

Винахід стосується мікропроцесорної схеми, відповідної обмежувальній частині пункту 1 формули винаходу, а також способу організації доступу до записаних в запам'ятовувачі даних чи програм згідно з обмежувальною частиною пункту 14 формули винаходу.

Мікропроцесорні схеми вказаного виду знаходять застосування переважно у так званих чіп-картках, тобто картках-посвідченнях, кредитних картках, розрахункових картках і подібних пристроях, оснащених інтегральною мікросхемою. Мікропроцесорні схеми можуть бути використані також у так званих модулях платіжного обороту, які зв'язуються зі згаданими вище картками через відповідні інтерфейси. Загалом мікропроцесорна схема може бути використана у персоналізованих кінцевих пристроях, таких, наприклад, як мобільні телефони, персональні чи кишенькові комп'ютери, придатні для використання у платіжному обороті.

Одна з важливих переваг таких карток полягає в численних можливостях використання з боку власника картки. Розміщені в інтегральній мікросхемі мікропроцесор і приналежні до нього запам'ятовуючі засоби дозволяють здійснювати велику кількість операцій обробки даних у самій картці.

Виробник картки може оснастити мікропроцесор записаною в постійну пам'ять операційною системою, яка бере на себе основні функції, наприклад, процедури порівняння введеного ззовні коду з записаним кодом, або подібні. Приналежні до мікропроцесора запам'ятовувачі служать - окрім запам'ятовування операційної системи - також для запису певних прикладних програм, які можуть складатися із кількох функцій, таких як параметри, необхідні, наприклад, для перевірки безпеки, і в кожному разі мають бути записані з дотриманням секретності.

Багатогранного використання карток можна досягнути шляхом використання операційної системи з приналежними програмами і відповідними інтерфейсами, а також резервування пам'яті чи області пам'яті для зовнішніх програм. Виробник картки надає в розпорядження користувача, тобто організації, що випускає картки, пам'ять або область пам'яті для розміщення зовнішньої програми. В цій зовнішній програмі вказана організація може закласти специфічні операції, які є незалежними від операційної системи і стосуються лише спеціальної організації.

Майбутній сценарій міг би бути таким, що у виготовлену чіп-картку записує свою зовнішню програму не лише одна єдина організація, але й кілька різних організацій записують свої програми. В цьому разі слід потурбуватися про те, щоб суттєві з точки зору безпеки дані, що є складовими операційної системи або окремої зовнішньої програми, були захищені від несанкціонованого доступу.

Можлива ситуація, коли зовнішня програма викликає фрагмент програми, що називається також бібліотекою програм, іншої зовнішньої програми. На основі звичайної організації пам'яті всі фрагменти програм можуть перебувати в так званій однаковій логічній адресній області. При цьому в адресному просторі області пам'яті розміщені як коди, так і дані зовнішньої програми. Сам код, у свою чергу, може складатися з певної кількості фрагментів програм, бібліотек чи функцій. Якщо, наприклад, зовнішня програма А, називана також прикладною програмою, поперемінно викликає програмну бібліотеку В зовнішньої програми В, а також програмну бібліотеку С іншої зовнішньої програми С в однаковому адресному просторі, то без додаткового захисного механізму програмна бібліотека В може під час процесу запису даних зруйнувати програмну бібліотеку С або під час процесу зчитування вивідати секретні дані програмної бібліотеки С. В результаті програмна бібліотека С стала б непридатною для подальшого використання іншими зовнішніми програмами, внаслідок чого цінність програмної бібліотеки була б втрачена.

Для вирішення цієї проблеми відоме розділення фрагментів програм, тобто програмних бібліотек, один від іншого в пам'яті апаратними засобами і надання доступу до бібліотек лише через операційну систему. Однак досягнута таким чином безпека потребує значних програмних і апаратних витрат. До того ж, знижується продуктивність порівняно з безпосереднім зв'язком між програмними бібліотеками. Таким чином, високий рівень безпеки стає причиною погіршення продуктивності.

Із EP 0512542 B1 відома мікропроцесорна схема для перешкоджання доступу до записаних у запам'ятовувачі даних чи програм, яка містить принаймні один мікропроцесор, запам'ятовуючий пристрій для операційної системи і принаймні один запам'ятовуючий пристрій для вільного програмування з індивідуальними зовнішніми програмами. При цьому передбачено кілька вільно програмуваних областей пам'яті, адресні простори яких однакові до щонайменше одного старшого розряду. Крім того, мікропроцесорна схема містить засоби, які кожного разу перед адресуванням області пам'яті завантажують у перший допоміжний регістр старший розряд, поставлений у відповідність області пам'яті, а старший розряд адресованої області пам'яті завантажують у другий допоміжний регістр, і потім здійснюють порівняння вмісту першого і другого допоміжних регістрів. При зміні вмісту допоміжного регістра формується сигнал блокування з метою встановлення факту виконання зовнішньої програми, яка здійснює звертання до забороненої адреси.

