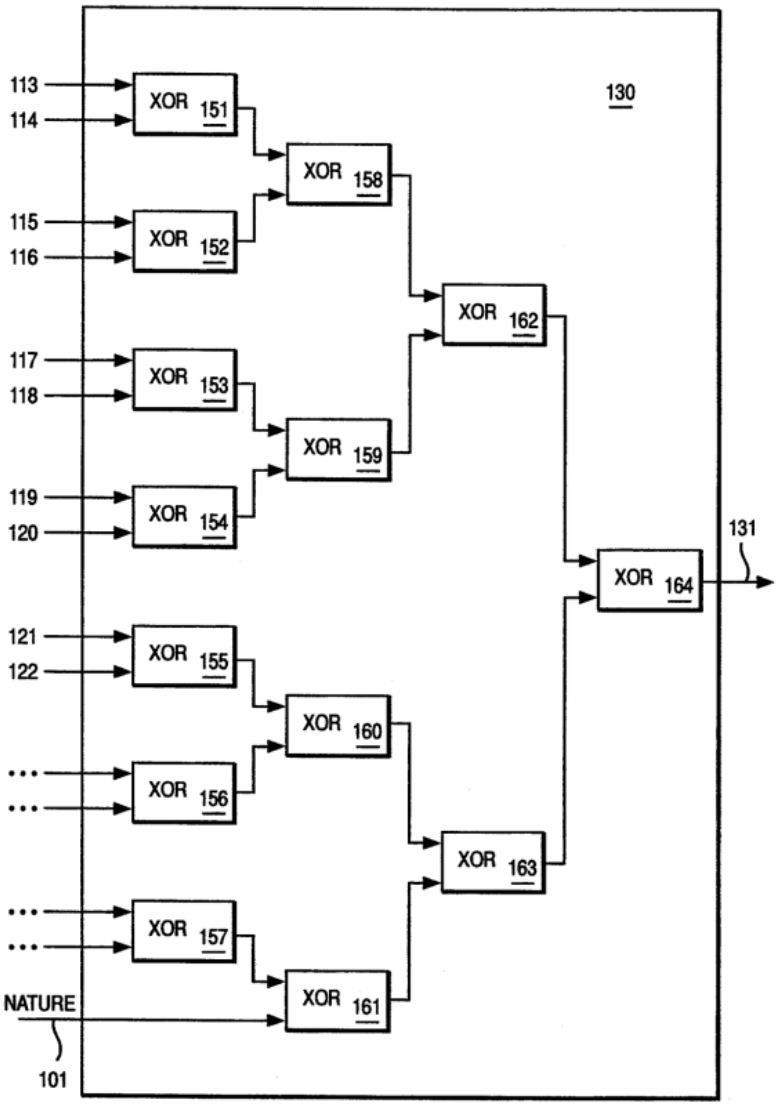

Рис 6.2. – Приклади БП

ВИСНОВОК

В ході виконання поставлених завдань були досліджені методи підвищення перешкодостійкості до відмови обчислювальних систем.

В роботі досліджено та проаналізовано існуючі завадостійкі архітектури; виявлено їх переваги та недоліки; запропоновано модифікації архітектур, що дозволило значно підвищити їх швидкодію; показано підвищення швидкодії пропонованих систем під впливом завад.

Комплекс отриманих результатів дозволяє зробити висновок, що поставлені задачі вирішено, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Затуліветер Ю.С., Фіщенко Є.А., Ходаковський І.А. Розробка та дослідження методів підвищення надійності розподілених обчислень. Журнал "Надійність" 2009 р. Випуск 1, с. 42-49.

2. Ленков С.В., Гунченко Ю.А., Савенчук В.В. Побудова та аналіз завадозахищених процесорних архітектур // Сучасна спеціальна техніка, м. Харків, 2014. - № 1(36). – С. 49 – 60.

3. Пат. № 6,571,363 USA Single event upset tolerant microprocessor architecture / Donald E. Steiss; заявл. December 15, 1999.

4. Уривський Л.О., Шмігель Б.О. Порівняння завадостійкості широкосмугових та вузькосмугових сигналів зв’язку в умовах низької енергетики. Науково-технічна конференція «Проблеми телекомунікацій»: Матеріали конференції. К.: КПІ, 2020.

5. Уривський Л.О., Шмігель Б.О. Визначення завадостійкості СКК в умовах низької енергетики. Науково-технічна конференція «Перспективи телекомунікацій»: Матеріали конференції. К.: КПІ, 2021.

6. Хеннесі, Джон, і Девід Паттерсон. Архітектура комп'ютера: кількісний підхід 5-е вид. 2016 р. 936 с.

7. Таненбаум Ендрю. Сучасні операційні системи 4-е вид. 2019 р. 1120 с.

8. Saurabh Gupta, Pradip Bose A Survey of Fault-Tolerant Techniques for High-Performance Computing. 2018 IEEE International Conference on Robotics and Automation (ICRA). URL: https://ieeexplore.ieee.org/document/8463147

9. Parthasarathy Ranganathan, Norman P. Jouppi Fault-Tolerant Architectures for Space and Terrestrial Applications. Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501). URL: https://ieeexplore.ieee.org/document/889430

10. The Race to Build a Fault-Tolerant Superconducting Quantum Computer URL: https://spectrum.ieee.org/fault-tolerant-quantum-computing

11. Fault Tolerant Computing URL: https://www.sciencedirect.com/topics/computer-science/fault-tolerant-computing

12. Пат. № 6,357,024 США електронної системи і методу для здійснення функціонального redundancy checking по повідомленню повідомлень мають відносно невеликі номери сигналів / Drew J., Dan S., Scott A.; заявл. August 12, 1998.

13. Пат. № 5,388,253 USA Processing system having device for testing the correct execution of instructions / Michel Jacob; Francois Poiraud; заявл. August 26, 1991.

14. Пат 67752 Україна, МПК (2006.01) G06F 12/08. Пристрій підвищення завадостійкості систем з програмним управлінням / Гунченко Ю.О., Мартинюк С.М., Лєнков С.В., Банзак О.В., Омельченко О.С., Купрацевич О.В.; власник Одеський національний університет ім. І.І. Мечнікова.

15. Пат 76984 Україна. МПК (2006.01) G06F 11/27. Відмовостійкий процесорний пристрій з підвищеною швидкодією / Гунченко Ю.О., Лєнков С.В., Кобозєва А.А., Мартинюк С.М., Борисенко І.І. - №u2012 07958, заявл. 27.06.2012, опубл. 25.01.2013, Бюл. №2.