Недоліком такого рішення є те, що принципово припиняється доступ до іншої зовнішньої програми. Взаємне використання певного фрагмента іншої зовнішньої програми неможливе. Тому в разі необхідності кілька зовнішніх програм повинні містити ідентичні фрагменти з однаковою функціональністю. В разі мобільних носіїв даних, які, як правило, мають обмежений розмір пам'яті, це є вкрай не вигідним.

В основі винаходу лежить задача розробки схеми і способу, яка і, відповідно, який простими засобами забезпечує доступ до зовнішніх програм.

Згідно з винаходом ця задача вирішена ознаками, наведеними в пунктах 1 і 14 формули винаходу.

Згідно з винаходом ця задача вирішена тим, що кожному адресному простору області пам'яті поставлена у відповідність біт-послідовність, що містить щонайменше одне право доступу. Іншими словами це означає, що кожен логічний адресний простір розділений на щонайменше один адресний блок, якому поставлена у відповідність біт-послідовність, що містить право доступу.

Таким чином, винахід передбачає апаратний захисний механізм, який дозволяє безпосередній зв'язок між фрагментами різних зовнішніх програм, які знаходяться в одному й тому самому логічному адресному просторі. Однак при цьому не потрібен контроль з боку операційної системи.

У переважній формі виконання винаходу кожній адресі адресного простору поставлена у відповідність біт-послідовність, що містить право доступу. Іншими словами це означає, що як кодовому рядку, так і слову даних

поставлена у відповідність біт-послідовність, що містить право доступу. Таким чином певним областям даних селективно може бути присвоєне певне право.

У відповідному варіанті біт-послідовності, що містять право на доступ, разом із адресами чи адресними блоками і кваліфікаторами (ідентифікаторами програм) зведені у таблиці, записані у пам'яті. При цьому адресний блок може бути позначений його початковою чи кінцевою адресою і довжиною блока. Альтернативно адресний блок може бути позначений нижньою і верхньою адресою. Кожному адресному блоку може бути поставлена у відповідність біт-послідовність, що містить право доступу. В разі, коли адресний блок містить лише одну адресу, кожній окремій команді чи кожній окремій адресі може бути поставлена у відповідність біт-послідовність, що містить право доступу.

Біт-послідовність, що містить право доступу, вміщує перше і друге право доступу, причому перше право доступу регулює доступ між двома областями пам'яті, а друге право доступу регулює доступ всередині однієї області пам'яті. Вважаючи, що в різних областях пам'яті розміщені різні зовнішні програми, можна дуже селективно регулювати доступ через фрагменти зовнішньої програми. Доступ може бути заборонений чи дозволений. Сам доступ може містити повноваження на здійснення запису чи зчитування, якщо йдеться про дані, до яких звертається зовнішня програма. Може бути також передбачене регулювання прав на випадок спроби однієї зовнішньої програми визвати код із іншої зовнішньої програми.

У переважній формі виконання винаходу третє право доступу (до даних) в біт-послідовності регулює доступ всередині області пам'яті, тобто в межах однієї програми. Це право пов'язане з другим правом доступу власної програми. Зокрема це можливо в разі необхідності захисту чутливих даних, наприклад, ініціалізаційних даних чи змінних, перед допуском певного фрагмента програми до зчитування чи запису всередині власної програми, коли друге право доступу виконуваного коду у власній програмі не встановлене. Однак зчитування чи запис до інших областей пам'яті, відведених лише для даних, що виникають в процесі виконання програм, може здійснюватися, наприклад, усіма фрагментами програм.

В іншій переважній формі виконання передбачений проміжний запам'ятовувач прав, в якому записані інші права доступу між двома областями пам'яті, причому операційна система має можливість зчитування чи запису прав доступу. Проміжний запам'ятовувач містить переважно дозволені присвоєння, тобто дозволені доступи, між двома різними областями пам'яті. Ця форма виконання передбачає введення таблиці доступу, яка може бути реалізована, наприклад, у формі кеша (проміжного запам'ятовувача). Присвоєння в таблиці можуть бути реалізовані або за допомогою кваліфікаторів (програмних ідентифікаторів) або за допомогою таблиці відповідності, якою встановлюються означення кваліфікаторів.

Прикладна програма А, яка складається із кількох фрагментів, може надати фрагменту В більше прав доступу, ніж фрагменту С. Проміжна пам'ять чи записана в проміжній пам'яті таблиця містить для цього, наприклад, дві колонки і кількість рядків, що залежить від складності програми чи кількості фрагментів програми. В рядку у першій колонці записаний кваліфікатор для фрагмента програми, до якого здійснюється звертання, а в другій колонці записаний кваліфікатор для фрагмента програми, якому має бути наданий доступ. В найпростішому варіанті реалізації ці кваліфікатори ідентичні програмним ідентифікаторам. Однак альтернативно може бути передбачена таблиця відповідності, за допомогою якої могли б бути означені програмні ідентифікатори із інших записів таблиці.

Записана в проміжному запам'ятовувачі прав таблиця доступу може бути реалізована різним чином. В разі двоколонкової таблиці кожна комбінація із різних фрагментів програми може виникати максимум один раз. Альтернативно таблиця може складатися із фрагментів програми, що відповідає кількості фрагментів програми. Дозволені доступи записуються у додаткові колонки таблиці.

В іншій переважній формі виконання винаходу передбачена область пам'яті, керована виключно операційною системою. Ця керована операційною системою область пам'яті, яка називається "пам'яттю для зберігання контексту" (Context Safe Area), служить для проміжного зберігання чутливих даних, які не можуть бути зчитані чи записані жодною іншою зовнішньою програмою чи фрагментом програми. Таким чином, область пам'яті, керована виключно операційною системою, є стеком, використовуваним зокрема тим кодом, третє право доступу якого надано у розпорядження лише особливо вибраним фрагментам програми.

У іншій формі виконання кожній області пам'яті може бути надана область адрес для проміжного зберігання даних, керування якою може бути здійснене лише програмою, записаною у відповідній області пам'яті. І цей проміжний запам'ятовувач (стек) служить для тимчасового зберігання особливо чутливих даних, які в жодному разі не мають бути зчитані іншою зовнішньою програмою.

За допомогою наведеного нижче опису способу виконання мікропроцесорної схеми стане зрозумілішим.

Відповідний винаходів спосіб організації доступу до записаних в запам'ятовувачі даних чи програм в мікропроцесорній схемі, що містить щонайменше один мікропроцесор, запам'ятовувач для операційної системи і щонайменше один запам'ятовувач для вільного програмування з індивідуальними зовнішніми програмами, причому в запам'ятовувачі для вільного програмування передбачено кілька областей пам'яті з відповідними адресними просторами, причому кожному адресному простору присвоєний кваліфікатор (програмний ідентифікатор), і причому в кожній області пам'яті записана одна зовнішня програма, який включає такі кроки:

- a) визначення першого програмного ідентифікатора (PID<sub>PC</sub>) тільки-но виконаної кодової команди зовнішньої програми за допомогою кваліфікатора поточної адреси,
- b) визначення другого програмного ідентифікатора (PID<sub>Ad</sub>) адресованої області пам'яті,
- c) порівняння першого і другого програмних ідентифікаторів,
- d) вибір першого чи другого права доступу (ACR) в залежності від результату порівняння у кроці c),
- e) оцінка права доступу,
- f1) продовження виконання програмного коду, якщо виконання кодової команди чи доступ до адресованої області пам'яті дозволено,
- f2) виклик програми обробки помилок, якщо виконання кодової команди чи доступ до адресованої області пам'яті не дозволено.

Таким чином, з використанням кваліфікатора (який надалі називається програмним ідентифікатором), наприклад, старшого розряду чи старших розрядів, може бути визначено, надається доступ до поточної чи іншої області пам'яті і, тим самим, до іншої зовнішньої програми, чи ні. В залежності від результатів цього аналізу виявляють і оцінюють перше чи друге право доступу. На основі цього способу можна певні фрагменти програм (бібліотеки програм) зробити доступними для інших зовнішніх програм.

В одній із форм виконання відповідного винаходу спосіб при однакових першому і другому програмних ідентифікаторах - в залежності від права доступу тільки-но виконаної кодової команди - вибирають друге або третє право доступу. Надані кодовій команді права доступу вказують, чи може вона звертатися також і до чутливих областей програмного коду. Якщо це так, то далі оцінюють присвоєне адресі вибране третє право доступу і визначають, чи може бути здійснена також і операція запису, чи лише операція зчитування. Таким чином забезпечують захист, який запобігає руйнуванню чутливих даних програмними бібліотеками тільки-но виконаних зовнішніх програм.

В іншій переважній формі виконання відповідного винаходу спосіб після кроку f2) виконують такі кроки:

g) перевірка проміжного запам'ятовувача прав на наявність запису, який представляє дозволений доступ області пам'яті з першим програмним ідентифікатором (PID<sub>PC</sub>) до області пам'яті, позначеної другим програмним ідентифікатором (PID<sub>ADR</sub>),

h) продовження виконання програмного коду, якщо такий запис у проміжному запам'ятовувачі прав є,

i) виклик програми обробки помилок, якщо такий запис відсутній.

Цей переважний варіант відповідного винаходу дозволяє за допомогою проміжного запам'ятовувача прав, наприклад, одному фрагменту В програми надати більше прав доступу, ніж іншому фрагменту С програми.

При звертанні фрагмента А програми до фрагмента В програми спочатку перевіряють наявність першого, другого і третього права доступу. Якщо доступ дозволений, його виконують згідно з відповідними правами. Якщо ж доступ не дозволено, вміст проміжного запам'ятовувача прав перевіряють на наявність запису для доступу фрагмента А програми до фрагмента В програми. В разі наявності такого запису фрагменту А програми надають, наприклад, розширене право доступу до фрагмента програми В. В разі відсутності такого запису у проміжному запам'ятовувачі прав запускають програму обробки помилок.

У переважній формі виконання відповідного винаходу спосіб кодова команда в кроці a) є командою переходу, причому тоді виконують такі кроки:

b) визначення другого програмного ідентифікатора (PID<sub>ADR</sub>) адресованої області пам'яті,

c) порівняння першого і другого програмних ідентифікаторів,

d) перехід на викликану адресу,

e1) продовження виконання програмного коду в адресованій області пам'яті, якщо перший і другий програмні ідентифікатори однакові,

e2) зчитування адресного вмісту адресованої області пам'яті, якщо перший і другий програмні ідентифікатори не однакові,

aa) виклик програми обробки помилок, якщо зчитаний адресний вміст не є командою входу,

bb) продовження виконання програмного коду, якщо зчитаний адресний вміст є командою входу.

Завдяки цьому рішенням забезпечуються переходи між програмними бібліотеками лише на попередньо жорстко задані адреси входів, які означені жорстко заданою командою входу. Завдяки цьому неконтрольований вхід в довільне місце програмного коду неможливий.

В іншій формі виконання способу адресу, під якою записано команду переходу, записують до проміжного запам'ятовувача.

Таким проміжним запам'ятовувачем є переважно керований лише операційною системою запам'ятовувач - згадана вище "пам'ять для зберігання контексту" (Context Safe Area).

Команда переходу є переважно сталою заданою біт-послідовністю. За її допомогою можна визначити - йдеться про адресу входу чи ні.

Крім того, у доцільній формі виконання кодовим командам, яким надано перше право виконання, як проміжний запам'ятовувач надають у розпорядження загальнодоступний проміжний запам'ятовувач (стек). Такою кодовою командою є програмний код, який не може звертатися до чутливих даних програми.

З іншого боку кодовими командами, яким надано друге право виконання, використовується поставлений у відповідність області пам'яті проміжний запам'ятовувач, керування яким може здійснюватися лише програмою, записаною у відповідній області пам'яті. Цей проміжний запам'ятовувач, який називають також "чутливим стеком", служить для проміжного запам'ятовування даних, які в жодному разі не мають бути зчитані чи перезаписані іншими програмними бібліотеками чи зовнішніми програмами.

Ознаки і переваги винаходу детальніше пояснюються далі на прикладі виконання з посиланнями на креслення. На них схематично зображено:

Фіг.1. поділ лінійного адресного простору з двома програмами А і В,

Фіг.2. перший приклад виконання таблиці, що містить права доступу,

Фіг.3. другий приклад виконання таблиці, що містить права доступу,

Фіг.4. принципова структура області пам'яті, причому кожній адресі поставлена у відповідність біт-послідовність, що містить право доступу,

Фіг.5. таблиця з можливими правами доступу,

Фіг.6. відповідний винаходові спосіб у формі блок-схеми програми,

Фіг.7. приклад виконання відповідної винаходові мікропроцесорної схеми.

Фіг.1 ілюструє проблематику, покладену в основу винаходу. В лінійному адресному просторі знаходяться дві програми А і В. Виходимо із того, що приналежні до програми А фрагменти програми чи програмні бібліотеки з відповідними змінними чи незмінними даними організовані таким чином, що вони не руйнуються внаслідок взаємного виклику і взаємного керування. В даній Фіг.1 програма А має область, яка містить лише командний код і позначена як Код А. Він може містити кілька незалежних одна від іншої функцій А1, А2. Вони

можуть викликатися взаємно. Крім того, програма А містить, наприклад, дві області даних А1, А2, які можуть містити змінні або сталі дані. Всі розміщені в програмі А дані можуть повністю керуватися і контролюватися командним кодом або функціями.

Відповідним чином побудована і програма В, причому на Фіг.1 для прикладу представлена лише одна область В даних. Наявні в програмі В функції також можуть взаємно викликати одна іншу, а також керувати і контролювати дані, записані в області В.

Відповідно до винаходу програма А і програма В знаходяться у відведеній кожній з них області пам'яті, кожна з яких відрізняється старшим розрядом чи розрядами або присвоєним програмним ідентифікатором (PID). Самозрозуміло, що всередині цієї області пам'яті зовсім не обов'язково командний код, а також дані мають бути розміщені блоками, як показано на Фіг.1.

Винахід відкриває можливість доступу функції програми В до даних програми А. Для прикладу на Фіг.1 зображено два таких звертання. Функція В1 звертається до області А2 даних, а функція В2 звертається до області А2 даних. Програма А, яка надає дані у розпорядження програми В, може згідно з винаходом селективно надавати певні права доступу до певних областей даних іншим програмам. Можливі права доступу представлені на Фіг.5. Ними є права запису-зчитування (W), право лише зчитування (R), а також відсутність права доступу, тобто ні зчитування, ні запису (-). Право доступу програми В до програми А може бути перевірено апаратними засобами шляхом порівняння програмного ідентифікатора (наприклад, старшого розряду чи розрядів) виконуваної кодової команди і адреси, до якої здійснюється звертання.

В залежності від результату порівняння програмних ідентифікаторів може бути встановлено, функція якої програми здійснює звертання до даних - виконуваної програми чи іншої програми. Відповідно до винаходу тепер кожній області даних (як кодовим командам, так і даним) поставлена у відповідність біт-послідовність, що містить право доступу.

Ця біт-послідовність може бути частиною рядка слова, як показано на Фіг.4. Додатково до адреси і закладеного в ньому значення такий рядок слова може бути подовжений на біт-послідовність права доступу (ACR). Альтернативно в пам'яті може бути передбачена таблиця, яка визначає область пам'яті, і містить поставлену у відповідність цій області пам'яті біт-послідовність з правом доступу. При цьому область адрес може бути задана верхньою і нижньою адресами (Фіг.2) або альтернативно початковою адресою і довжиною адресного блока (Фіг.3).

Біт-послідовність, що містить права доступу, містить щонайменше два права доступу: перше право доступу, яке регулює доступ між двома програмами, і друге право доступу, яке регулює доступи в межах однієї програми. Послідовність, в якій права доступу розміщені в межах біт-послідовності (ACR), може бути вибрана довільною. У переважному варіанті біт-послідовність, що містить права доступу, вміщує ще одне, третє право доступу, яке регулює доступ в межах однієї програми. На відміну від другого права доступу, таким чином позначають адресні області, які містять чутливі дані, наприклад, ініціалізаційні дані. Для того, щоб кодова команда могла зчитувати чи записувати такі дані, така кодова команда також потребує спеціального позначення.

Відповідна Фіг.2 біт-послідовність, що містить права доступу, вміщує перше, друге і третє права доступу, причому вони занесені до таблиці у вказаній послідовності справа наліво у колонці ACR. Область А даних, що містить, наприклад, ініціалізаційні дані, може бути зчитана функціями зовнішньої програми (перше право доступу: R), тоді як функції даної програми потребують навіть права запису. Це дійсно як для таких кодових команд, що позначені нормальним, першим правом виконання ( $x_n$ ), так і для таких, що містять повноваження на виконання чутливих даних ( $x_s$ ) (друге право виконання).

Крім того, із Фіг.2 видно, що в області А1 даних запис можуть здійснювати функції як виконуваної програми А, так і функції зовнішніх програм (перше, друге і третє право доступу: W).

З використанням прав доступу, поставлених у відповідність окремим областям даних лінійного адресного простору, простим чином дані чи функції можуть бути зроблені доступними іншим програмам. Якщо зовнішній програмі доступ заборонено, що встановлюється правом доступу, то запускається програма обробки помилки. Таким чином програміст сам може захистити кодові команди і вибрані дані однієї програми від здійснення запису іншою програмою.

Інший відповідний винахोdв захисний механізм стосується контролю викликів функцій із зовнішніх програм. Для запобігання неконтрольованому ходу виконання програми і можливому пошкодженню власних даних внаслідок перезаписування функція всередині програми іншою програмою може бути переведена лише на певні адреси входу. Ці адреси входу позначені попередньо заданими біт-послідовностями. При отриманні команди входу вказана адреса перевіряється на наявність біт-послідовності. Якщо вона відсутня, запускається програма обробки помилок. Якщо очікувана біт-послідовність є, то викликана функція може виконуватися.

Оскільки функції в рамках програми можуть бути викликані з будь-якого місця, при кожному виклику функції слід записувати адресу повернення. Коли під час обробки кодової команди функції зустрічається інструкція повернення, лічильник програм встановлюється на попередньо записану адресу повернення. Шляхом маніпулювання з адресою повернення міг би бути небажаним чином змінений хід виконання програми.

Для вирішення цієї проблеми винахід пропонує захищати адреси повернення від внесення записів іншими програмами шляхом автоматичного запису адрес повернення при виклику функції до проміжної пам'яті, до якої має доступ лише операційна система. При цьому достатньо записувати до захищеної від запису проміжної пам'яті лише ті адреси повернення, які відповідають викликам функцій із інших програм. На противагу цьому при викликах функцій всередині тієї самої програми може бути використана загальнодоступна проміжна пам'ять. Однак при цьому слід забезпечити, щоб ці адреси повернення не викликалися із поточної програми. Як подальша альтернатива, адреса повернення при виклику функції може бути автоматично записана до проміжного регістру, причому старе значення проміжного регістру відкладається до окремого стеку з адресами повернення. Потім доступ до проміжного регістру для викликаної програми має бути здійснений так само обмежено, як і до керованої операційною системою проміжної пам'яті.

Відповідний винахідові спосіб ще раз пояснюється з використанням Фіг.6. Команду із кеша інструкцій завантажують до центрального процесора, а програмний ідентифікатор ( $PID_{PC}$ ) зберігають у внутрішньому регістрі центрального процесора. В наступному кроці здійснюють перевірку, чи є кодова команда командою переходу. Якщо це так, визначають програмний ідентифікатор викликаної адреси ( $PID_J$ ). При збігові обох програмних ідентифікаторів здійснюють перехід на викликану адресу і виконання програми продовжується. Однакові програмні ідентифікатори означають, що перехід здійснюється всередині якраз активної програми. В разі різних програмних ідентифікаторів здійснюють перехід на викликану адресу і зчитують її вміст. Якщо цей вміст адреси є очікуваною командою входу, тобто попередньо заданою біт-последовністю, програмний код продовжують. В іншому разі викликають програму обробки помилок.

Якщо кодова команда є не командою переходу, а командою з доступом до даних, то із кеша даних в центральний процесор завантажують і оцінюють програмний ідентифікатор викликаної адреси. Якщо програмні ідентифікатори  $PID_{PC}$  і  $PID_{Adr}$  не збігаються, то йдеться про спробу доступу до даних з боку зовнішньої програми. Таким чином виявляють і оцінюють перший біт доступу або із адреси кеша даних, або із таблиці. Якщо дозволено доступ, наприклад, зчитування даних, причому перше право доступу є R або W, то виконання програмного коду продовжують. Якщо ж перший біт доступу у вказаному прикладі не надає права ні для зчитування, ні для запису, викликають програму обробки помилок.

Замість виклику програми обробки помилок в цьому місці може бути здійснена подальша перевірка прав. Тому у таблиці, записаній у проміжному запам'ятовувачі прав, перевіряють записи на наявність пари прав із викликаного чи викликаючого фрагмента програми. Якщо така пара прав є, то фрагменту програми з програмним ідентифікатором  $PID_{PC}$  надають, наприклад, розширений доступ до фрагмента програми з програмним ідентифікатором  $PID_{ADR}$ . Таким чином, є можливість одному фрагменту програми надати більше прав доступу, ніж іншому фрагменту програми з такими ж повноваженнями. Однак крім того існує описана вище процедура розпізнавання для чутливих прав доступу (третій біт доступу). Якщо ж в проміжному запам'ятовувачі прав не буде виявлено жодного запису з програмними ідентифікаторами  $PID_{PC}$  і  $PID_{ADR}$ , то запускають програму обробки помилок.

Описана на початку перевірка прав може бути виконана також паралельно до описаної тут розширеної перевірки прав. Доступ до викликаного фрагмента програми буде дозволений тоді, коли нормальна чи розширена перевірка прав дасть позитивний результат. В разі негативного результату обох перевірок прав запускають програму обробки помилок.

В разі збігу програмних ідентифікаторів  $PID_{PC}$  і  $PID_{ADR}$  йдеться про доступ всередині якраз активної програми. Якщо кодова команда має право виконання, яке лише дозволяє доступ до нормальних даних, із кеша даних зчитують і оцінюють другий біт доступу до адреси. Якщо кодова команда має право обробляти також і чутливі дані, що позначається другим правом виконання ( $x_s$ ), то із біт-последовності, що містить права доступу, зчитують і оцінюють третє право доступу. В разі наявності доступу в обох випадках виконання програмного коду продовжують. В іншому разі здійснюють виклик програми обробки помилок.

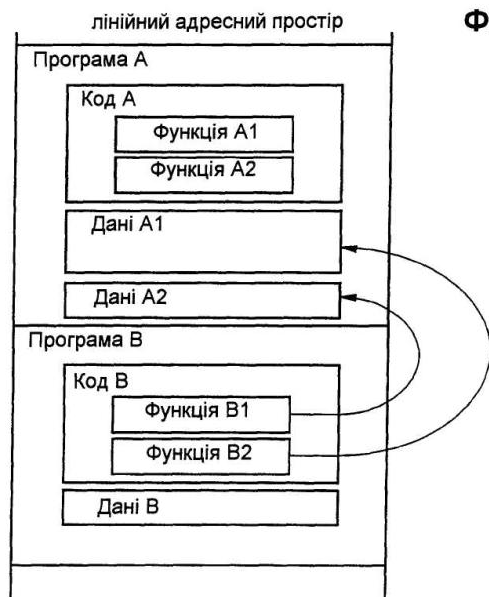
На Фіг.7 представлена структурна схема відповідної винахідові мікропроцесорної схеми. Мікропроцесор 1 містить блок 3 керування, який через керуючу шину 10 зв'язаний з запам'ятовувачем 2. Запам'ятовувач 2 має призначену для операційної системи область 21, для прикладу три області пам'яті 22, 23, 24 для зовнішніх програм, керований виключно операційною системою проміжний запам'ятовувач 25, використовуваний усіма програмами проміжний запам'ятовувач 26, а також проміжні запам'ятовувачі 27, 28, 29 (так звані "чутливі стеки"), поставлені у відповідність областям пам'яті 22, 23, 24 відповідно і керовані лише програмами, розміщеними у цих областях пам'яті. Крім того, мікропроцесор має центральний процесорний блок 6 (CPU), кеш 4 інструкцій, кеш 5 даних, а також обчислювальні блоки 7, 8 з допоміжними регістрами 72, 82 відповідно. Мікропроцесор 1 містить також блок 9 оцінки, зв'язаний з обчислювальними блоками 7, 8 і арифметико-логічним обчислювальним блоком 6.

За допомогою обчислювального вузла 7 здійснюється визначення програмного ідентифікатора адреси тільки-но виконаної кодової команди і запис до допоміжного регістра 72. Для цього адреса із кеша 4 інструкцій через арифметико-логічний обчислювальний блок 6 подається до обчислювального блока 7. Відповідним чином обчислювальним блоком 8 визначається програмний ідентифікатор адреси, що підлягає виконанню, і відкладається в допоміжному регістрі 82. Адреса, що підлягає виконанню, подається до обчислювального блока 8 через арифметико-логічний блок 6 із кеша даних. Вмісти регістрів 72, 82 подаються на блок 9 оцінки і там порівнюються між собою. Блок оцінки формує твердження стосовно того, чи має бути здійснений доступ між двома програмами, чи всередині однієї програми. На блок 9 оцінки подаються також права доступу кодової команди, а також адреси, що підлягає виконанню. На основі цієї інформації блок 9 оцінки виконує зображену на Фіг. 6 програму і приймає рішення стосовно дозволу чи заборони доступу. Цей результат подається до центрального процесорного блока.

Позиційні позначення

- 1 Мікропроцесор
- 2 Запам'ятовувач
- 3 Блок керування
- 4 Кеш інструкцій
- 5 Кеш даних
- 6 Центральний процесорний блок
- 7 Обчислювальний блок
- 8 Обчислювальний блок
- 9 Блок оцінки
- 10 Керуюча шина
- 21 Область пам'яті
- 22 Область пам'яті
- 23 Область пам'яті

- 24 Область пам'яті
- 25 Проміжний запам'ятовувач
- 26 Проміжний запам'ятовувач
- 27 Проміжний запам'ятовувач
- 28 Проміжний запам'ятовувач
- 29 Проміжний запам'ятовувач
- ACR Біт-послідовність, що містить права доступу
- 72 Перший допоміжний регістр
- 82 Другий допоміжний регістр



ФІГ. 1

нижня адреса	верхня адреса	ACR	
		W W R	Дані А2 (ініціалізаційні дані)
		W W W	Дані А2
		- Xn -	Функція А2
		- - Xn	Функція А1
		W W R	Дані В
		Xs - -	Функція В2
		Xs - -	Функція В1

ФІГ. 2

Початкова адреса	Довжина адресного блоку	ACR	
		W W R	Дані А2 (ініціалізаційні дані)
		W W W	Дані А2
		Xn - -	Функція А2
		- - Xn	Функція А1
		W W R	Дані В
		Xs - -	Функція В2
		Xs - -	Функція В1

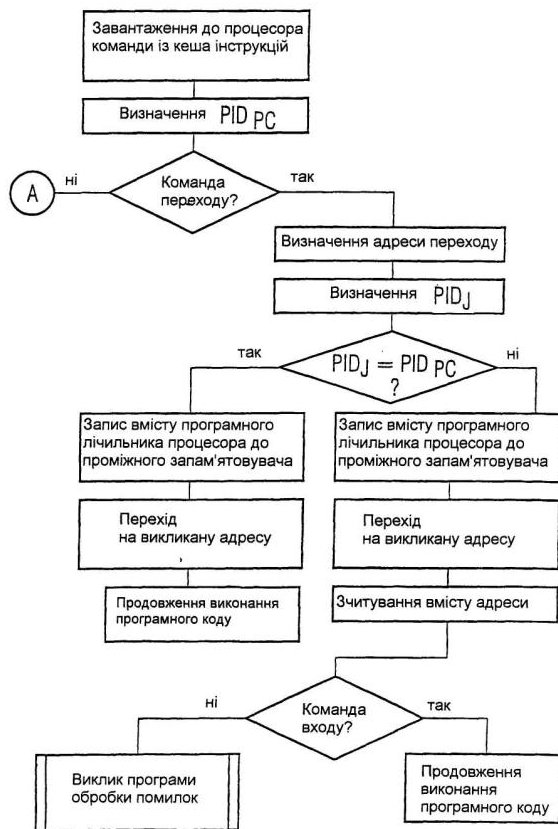
ФІГ. 3

Нижня адреса	Верхня адреса	ACR
		R W
		R W W

ФІГ. 4

Можливі права доступу	
W	Право запису/зчитування
R	Право лише зчитування
-	Жодного права
Xn	Право виконання
Xs	Право виконання

ФІГ. 5



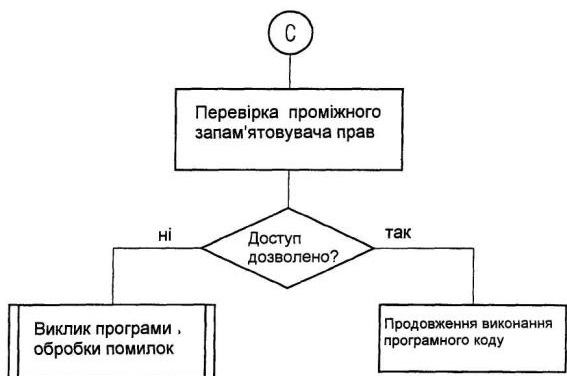
ФІГ. 6А



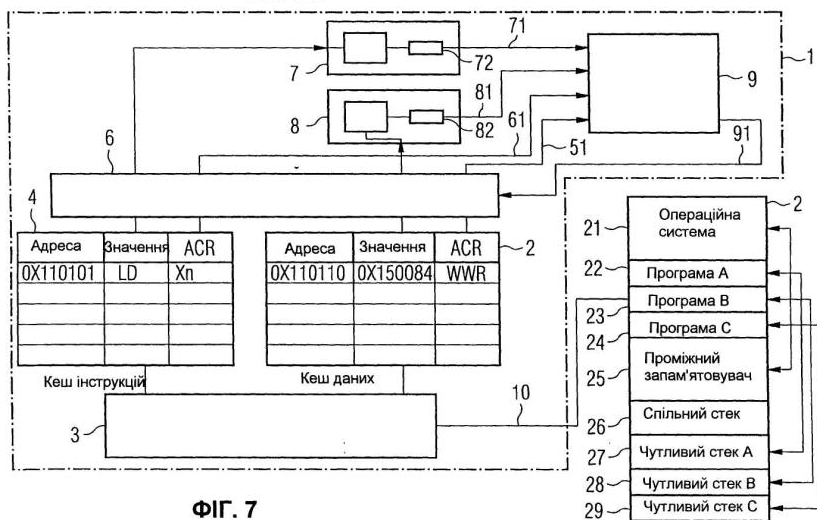
ФІГ. 6В



ФІГ. 6С



ФІГ. 6D



ФІГ. 7